НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Теплоенергетичний факультет
Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено:
Завідувач кафедри
_____ Олександр Коваль
«___»_____2020 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Геометричне моделювання в інформаційних системах»

спеціальності 122 «Комп'ютерні науки та інформаційні технології»
на тему: «Інструментальні засоби підключення реляційної бази даних до онтології предметної області»

Виконав (-ла):
студент (-ка) IV курсу, групи ТР-61
Шестєріков Ілля Валерійович _____

Керівник:
Ст. викладач Дацюк Оксана Антонівна _____

Рецензент:
_____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2020 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки 122 Комп’ютерні науки та інформаційні технології

Спеціалізація Геометричне моделювання в інформаційних системах

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр Коваль
(підпис)
”___”_____2020р.

ЗАВДАННЯ
на дипломну роботу студенту

_____Шестєрікову Іллі Валерійовичу_____
(прізвище, ім’я, по батькові)

1. Тема роботи ____“Інструментальні засоби підключення реляційної бази даних до онтології предметної області”_____
_____
_____

керівник роботи_____старший викладач Дацюк Оксана Антонівна_____
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”25” травня 2020р. № **1168-с**

2. Строк подання студентом роботи _____
3. Вихідні дані до роботи _персональний комп’ютер під керуванням операційної системи Windows, мова програмування JavaScript, мова програмування Java
4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) _проаналізувати існуючі програмні рішення та можливі засоби реалізації взаємодії, обґрунтувати обрані програмні застосунки та шляхи розробки програмних додатків, розробити програмне забезпечення, розробити користувацький інтерфейс, зробити висновки за результатами роботи_____
5. Перелік ілюстративного матеріалу
архітектура проекту, графічне представлення інтерфейсу, приклади роботи програмного модулю_____

7. Дата видачі завдання ”___”_____202___ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	1.10.2019	
2.	Вивчення та аналіз задачі	14.10.2019-23.12.2019	
3.	Розробка архітектури та загальної структури системи	2.02.2020-3.03.2020	
4.	Розробка структур окремих підсистем	4.03.2020-14.04.2020	
5.	Програмна реалізація системи	18.04.2020-14.05.2020	
6.	Оформлення пояснювальної записки	16.05.2020-3.06.2020	
7.	Захист програмного продукту	9.09.2020	
8.	Передзахист	9.09.2020	
9.	Захист	15.06.2020	

Студент _____ ___Щестєріков І. В._____
(підпис) (прізвище та ініціали,)

Керівник роботи _____ _____Дацюк О. А._____
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

Обсяг дипломної роботи складає 57 сторінок, 13 рисунків, 3 додатки та 5 посилань. Мета роботи – створити програмний додаток, який дозволяє підключити реляційну базу даних до онтології предметної області для проведення обміну даними. У ході роботи було розглянуто редактор існуючі рішення, а саме редактор онтологій Protégé. Проаналізовано підходи до реалізації взаємодії онтології та реляційних баз даних. Розроблено систему спільної роботи онтології та реляційної бази даних. Ключові слова: онтологія, реляційна база даних, Protégé, OWL, class, property, rdf mapping individuals, SPARQL.

ABSTRACT

The volume of the thesis is 57 pages, 13 figures, 3 appendices and 5 references. The purpose of the work is to create a software application that allows you to connect a relational database to the ontology of the subject area. In the course of the work, the editor of the existing solutions, namely the editor of ontologies Protégé, was considered. Approaches to the implementation of the interaction of ontology and relational databases are analyzed. A system of joint work of ontology and relational database has been developed. Keywords: ontology, relational database, Protégé, OWL, class, property, rdf mapping individuals, SPARQL.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API — Application programming interface

UML — Unified Modeling Language

OBDB — Ontologies Based Databases

OWL — Web Ontology Language

RDF — Resource Description Framework

RDFS — Resource Description Framework Schema

SQL — Structured Query Language

SPARQ — is an RDF query language

JDK — Java Development Kit

IDE — Integrated Development Environment

XML — Extensible Markup Language

JVM — Java Virtual Machine

MVC — Model View Controller

JDBC — Java DataBase Connectivity

REST — Representational State Transfer

HTTP — Hyper Text Transfer Protocol

СКБД — Система керування базами даних

RxJS — Reactive Extensions For JavaScript

ORM — Object-Relational Mapping

URI — Uniform Resource Identifier

HTML — HyperText Markup Language

CSS — Cascading Style Sheets

W3C — World Wide Web Consortium

ВСТУП

У контексті комп'ютерних та інформаційних наук онтологія визначає набір репрезентативних примітивів, за допомогою яких можна моделювати область знань чи дискурсу. Представницькі примітиви – це типи класів (або множин), атрибутів (або властивостей) та відносин (або відносин між членами класу). Визначення репрезентативних примітивів містять інформацію про їх значення та обмеження щодо їх логічно послідовного застосування. У контексті систем баз даних онтологію можна розглядати як рівень абстрагування моделей даних, аналогічних ієрархічній та реляційній моделям, але призначених для моделювання знань про індивідів, їх атрибути та їх відносини до інших осіб [1]. Онтології зазвичай задаються мовами, які дозволяють абстрагуватися від структур даних та стратегій реалізації. На практиці мови онтологій виразніше підходять до логіки першого порядку, ніж мови, що використовуються для моделювання баз даних. З цієї причини, як говорять, онтології знаходяться на "семантичному" рівні, тоді як схема бази даних є моделями даних на "логічному" або "фізичному" рівні. Через свою незалежність від моделей даних нижчого рівня онтології використовуються для інтеграції різнорідних баз даних, що дозволяє взаємодіяти між різними системами та визначати інтерфейси для незалежних служб, заснованих на знаннях. Не зважаючи на це, онтологія може містити певні недоліки пов’язані з використанням класів та їх відношень [2]. Тому для роботи з онтологіями потрібні певні навички, а заповнення даними потребує витрачання великого часу. Проблемою може стати, як занесення даних, так і робота з великим об’ємом інформації. Реляційні бази даних (РБД) з цим справляються краще. Існують певні подібності між онтологіями та базами даних. Завдяки цим паралельним можливостям ми можемо перетворити базу даних в онтологію і навпаки. Цей тип передачі можливий, коли інформація, що зберігається в онтології, відповідає даним, що зберігаються в базі даних. Такий підхід роботи

з даними може автоматизувати роботу та за зайняти меньше часу на виконання задач. Програмний продукт був реалізований за допомогою мов програмування Java, JavaScript та фреймворків Angular 7, Spring. Для аналізу файлу онтології була вибрана бібліотека edu.Standford.Protege. У завдання дипломного проекту входять:

- ознайомитися із існуючими рішеннями та виявити основні недоліки, які виникають при роботі з даними в онтології;
- створити ієрархію онтології;
- розробити конвертацію типів даних із онтології до реляційної бази даних;
- розробити програмний продукт підключення онтології до реляційної бази даних.

1 ЗАДАЧА РОЗРОБКИ ІНСТРУМЕНТАЛЬНОГО ЗАСОБУ ПІДКЛЮЧЕННЯ РЕЛЯЦІЙНОЇ БАЗИ ДАНИХ ДО ОНТОЛОГІЇ ПРЕДМЕТНОЇ ОБЛАСТІ

Метою розробки є створення програмного додатку, який дозволяє підключитися до існуючої бази даних маючи модель в онтології або перенести дані із онтології .owl формату реляційної бази даних до реляційної бази даних. Покращити процес систематизованого відображення онтології порівняно з аналогами.

Призначенням даного програмного засобу є відображення даних використовуючи перенесенні екземпляри класів онтології до реляційної бази даних або отриманні дані з підключеної онтології.

Програмний засіб не залежить від операційної системи комп'ютера, так як це WEB-додаток використати його може кожний в кого є браузер.

Вхідні дані: файл онтології у форматі *.owl, SPARQL-запит переведений SQL-запит.

Вихідні дані: результат виконання запиту.

Необхідними можливостями, які має забезпечувати модуль, є:

- завантаження файлу онтології з комп'ютера користувача;
- аналіз файлу онтології;
- підключення до реляційної бази даних;
- переведення SPARQL-запиту в SQL-запиту для отримання даних з реляційної бази даних;
- відображання результату запиту.

2 АНАЛІЗ ПРОБЛЕМИ ПІДКЛЮЧЕННЯ РЕЛЯЦІЙНОЇ БАЗИ ДАНИХ ДО ОНТОЛОГІЇ

2.1 Опис предметної області

Онтологія тлумачиться як явна специфікація концептуалізації, тобто абстрактного представлення предметної області, спільне розуміння певної сфери зацікавленості. Це угода про спільне використання понять, що містить засоби подання предметних знань. На формальному рівні онтологія складається з наборів понять і тверджень про ці поняття, на основі яких можна будувати класи, об'єкти, відношення, функції та теорії. Онтологія, як зразок домовленості про семантику предметної області, сприяє встановленню коректних зв'язків між значеннями елементів предметної області, створюючи умови для їх спільного використання. Онтологія – база знань спеціального виду з семантичною інформацією певної предметної області [3]. Компоненти, з яких складаються онтології, залежать від парадигми подання. Але практично всі моделі онтологій містять певні концепти (поняття, класи), властивості концептів (атрибути, ролі), відношення між концептами (залежності, функції) та додаткові обмеження, що визначаються аксіомами (Рисунок 2.1).

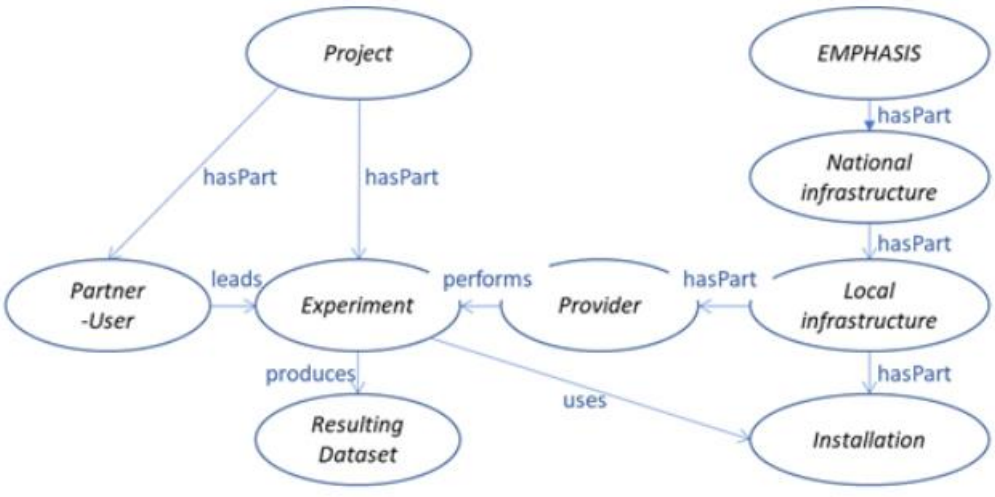

Рисунок 2.1 – Представлення онтології у вигляді графу

Концептом може бути опис задачі, функції, дії, стратегії, процесу міркування тощо. Формально онтологія складається з термінів, організованих у таксономію, їх визначень і атрибутів, а також пов'язаних з ними аксіом та правил виведення. Формальна модель онтології (O) – це упорядкована трійка O = < T, R, F >, де T – скінченна множина термінів предметної області, яку описує онтологія O; R – скінченна множина відношень між термінами заданої предметно області; F – скінченна множина функцій інтерпретації, заданих на термінах і/або відношеннях онтології O [4] (Рисунок 2.2).

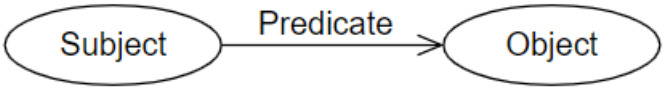

Рисунок 2.2 – Триплет онтології

Класи використовуються для групування екземплярів, які мають щось спільне для того, щоб звернутися до них. Отже, класи по суті представляють сукупність осіб. У моделюванні на заняттях часто використовують позначення набору предметів, що складаються з поняття людського мислення.

Властивості відповідно до того, чи відносять вони індивідів до індивідів (властивості об'єкта) або індивідів до типів даних (властивості типу даних). Властивості типу даних можуть варіюватися за літералами RDF або простими типами, визначеними відповідно до типів даних XML Schema. Найпопулярніші типи які використовуються в онтологіях:

- http://www.w3.org/2001/XMLSchema#boolean,
- http://www.w3.org/2001/XMLSchema#byte,
- http://www.w3.org/2001/XMLSchema#dateTime,
- http://www.w3.org/2001/XMLSchema#dateTimeStamp,
- http://www.w3.org/2001/XMLSchema#decimal,
- http://www.w3.org/2001/XMLSchema#double,
- http://www.w3.org/2001/XMLSchema#float,
- http://www.w3.org/2001/XMLSchema#int,
- http://www.w3.org/2001/XMLSchema#integer,
- http://www.w3.org/2001/XMLSchema#short,
- http://www.w3.org/2001/XMLSchema#string.

Хоча онтології надають багатий набір інструментів для моделювання даних, їх зручність використання має певні обмеження.

Одним з таких обмежень є доступні конструкції властивостей. Наприклад, надаючи потужні конструкти класів, остання версія мови онтологій Web - OWL2 має дещо обмежений набір конструкцій властивостей.

Ще однин недолік пов'язаний із тим, як OWL використовує обмеження. Вони також служать для визначення того, як слід структурувати дані, і запобігають додаванню даних, які не відповідають цим обмеженням. Але це не завжди вигідно. Часто дані, імпортовані з нового джерела у триплеторій RDF, структурно не відповідають обмеженням, встановленим за допомогою OWL. Отже, ці нові дані повинні бути змінені, перш ніж інтегруватися з тим, що вже завантажено в триплетовий магазин.

Новою альтернативою для використання онтологій для моделювання даних є використання мови обмежень форми (SHACL) для перевірки графіків RDF щодо набору обмежень. Форма визначає метадані про тип ресурсу – як він використовується, як ним слід користуватися та як його потрібно використовувати. Для OWL, для перевірки даних може застосовуватися SHACL. Однак, на відміну від OWL, SHACL може застосовуватися для перевірки даних, які вже доступні в триплетовому магазині[5].

Бази даних включають концептуальні моделі та інформаційні ресурси, які разом можуть бути сприйняті як сховище концептуалізації онтології. На основі аналізу формальних відповідних зв’язків між реляційними базами даних та онтологіями OWL: реляційна база даних містить кілька таблиць, таблиця містить декілька полів і записів – це набір значення поля, тоді як онтологія OWL містить кілька класів, клас містить кілька властивостей і екземплярів – це сукупність значень властивостей. Формально відповідні зв'язки між таблицями, полями та записами у реляційних базах даних та класах, властивостями та екземплярами в онтологіях OWL дозволяють перетворити одну схему в іншу [6]. Відповідні зв’язки між компонентами реляційної бази даних та компонентами онтології показані на рисунку 2.3.

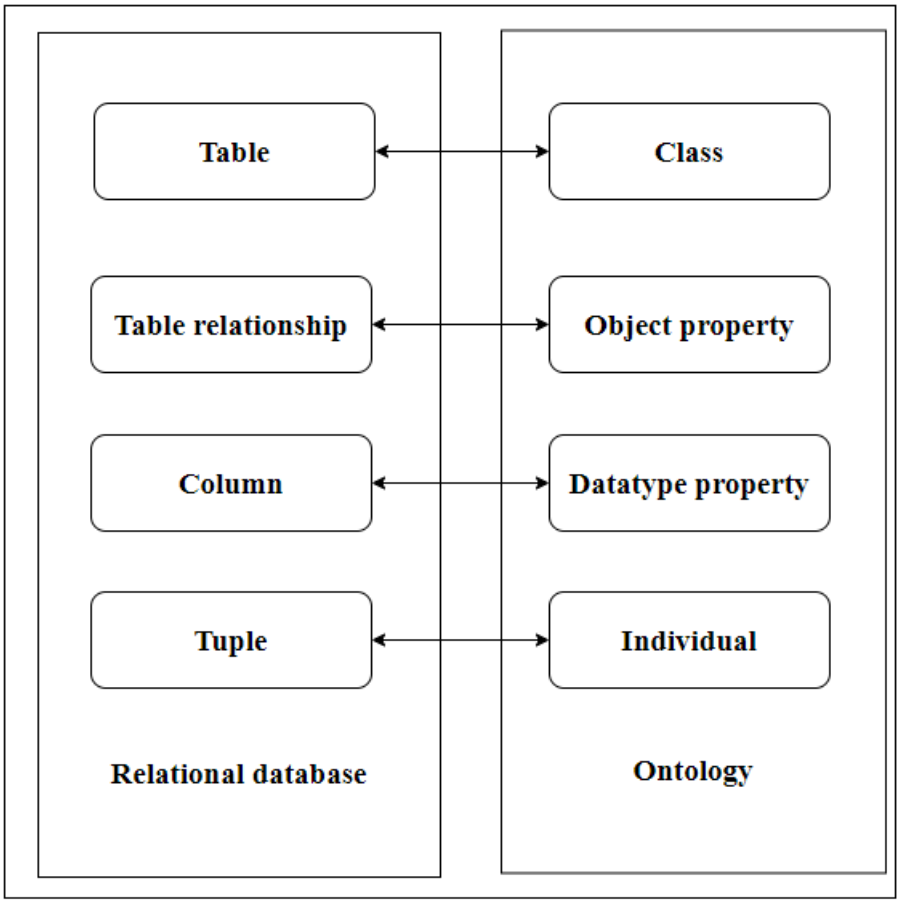

Рисунок 2.4 – Зв’язки між компонентами реляційної бази даних та онтології

Існують різні типи додатків, які забезпечують взаємодію реляційних баз даних та онтологій. Більшість з них побудовані на таких підходах до вирішення задач:

- інтеграція існуючих баз даних та онтологій;
- створення бази даних з існуючої онтології;
- вилучення доменної онтології з існуючої бази даних.

Protégé – це вільний, безкоштовний редактор онтологій і інструментарій для побудови баз знань (Рисунок 2.4).

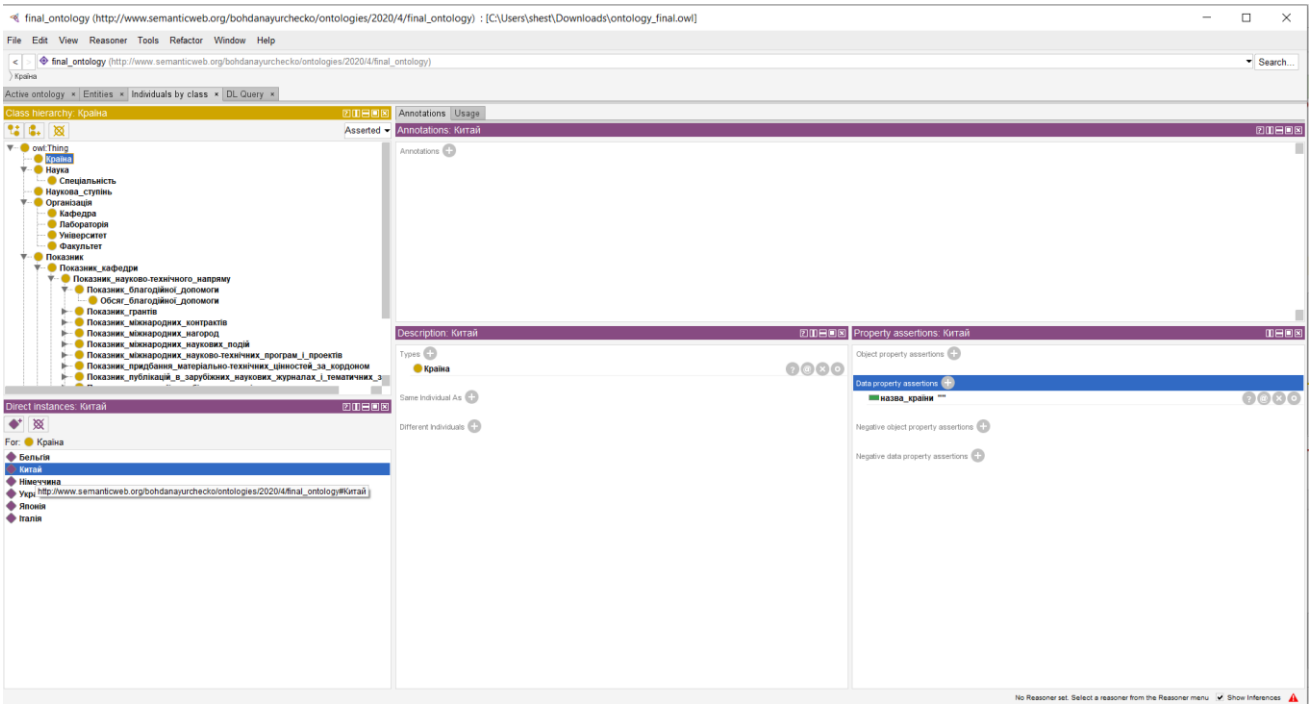

Рисунок 2.4 –Програма Protégé

Платформа Protégé підтримує два основних способи моделювання онтологій за допомогою редакторів Protégé-Frames і Protégé-OWL. Онтології, побудовані в Protégé, можуть бути експортовані в безліч форматів, включаючи RDF (RDF Schema), OWL і XML Schema.

Protégé має відкриту, легко розширювану архітектуру за рахунок підтримки модулів розширення функціональності.

Protégé підтримується значним співтовариством, що складається з розробників і вчених, урядових і корпоративних користувачів, що використовують його для вирішення завдань, пов'язаних зі знаннями, в таких різноманітних галузях, як біомедицина, збір знань і корпоративне моделювання.

2.2 Існуючі рішення

Ontop – випущена система OBDA з відкритим кодом за ліцензією Apache, розробленою у Вільному університеті Бозен-Больцано. Система Ontop виставляє реляційні бази даних у вигляді віртуальних графіків RDF,

пов'язуючи терміни (класи та властивості) в онтології з джерелами даних за допомогою відображення. Цей віртуальний графік RDF можна запитувати за допомогою SPARQL шляхом перекладу SPARQL-запитів у SQL-запити через реляційні бази даних. Цей процес перекладу прозорий для користувача. Ontop відповідає на запити SPARQL кінцевого користувача, переписуючи їх у SQL-запити та делегуючи виконання SQL-запитів джерелам даних. При такому підході немає необхідності застосовувати правила до джерел даних, щоб матеріалізувати всі факти, пов'язані з онтологією [7].

D2RQ – це інструмент, який використовується для перетворення вмісту, що зберігається у реляційній базі даних, у графі RDF, доступний лише для читання. D2RQ пропонує можливість створення відношення таблиць до класів онтології, рядки від таблиць до екземплярів онтології, стовпці з властивостями типу даних та зовнішні ключі до властивостей об'єкта. Jena – це інструмент, який реалізується в мові програмування Java. Він може бути використаний для перекладу конструкцій та висловлювань семантичної павутини в класи, об'єкти, атрибути та методи Java. Серед артефактів, якими користується Jena, найбільш важливими є такі: теми, предикати, об'єкти, заяви, дані, запити та результати, аргументи та правила. Запити записуються на SPARQL, що є мовою запитів рекомендованою W3C для RDF [8].

Інший підхід, який може бути використаний для нанесення реляційної бази даних на онтологію, описаний у [9]. Цей підхід використовує R2RML (RDB to RDF Mapping Language). Як випливає з назви, R2RML - це мова, яка забезпечує можливість перегляду реляційної бази даних в моделі даних RDF. Вхідні дані для відображення R2RML - це наявна база даних, яка містить інформацію, яка заповнює графік RDF. Серед функцій, які надає R2RML, є такі: він може зіставляти таблиці з бази даних, він може створювати атрибут R2RML, він може зв'язати дві таблиці, а також може перекласти коди типу бази даних на IRI і так далі.

2.3 Висновки до розділу

В даному розділі описано основні поняття пов’язані з онтологіями. Було розглянуто основні існуючі підходи до сумісної роботи з даними онтології та реляційної бази даних. Були наведені зв’язки між компонентами реляційної бази даних та компонентами онтології. Проаналізований редактор онтологій Protégé, його функціональні можливості. Простежується велика кількість відмінностей та подібностей між онтологіями та базами даних. Було покращене розуміння двох концепцій.

Перша відмінність, яку слід зазначити – власне дизайн. У цьому відношенні база даних створюється з нуля, тоді як онтологія повторно використовує вже наявні онтології. Інший момент розбіжності полягає в способі представлення знань. Різниця заключається в організації структур даних. Онтологія це представлення предметної області. РДБ – це структурована модель даних яка не відображає взаємозв’язок між об’єктами предметної області.

3 ЗАСОБИ РОЗРОБКИ

3.1 Мова програмування Java

Java – мова програмування високого рівня, розроблена Sun Microsystems. Спочатку вона був зроблена для розробки програм для приладів і портативних пристроїв, але згодом став популярним вибором для створення веб-додатків .

Синтаксис Java схожий на C++ , але суворо є об'єктно-орієнтованою мовою програмування . Наприклад, більшість програм Java містять класи, які використовуються для визначення об'єктів, та методи, які призначаються окремим класам. Java також відома тим, що є більш суворою, ніж C++, тобто змінні та функції повинні бути чітко визначені. Це означає, що вихідний код Java може створювати помилки або "винятки" легше, ніж інші мови, але також обмежує інші тип помилок, які можуть бути викликані невизначеними змінними або непризначеними типами.

На відміну від виконуваних файлів Windows (файли .EXE) або програм Macintosh (файли .APP), програми Java не запускаються безпосередньо операційною системою. Натомість програми Java інтерпретуються віртуальною машиною Java або JVM, яка працює на декількох платформах. Це означає, що всі програми Java є багатоплатформенними та можуть працювати на різних платформах, включаючи комп'ютери Macintosh, Windows та Unix. Однак JVM повинен бути встановлений, щоб програми Java або аплети взагалі працювали. На щастя, JVM включений як частина Java Runtime Environment (JRE), яка доступна для безкоштовного звантаження [10].

У створенні мови Java було п'ять основних цілей:

- Вона повинна бути простою, об'єктно-орієнтованою та звичною.
- Вона повинна бути надійною.
- Вона повинна бути нейтральною для архітектури та переносною.
- Вона повинна виконуватись з високою продуктивністю.
- Вона повинна бути інтерпретованою, потоковою та динамічною.

3.2 Технологія Spring Framework

Spring Framework є фреймворк і інверсія керування контейнера для платформи Java. Основні функції фреймворку можуть використовуватися будь-яким додатком Java, але є розширення для побудови веб-додатків на платформі Java EE (Enterprise Edition). Хоча фреймворк не нав'язує жодної конкретної моделі програмування, вона стала популярною у спільноті Java як доповнення або навіть заміна моделі Enterprise JavaBeans (EJB). Spring Framework є відкритим кодом.

Spring Framework включає кілька модулів, які надають цілий спектр послуг:

- контейнер spring core: це базовий модуль spring і забезпечує весняні контейнери (beanfactory та applicationcontext);
- програмно-орієнтоване програмування: дозволяє реалізувати наскрізні проблеми;
- аутентифікація та авторизація: налаштовані процеси безпеки, які підтримують цілий ряд стандартів, протоколів, інструментів та практик через підпроект spring security (раніше acegi security system for spring);
- доступ до даних: робота з реляційними системами управління базами даних на платформі java з використанням підключення до бази даних java (jdbc) та об'єктно-реляційних інструментів відображення та з базами даних nosql;
- інверсія контейнера управління: конфігурація компонентів програми та управління життєвим циклом об'єктів java, що здійснюється головним чином за допомогою введення залежності;
- повідомлення: конфігураційна реєстрація об'єктів слухача повідомлень для прозорого споживання повідомлень з черг повідомлень через службу повідомлень java (jms), покращення надсилання повідомлень через стандартні api jms;
- модель-вистава-контролер: http – і сервлет засноване рамки надання гачки для розширення і налаштування для веб - додатків і restful (репрезентативний стан передачі) web – сервісів;

- фреймворк віддаленого доступу: конфігураційний віддалений виклик процедур (rpc) - стильний обмін об'єктами java по мережах, що підтримують виклик віддаленого методу java (rmi), corba (загальна об'єктна запит брокера архітектури) та протоколи на базі http , включаючи веб-сервіси (soap);
- управління транзакціями: об'єднує кілька api управління транзакціями та координує транзакції для об’єктів java;
- віддалене управління: конфігураційне опромінення та управління об'єктами java для локальної або віддаленої конфігурації через розширення управління java (jmx);
- тестування: класи підтримки для написання одиничних тестів та інтеграційних тестів.

3.3 Мова програмування JavaScript

JavaScript (JS) – p це невибаглива до ресурсів мова програмування з функціями першого класу, код якої інтерпретується та компілюється під час виконання. Хоча JavaScript насамперед відома як скриптова мова для веб-сторінок, вона також використовується у багатьох небраузерних середовищах на кшталт Node.js, Apache CouchDB та Adobe Acrobat. JavaScript – прототип-орієнтована динамічна мова, що має декілька парадигм та підтримує об'єктно-орієнтований, імперативний та декларативний (тобто функціональне програмування) стилі [11].

Переваги використання JavaScript :

- Оскільки JavaScript є "інтерпретованою" мовою, це скорочує час, необхідний для компіляції іншими мовами програмування, такими як Java. JavaScript – це також сценарій на стороні клієнта, що прискорює виконання програми, оскільки це економить час, необхідний для підключення до сервера.
- JavaScript легко зрозуміти та вивчити. Структура проста як для користувачів, так і для розробників. Ця мова також дуже легка для імплементування, заощадивши розробників та чималі гроші на розробку динамічного контенту для Інтернету.

- Оскільки всі сучасні браузери підтримують JavaScript, його можна побачити майже скрізь. Усі відомі компанії використовують JavaScript як інструмент, включаючи Google, Amazon, PayPal тощо.
- JavaScript прекрасно працює з іншими мовами програмування, тому численні розробники вважають за краще його розробляти безліч додатків. Ми можемо вбудовувати його на будь-яку веб-сторінку або всередині сценарію іншої мови програмування.
- Оскільки JavaScript працює на стороні клієнта, перевірка даних можлива в самому браузері, а не надсилається на сервер. У разі будь-якої невідповідності весь веб-сайт не потрібно перезавантажувати. Оглядач оновлює лише вибраний сегмент сторінки.
- JavaScript надає розробникам різні інтерфейси для створення привабливих веб-сторінок. Перетягування компонентів або повзунків може надавати багатий інтерфейс веб-сторінкам. Це призводить до поліпшення інтерактивності користувачів на веб-сторінці.
- JavaScript покращує ефективність веб-сайтів та веб-додатків за рахунок зменшення довжини коду. Коди містять менше накладних витрат із використанням різних вбудованих функцій для циклів, доступу DOM тощо.

Недоліки JavaScript:

- Оскільки JavaScript-код можна побачити користувачеві, інші можуть використовувати його в шкідливих цілях. Ці практики можуть включати використання вихідного коду без автентифікації. Крім того, дуже просто розмістити на сайті якийсь код, який загрожує безпеці даних через веб-сайт.
- Браузер по-різному інтерпретує JavaScript у різних браузерах. Таким чином, код повинен бути запущений на різних платформах перед публікацією. Старі браузери не підтримують деякі нові функції, і ми також повинні їх перевірити.
- Хоча деякі редактори HTML підтримують налагодження, він не настільки ефективний, як інші редактори, такі як редактори C/C++ . Крім того, оскільки браузер не показує помилок, розробнику важко виявити проблему.

- JavaScript підтримує лише одне успадкування, а не множинне спадкування. Для деяких програм може знадобитися ця об'єктно-орієнтована мовна характеристика.
- Помилка одного коду може зупинити візуалізацію всього коду JavaScript на веб-сайті. Для користувача це виглядає так, ніби JavaScript не був присутній. Однак браузери вкрай терпимі до цих помилок.

3.4 Технологія Angular 7

Angular – це найпопулярніший фреймворк та платформа JavaScript для розробки веб-додатків для мобільних пристроїв та настільних ПК або додатків для однієї сторінки (SPA) на стороні клієнта.

Переваги Angular:

- Архітектура Model-View-Controller не тільки надає значення рамці під час створення додатка на стороні клієнта, але й встановлює основу для інших функцій, таких як прив'язка даних та сфери застосування.
- За допомогою архітектури MVC можна ізолювати логіку програми від рівня інтерфейсу та підтримувати розділення проблем. Контролер отримує всі запити на додаток і працює з моделлю, щоб підготувати будь-які дані, необхідні для перегляду. Перегляд використовує дані, підготовлені контролером, і відображає остаточну презентабельну відповідь.
- Деякі з великих веб-додатків містять безліч компонентів. Angular спрощує спосіб керування цими компонентами, навіть якщо новий програміст приєднається до проекту після того, як процес розробки вже розпочався. Архітектура побудована таким чином, що допомагає програмісту легко знаходити та розробляти код.
- Модуль – це механізм, який об'єднує пов'язані директиви, компоненти та сервіси таким чином, що їх можна поєднувати з іншими модулями для створення програми. Додаток на основі Angular може розглядатися як головоломка, де кожен модуль є необхідним, щоб мати можливість побачити повну картину. Існує

ряд способів додавання різних елементів до модуля. Angular вирішує проблему глобальної експлуатації функцій, обмежуючи сферу застосування всіх функцій модулем, в якому вона була визначена та використовується.

- Службі або компоненту іноді можуть знадобитися інші залежні служби для виконання завдання. Для виконання цих залежностей використовується шаблон дизайну ін'єкцій в залежність. Це розділяє завдання між різними службами. Клієнтська служба не створить залежний об'єкт, скоріше він буде створений і введений Angular провайдером. Angular провайдером відповідає за створення службових примірників та введення їх у такі класи, як компоненти та служби.

3.5 Бібліотека edu.stanford.protege

API Protege-OWL – це бібліотека Java з відкритим кодом для мови веб-онтології (OWL) та RDF (S). API надає класи та методи для завантаження та збереження файлів OWL, запиту та маніпулювання моделями даних OWL та проведення міркувань на основі двигунів Description Logic. Крім того, API оптимізовано для реалізації графічних інтерфейсів користувача.

API призначений для використання в двох контекстах:

- Для розробки компонентів, які виконуються всередині користувальницького інтерфейсу редактора Protege
- Для розробки автономних додатків (наприклад, додатків Swing, сервлетів або плагінів Eclipse)

3.6 Середовище розробки

IntelliJ IDEA – це особливе середовище програмування або інтегроване середовище розробки (IDE), багато в чому призначене для Java. Це середовище використовується спеціально для розробки програм. Він розроблений компанією під назвою JetBrains, яку офіційно називали IntelliJ. Він доступний у двох виданнях: Community Edition, що має ліцензію Apache 2.0, і комерційне

видання, відоме як Ultimate Edition. Обидва вони можуть бути використані для створення програмного забезпечення, яке можна продати. Що робить IntelliJ IDEA настільки відмінним від своїх аналогів, це його легкість у використанні, гнучкість та міцний дизайн. IntelliJ IDEA був розроблений JetBrains, раніше відомим як IntelliJ. Він був вперше випущений у 2001 році, і він похвалився такими функціями, як розширена навігація по коду та можливість переробити коди, які зробили його дуже популярним. Він навіть отримав відзнаку того, що він був визнаний найкращим інструментом програмування на базі Java в 2010 році, переклавши такі усталені інструменти, як NetBeans, Eclipse та JDeveloper. Середовище розробки з відкритим кодом для Android, випущене Google у 2014 році, також базується на IntelliJ IDEA. IDE підтримує багато інших мов програмування, таких як Python, Lua та Scala.

Найбільша причина, яку він вважає одним з найкращих інструментів програмування, заснованого на Java, - це його функції допомоги, що робить його простим у використанні та робить створені ним програми дуже добре розробленими. Він також має розширені функції перевірки помилок, що дозволяє швидше і простіше перевірити помилки [12].

JetBrains WebStorm – інтегроване середовище розробки для JavaScript, HTML та CSS від компанії JetBrains, розроблена на основі платформи IntelliJ IDEA. WebStorm є спеціалізованою версією PhpStorm, пропонуючи підмножину з його можливостей. WebStorm постачається з перед-установленим плаґінами JavaScript (такими як для Node.js), котрі доступні для PhpStorm безоплатно.

WebStorm підтримує мови JavaScript, CoffeeScript, TypeScript та Dart.

WebStorm забезпечує автодоповнення, аналіз коду на льоту, навігацію по коду, рефакторинг, зневадження та інтеграцію з системами управління версіями. Важливою перевагою інтегрованого середовища розробки WebStorm є робота з проектами (у тому числі, рефакторинг коду JavaScript, що міститься в різних файлах і теках проекту, а також вкладеного в HTML). Підтримується множинна вкладеність (коли в документ на HTML вкладений скрипт на

Javascript, в який вкладено інший код HTML, всередині якого вкладений Javascript) – в таких конструкціях підтримується коректний рефакторинг.

3.6 Висновки до розділу

У даному розділі було розглянуто інструменти та технології, що використовувались для розробки програмного продукту підключення реляційних баз даних до онтології предметної області. Розроблений додаток був розділений на дві частини: інтерфейс користувача та серверну. Серверна частина була розроблена на об’єктно-орієнтованій мові програмування Java з використанням фреймворка Spring. Для розробки інтерфейсу користувача було обрано мову програмування JavaScript та фреймворку Angular 7, що дозволило скоротити час на розробку додатку та забезпечити модульність. Розробка проводилась у популярних інтегрованих середовищах розробки IntellijIDEA та WebStorm які розробляє компанія JetBrains. Для аналізу файлу онтології було обрано бібліотеку edu.stanford.protege.

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

4.1 Шаблон проектування MVC

Model-View-Controller (MVC) – це архітектурна схема, яка розділяє додаток на три основні логічні компоненти Model, View і Controller. Звідси абревіатура MVC. Кожен компонент архітектури побудований для обробки конкретного аспекту розвитку програми. MVC відокремлює бізнес-логіку та рівень презентації один від одного. Традиційно його використовували для графічних інтерфейсів користувача (GUI) настільних ПК (Рисунок 4.1). У наш час архітектура MVC стала популярною для розробки веб-додатків, а також мобільних додатків [13].

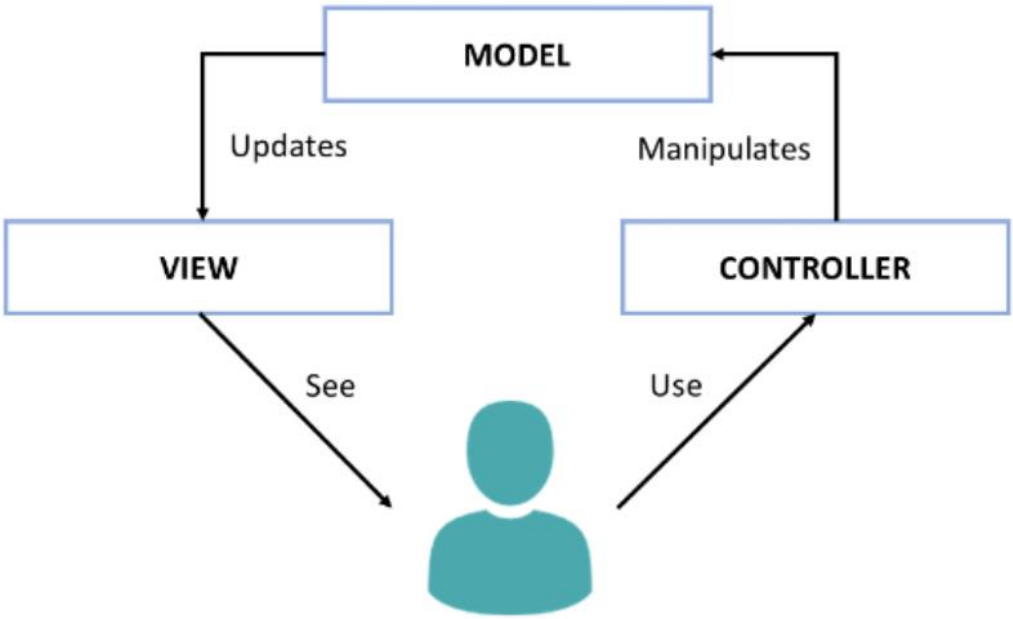

Рисунок 4.1– Графічне представлення роботи шаблону MVC

Вид (View) – це частина програми, яка представляє представлення даних.

Перегляди створюються за даними, зібраними з даних моделі. Перегляд вимагає, щоб модель надавала інформацію так, щоб вона нагадувала вихідну презентацію користувачеві.

Візуалізація також представляє дані з чатів, діаграм та таблиці. Наприклад, будь-який перегляд клієнта буде включати всі компоненти інтерфейсу користувача, наприклад текстові поля, спади тощо.

Контролер (Controller) – це та частина програми, яка обробляє взаємодію з користувачем. Контролер одержує вхідні дані й перетворює їх на команди для моделі чи вигляду.

Контролер надсилає команди моделі для оновлення свого стану (Наприклад, збереження конкретного документа). Контролер також надсилає команди до пов'язаного з ним перегляду для зміни подання перегляду (наприклад, прокрутка певного документа).

Компонент моделі (Model) зберігає дані та пов'язану з цим логіку. Він представляє дані, що передаються між компонентами контролера або будь-якою іншою пов'язаною бізнес-логікою. Наприклад, об'єкт Controller отримає інформацію про клієнта з бази даних. Він маніпулює даними та надсилає назад до бази даних або використовує їх для надання тих же даних.

Він відповідає на запит від інтерфейсу, а також відповідає на вказівки контролера щодо оновлення. Це також найнижчий рівень структури, який відповідає за збереження даних.

4.2 Архітектура проекту

REST – це архітектурний стиль програмного забезпечення, який визначає набір обмежень, які будуть використані для створення веб-служб. Веб-сервіси, які відповідають архітектурному стилю REST, називаються послугами RESTful Web, забезпечують сумісність між комп'ютерними системами в Інтернеті [14].

Архітектура проекту була розділена на 3 шари, а саме:

- шар зв'язку з реляційною базою даних;
- шар бізнес логіки;
- шар представлення інтерфейсу користувача.

Основні дані, які вводить користувач в систему є файл онтології та SPARQL-запит, який в свою чергу транслюється в SQL-запит до створеної реляційної бази. У користувача є можливість отримати результат виконання запиту (Рисунок 4.2).

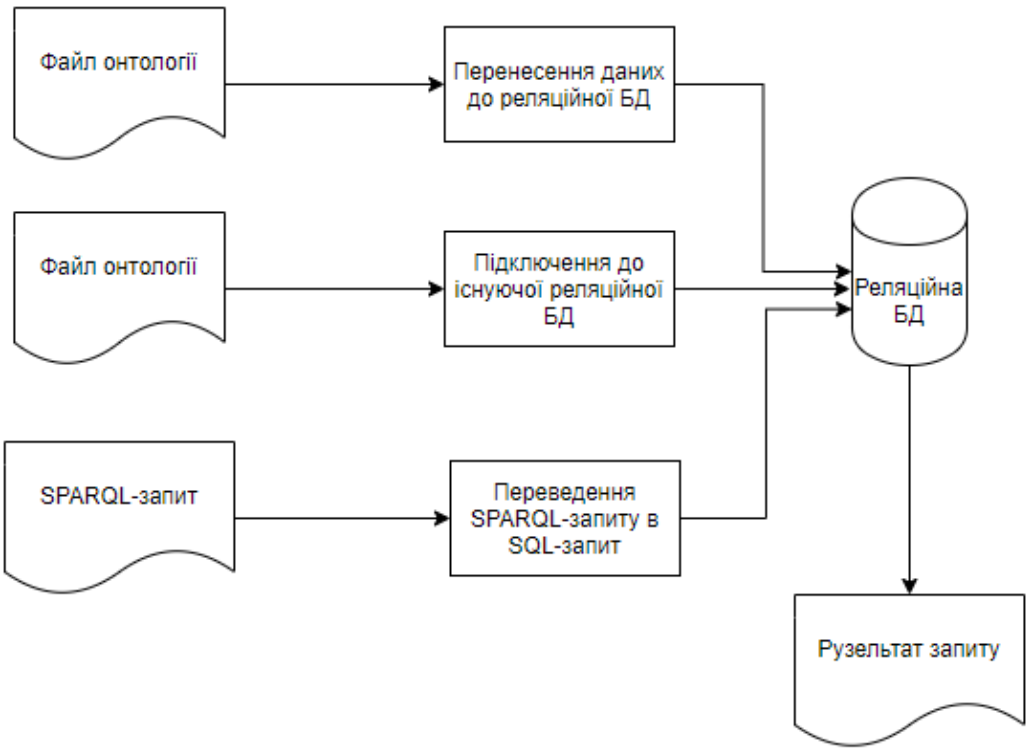

Рисунок 4.2 – Процес підключення реляційної БД до онтології

Результатом виконання запиту може бути файл завантажений з інтерфейсу або його можливо переглянути в інтерфейсі програмного засобу

4.3 Бізнес-логіка

Бізнес логіка в проекті представлена за допомогою трьох сервісів:

- FileStorageService – відповідає за збереження;
- DataBaseService – відповідає за створення таблиць та виконання запитів;
- OwlProcessor – відповідає за створення команд перенесення даних з онтології до БД.

Основну функціональність сервісів та репозиторії які вони використовують можна розглянути на рисунку 4.3.

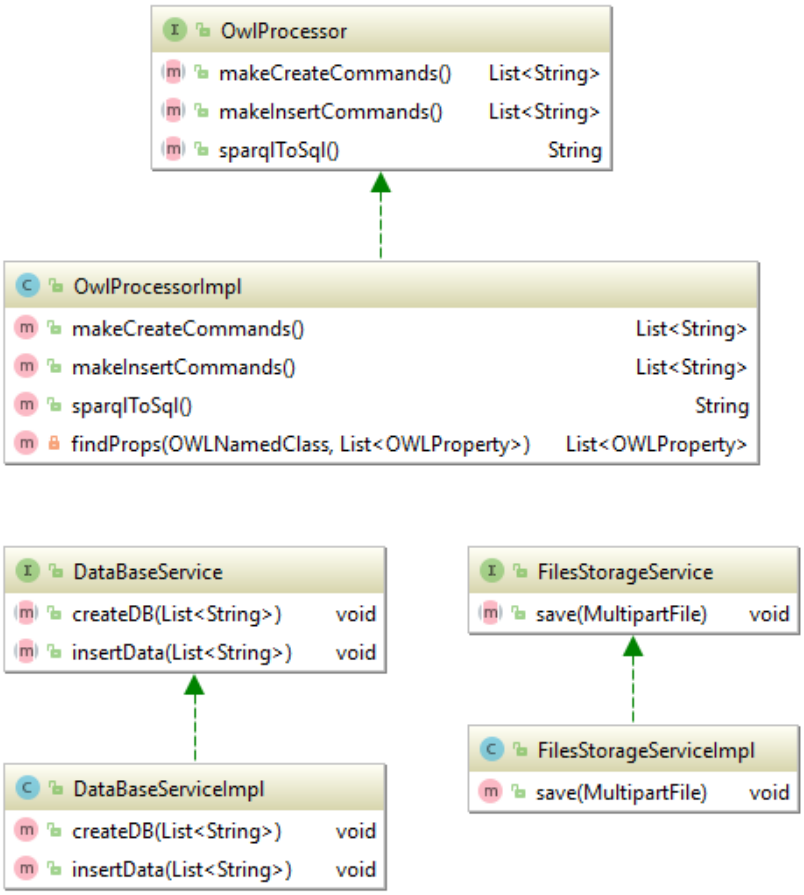

Рисунок 4.3 – Функціонал сервісу

Для створеня реляційної БД у сервісах OwlProcessor та DataBaseService наявні методи makeCreateCommands() , makeInsertCommands(), createDB(List<String>), insertData(List<String>). Алгоритм представлений на рисунку 4.4.

4.4 Зв’язок з інтерфейсом користувача

Зв’язок серверної частини та інтерфейсу користувача налаштований через контролери. Спочатку запит на сервері обробляється за допомогою DispatcerServlet, а далі на основі шляху та типу запиту відправляється на відповідний контролер.

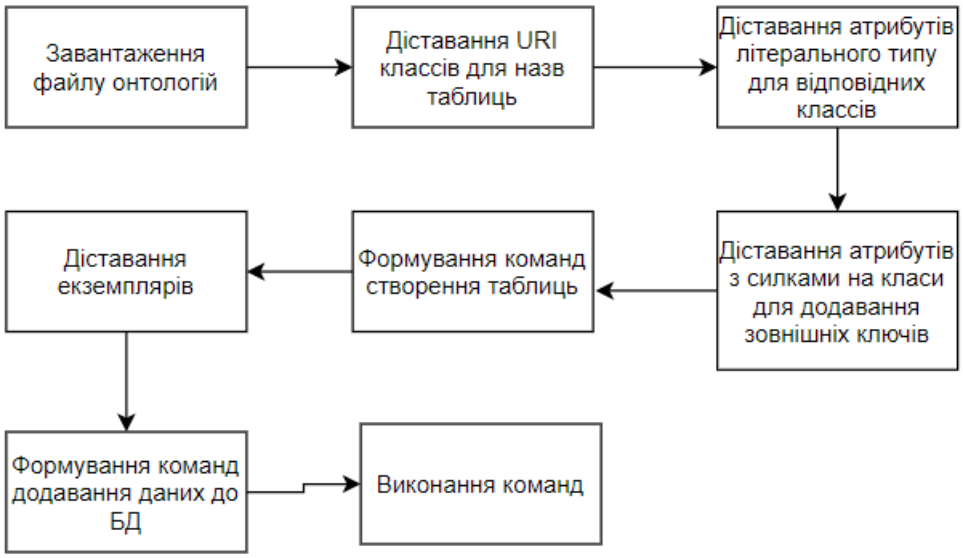

Рисунок 4.4 – Алгоритм обробки

Всього в розробленому додатку 5 контролерів:

- /upload-file (POST);
- /create-db (POST);
- /insert-individuals (POST);
- /execute-sparql (POST);
- /download-result (GET).

Всього в додатку використовується два типи запитів POST та GET. POST використовується для надсилання даних на сервер для створення / оновлення ресурсу. GET використовується для запиту даних із визначеного ресурсу та відображення відповіді користувачу.

4.5 Створення клієнтського інтерфейсу

Інтерфейс користувача створено за допомогою мови розмітки гіпертексту HTML, мови програмування JavaScript та фреймворку Angular.

Angular – це структура дизайну прикладних програм та платформа розробки для створення ефективних та складних програм на одній сторінці.

В якості графічного інтерфейсу створено веб-сторінки з усіма необхідними полями, таблицями, вказівками та кнопками. Тоді було написано

обробники подій для кнопок, в яких викликаються методи для підготовки даних до відправки запитів на сервер, на я кому виконується бізнес-логіка. Це надає можливість зручної взаємодій користувача та системи, виконання запитів та отримання результатів.

4.6 Висновки до розділу

В даному розділі описано особливість архітектури даного програмного продукту. Були наведені схеми роботи основних модулів додатку. Модулі серверної частини:

- модуль завантаження файлу онтології до системи – призначений для завантаження та зберігання файлу онтології на сервері для подальшої обробки;
- модуль обробки .owl файлу – призначений для аналізу та обробки моделі збереженої в файлі формату .owl, а саме зчитування класів, атрибутів, та посилань. Також модуль використовується для побудови команд створення таблиць, занесення даних за необхідністю, та переведення SPARQL-запиту до SQL-запиту;
- модуль підключення до БД – призначений для забезпечення роботи системи з базою даних.

Також у розділі наведений алгоритм роботи бізнес-логіки: побудови команд, та представлення даних для відправки до клієнтської частини. Описано схема контролерів, на які відправляються запити з клієнтської частини.

5 РОБОТА КОРИСТУВАЧА З ПРОГРАМОЮ

5.1 Підключення бази даних до онтології

Для роботи із програмним забезпеченням достатньо завантажити її на веб сервер і розгорнути додаток. З цього часу при правильних налаштуваннях сервера у користувача буде надано доступ до нього. Також можливо тестувати додаток локально на комп'ютері користувача. Цев можна зробити відкривши серверну частину через Intelij IDEA там частину клієнтського інтерфейсу через WebStorm та запустити їх натиснувши кнопку «Run». Після цього користувачу необхідно відкрити браузер та перейти за адресою http://localhost:4200/ .

Перший крок роботи із системою є завантаження файлу із даними онтології у форматі .owl. Файл такого формату можна отримати зберігши онтологію за допомогою будь-якого редактора онтологій, наприклад Protégé. Початкова сторінка зображена на рисунку 5.1.

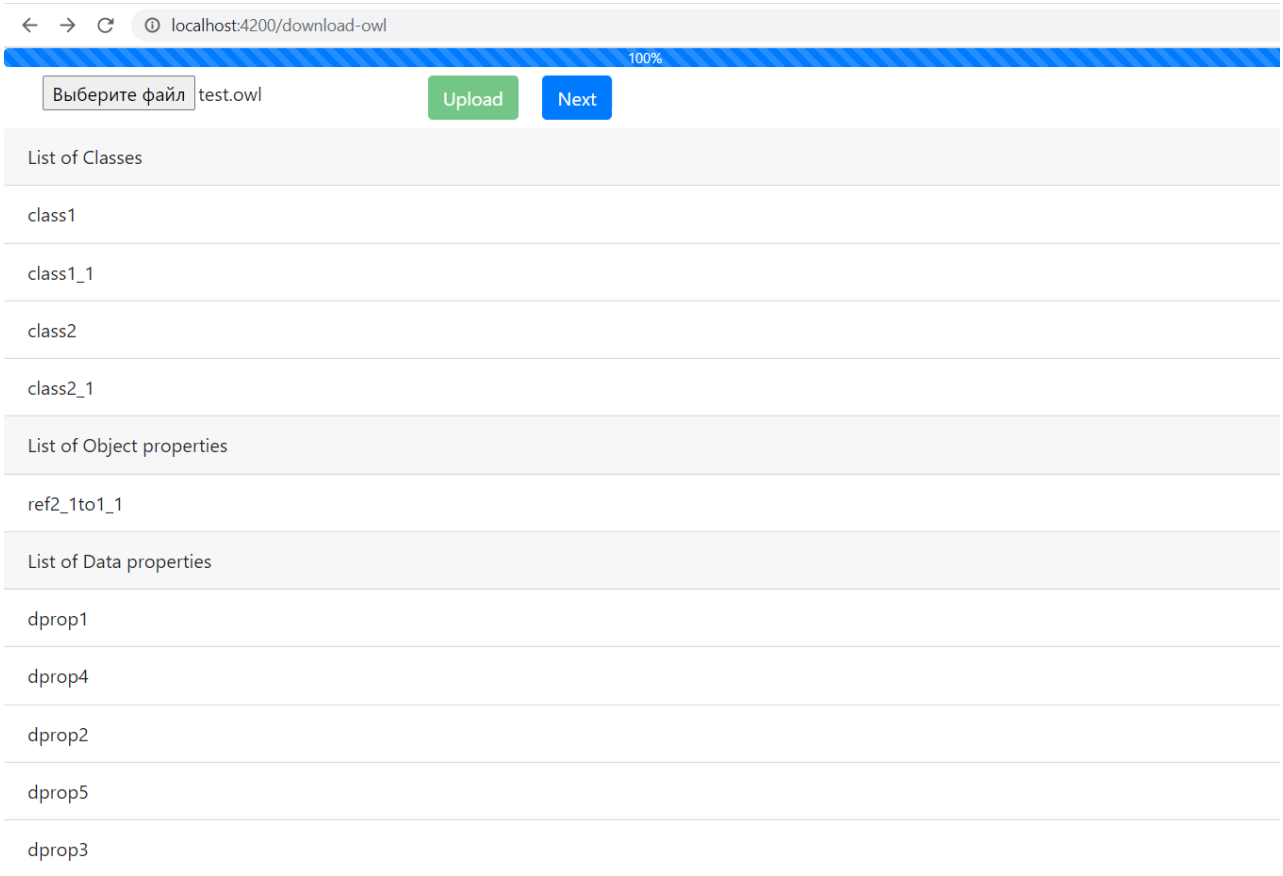

Рисунок 5.1 – Початкова сторінка

Після завантаження файлу онтології буде відображено проаналізовані класи та їхні властивості. Коли користувач перегляне чи всі дані правильно проаналізовані він може перейти на наступний етап створення таблиць та занесення даних до бази даних, натиснувши кнопку «Next» (Рисунок 5.2).

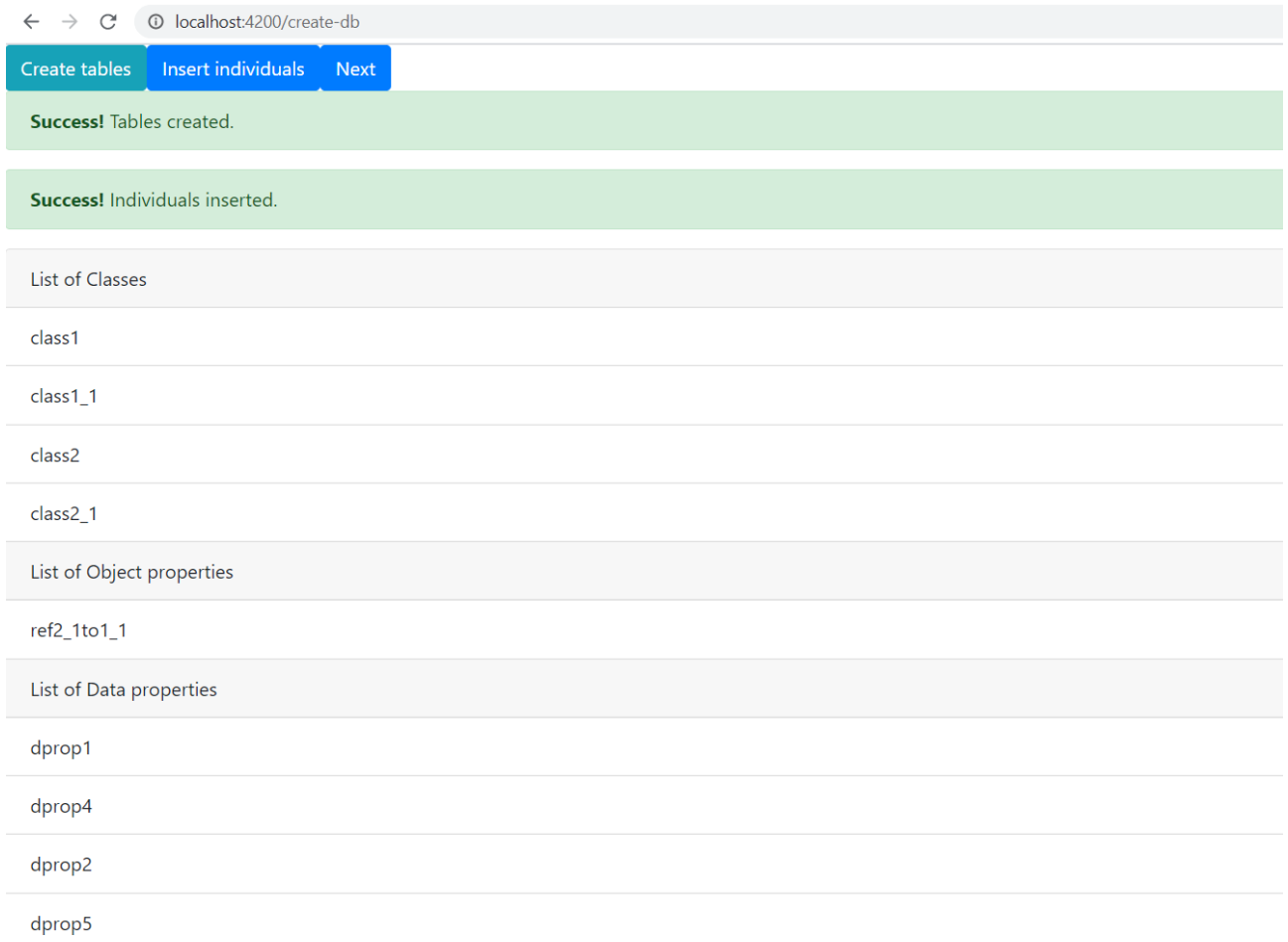

Рисунок 5.2 – Підтвердження створення бази даних

Після натискання кнопки “Next” на сторінці підтвердження занесення даних користувач перейде на сторінку написання тестового SPARQL-запиту.

Користувач зможе ввести запит та підтвердити виконання. Після цього буде відображено згенерований SQL-запит та результат виконання останнього (Рисунок 5.3). Також користувачу буде запропоновано зберегти файл з результатом.

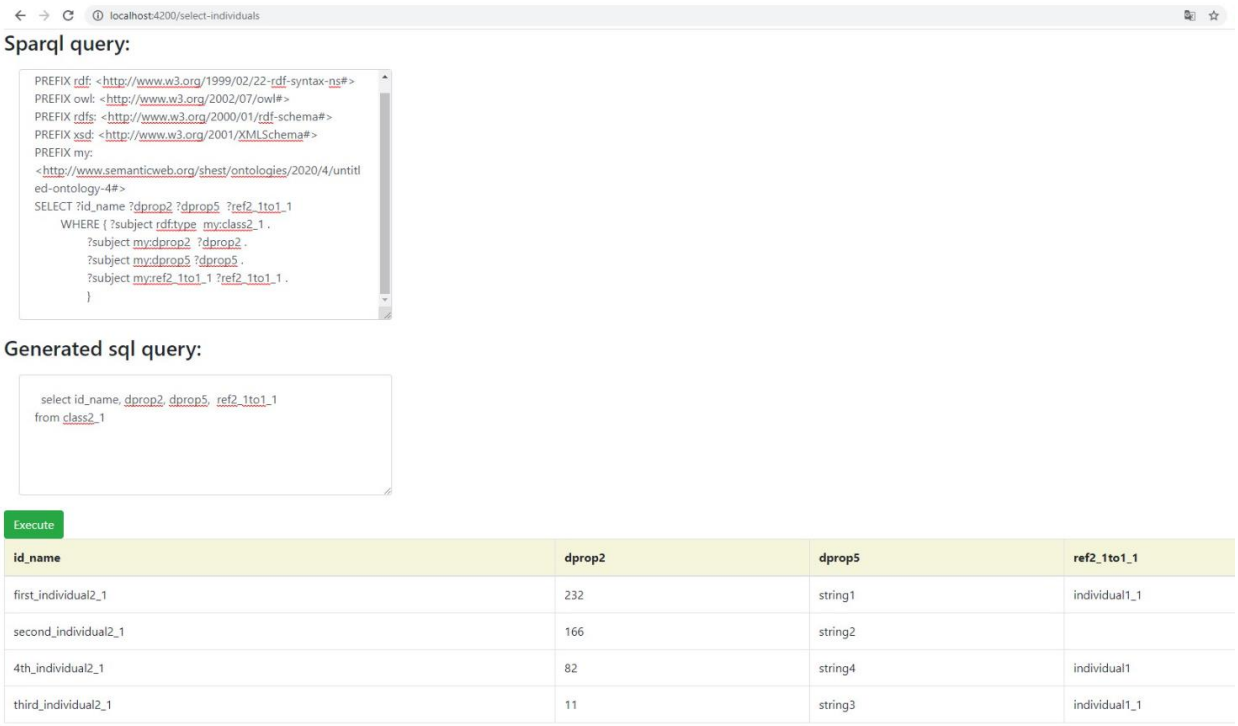

Рисунок 5.3 – Результат виконання запиту

5.2 Перевірка коректності міграції даних

Для перевірки отримання всіх даних можна відкрити файл формату .owl в текстовому редакторі. В файлі будуть міститися екземпляри онтології (Рисунок 5.4).

```
<owl:NamedIndividual rdf:about="http://www.semanticweb.org/shest/ontologies/2020/4/untitled-ontology-4#first_individual2_1">
    <rdf:type rdf:resource="http://www.semanticweb.org/shest/ontologies/2020/4/untitled-ontology-4#class2_1"/>
    <dprop2 rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">232</dprop2>
    <dprop5 rdf:datatype="http://www.w3.org/2001/XMLSchema#string">string1</dprop5>
    <ref2_1to1_1 rdf:resource="http://www.semanticweb.org/shest/ontologies/2020/4/untitled-ontology-4#individual1_1"/>
</owl:NamedIndividual>

<!-- http://www.semanticweb.org/shest/ontologies/2020/4/untitled-ontology-4#second_individual2_1 -->

<owl:NamedIndividual rdf:about="http://www.semanticweb.org/shest/ontologies/2020/4/untitled-ontology-4#second_individual2_1">
    <rdf:type rdf:resource="http://www.semanticweb.org/shest/ontologies/2020/4/untitled-ontology-4#class2_1"/>
    <dprop2 rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">166</dprop2>
    <dprop5 rdf:datatype="http://www.w3.org/2001/XMLSchema#string">qwerty</dprop5>
</owl:NamedIndividual>

<!-- http://www.semanticweb.org/shest/ontologies/2020/4/untitled-ontology-4#4th_individual2_1 -->

<owl:NamedIndividual rdf:about="http://www.semanticweb.org/shest/ontologies/2020/4/untitled-ontology-4#4th_individual2_1">
    <rdf:type rdf:resource="http://www.semanticweb.org/shest/ontologies/2020/4/untitled-ontology-4#class2_1"/>
    <dprop2 rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">82</dprop2>
    <dprop5 rdf:datatype="http://www.w3.org/2001/XMLSchema#string">string4</dprop5>
    <ref2_1to1_1 rdf:resource="http://www.semanticweb.org/shest/ontologies/2020/4/untitled-ontology-4#individual1"/>
</owl:NamedIndividual>
```

Рисунок 5.4 – Перевірка даних в файлі онтології

Також для більш точної перевірки слід виконати SPARQL-запит. Для цього можна використати Protégé (Рисунок 5.5). В таблиці ми бачимо ті самі дані, що і у розробленому додатку.

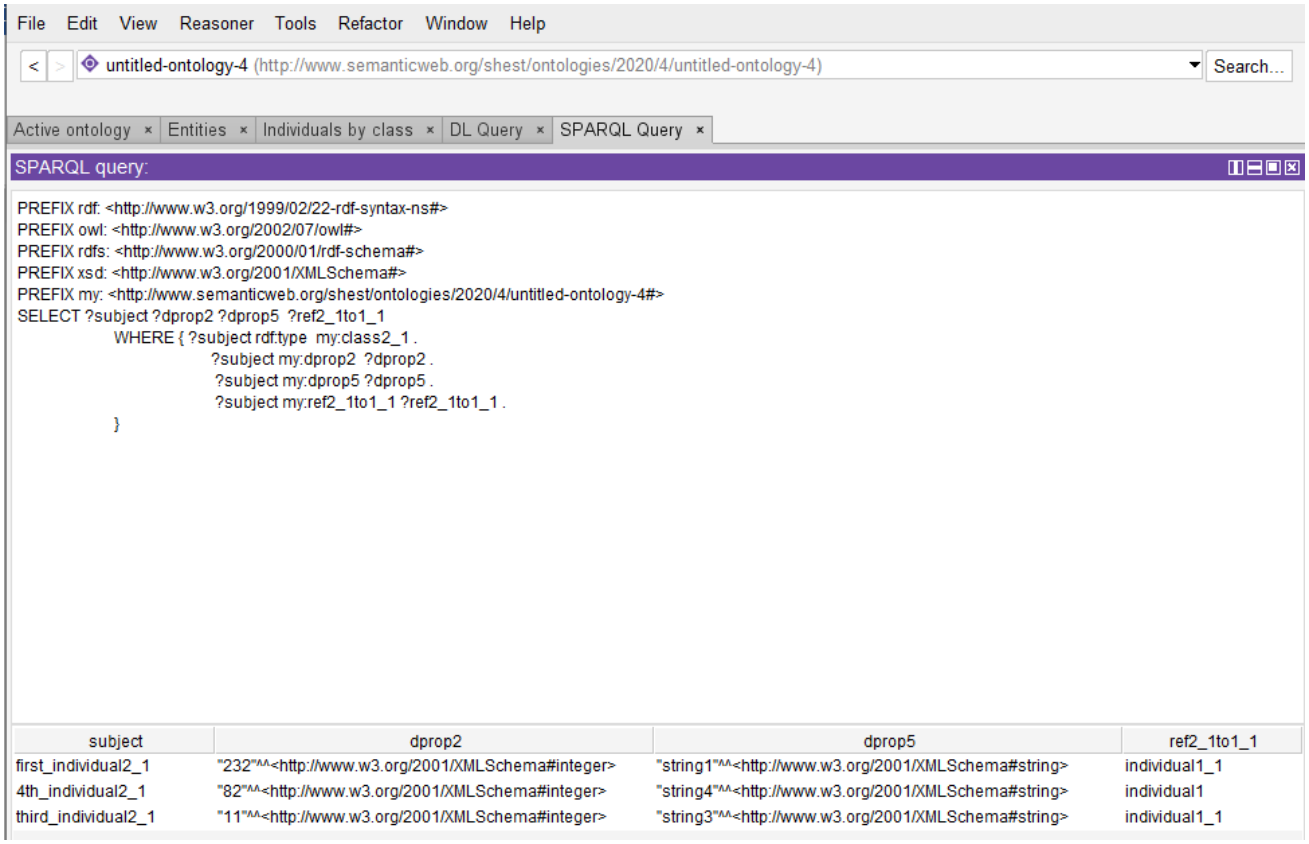

Рисунок 5.5 – Перевірка виконання запиту до онтології

5.3 Висновки до розділу

В даному розділі представлений інтерфейс користувача розробленого програмного забезпечення. Як зазначалося в постановці задач додаток має всі необхідні можливості. В процесі було завантажено файл онтології та здійснена міграція моделі даних до реляційної бази даних

Після підключення до бази даних було виконано запит та отримано результат з БД.

Була проведена перевірка, яка показала що всі повернені дані співпадали з файлом онтології. Також, виконавши перевірку за допомогою Protégé, можна було побачити що SPARQL-запит був трансльований правильно.

Функції системи :

- відкрити любий файл .owl для створення РБД ;
- згідно файлу онтології створити РБД;
- перенесення даних з файлу онтології до РБД;
- підключення вже створеної БД для формування запитів;
- формування SPARQL запитів і конвертація їх в SQL;
- виконання запитів;
- візуалізація результату.

ВИСНОВКИ

Під час виконання дипломної роботи було розглянуто існуючі підходи до підключення реляційної бази даних до онтології. Було обрано підхід з використанням однієї концептуальної моделі БД і онтології. Де реляційна БД використовується для зберігання даних, представлених в онтології. Також є можливість міграції даних з онтології до бази даних.

В результаті розробки було створено програмний додаток, який дозволяє перенести дані із онтології .owl формату до реляційної бази даних та зіставити модель онтології з БД.

Розроблений додаток не залежить від операційної системи встановленої на комп'ютері. Так як це WEB-додаток, використати його може кожний в кого є браузер.

Вхідні дані: файл онтології з розширенням *.owl, SPARQL-запит переведений SQL-запит.

Вихідні дані: результат виконання запиту.

Основні завдання які були вирішені при розробці програмного продукту являються:

- завантаження файлу онтології з комп'ютера користувача;
- аналіз файлу онтології;
- підключення до реляційної бази даних;
- переведення SPARQL-запиту в SQL-запиту для отримання даних з реляційної бази даних;
- відображання результату запиту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Gruber, T. (2008). Liu, Ling; Özsu, M. Tamer (eds.). Ontology. Encyclopedia of Database Systems. Springer-Verlag. ISBN 978-0-387-49616-0

2. Добров Б. В., Иванов В.В., Лукашевич Н.В., Соловьев В.Д. Онтологии и тезаурусы: модели, инструменты, приложения. — М.: Бином. Лаборатория знаний, 2009. — 173 с.

3. Gómez-Pérez A., Moreno A., Pazos J., Sierra-Alonso A. Knowledge Maps: An essential technique for conceptualisation // Data & Knowledge Engineering, 2000, V. 33(2). – P. 169 - 190.

4. Технології менеджменту знань (Серія "Консолідована інформація". Випуск 2) — Львів: Видавництво Львівської політехніки — 195 с.

5. What are Ontologies? [Електроний ресурс] — Режим доступу до ресурсу: https://www.ontotext.com/knowledgehub/fundamentals/what-are-ontologies/

6. Sedighi, Saeed & Javidan, Reza. (2011). Semantic query in a relational database using a local ontology construction. South African Journal of Science. 108. 97-107. 10.4102/sajs.v108i11/12.1107.

7. Kontchakov, R.; Rezk, M.; Rodriguez-Muro, M.; Xiao, G.; and Zakharyaschev, M. 2014. Answering SPARQL queries over databases under OWL 2 QL entailment regime. In Proc. of ISWC, Part I, volume 8796 of LNCS, 552–567. Springer.

8. Cioara, Tudor & Moldovan, Dorin & Antal, Marcel & Valea, Dan & Pop, Claudia & Anghel, Ionut & Salomie, Ioan. (2015). Tools for Mapping Ontologies to Relational Databases: A Comparative Evaluation. 10.1109/ICCP.2015.7312609.

9. N. Konstantinou, D. Kouis, N. Mitrou, “Incremental Export of Relational Database Contents into RDF Graphs”, 4th International Conference on Web Intelligence, Mining and Semantics (WIMS’14), June 2014

10. JAVA [Електронний ресурс] – Режим доступу до ресурсу: https://techterms.com/definition/java

11. JavaScript [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/uk/docs/Web/JavaScript

12. Intellij IDEA [Електронний ресурс] – Режим доступу до ресурсу: https://www.techopedia.com/definition/7755/intellij-idea

13. A. Elsaadany and K. Abbas, “Development and implementation of e-learning system in smart educational environment,” in 2016 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2016, pp. 1004–1009

14. Fielding, Roy Thomas. Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, University of California, Irvine, 2000.

ДОДАТОК 1

Інструментальні засоби підключення реляційної бази даних до онтології предметної області

Специфікація

УКР.НТУУ“КПІ”.ТР61124_20Б

Аркушів 2

2020

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ«КПІ ім. Ігоря Сікорського».ТР61124_20Б 81-1	Записка	Пояснювальна записка
Компоненти		
УКР.НТУУ«КПІ».ТР61124_20Б 12-1	Текст програмного Модуля	ontologyFront,js
УКР.НТУУ«КПІ».ТР61124_20Б 12-2	Текст програмного модуля	OwlProcessorImpl.java
УКР.НТУУ«КПІ».ТР61124_20Б 12-3	Текст програмного модуля	DataBaseService.java
УКР.НТУУ«КПІ».ТР61124_20Б 12-4	Текст програмного модуля	FileStorageService.java
УКР.НТУУ«КПІ».ТР5168_20Б 13-2	Опис програми	OwlProcessorImpl.doc

ДОДАТОК 2

Інструментальні засоби підключення реляційної бази даних до онтології предметної області

Текст програмного модуля

УКР.НТУУ«КПІ».ТР61124_20Б 12-2

Аркушів 10

2020

```
        @Component
        //створення команд для аналізу файлу онтології
public class OwlProcessorImpl implements OwlProcessor {
    private final JdbcTemplate jdbcTemplate;

    @Autowired
    public OwlProcessorImpl(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }
// отримання классів та атрибутів онтології
    public Response getNodes(String filePath)throws DataAccessException,
Exception{
        String uri1 = new
File("src/main/resources/uploads/"+filePath).toURI().toString();
//вся модель онтології
        OWLModel owlModel = ProtegeOWL.createJenaOWLModelFromURI(uri1);

        List<String> classes =
(List<String>)owlModel.getUserDefinedOWLNamedClasses().stream().map(c ->
((OWLNamedClass)c).getBrowserText()).collect(Collectors.toList());
        List<String> dprops =
(List<String>)owlModel.getUserDefinedOWLDatatypeProperties().stream().map(c -
> ((OWLProperty)c).getBrowserText()).collect(Collectors.toList());
        List<String> oprops =
(List<String>)owlModel.getUserDefinedOWLObjectProperties().stream().map(c ->
((OWLProperty)c).getBrowserText()).collect(Collectors.toList());
        return new Response(classes, oprops, dprops, null,null,null);
    }
```

-3-

```
                // створення команд для створення таблиць
        @Override
    public List<String> makeCreateCommands(String file) throws
DataAccessException, Exception{
        //мапа для переведення типів онтолгії до SQL
        HashMap<Class, JDBCType> types = new HashMap<Class, JDBCType>();
        types.put(Integer.class, JDBCType.INTEGER);
        types.put(Float.class, JDBCType.FLOAT);
        types.put(String.class, JDBCType.VARCHAR);
        types.put(Double.class, JDBCType.DOUBLE);
        types.put(Date.class, JDBCType.DATE);
        types.put(Instance.class, JDBCType.VARCHAR);

        String uri1 = new File(file ).toURI().toString();

        OWLModel owlModel = ProtegeOWL.createJenaOWLModelFromURI(uri1);
        List<String> result = new ArrayList<>();

        List<OWLNamedClass> classes =
(List<OWLNamedClass>)owlModel.getUserDefinedOWLNamedClasses();
        List<OWLProperty> dprops =
(List<OWLProperty>)owlModel.getUserDefinedOWLDatatypeProperties();
        List<OWLProperty> oprops =
(List<OWLProperty>)owlModel.getUserDefinedOWLObjectProperties();
        List<OWLIndividual> inds =
(List<OWLIndividual>)owlModel.getUserDefinedRDFIndividuals(true);
```

```
        System.out.println(classes);

        System.out.println(dprops.get(0).getProtegeType());

System.out.println(Integer.class.equals(dprops.get(0).getValueType().getJavaType()
));
      System.out.println(dprops.get(0).getRDFType());
      System.out.println(dprops.get(0).getDirectType());

      System.out.println(oprops);
      System.out.println(inds);

      for (Iterator it = classes.iterator(); it.hasNext();) {
         OWLNamedClass cls = (OWLNamedClass) it.next();
         Collection instances = cls.getInstances(false);
         System.out.println("Class " + cls.getBrowserText() + " (" + instances.size() +
")");

         for (Iterator jt = instances.iterator(); jt.hasNext();) {
            OWLIndividual individual = (OWLIndividual) jt.next();
            System.out.println(" - " + individual.getBrowserText());
            for (OWLProperty o: oprops)
               System.out.println(" -- " + individual.getPropertyValue(o));
            System.out.println(" - " + individual.getReferences());

         }
      }
```

```
        System.out.println(
(List<OWLIndividual>)owlModel.getUserDefinedRDFIndividuals(true));

        for (OWLNamedClass clazz : classes) {
            String createString =
                    "CREATE TABLE " + clazz.getBrowserText()
                            +
                            "(\n" +
                            "   id_name VARCHAR(50) PRIMARY KEY      NOT NULL,\n";
            for (OWLProperty p: findProps(clazz, dprops)) {
                createString += "   " + p.getBrowserText() + " " +
types.get(p.getValueType().getJavaType()) + ",\n";
            }
            for (OWLProperty p: findProps(clazz, oprops)) {
                createString += "   " + p.getBrowserText() + " " +
types.get(p.getValueType().getJavaType()) + ",\n";
            }
            createString = createString.substring(0, createString.length() - 2);
            createString += ");";
            System.out.println(createString);
            result.add(createString);
          //  jdbcTemplate.execute(createString);
        }

        for (OWLProperty p: oprops) {
            String alterString = "ALTER TABLE ";
            alterString += p.getDomain(false).getBrowserText() + " ADD CONSTRAINT
fk_" + p.getDomain(false).getBrowserText() +
```

```
        p.getRange().getBrowserText()

                + " FOREIGN KEY (" + p.getBrowserText() + ") REFERENCES " +
p.getRange().getBrowserText() + "(id_name)";
            System.out.println(alterString);
            result.add(alterString);

          // jdbcTemplate.update(alterString);
        }



        return result;
    }

         //створення команд міграції даних
    @Override
    public List<String> makeInsertCommands(String file) throws
DataAccessException, Exception {
        String uri1 = new File(file).toURI().toString();
        OWLModel owlModel = ProtegeOWL.createJenaOWLModelFromURI(uri1);
        List<OWLIndividual> inds =
(List<OWLIndividual>)owlModel.getUserDefinedRDFIndividuals(true);
        List<String> result = new ArrayList<>();
        for (OWLIndividual in : inds) {
            String insertString = "INSERT INTO " +
in.getProtegeType().getBrowserText() + "(id_name,";
            for (String s : (List<String>)in.getRDFProperties().stream()
                    .map(p -> ((RDFResource)p).getBrowserText())
                    .filter(s ->!s.equals("rdf:type"))
```

```
                .collect(Collectors.toList())) {
            insertString += s + ",";
        }
        insertString = insertString.substring(0, insertString.length() - 1);
        insertString += ") VALUES ('" + in.getBrowserText() +"',";
        for (String s : (List<String>)in.getRDFProperties().stream()
                .filter(p ->
!((RDFProperty)p).getFrameID().getName().equals("http://www.w3.org/1999/02/22-
rdf-syntax-ns#type"))
                .map(p -> in.getPropertyValue((RDFProperty) p) instanceof RDFObject
                        ? ((RDFObject)in.getPropertyValue((RDFProperty)
p)).getBrowserText()
                        : in.getPropertyValue((RDFProperty) p))
                .map(p -> p = """+p+""")
                .collect(Collectors.toList()) ){
            insertString += s + ",";
        }
        insertString = insertString.substring(0, insertString.length() - 1);
        insertString +=  ")";
        System.out.println(insertString);
        result.add(insertString);
        //jdbcTemplate.update(insertString);

    }
    return result;
}

    //переведення спаркл в скл
@Override
```

```
    public String sparqlToSql(String file, String queryString) throws
DataAccessException, Exception {
        Query query = QueryFactory.create(queryString);

        List<String> resultVars = query.getResultVars();
        ElementGroup queryPattern = (ElementGroup)query.getQueryPattern();
        ElementPathBlock elementPathBlock = (ElementPathBlock)
queryPattern.getElements().get(0);
        List<TriplePath> triplePaths =  elementPathBlock.getPattern().getList();
        String clazz = triplePaths.stream()
                .filter(p -> "type".equals(p.getPredicate().getLocalName()))
                .findFirst().get()
                .getObject().getLocalName();

        String selectString = "select id_name as name, ";
        for (TriplePath r: triplePaths) {
           if (resultVars.contains(r.getObject().toString().substring(1))
                   && !"type".equals(r.getPredicate().getLocalName())){
               selectString += r.getPredicate().getLocalName() + ", ";
           }

        }
        selectString = selectString.substring(0, selectString.length() - 2);
        selectString += " from " + clazz + " ";

        System.out.println(selectString);
        return selectString;
    }
```

```
    private List<OWLProperty> findProps(OWLNamedClass clazz,
List<OWLProperty> dprops) {
        List<OWLProperty> result = new ArrayList<>();
        dprops.stream().filter(p ->
p.getDomains(false).contains(clazz)).collect(Collectors.toList());
        result.addAll(dprops.stream().filter(p ->
p.getDomains(false).contains(clazz)).collect(Collectors.toList()));
        for (OWLNamedClass superClazz :
(List<OWLNamedClass>)clazz.getSuperclasses(true)) {
            result.addAll(findProps(superClazz, dprops));
        }
        return result;
    }
}

@RestController
@CrossOrigin(origins = "*", maxAge = 3600)
@RequestMapping("/api/")
public class Controller {
    private final OwlProcessor owlProcessor;
         private final FilesStorageService filesStorageService;
    private final DataBaseService dataBaseService;

    @Autowired
    public Controller(OwlProcessor owlProcessor, FilesStorageService
filesStorageService, DataBaseService dataBaseService) {
        this.owlProcessor = owlProcessor;
```

```
        this.filesStorageService = filesStorageService;
        this.dataBaseService = dataBaseService;

    }

    @GetMapping("/create-db")
    public ResponseEntity<?> createDB(@RequestParam("file") String file) {

        try {
            dataBaseService.createDB(owlProcessor.makeCreateCommands(file));
            return ResponseEntity.status(HttpStatus.OK).body(new
Response(null,null,null,null,null,null));
        } catch (Exception e) {
            return ResponseEntity.ok("not ok");
        }
    }

    @GetMapping("/insert-individuals")
    public ResponseEntity<?> insertIndividuals(@RequestParam("file") String file) {

        try {
            dataBaseService.insertData(owlProcessor.makeInsertCommands(file));
            return ResponseEntity.status(HttpStatus.OK).body(new
Response(null,null,null,null,null,null));
        } catch (Exception e) {
            return ResponseEntity.ok("not ok");
        }
    }
```

```
    @PostMapping("/uploadFile")
    public ResponseEntity<Response> uploadFile(@RequestParam("file")
MultipartFile file) {

        String filename = filesStorageService.save(file);
        return
ResponseEntity.status(HttpStatus.OK).body(owlProcessor.getNodes(filename));
    }
    @GetMapping("/execute-sparql-query")
    public ResponseEntity<Response> executeSparqlQuery(@RequestParam("file")
String file, @RequestParam("query") String query) {
        String sqlQuery = owlProcessor.sparqlToSql(file, query);
        List<String> f=QueryFactory.create(query).getResultVars();
        List<Map<String,Object>> o = dataBaseService.executeQuery(sqlQuery);
        return ResponseEntity.status(HttpStatus.OK).body(new
Response(null,null,null,o,f, null));
    }
}
```

ДОДАТОК З

Інструментальні засоби підключення реляційної бази даних до онтології предметної області

Опис програмного модуля

УКР.НТУУ“КПІ”.ТР5168_20Б 13-2

Аркушів 5

2020

АНОТАЦІЯ

Метою роботи було створення додатку для підключення реляційної бази даних до онтології предметної області. Система надає можливість підключення реляційної бази даних до онтології предметної області для виконання операцій над даними. Додаток забезпечує можливість виконувати SPARQL-запити до реляційної бази даних.

ЗМІСТ

1 ВІДОМОСТІ ПРО ПРОГРАМНИЙ МОДУЛЬ

Даний програмний модуль розроблено у середовищі IntelliJ IDEA та WebStrom, використовуючи об’єктно-орієнтовану мову програмування Java та не типізовану мову програмування JavaScript фреймворки Spring, , Angular7 та деякі додаткові бібліотеки.

Програмний модуль призначений для побудови команд створення БД, занесення даних, та SPARQL-запиту в SQL-запит.

1.1. Опис логічної структури

Було розроблено веб-додаток, основною задачею якого є підключення реляційної бази даних до онтології предметної області. Основними проблемами, які розв’язує дана робота є налагодження сумісної роботи між БД та онтологією, звідси постає проблема в конвертації типів даних БД та онтології, та правильності конвертації SPARQL-запиту в SQL-запит.

1.2. Вхідні та вихідні дані

Вхідними даними для системи є файл онтології у форматі .owl, параметри підключення до реляційної бази даних та SPARQL -запит.

Вихідними даними є конвертований запит та результат виконання запиту.

2 ВИКОРИСТАНІ ТЕХНІЧНІ ЗАСОБИ

Програмний модуль було протестовано в браузері Google Chrome 83.0 на персональному комп’ютері, який працює на базі процесору 1,8 GHz Intel Core i5 та має 12 Гб оперативної пам’яті. Розроблене програмне забезпечення є кросбраузереним та кросплатформенним, що дозволяє запускати його на комп’ютерах будь-якої потужності та в будь-яких сучасних браузерах.