

**Akademia Górniczo-Hutnicza
im. Stanisława Staszica w Krakowie**

Wydział Fizyki i Informatyki Stosowanej



PRACA MAGISTERSKA

ADRIAN KAŁUZIŃSKI

**ANALIZA MOŻLIWOŚCI ZASTOSOWAŃ GRAFOWYCH BAZ
DANYCH W BUDOWANIU I PRZESZUKIWANIU SEMANTIC
WEB**

PROMOTOR:
dr inż. Antoni Dydejczyk

Kraków 2015

OŚWIADCZENIE AUTORA PRACY

OŚWIADCZAM, ŚWIADOMY ODPOWIEDZIALNOŚCI KARNEJ ZA POŚWIADCZENIE NIEPRAWDY, ŻE NINIEJSZĄ PRACĘ DYPLOMOWĄ WYKONAŁEM OSOBIŚCIE I SAMODZIELNIE, I NIE KORZYSTAŁEM ZE ŹRÓDEŁ INNYCH NIŻ WYMNIENIONE W PRACY.

.....

PODPIS

Kraków, 27 września 2015

Tematyka pracy magisterskiej i praktyki dyplomowej Adriana Kałuzińskiego, studenta V roku studiów kierunku informatyka stosowana, specjalności Grafika komputerowa i przetwarzanie obrazów

Temat pracy magisterskiej: Analiza możliwości zastosowań grafowych baz danych w budowaniu i przeszukiwaniu Semantic Web

Opiekun pracy: dr inż. Antoni Dydejczyk

Recenzenci pracy:

Miejsce praktyki dyplomowej: WFiIS AGH, Kraków

Program pracy magisterskiej i praktyki dyplomowej

1. Omówienie planu pracy magisterskiej z opiekunem.
2. Zebranie i zapoznanie się z istniejącymi standardami i literaturą dotyczącą Semantic Web.
3. Wybranie dostępnej na rynku grafowej bazy danych i zbadanie możliwości jej wykorzystania pod względem czasowym i finansowym.
4. Realizacja praktycznej części pracy.
5. Analiza wyników i omówienie części praktycznej z opiekunem.
6. Opracowanie redakcyjne pracy.

Termin oddania w dziekanacie:

.....
(podpis kierownika katedry)

.....
(podpis opiekuna)

Spis treści

1. Wstęp	10
2. The Semantic Web	11
2.1. The Semantic Web	11
2.2. Ontologia	12
2.2.1. Przykład ontologii	12
2.3. Semantic Web Cake	13
2.3.1. UTF-8, Unicode	13
2.3.2. Uniform Resource Identifier	14
2.3.3. Internationalized Resource Identifier	14
2.3.4. XML.....	15
2.3.5. Resource Description Framework (RDF)	16
2.3.6. RDF Schema	19

2.3.7. SPARQL.....	19
2.3.8. OWL.....	21
2.4. Przykładowe aplikacje zgodne z Semantic Web.....	22
2.4.1. GeoNames.....	22
2.4.2. DBPedia	23
3. Wykorzystane technologie	24
3.1. Standardy aplikacji webowych	24
3.1.1. JavaScript	24
3.1.2. REST	24
3.1.3. MVC.....	25
3.1.4. AJAX.....	26
3.1.5. DOM	26
3.1.6. JSON	28
3.2. Użyte aplikacje i biblioteki	30
3.2.1. Spring Framework.....	30
3.2.2. Apache Maven	31
3.2.3. JDBC - połączenie z bazą danych.....	32
3.2.4. Technologie wykorzystane technologie przy tworzeniu interfejsu graficznego.	32
4. Grafowa baza danych Neo4j	34
4.1. Bazy NoSQL.....	34
4.2. Relacyjny model danych.....	34
4.3. Bazy NoSQL.....	35
4.3.1. Cechy NoSQL	37
4.3.2. Klasyfikacja baz NoSQL.....	38
4.4. Graf	39
4.5. Grafowa baza danych Neo4j	40
4.5.1. Struktura danych w bazie Neo4j	40
4.5.2. Inne grafowe bazy danych istniejące na rynku	41
4.5.3. Możliwości Neo4j	43
4.5.4. Język Cypher.....	43
4.5.5. CREATE.....	44
4.5.6. MATCH.....	44
4.5.7. WHERE	44
4.5.8. Interfejsy dostępu do bazy danych Neo4j	45
4.5.9. JDBC	49
4.6. Wady Neo4j	49
5. Opis aplikacji	50
5.1. Funkcjonalność aplikacji	50
5.1.1. Założenia funkcjonalności	50
5.1.2. Klasa.....	50
5.1.3. Atrybut	51
5.1.4. Relacja.....	52
5.1.5. Metadane.....	53
5.2. Wyszukiwanie	54
5.3. REST API	55

5.4. Diagram klas	55
5.5. Opis pełnego interfejsu REST i klas.....	55
5.6. Zaimplementowana Ontologia PACSO - opis co to jest.....	55
6. Zakończenie	56
6.1. Napotkane problemy i możliwości rozwinięcia funkcjonalności.....	56
6.2. Wyniki pracy	56
7. Bibliografia	57
I Dodatek A	61
7.1. Szczegóły implementacyjne.....	62
7.2. Zawartość dołączonego do pracy nośnika	62
7.2.1. Struktura katalogów aplikacji	62
7.3. Instrukcja wdrożeniowa	62
7.4. Interfejs REST dla Semantic Web	63

Recenzja Opiekuna

Recenzja 2

1. Wstęp

Od ponad dekady trwają badania nad nowym modelem przetwarzania danych z wielu źródeł jakim jest Semantic Web. Rozwój technologii w ostatnich latach spowodował olbrzymi wzrost danych w Internecie. W sierpniu 2015 stron indeksowanych przez największe przeglądarki (Google, Bing) było 4,77 miliarda.

Za sprawą prostych w użyciu i w pełni interaktywnych portali szeroko pojęta wiedza stała się dostępna dla każdego. Taka ilość informacji niestety pociąga za sobą problemy. Coraz trudniejsze staje się przeszukiwanie sieci przy użyciu standardowego przeszukiwania opartego o słowa kluczowe. Zamiast naturalnych zapytań takich jak „W którym roku i przez kogo założono Akademię Krakowską?” należy korzystać raczej z „Akademia Krakowska założenie”. Konieczność maksymalnego upraszczania zapytań utrudnia wyszukiwanie tego, co jest potrzebne.

Jednym z rozwiązań problemu jest pełna analiza kontekstu wprowadzanych zapytań. Projekt Semantic Web wchodzący w skład nowopowstałego standardu Web 3.0 ma pomóc w kontroli informacji. Aplikacje na nim oparte „starają się” zrozumieć zaimki pytające, znaleźć kontekst i przeprowadzić stosowne wnioskowanie.

Jak zostanie zaprezentowane w pracy, sieć semantyczna jest rozbudowanym grafem skierowanym etykietowanym. Od zaproponowania w 1970 roku przez Edgar F. Codda modelu relacyjnego organizacji danych, relacyjne bazy danych stały się dominujące w informatyce. Rozwój aplikacji, w których dane nie są tabelami, a skomplikowanymi sieciami zależności, takich jak serwisy społecznościowe, wymusił badania nad alternatywnymi. Jedną z alternatyw są grafowe bazy danych, które przechowują dane w postaci grafów skierowanych etykietowanych. Struktura danych przechowywanych w tej bazie dokładnie odpowiada założeniom Semantic Web. Podjęto zatem próbę zbadania czy grafowe bazy danych mogą uprościć składowanie danych w sieciach semantycznych.

Obecna wersja Semantic Web jest zbudowana na dość dużym zbiorze technologii zwanym Semantic Web Cake. Celem niniejszej pracy jest przegląd literaturowy z zakresu Semantic Web a następnie analiza możliwości zastosowania grafowych baz danych w budowaniu i przeszukiwaniu sieci semantycznej, bez konieczności zastosowania wszystkich elementów rozbudowanego Semantic Web Cake. Przeprowadzono badanie funkcjonalności dostarczanych przez bazę Neo4j i w ramach pracy powstała aplikacja, która zezwala na zbudowanie modelu sieci, uzupełnianie go i przeszukiwanie. Praca magisterska składa się z 3 rozdziałów. Pierwszy rozdział jest wstępem teoretycznym opisującym ideę Semantic Web, Web 3.0 oraz aktualny trend jej rozwoju. Drugi rozdział jest opisem technologii wykorzystanych przy rozwiązaniu problemu. Trzeci rozdział wprowadza do funkcjonalności grafowych baz danych. Ostatnia część pracy jest opisem samej aplikacji.

2. The Semantic Web

2.1. The Semantic Web

The Semantic Web jest to projekt, który ma umożliwić szybkie przetwarzanie danych z różnych źródeł w taki sposób, że ich znaczenie będzie interpretowane a kontekst danych zostanie „zrozumiany” przez programy i urządzenia. Jednym z założeń Semantic Web¹ jest pojawienie się zbiorów danych i wyszukiwarek ~~do nich~~, które umożliwią użytkownikowi uzyskanie dostępu do danych na podstawie zrozumiałego kontekstu, a nie jak ~~dotychczas poprzez~~ słowa kluczowe, adres ~~strony, umieszczenie~~ w katalogach. [1][2] Fundamentem jest skorzystanie z założenia, iż większość danych w otaczającym nas świecie jest w jednoznaczny sposób powiązane ze sobą. Zamiast, jak to się czyni obecnie, przekazywania samych zbiorów danych, maszyny będą się ze sobą komunikowały poprzez uzgodniony wzór meta-danych określających kontekst a w przyszłości także naturalne wnioskowane (jeżeli A jest bratem B, a B jest bratem C, to A jest rodzeństwem z C). Obrazowym zastosowaniem komercyjnym Semantic Web jest wyszukiwarka hoteli. W sytuacji takiej hotele muszą udostępnić zunifikowany interfejs dostępowy do swoich danych. Same dane również muszą być prawidłowo wzbogacone meta-danymi. W takiej sieci użytkownik wysłałby żądanie, że szuka hotelu w pewnej miejscowości a system wykonałby je poprzez przejrzenie witryn hoteli i ich cenników. Kluczowym założeniem tego systemu jest to, że system dysponuje wyłącznie listą hoteli, a wszystkie informacje kluczowe (w tym przypadku lokalizacja) są umieszczone na witrynach samych hoteli. Warunkiem istnienia takiego systemu jest konieczność wprowadzenia ujednoliconego opisu danych w sieci.

Z założenia projekt ten, oznaczany jako Web 3.0 ma stopniowo wypierać obecnie dominującą i powszechnie nam znaną Web 2.0. [3]

Semantic Web a Web 3.0

Semantic Web ma służyć jako podstawa technologiczna do rozwoju nowego standardu stron internetowych oznaczany jako Web 3.0.

Aby w pełni podkreślić znaczenie Semantic Web w przyszłym rozwoju Internetu należy przedstawić poszczególne etapy rozwoju WWW. Rozwój Web przedstawia się następująco:

- Web 1.0 był pierwszą generacją witryn internetowych, ~~które były~~ nakierunkowane głównie na bierny odczyt ze strony użytkowników. Właściciele witryn umieszczali na nich pewne informacje, których treść nie mogła być współtworzona przez użytkowników. Własność intelektualna danych umieszczonych na serwisach była oczywista (właściciel witryny), a interakcja użytkowników ograniczona była do minimum w postaci księgi gości i komentarzy.
- Web 2.0 wdrażana jest stopniowo od pierwszej dekady XXI wieku wraz rozpowszechnieniem takich technologii jak SOAP, AJAX, REST. Jest ona nakierkowana na współtworzenie danych przez sieci użytkowników. Architektura zamiast wcześniejszej klient-serwer przypomina układ peer-to-peer. Najbardziej typowymi przedstawicielami tego trendu są witryny takie jak Wikipedia oraz

¹W literaturze brakuje polskiego odpowiednika dla The Semantic Web. W wolnym tłumaczeniu z języka angielskiego jest to Sieć Semantyczna. Warto przy tym wspomnieć, że choć nazwa jest ładząco podobna do sieci semantycznych stosowanych w sztucznych sieciach neuronowych i automatyce [7], nie są to pojęcia równoważne (mimo że obie sieci semantyczne przedstawia się w postaci grafu skierowanego etykietowanego).

wszystkie serwisy społecznościowe takie jak Facebook i Twitter. Producent aplikacji udostępnia pewną platformę komunikacyjną i dostarcza podstawową treść, która zostanie następnie rozbudowywana przez użytkowników. Za moderację treści odpowiadają uprzywilejowane osoby z tej społeczności, którymi nie są najczęściej przedstawiciele producenta aplikacji. [4][5]

- Web 3.0 jest obecnie w powijakach i ma stanowić rozszerzenie istniejących technologii ze standardu Web 2.0. Głównym motorem rozwoju ma być połączenie już istniejących, rozproszonych po systemach danych. Nie jest to rewolucja technologiczna a raczej powolna ewolucja, gdyż Web 3.0 także ma korzystać z istniejącego od lat protokołu HTTP. Jak pisze wynalazca World Wide Web i dyrektor konsorcjum W3C w magazynie American Scientific, Timothy "Tim" Berners-Lee: „*Semantic Web nie jest niezależną Web ale rozszerzeniem istniejącej, w której informacje mają dobrze zdefiniowane znaczenie, pozwalając komputerom i ludziom lepiej współpracować ze sobą współpracować*”. [6] Idea nie jest nowa bo ma już 14 lat ale powstanie nowych technologii takich jak OWL, RDF i RDF Schema pozwala stopniowo wdrażać ją w życie. Rozpowszechnienie w ostatnich latach standardu Unicode usuwa narzut związany ze zmianą kodowania znaków. Wynikiem odczuwalnym przez użytkowników internetu będzie jeszcze lepsze dopasowanie wyników wyszukiwań. Użytkownicy systemów Web 3.0 uzyskają odpowiedzi na zapytania nawet wtedy, gdy nie będą precyzyjne.

2.2. Ontologia

Istotnym pojęciem projektu Semantic Web jest pojęcie ontologii. Termin ten wywodzący się z języka języka greckiego, złożony jest ze słów ontos- – istnienie oraz -logia czyli nauka, wiedza. [23]. Ontologia jest to pewna część wiedzy złożona ze zbioru pojęć (konceptów - klas, instancji) oraz połączeń istniejących pomiędzy nimi.

Służy ona to tworzenia opisów wiedzy zrozumiałych dla ludzi. Powinna ona zawierać:

- Pojęcia wchodzące w skład tej dziedziny
- Cechy i związki pomiędzy własnościami
- Relacje pomiędzy pojęciami
- Obiekty wchodzące w skład dziedziny

W skrócie, ontologia opisuje jak zbudować modele dla pewnego wydzielonego zakresu wiedzy. Tworzenie ontologii umożliwia szereg technologii takich jak RDF, OWL, XML, OCML i RDS Schema. Część z nich zostanie opisana w następnych rozdziałach tej pracy. [4]

2.2.1. Przykład ontologii

W przypadku tworzenia własnej ontologii należy na początku skupić się na podstawowych elementach ontologii czyli klasach, ich cechach oraz instancjach. Te parametry ontologii zostaną pokrótce opisane. Dla osób znających podstawy programowania obiektowego, ontologia okaże się czymś intuicyjnym.

- Klasa jest podstawową jednostką umożliwiającą identyfikację obiektów wchodzących w skład ontologii. Przykładem klasy może być Pojazd reprezentujący abstrakcyjny zbiór wszystkich pojazdów. Zbiór klas definiuje domenę ontologii. Klasa może zawierać swoje podklasy,
- Podklasa jest dalej abstrakcyjny, ale nieco bardziej precyzyjny opis grupy elementów. Przykładem podklasy mogą być Coupé, Kombi, Minivan ale też Samochód Spalinowy, Samochód Hybrydowy, Samochód Elektryczny.

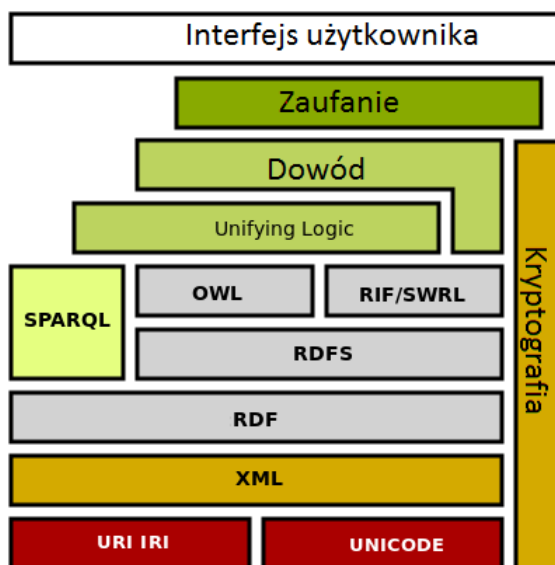
- Instancje są to realni przedstawiciele klas. Instancją jest na przykład Porsche Carrera 911, które jest zarówno klasy Coupé, jak i Samochodem Spalinowym. Będąc instancją podklasy Samochód Spalinowy staje się też instancją klasy Samochód. W związku z tym instancja może mieć więcej niż jedną klasę. Pojedyncza instancja jest najbardziej przyzyjnym conceptem wiedzy w ontologii.
- Cechy (bądź też atrybuty) są to własności opisujące możliwości klas i podklas. W przypadku aut mogą to być masa, moc silnika, długość. Instancja może posiadać cechę, której nie posiada sama klasa (jest to istotna różnica pomiędzy definicją instancji w programowaniu obiektowym, a ontologiami). Niekiedy na klasy można narzucić także dodatkowe ograniczenia. [24]

Tworzenie **ontologii zrealizowania** następujących kroków:

1. Zdefiniowania klas należących do dziedziny,
2. Ustalenie hierarchii między klasami, a podklasami,
3. Ustalenie dostępnych atrybutów klas,
4. Wypełnienie wartości atrybutów w instancjach. [24]

2.3. Semantic Web Cake

Obecną strukturę Semantic Web **graficznie przedstawia się w** postaci piramidy nazywanej Semantic Web Stack lub Semantic Web Cake, co w dosłownym tłumaczeniu oznacza odpowiednio Stos Sieci Semantycznej lub Ciasto Sieci Semantycznej. Jest to hierarchiczny zbiór technologii i standardów W3C, które mogą być wykorzystane do rozwoju witryn opartych o Semantic Web. Lista wspieranych elementów jest systematycznie rozwijana. ~~Już na pierwszy rzut oka widać, że stos ten jest rozbudowany co skłoniło autora tej pracy do prób zmniejszenia liczby potrzebnych technologii do minimum.~~



Rysunek 2.1: Semantic Web Stack. Został on opracowany w oparciu o ilustrację autorstwa Tima Berners-Lee [9] oraz rozszerzenie ze ten pochodzi ze strony Wikipedia [8].

2.3.1. UTF-8, Unicode

Unicode jest to system kodowania, reprezentacji i obsługi tekstu, który docelowo ma obsługiwać wszystkie pisma na świecie. Standard bardzo szybko się rozwija, a jego najnowsza, **ósmą**, wersja powstała w czerwcu 2015 roku i obsługuje **120000 znaków ze 129 systemów zapisu**. Ta popularność w

naturalny sposób uzasadnia umieszczenie go jako podstawę Semantic Web Stack. Unicode jest użyty jako podstawa bytecode'u języka programowania Java, który został wykorzystany przy tworzeniu tej pracy.

UTF-8 – jest to jeden z systemów kodowania Unicode, wykorzystujący od 8 do 32 bitów do zapisania znaku. UTF-8 jest kompatybilny wstecz z ASCII. [13] Jest to najpopularniejszy (w lipcu 2015 wykorzystuje go 84.6% witryn) standard kodowania stosowany w internecie. [14]

2.3.2. Uniform Resource Identifier

URI (ang. Uniform Resource Identifier) jest standardem **internetowym umożliwiającym** łatwą identyfikację zasobów w sieci. URI jest to łańcuch znaków w formacie zdefiniowanym przez standard RFC 2396 [18]. **Ten ciąg znaków** jest złożony z lokatora zasobu URL (ang. Uniform Resource Locator) lub/i nazwy zasoby URN (ang. Uniform Resource Name).



Rysunek 2.2: Przykład Uniform Resource Identifier

Pełna składnia URI wygląda następująco:

<schemat> : <część hierarchiczna> [? <zapytanie>] [# <fragment>]

gdzie elementy otoczone nawiasami kwadratowymi są opcjonalne:

- schemat określa protokół np. **http**,
- część hierarchiczna jest złożona z nazwy hosta (np. **agh.edu.pl**). ~~Ewentualnie może zawierać dodatkowe~~ nazwę użytkownika i jego hasło oddzielone dwukropkiem (*użytkownik:hasło@host*). Notację złożoną z nazwy użytkownika i hasła niekiedy stosuje się przy dostępie do zdalnych baz danych. Ma to na przykład miejsce w serwisie MongoLab udostępniającą przestrzeń dyskową w ramach chmury typu Database-as-a-Service:

`mongodb://<dbuser>:<dbpassword>@ds039311.mongolab.com:39311/database_name`

- zapytanie zaczyna się od znaku zapytania i ma postać **par klucz wartość** oddzielonych ampersandami: `?klucz1=wartosc1&klucz2=wartosc2`
- fragment jest zależny od typu zasobu i jest jego częścią. Interpretacja może być różna - w przypadku dokumentów tekstowych może to być np. **rozdział książki lub [18]**

2.3.3. Internationalized Resource Identifier

IRI (ang. Internationalized Resource Identifier) jest standardem **internetowym zakładającym** internacjonalizację a tym samym pozwalającym na szybką lokalizację zasobów w witrynach internetowych. Ma on zastąpić standard URI [10]. IRI jest jednoznacznym łańcuchem znaków, zapisanym zgodnie ze składnią określoną w standardzie określającym zarówno adres, jak i nazwę oczekiwanego zasobu. Standard IRI jest zbudowany w oparciu o kodowanie znaków ISO/IEC 10646 **Universal Coded Character Set** **jest zsynchronizowany** ze znanym Unicode (także występującym w Semantic Web **Stack**), co **zezwala** na stosowanie nazw z alfabetów innych niż angielski. Jest on zgodny z historycznie stosowanym standardem **URI zezwalając** na mapowanie znaków Unicode do ich kodów ASCII i odwrotnie. IRI **zezwala** na zamianę znaków międzynarodowych (np. znaki <, > stosowane w XML) do kodowania opartego o **procencie tylko gdy to jest** konieczne. Przykładowo adres w notacji XML

`http://www.example.org/red%09rosé#red`

jest konwertowany do

`http://www.example.org/red%09ros%C3%A9#red`

zamiast

`http%3A%2F%2Fwww.example.org%2Fred%2509ros%C3%A9%23red`

co miałyby miejsce przy konwersji naiwnej. Konwersja naiwna mogłaby spowodować nieprawidłowe działanie przeglądarek. Standard IRI opisany jest w dokumencie RFC 3987. [10]

2.3.4. XML

XML (ang. Extensible Markup Language, Rozszerzalny Język Znaczników) jest to uniwersalny język znaczników zaprojektowany w 1996 i przeznaczony do reprezentowania różnych danych w strukturalizowany sposób. Jego cechą charakterystyczną jest to, że kodowanie w XML generuje dokument, który jest czytelny zarówno dla człowieka, jak i dla maszyny. Obecnie W3C rekomenduje zarówno XML 1.0 w edycji piątek, jak i XML 1.1 w edycji drugiej a wszystkie parsery powinny być zgodne z obiema wersjami [11][12]. Główną różnicą pomiędzy XML 1.1 a XML 1.0 jest to, że XML 1.1 wymusza korzystanie z najnowszej wersji standardu Unicode. [12]

Specyfikacja języka znaczników wprowadza terminologię określającą podstawowe składniki dokumentu XML.

- **Deklaracja XML** - dokument XML może rozpocząć się od deklaracji opisującej siebie tj. wersja języka, użyte kodowanie i flagę czy zawiera odwołanie do innych dokumentów XML.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

- **Znaczniki** (ang. tags) - składnia znaczników istnieje w 2 wersjach:

- Istnieje znacznik otwierający i domykający. Znacznik zaczyna się od <, a kończy na >. Znacznik domykający zaczyna się od znaków </. Tekst pomiędzy znakami <,> określa nazwę znacznika np. <osoba></osoba>. Nazwa znacznika otwierającego i domykającego jest taka sama i nie może zawierać znaków zastrzeżonych &,<,>,'',''. (zamiast nich należy użyć w zastępstwie symbolami < " itp. znanymi z HTML). Istotny jest fakt, że XML jest czuły na wielkość znaków - </Osoba> nie może być domknięciem dla <osoba>.
- Znacznik może być pustym elementem <osoba/>

- **Element** - pomiędzy znacznikami można umieścić pewne dane, przy czym mogą być one (i często są) elementami zagnieżdżonymi. W poniższym przykładzie uczelnia zawiera w sobie 2 zagnieżdżone elementy typu wydział:

```
<uczelnia><wydział>WFIIS</wydział></wydział></uczelnia>
```

- **Atrybut** - każdy element może posiadać atrybuty wpisane pomiędzy apostrofami lub w cudzysłowie. Atrybut określa pewne własności danego elementu. [11]

```
<uczelnia nazwa="Akademia_Górnicz-Hutnicza_im._Stanisława_Staszica">
```

XSD

XML Schema to standard zdefiniowany przez W3C, który ma zastosowanie w definicji struktury dokumentu XML. Dokumenty XML schema także są strukturami XML co daje dużą przejrzystość w jego odczycie. Poniżej znajduje się przykład dokumentu opisującego adres. Dokument XML, w którego nagłówku ustalono zgodność z dokumentem XSD musi mieć format dokładnie taki jak XSD definiuje. W przeciwnym razie zostanie głośzony błąd.


```

<?xml version="1.0" encoding="utf-8"?>
<xs:schema elementFormDefault="qualified"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Adres">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Ulica" type="xs:string" />
        <xs:element name="NumerBudynku" type="xs:string" />
        <xs:element name="NumerLokalu" use="optional" type="xs:string" />
        <xs:element name="Miasto" type="xs:string" />
        <xs:element name="KodPocztowy" type="PolskiKodPocztowy" />
        <xs:simpleType name="PolskiKodPocztowy">
          <xs:restriction base="xs:string">
            <xs:pattern value="[0-9]{2}-[0-9]{3}" />
          </xs:restriction>
        </xs:simpleType>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>

```

Powyższy schemat XSD wymusza od korzystającego z niego dokumentu XML aby istniał węzeł adres, w którym:

- znajdują się obowiązkowe węzły Ulica, NumerBudynku, Miasto, KodPocztowy, przy czym KodPocztowy jest typu enumerowanego PolskiKodPocztowy,
- mogą znajdować się dodatkowe węzły NumerLokalu
- PolskiKodPocztowy jest nowym typem zbudowanym w oparciu o xs:string (łańcuch znakowy) i może przyjmować tylko wartości zgodne z wyrażeniem regularnym [0-9]{2}-[0-9]{3} (dwie cyfry, myślnik i trzy cyfry co odpowiada kodom pocztowym stosowanym w Polsce)

Przykładowy dokument zgodny z powyższym schematem mógłby wyglądać tak:

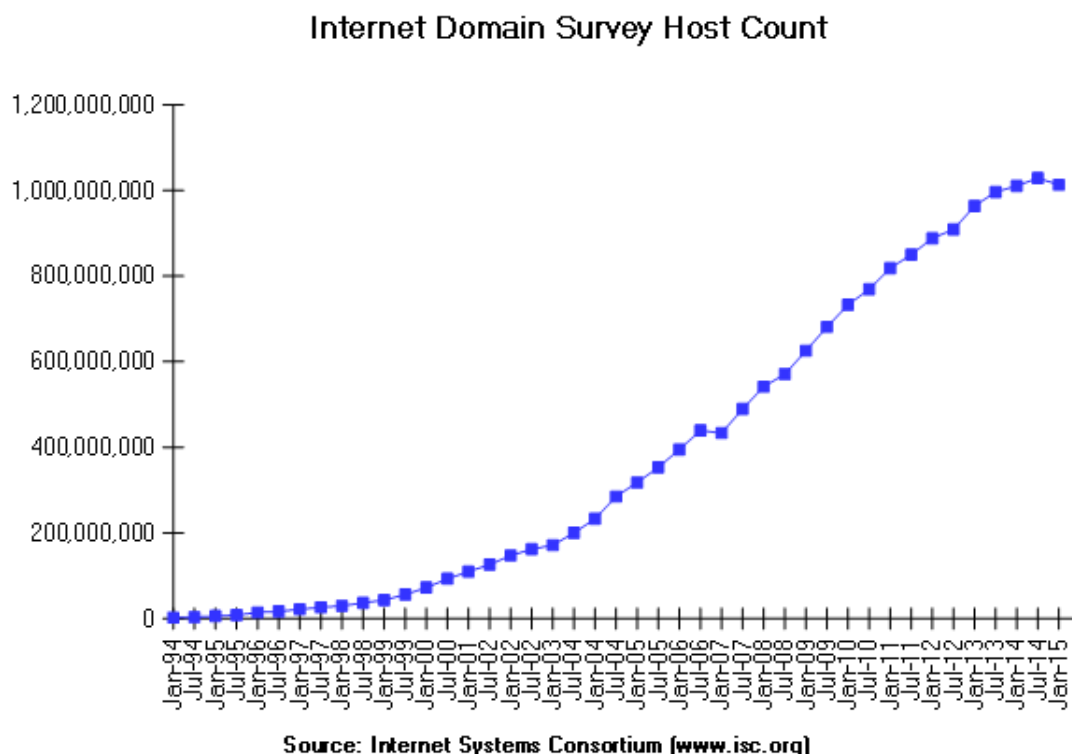
```

<?xml version="1.0" encoding="utf-8"?>
<Adres
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="Adres.xsd">
  <Ulica>Sienkiewicz</Street>
  <NumerBudynku>13A</NumerBudynku>
  <NumerLokalu>1</NumerLokalu>
  <Miasto>Kraków</Miasto>
  <KodPocztowy>30-233</PostCode>
</Adres>

```

2.3.5. Resource Description Framework (RDF)

Zbiór danych w internecie wzrasta na poziomie trudnym do oszacowania. Według danych Internet Systems Consortium, w styczniu 2015 zarejestrowanych było ponad 1,2 mld hostów.



Rysunek 2.3: Liczba hostów na świecie w skali logarytmicznej. Rysunek pochodzi z witryny Internet Systems Consortium. [16]

Taka ilość danych (tekstu, zdjęć, wideo **itd.**) wymaga wprowadzenia ujednoliconego standardu opisu czyli **meta-danych**. Jedną z prób opisu danych w internecie jest standard RDF. Próbuje on opisać istniejące już dane bez konieczności ich modyfikacji.

RDF (ang. Resource Description Framework) - język ze składnią opartą zazwyczaj na XML, umożliwiający opis zasobów sieci Web. Podobnie jak wiele innych standardów stosowanych w **internecie został** opracowany przez konsorcjum W3C i w wersji 1.1 **ustalony** jako rekomendacja 25 lutego 2014 roku. Jest to język dający podłoże technologiczne do zrealizowania koncepcji ujednolicenia opisu danych. Nie jest to opis danych przyjazny człowiekowi, **ale** jest czytelny dla komputerów [1]. Pierwszym wprowadzonym RDF był RDF/XML czyli RDF przechowywany w dokumencie XML. Z czasem powstały również inne notacje, jednakże tylko RDF/XML jest rekomendacją W3C. [21]

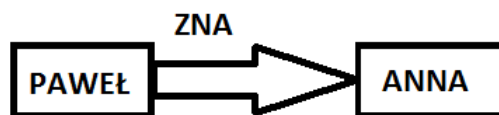
Trójki

~~Jest to taki model danych, który stara się opisać wszystkie dane użyte w Semantic Web definiując relacje pomiędzy nimi.~~ Wyrażenia opisujące dane przechowuje pod postacią tak zwanych **trójek** (ang. triples). [17] Graficzna realizacja tej idei jest grafem skierowanym, w którym zarówno krawędzie, jak i węzły są etykietowane. [19] Faktyczne dane są przechowywane pod postacią węzłów, a relacje oznaczamy jako krawędzie.

Trójka jest złożona z następujących elementów:

- Podmiot (ang. subject) - aktualnie opisywany zasób **b**
- Predykat/orzeczenie (ang. predicate) - własność lub zależność podmiotu
- Obiekt/dopełnienie (ang. object) - wartość własności lub zależności podmiotu zgodna z predykatem **m**

Trójka podmiot-predykat-obiekt daje nam stwierdzenie (ang. statement):



Rysunek 2.4: Przykład trójki

Stwierdzenie „Paweł zna Annę” można przedstawić jako trójkę Paweł-Zna-Anna, gdzie Paweł to podmiot, Zna to predykat a Anna to obiekt.

Trójka w jednoznaczny sposób umożliwia uzyskanie pozostałych dwóch wartości znając tylko jedną z nich. Taka prosta struktura daje to nam możliwość zdefiniowania takich zapytań jak:

1. Kto zna Annę?
2. Kogo zna Paweł?
3. Kto zna kogo?
4. Co wiąże Pawła i Annę?

Przykład modelu RDF

Poniżej znajduje się przykład modelu RDF w notacji RDF/XML opisującego przykładowy zasób o nazwie bluza.

```

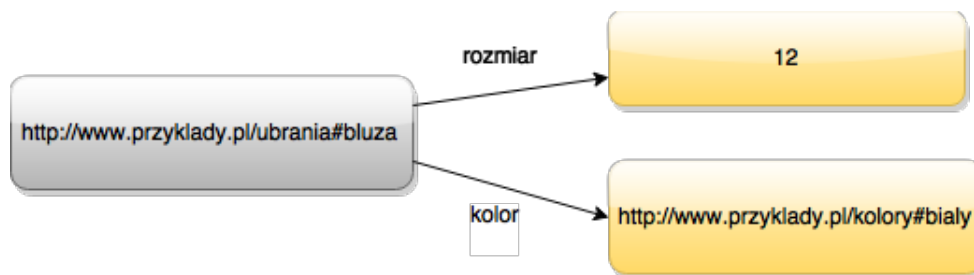
<?xml version="1.0" ?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    <rdf:Description rdf:about="http://www.przyklady.pl/ubrania#bluza">
        <rozmiar>12</rozmiar>
        <kolor rdf:resource="http://www.przyklady.pl/kolory#bialy"/>
    </rdf:Description>
</rdf:RDF>
  
```

Jak łatwo zauważyć, modele RDF można przedstawić w postaci dokumentów XML. Na podstawie powyższego przykładu można wyrazić następujące stwierdzenia:

- Zasób *http://www.przyklady.pl/ubrania#bluza* ma rozmiar 12
- Zasób *http://www.przyklady.pl/ubrania#bluza* ma kolor, o wartości znajdującej się w zewnętrznym źródle i wynoszącej *http://www.przyklady.pl/kolory#bialy*

Podmiotem jest *http://www.przyklady.pl/ubrania#bluza*, natomiast predykatami rozmiar z obiektem 12 i kolor z obiektem *http://www.przyklady.pl/kolory#bialy*. Dokument taki zawiera w sobie dwie trójki.

Po wizualizacji takiego modelu otrzymamy następujący graf skierowany.



Rysunek 2.5: RDF dla zasobu bluza

2.3.6. RDF Schema

RDF Schema (RDFS) jest językiem reprezentacji wiedzy, powstałym na bazie RDF. RDF służy za bazę opisu relacji pomiędzy danymi i opisuje ją jako graf skierowany. RDF Schema jest rozszerzeniem tego standardu wprowadzając dodatkowe pojęcia systematyzujące kształt grafu. Zapisane za pomocą RDF Schema struktury, nazywane są słownikami RDF (ang. RDF vocabularies). Język RDF Schema jest rekomendacją W3C w zastosowaniach Semantic Web. [20] Wprowadzenie dodatkowych pojęć klas (ang. classes), podklas (ang. subclasses) i typów pozwala na bardziej precyzyjny opis danych.

Przykład RDF z poprzedniej sekcji można rozszerzyć o elementy RDF Schema.

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"

    <rdfs:Class rdf:ID="Odzież"></rdfs:Class>
    <rdfs:Class rdf:ID="OdzieżMęska">
        <rdfs:subClassOf rdf:resource="#Odzież" />
        <rdfs:description>Jest to odzież męska</rdfs:description>
    </rdfs:Class>
    <rdfs:Class rdf:ID="OdzieżDamska">
        <rdfs:subClassOf rdf:resource="#Odzież" />
        <rdfs:description>Jest to odzież damska</rdfs:description>
    </rdfs:Class>

    <rdf:Description rdf:about="http://www.przyklady.pl/ubrania#bluza">
        <rdf:type rdf:resource="#OdzieżMęska" />
        <rozmiar rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1
        <kolor rdf:resource="http://www.przyklady.pl/kolory#bialy" />
    </rdf:Description>
</rdf:RDF>
```

Wprowadzono trzy nowe klasy: *Odzież*, *OdzieżMęska* i *OdzieżDamska*, przy czym *OdzieżMęska* i *OdzieżDamska* są podklasami *Odzieży*. Jest to zachowanie podobne do dziedziczenia znanego z programowania obiektowego. Należy przy tym zwrócić uwagę na fakt, iż jeden wierzchołek może należeć równocześnie do dowolnej liczby klas.

RDF Schema umożliwia narzucenie ograniczeń typów na dane. W tym przypadku narzucono ograniczenie typu rozmiaru do liczby całkowitej (*rdf:datatype="http://www.w3.org/2001/XMLSchema#int"*).

2.3.7. SPARQL

Mimo, że najczęściej wykonywaną operacją na dokumentach jest ich odczyt, to RDF nie definiuje żadnego sposobu dostępu do danych. Aby sprostać temu wymaganiu powstał język zapytań SPARQL.

SPARQL (ang. SPARQL Protocol and RDF Query Language) - jest to protokół i język zapytań dla RDF. Umożliwia przeszukiwanie i manipulację danymi przechowywanymi w formacie Resource Description Framework.

Przez W3C został oznaczony jako oficjalna rekomendacja. Składnia tego języka ze względu na słowa kluczowe w pewnym stopniu przypomina SQL, gdyż dokumenty RDF można potraktować jako tabelę złożoną z 3 kolumn - kolumny podmiotów, kolumny predykatów i kolumny obiektów.

jednakże ze względu na duże różnice pomiędzy relacyjnymi bazami danych a dokumentami RDF, nie jest to podobieństwo duże.²

Zapytania SPARQL zezwalają na uzyskanie zasobu wchodzącego w skład węzła grafu RDF, a także uzyskać informacje o relacjach łączących go z innymi zasobami. Wprowadzenie zmiennych poprzedzonych znakami specjalnymi \$, ? ułatwia tworzenie zapytań. Klauzula WHERE narzuca ograniczenie na oczekiwany podzbiór wyników.

SPARQL dostarcza zbiór operacji takich jak JOIN, AGGREGATE umożliwiających przeprowadzenie wstępnej analizy danych. Język dostarcza 4 zapytania różniące się przede wszystkim oczekiwanym formatem wyjściowym:

- **SELECT** - zwraca nieprzetworzone dane w formie tabeli. Zapytanie analogiczne do selekcji z tradycyjnych, relacyjnych baz danych
- **ASK** - stosowane w przypadkach zapytań logicznych. Zwraca ono dane w formie prawda/fałsz
- **CONSTRUCT** - zapytanie służące do pobrania opisu modelu RDF. Zwraca dane w postaci czy-stego obiektu RDF/XML.
- **DESCRIBE** - zwraca pojedynczy graf RDF zawierający dane o zasobach. Jest to najprostsze z zapytań, gdyż nie wymaga klauzuli WHERE oczekiwanej w pozostałych (wystarczy podać tylko adres węzła). [22]

Ogólny format zapytania wygląda następująco:

```
(SELECT | ASK | CONSTRUCT | DESCRIBE)
[? subject] [? predicate] [? object]
WHERE {
    ?subject predicate ?object
}
```

Poniżej umieszczono rzeczywisty przykład użycia języka SPARQL w aplikacji DBPedia zgodnej z Semantic Web.³ DBPedia jest to serwis, który pobiera dane z Wikipedii i udostępnia je przy użyciu interfejsu operującego na języku SPARQL. Przykładowe zapytanie „Czy Polska jest większa niż Hiszpania?” z serwisu DBPedia można zapisać w ten sposób:

```
PREFIX prop: <http://dbpedia.org/property/>
ASK {
  <http://dbpedia.org/resource/Poland> prop:areaTotal ?poland .
  <http://dbpedia.org/resource/Spain> prop:areaTotal ?spain .
  FILTER(?poland > ?spain).
}
```

Zapytanie do sprawdzi zasoby <http://dbpedia.org/resource/Poland>, <http://dbpedia.org/resource/Spain> w poszukiwaniu pola areaTotal, a następnie zwróci wartość zapytania logicznego, czy wartość areaTotal dla Polski jest większa niż dla Hiszpanii.

²Warto w tym momencie zaznaczyć, że istnieją narzędzia pozwalające na konwersję języka SPARQL do SQL takie jak D2R SERVER dostępne pod adresem <http://http://d2rq.org/d2r-server>. Ateńska Politechnika Narodowa udostępniła nawet narzędzie zamieniające zapytania SPARQL na języki XQuery - <http://www.dblab.ntua.gr/~bikakis/SPARQL2XQuery.html>

³Strona internetowa serwisu DBPedia to: <http://dbpedia.org/>

2.3.8. OWL

Na potrzeby bardziej wymagających zastosowań Semantic Web zaprojektowano język **OWL** będący rozszerzeniem funkcjonalności dostarczanej przez RDF i podobnie jak **on jest** oparty o język XML. Język dostarcza bibliotekę klas i większe obostrzenia nałożone na wartości typów. [35]

Wyróżnia się trzy warianty języka OWL:

- OWL Lite - używany do prostych struktur, w których najistotniejsza jest prędkość przetwarzania. Daje dużą dowolność w opisie klas. Ograniczenia są niewielkie i są to więzy licznosciowe (np. ustalenie mocy zbioru na 0 lub 1). OWL Lite nie dopuszcza klas wyliczeniowych.
- OWL DL - wprowadza podstawowe mechanizmy wnioskowania i oczekuje ograniczeń nałożonych na używane słownictwo RDFS (np. rozdzielenie własności, klas). Nie jest on w pełni zgodny z RDF.
- OWL Full - Stanowi zbiór konstruktorów, które nie posiadają żadnych ograniczeń. Powoduje to przełożenie się na niską efektywność i brak rozstrzygalności. Wariant ten daje pełną dowolność w definicji ontologii, ale wolność ta powoduje, że nie istnieje pełne wsparcie dla wnioskowania.

Przykładem więzów ilościowych dla klas jest **nałożenie ograniczenie takie**, że dane auto może mieć tylko jeden rok produkcji:

```
<owl:Class rdf:ID="RocznikAuta">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#rokProdukcji"/>
      <owl:cardinality>1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

OWL wprowadza własności, które można nałożyć na obiekty:

- Przechodność (TransitiveProperty) - wprowadza wnioskowanie jeśli elementy nie są połączone bezpośrednio. Jeśli element A jest składnikiem elementu B, a element B jest składnikiem elementu C, to element A wchodzi w skład elementu C. Przykładem przechodności może być Zerowe Prawo Termodynamiki i wnioskowanie dotyczące temperatur obiektów.

```
<owl:TransitiveProperty rdf:ID="podElement">
<rdfs:domain rdf:resource="#Element"/>
<rdfs:range rdf:resource="#Element"/>
</owl:TransitiveProperty>
```

- Symetryczność (SymmetricProperty) - wprowadza symetrię pomiędzy obiektami. Jeżeli OsobaA jest przyjacielem OsobyB to można wywnioskować, że osoba B jest także przyjacielem osoby A

```
<owl:SymmetricProperty rdf:ID="jestPrzyjacielem">
<rdfs:domain rdf:resource="#Osoba"/>
<rdfs:range rdf:resource="#Osoba"/>
</owl:SymmetricProperty>
```

- Odwrotność (ObjectProperty) - określa naturalną zależność pomiędzy dwoma obiektami w hierarchii drzewiastej. Jeżeli wiemy, że obiekt A ma rodzica B, to wnioskujemy, że rodzic B na pewno ma dziecko A.

```
<owl:ObjectProperty rdf:ID="maDziecko">
<owl:inverseOf rdf:resource="#maRodzica"/>
</owl:ObjectProperty>
```

- Funkcyjność - własność może mieć jedną unikatową wartość Y dla każdej instancji X. Dobrym odniesieniem jest to, że kobieta może mieć (w większości kultur) maksymalnie jednego męża. Stąd wiemy, że jeżeli pewna osoba jest jej mężem, to inna na pewno nią nie jest.

```
<owl:FunctionalProperty rdf:ID="maMęża">
<rdfs:range rdf:resource="#Mężczyzna" />
<rdfs:domain rdf:resource="#Kobieta" />
</owl:FunctionalProperty>
```

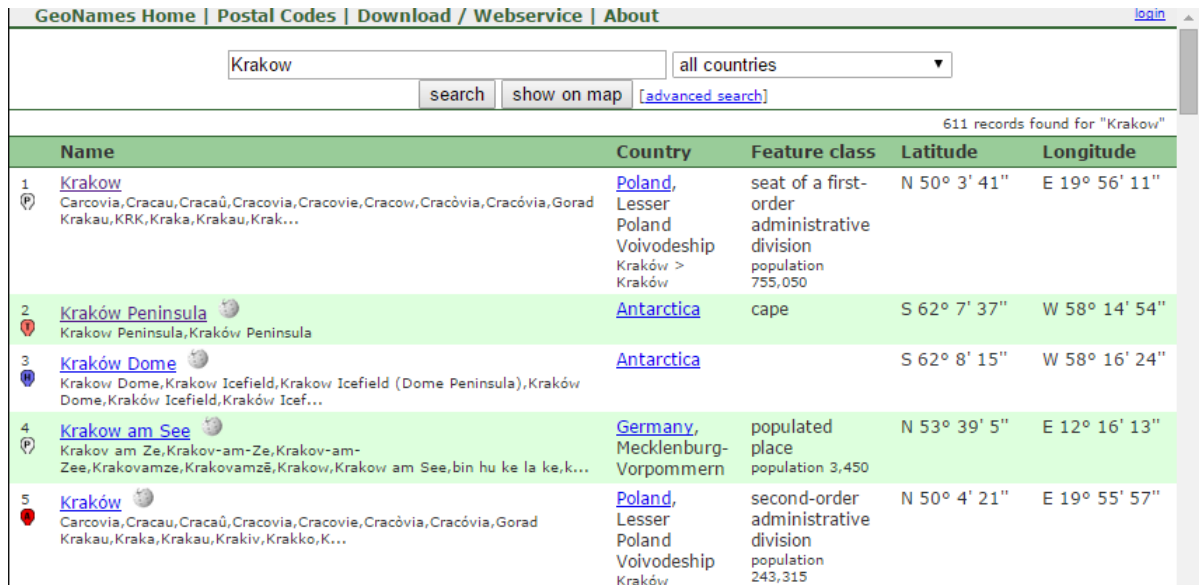
OWL wprowadza także szereg operatorów zbiorowych na klasach takich jak intersekcja (przecięcie). Poniżej znajduje się klasa Blondynka, która jest zbudowana jako część wspólna obiektów, które należą do klas Kobieta i Blondyn. OWL naturalnie wprowadza pozostałe operacje na zbiorach takie jak dopełnienie, unia (suma) oraz dziedziczenie opisane w poprzednim rozdziale.

```
<owl:Class rdf:ID="Blondynka">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:resource="Kobieta">
    <owl:Class rdf:resource="Blondyn">
  </owl:intersectionOf>
</owl:Class>
```

2.4. Przykładowe aplikacje zgodne z Semantic Web

2.4.1. GeoNames

GeoNames jest darmową geograficzną bazą danych dostępną poprzez otwarte usługi sieciowe (ang. web services) zbiór 10 milionów nazw. Przechowywane nazwy mają dodatkowe atrybuty takie jak tłumaczenia do innych języków, współrzędne geograficzne, populacja, podział polityczny itd. Każdy z zasobów jest dostępny poprzez unikalne URI i wyświetlony jako dokument tekstowy HTML. Można uzyskać opis dokumentu w postaci RDF.



Name	Country	Feature class	Latitude	Longitude
1 Krakow Carcovia, Cracau, Cracaù, Cracovia, Cracovie, Cracow, Cracòvia, Cracòvia, Gorad Krakau, KRK, Kraka, Krakau, Krak...	Poland , Lesser Poland Voivodeship Kraków > Kraków	seat of a first-order administrative division population 755,050	N 50° 3' 41"	E 19° 56' 11"
2 Kraków Peninsula Krakow Peninsula, Kraków Peninsula	Antarctica	cape	S 62° 7' 37"	W 58° 14' 54"
3 Kraków Dome Krakow am Ze, Krakow Icefield, Krakow Icefield (Dome Peninsula), Kraków Dome, Kraków Icefield, Kraków Icef...	Antarctica		S 62° 8' 15"	W 58° 16' 24"
4 Krakow am See Krakow am Ze, Krakov-am-Ze, Krakov-am-Zee, Krakovamze, Krakovamzë, Krakow, Krakow am See, bin hu ke la ke, k...	Germany , Mecklenburg-Vorpommern	populated place population 3,450	N 53° 39' 5"	E 12° 16' 13"
5 Kraków Carcovia, Cracau, Cracaù, Cracovia, Cracovie, Cracòvia, Cracòvia, Gorad Krakau, Kraka, Krakau, Krakiv, Krakko, K...	Poland , Lesser Poland Voivodeship Kraków	second-order administrative division population 243,315	N 50° 4' 21"	E 19° 55' 57"

Rysunek 2.6: Zrzut ekranu aplikacji GeoNames dla wyszukiwania instancji związanych z Krakowem. <http://www.geonames.org/search.html?q=Krakow>

2.4.2. DBPedia

O DBPedi wspomniano podczas omawiania języka zapytań SPARQL. Projekt ten, rozpoczęty przez pracowników Wolnego Uniwersytetu Berlina oraz Uniwersytetu w Lipsku, ma na celu powiązanie ze sobą danych z Wikipedii, usystematyzowanie **jej** i udostępnienie. Poza artykułami z Wikipedii zawiera **usystematyzowane** informacje pochodzące z innych źródeł **danych takich** jak CIA World Fact Book, Project Gutenberg i Eurostat. Korzystając z otwartego interfejsu przyjmującego zapytania SPAQRL, można uzyskać dostęp do wiedzy z tych źródeł, wliczając w to czysty tekst, tabele, zdjęcia i współrzędne geograficzne. Interfejs strony pozwala na zadawanie pytań logicznych (**patrz przykład** „Czy Polska ma większe pole powierzchni niż Hiszpania?”). Nawijając do opisu ontologii na witrynie DBPedi, model wiedzy zbudowany jest z 4,2 mln instancji reprezentujących 685 klas o 2795 własnościach.

dbo:anthem	▪ dbr:Poland_Is_Not_Yet_Lost
dbo:areaTotal	▪ 312602266860.239075 (xsd:double) ▪ 312679000000.000000 (xsd:double)
dbo:currency	▪ dbr:Polish_złoty
dbo:demonym	▪ PolePolish
dbo:ethnicGroup	▪ dbr:Poles ▪ dbr:Lemkos ▪ dbr:Kashubians ▪ dbr:Romani_people ▪ dbr:Ukrainians_in_Poland ▪ dbr:German_minority_in_Poland ▪ dbr:Silesians ▪ dbr:Belarusian_minority_in_Poland
dbo:ethnicGroupsInYear	▪ 2011-01-01 (xsd:date)
dbo:flag	▪ Flag of Poland.svg
dbo:foundingDate	▪ 0966-04-14 (xsd:date) ▪ 1939-09-01 (xsd:date)
dbo:governmentType	▪ dbr:Parliamentary_republic
dbo:language	▪ dbr:Polish_language
dbo:leader	▪ dbr:Ewa_Kopacz ▪ dbr:Bronisław_Komorowski
dbo:leaderTitle	▪ President ▪ Prime Minister
dbo:longName	▪ Republic of Poland
dbo:percentageOfAreaWater	▪ 3.070000 (xsd:float)

Rysunek 2.7: Zrzut ekranu DBPedi dla węzła dotyczącego Polski: <http://dbpedia.org/page/Poland>

3. Wykorzystane technologie

3.1. Standardy aplikacji webowych

Większość współczesnych aplikacji internetowych bazuje na szeregu popularnych standardów architektury aplikacji. **Poniżej znajduje się omówienie** najczęściej pojawiających się technologii i wzorców.

3.1.1. JavaScript

Dużo akcji wykonywanych w dynamicznych aplikacjach internetowych musi być zainicjowana po stronie przeglądarki klienta. Od 1995 roku jednym z dominujących skryptowych języków **programowania jest** wieloparadygmataowy JavaScript będący implementacją specyfikacji ECMAScript. Skrypty tego języka są używane do dostarczenia interaktywności witryn internetowych w taki **sposób jak** reakcja na kliknięcia, **walidacja formularzy** i ogólna modyfikacja drzewa DOM. Składnia języka bardzo przypomina język C. Poniżej jest przykładowa funkcja licząca silnię liczby naturalnej.

```
function factorial(n)
{
    if (!n) {
        return 1;
    }
    return n * factorial(n - 1);
}
```

3.1.2. REST

REST (ang. Representational State Transfer - zmiana stanu poprzez reprezentacje) jest to architektura oprogramowania nakierowana na tworzenie skalowalnych serwisów internetowych. Roy T. Fielding, jeden z autorów protokołu HTTP definiuje REST jako „*skoordynowany zestaw stałych architektonicznych, które próbują zminimalizować opóźnienia i komunikację siecią przy równoczesnej maksymalizacji niezależności i skalowalności implementacji komponentów.*” [28]

Jadną z podstaw uniknięcia opóźnień jest operowanie **jak na najniższych warstwach** modelu OSI (ang. Open Systems Interconnection). ~~W związku z tym powszechnie przyjęło się, aby skorzystać z już istniejących rozwiązań (choć nie wszystkie implementacje REST z tego korzystają), a mianowicie ze standardu Hypertext Transfer Protocol (HTTP).~~

REST wyróżnia 3 podstawowe klasy elementów:

Elementy danych - Kluczowym aspektem w technologii REST, jak wskazuje jego nazwa, jest stan danych. Komponenty technologii wymieniają informację, którą jest właśnie aktualny stan elementów danych. Zakłada się, że dane mogą zmieniać się w czasie.

Te dane są reprezentowane jako **zasoby** (ang. resource), przy czym dowolna informacja może być zasobem - plik, lista aktywnych użytkowników, a nawet odpowiedź pusta. Użytkownik korzystający z systemu REST uzyskuje dostęp do zasobów przez identyfikatory zasobów (ang. resource identifiers).

Mimo wstępnie enigmatycznej nazwy, jest to najczęściej, opisany w poprzednich rozdziałach URI. Elementy danych są opatrzone metadanymi opisującymi zawartość (np. PDF, JPG) i niekiedy także wpisami takimi jak: ostatnia modyfikacji i suma kontrola.

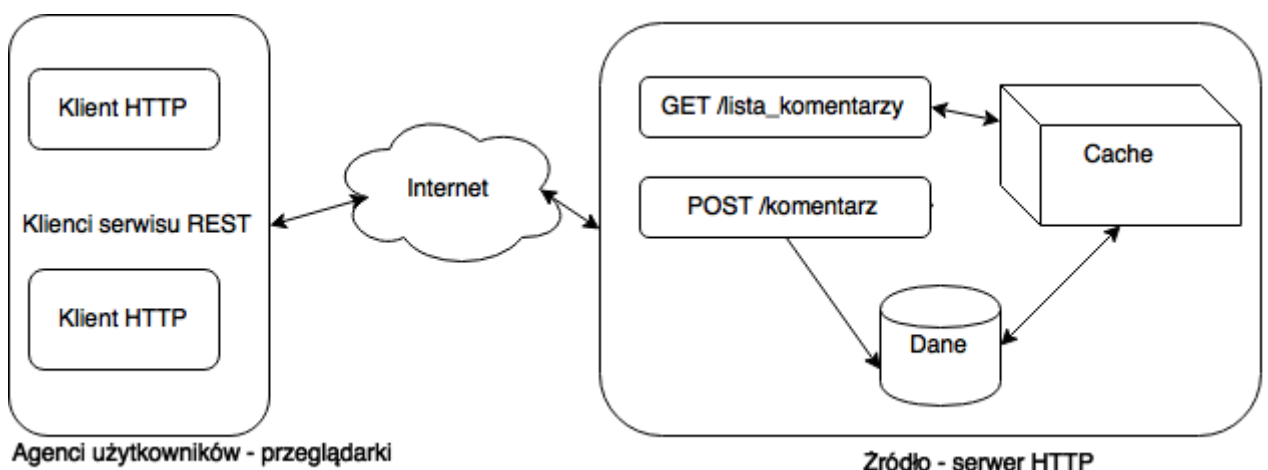
Aby uzyskać dostęp do zasobów korzysta się ze standardowych metod HTTP: GET, POST, PUT i DELETE. GET zwyczajowo używa się do żądania o dane, POST do wprowadzania nowych lub żądania przy użyciu bardziej rozbudowanych warunków precyzujących zapytanie, PUT do aktualizacji i DELETE do ich usuwania.

Elementy łączące, czyli **konektory**, zarządzają komunikacją sieciową w imieniu komponentów. W ogólności jest to abstrakcyjny interfejs komunikacyjny upraszczający implementację aplikacji. Zewnętrzne elementy wysyłające żądania pomiędzy sobą nie muszą znać wewnętrznej struktury aplikacji i katalogów, aby uzyskać dostęp do określonej informacji. Konektory zapewniają funkcjonalności takie jak cache i przetwarzanie równoległe, które jest łatwo dostępne dzięki bezstanowości REST. Wyróżnia się następujące elementy łączące:

- klient opowiedzialny za wysyłanie zapytań i otrzymywanie odpowiedzi
- serwer - odpowiedzialny za nasłuch i wysyłanie odpowiedzi
- cache - może być umieszczone po stronie serwera lub klienta w celu przyspieszenia przetwarzania
- enkoder - zamienia identyfikatory zasobów na adresy sieciowe.

Elementy przetwarzające - komponenty. Służą do identyfikacji ról w aplikacjach. Jeden komponent może implementować kilka konektorów.

- Agent użytkownika (ang. user agent) - inicjuje zapytanie i jest odbiorcą odpowiedzi. Typowym agentem jest przeglądarka internetowa.
- Server źródłowy (ang. origin server) - dzięki konektorom takim jak serwer, enkoder i cache odbiera żądania i odpowiada na nie. Typowym serwerem jest serwer HTTP (np. Apache) [27]



Rysunek 3.1: Architektura REST

3.1.3. MVC

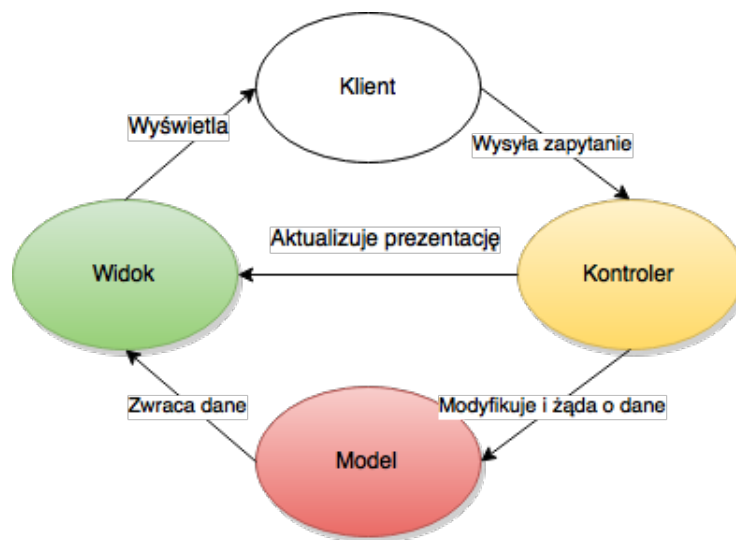
MVC (ang. Model-View-Controller - Model-Widok-Kontroler) jest to architektoniczny wzorec oprogramowania organizujący strukturę plików i wymianę informacji w aplikacjach posiadających graficzne interfejsy użytkownika.

Dzieli on daną aplikację na trzy części, rozdzielając logikę biznesową aplikacji od wizualizacji danych prezentowanych u użytkownika.

Składa się on z następujących części:

- **model** - **obsługujący** zachowanie aplikacji na poziomie logiki biznesowej - **analiza danych**, połączenie z bazą danych, bezpieczeństwo informacji. Służy do aktualizacji i pobierania danych ze źródeł.
- **widok** - **wyświetlenie** już przetworzonej informacji użytkownikowi np. w postaci tabeli lub wykresów. Jedna informacja pobrana z modelu może być wyświetlona na kilka sposobów.
- **kontroler** - zbiera żądania od użytkownika i przekazuje do modelu, generuje widoki na podstawie danych uzyskanych z modelu.

Model ten pozwala na prostą izolację danych od **widoku** co zapewnia mniejszą podatność na błędy i elastyczność w przypadku rozbudowy programu. Modyfikacja logiki biznesowej nie wymusza zmiany plików UI **co ułatwia** jasny podział pracy w zespołach. Model MVC w intuicyjny sposób można połączyć z REST, co spowodowało **na łączenie** obu standardów w projektach takich jak Node.js. [29]



Rysunek 3.2: Schemat modelu MVC

3.1.4. AJAX

AJAX (ang. Asynchronous JavaScript and XML, asynchroniczny JavaScript i XML) – technika tworzenia aplikacji webowych, której cechą jest to, że komunikacja klienta z serverem odbywa się w sposób asynchroniczny. Efektem tego jest praca w przeglądarce bez konieczności przeładowania całego dokumentu, a jedynie jego zaktualizowanej części. Wpływa to na dynamikę aplikacji (większą płynność obsługi) przy znacznym obciążeniu serwera ze względu na rzadsze odwołania do baz danych – **pobierane** są tylko te dane, które mogły ulec zmianie po ostatniej akcji użytkownika. Dla użytkownika jedyny odczuwalny skutek uboczny to oczekiwanie na przeładowanie części witryny, **ale w tym czasie strona** nie jest zablokowana i można operować na elementach już załadowanych.

W skład technologii AJAX wchodzi 5 podstawowych elementów (warto przy tym zwrócić uwagę, że wszystkie te elementy mogą być różne w zależności od potrzeb, więc AJAX jest głównie ogólną taktyką asynchronicznej wymiany danych w aplikacjach internetowych): [32]

3.1.5. DOM

DOM (ang. Document Object Model, Obiektowy model dokumentu) - jest to sposób reprezentacji rozbudowanych dokumentów (X)HTML i XML w postaci modelu obiektowego niezależnego od języka programowania i platformy. [33] Zakłada się, że **zagnieżdżona struktura** znaczników w takich **dokumentach można** przedstawić jako standardowe drzewo. Definiuje on zbiór interfejsów i klas umożliwiających,

poprzez parsowanie, bezpośredni dostęp do elementów struktury dokumentu. Aplikacje operujące na tym modelu mogą dowolnie **tworzyć modyfikować** (np. zmieniać wartości atrybutów) i usuwać węzły tego drzewa. . Możliwość bezpośredniej modyfikacji drzewa DOM pozwala na asynchroniczną aktualizację informacji, pomiędzy przeglądarką, a serwerem.

XMLHttpRequest

XMLHttpRequest – klasa zezwalająca na asynchroniczne przesyłanie **danych zaprojektowana** przez korporację Microsoft [31]. To ona powoduje, że oczekiwanie na porcję danych nie blokuje strony. Klasa ta nie jest standardem. W trakcie pisania niniejszej pracy nie jest to jeszcze rekomendacja W3C, a szkic (ang. *working draft*) więc implementacja może zależeć od danej przeglądarki.[30]

W celu uniknięcia problemów z **kompatybilnością zaleca** się korzystanie z bibliotek takich jak jQuery. Obiekty tej klasy mają szereg atrybutów:

- **readyState** - Zawiera jeden ze stanów obiektu: 0 - zapytania niezainicjowane; 1 - cel otwarty; 2 - żądanie wysłane; 3 - odbieranie odpowiedzi; 4 - żądanie zakończone
- **onreadystatechange** - Określa wywołanie zwrotne (ang. *callback*) do funkcji wykonywanej przy zmianie **własności readyState**.
- **responseText** - Jest to ciąg znaków będący odpowiedzią
- **status** - jest to odpowiedź protokołu HTTP w postaci kodu (np. 503 dla „Forbidden”, 404 dla „Not Found”, 200 dla „OK”).

W oparciu o obiekty tej klasy powstają duże aplikacje **internetowe takie** jak Google Gmail.

JavaScript

JavaScript – pierwotnym językiem użytym w AJAX był właśnie język JavaScript, ale nic nie stoi na przeszkodzie, aby użyć innego języka (np. VBScript). Jedynym wymaganiem **jest aby** język ten pracował po stronie klienta, w jego przeglądarce. Powszechność języka JavaScript i jego wsparcie przez znane przeglądarki powoduje, że jest językiem dominującym **w kliencie części** technologii AJAX.

CSS

CSS - Ze względu na wielokrotne użycie stylów w aplikacji dostarczonej do tej pracy, konieczne jest opisanie CSS. CSS (ang. Cascading Style Sheets - Kaskadowe arkusze stylów) jest to język służący do opisu formy prezentacji stron WWW. CSS został opracowany przez W3C w 1996 r. Arkusz CSS to zbiór reguł ustalających sposób wyświetlenia przez przeglądarkę internetową **zawartość** wybranego elementu (X)HTML lub XML. Jednym z założeń CSS było oddzielenie struktury dokumentu (faktycznych danych) od jego wyglądu. Zmniejsza to rozmiar dokumentu (przez uogólnione reguły stylizacji) i poziom skomplikowania dokumentu. Kilka dokumentów może mieć jeden styl CSS. Jest to kolejna technologia, którą łatwo wykorzystać w aplikacjach opartych na wzorcu MVC.

Style te wprowadzają pojęcie takie jak klasa czyli zbiór znaczników, które łączy jakaś wspólna cecha.

Ogólna składnia CSS wygląda następująco:

```
selektor {  
    atrybut : wartość ;  
    atrybut2 : wartość2 ;  
}
```

gdzie selektor określa, który obiekt ma mieć zmodyfikowany styl. Atrybut wskazuje na to jaka część stylistyki dokumentu ma być zmieniona (kolor, margines, zaokrąglenia, krawędzie itd.).

Przykładowo arkusz

```
div {  
    color : blue ;
```

```

        border-radius: 25px;
    }
    .clientVip
    {
        background-color: red;
        text-transform: uppercase;
    }

```

spowoduje ustawienie koloru we wszystkich blokach `<div></div>` na niebieski i zaokrąglenie krawędzi okręgiem o promieniu 25 pikseli oraz ustawienie wielkości wszystkich liter w elementach klasy `clientVip` ma wielkie i umieszczenie ich na tle czerwonym.

Oczywiście aplikacje Web 3.0 nie muszą mieć żadnej stylizacji, ale wygoda korzystania z aplikacji jest jednym z założeń aplikacji działających w Semantic Web. Czytelny UI ~~jak~~ w sposób bezpośredni przyspiesza korzystanie z aplikacji, a tym samym wyszukiwanie danych.

XML w AJAX

Pierwotne założenie polegało na tym, że obiekty klasy XMLHttpRequest zwracały drzewa XML, które należało wstawić do strony. W rzeczywistości jednak zamiast dokumentów XML można zwracać także czysty tekst, dokumenty formatu JSON, a nawet skrypty gotowe do wykonania. Aby jednak uniknąć nieporozumień, zarówno po stronie kodu serwera, jak i klienta należy ustawić nagłówki HTTP Accept Header i Content-Type adekwatne do przesyłanych danych.

3.1.6. JSON

XML jest formatem czytelnym zarówno dla człowieka, jak i komputera. Niestety mnogość danych przesyłanych przez sieć wymusza wprowadzenie pewnych uproszczeń. Jak można zauważyć na przykładzie

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<uczelnie>
<uczelnia nazwa="Wyższa_Szkoła" rok="1925">
    <wydział nazwa="Wydział_Przykładowy"/>
</uczelnia>
<uczelnia nazwa="Uczelnia">
    <wydział nazwa="Wydział_Pierwszy" rok="1922">
        <katedry>
        <katedra nazwa="Katedra_Fizyki_Współczesnej" rok="1944"/>
        <katedra nazwa="Katedra_Informatyki" rok="1998"/>
        </katedry>
    </wydział>
    <wydział nazwa="Wydział_Inny" rok="1879"></wydział>
</uczelnia>
<uczelnie>

```

Sam opis prostego dokumentu tj. znaczniki może zużywać istotną objętość pamięci. Należy zauważyć, że od aplikacji Semantic Web oczekujemy wyników w czasie rzeczywistym, więc należy wprowadzać optymalizacje gdziekolwiek jest to możliwe. Technologia AJAX zezwala na przysyłanie dowolnego typu danych i warto z tego skorzystać. Dokument RFC 4627 [34] wprowadził pewien lekki format wymiany danych JSON (ang. JavaScript Object Notation - Notacja Obiektu JavaScript). Został on zaprojektowany w oparciu o opis obiektów w języku JavaScript, ale jest zupełnie niezależny od technologii i powszechne języki takie jak PHP, Python, C++ i Java wspierają go.

Dokument JSON jest literałem obiektu języka JavaScript, co jest strukturą danych znaną jako tablica asocjacyjna otoczona nawiasami klamrowymi. Dokumenty JSON nie mogą zawierać kodu wykonywalnego (wszystko jest danymi kodowanymi w Unicode w systemie UTF-8). Nazwy właściwości w standar-

dzie w tej tablicy są otoczone cudzysłowami, lecz niektóre języki tego nie wymuszają. Dla zapewnienia zgodności lepiej jest stosować cudzysłowy przy definicji kluczy w tablicy.

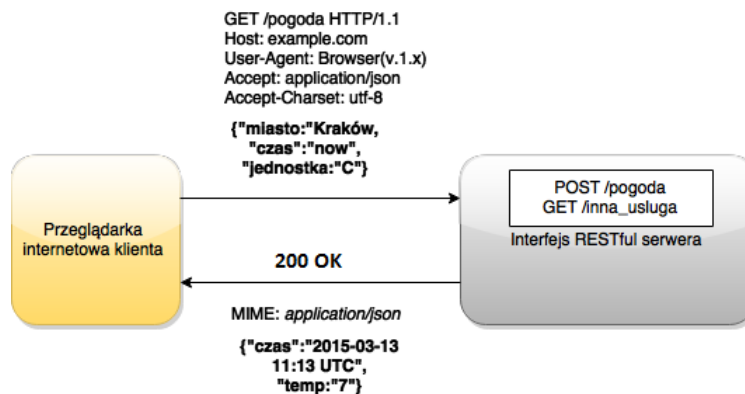
Standard dopuszcza następujące typy danych:

- Łańcuch tekstowy
- Liczba zmiennoprzecinkowa
- Wartość logiczna - true lub false
- Wartość pusta - null
- Tablica (lista) powyższych otoczona nawiasami kwadratowymi ["Test", "Wartość"]
- Zagnieżdżony obiekt typu JSON

Korzystając z powyższych cech można przekonwertować z XML na JSON dokument przedstawiony na początku podsekcji:

```
{ "uczelnie" : [  
  {  
    "nazwa" : "Wyższa_Szkoła",  
    "rok" : 1925,  
    "wydział" : { "nazwa": "Wydział_Przykładowy" }  
  }, {  
    "nazwa" : "Uczelnia",  
    "rok" : 1922,  
    "wydział" : { "nazwa": "Wydział_Pierwszy", "rok": 1975,  
      "katedry": [  
        { "nazwa": "Katedra Fizyki Współczesnej", "rok": 1944 },  
        { "nazwa": "Katedra Informatyki", "rok": 1998 } ]  
      }  
    }  
  ]  
}
```

Porównując obiekty XML i JSON różnica w objętości nie przekroczy 20-30% procent, gdyż polega ona głównie na minimalizacji liczby znaczników, jednakże biorąc pod uwagę ilość klientów obsługiwanych dziennie przez serwery, różnica ta może sięgać gigabajtów transferu dziennie. Przesiębiorstwa operujące na rynku komercyjnym udostępniające infrastrukturę serwerową pobierają opłaty proporcjonalnie do wykorzystanego transferu. Jest to jeden z czynników, który spowodował rozpowszechnienie standardu JSON.



Rysunek 3.3: Przykład użycia JSON w REST. Pod odpowiedni URI zostaje wysłane żądanie metodą POST z żądaniem o uzyskaniu temperatury w Krakowie w stopniach Celsjusza. Serwer dopasowuje żądanie do dostępnych usług i w odpowiedzi zwraca dokument JSON zawierający oczekiwane informacje. Taką funkcjonalność można zaimplementować w ramach technologii AJAX, co spowoduje, że aplikacja (np. widget na urządzeniu mobilnym) będzie miała systematycznie odświeżaną temperaturę adekwatną do aktualnej lokalizacji.

Autor pracy chciałby w tym momencie zwrócić istotną uwagę, iż dokumenty formatu JSON są domyślną odpowiedzią zwracaną przez bazę danych Neo4j co jest dodatkowym atutem przemawiających za zastosowaniem tego typu w pracy.

3.2. Użyte aplikacje i biblioteki

Aplikacja została zaprojektowana w taki sposób, który umożliwia jej przyszły rozwój i wymaga jak najmniejszego wykładu pracy przez użytkowników. Wybrano technologie, które są aktualnie powszechnie używane w środowisku programistów. [39][38] Jako podstawowy język programowania wybrano język Java 7.

Jednym z celów było osiągnięcie maksymalnej prostoty wdrożenia programu na nowym komputerze.

3.2.1. Spring Framework

Spring Framework [40] jest darmowym szkieletem umożliwiającym prostsze tworzenie aplikacji opartych o platformę Java Enterprise Edition. Technologia Spring dostarcza, między innymi, rozwiązania typowych problemów pojawiających się podczas tworzenia aplikacji webowych. Przykładowo dostarcza komponenty pozwalające na tworzenie programów zgodnie z architekturą MVC, programowanie aspektowe, szablon obsługi transakcyjności oraz mechanizmy dostępu do baz danych i zabezpieczania aplikacji przed typowymi atakami. Szczególnie cenne podczas pisania tej pracy okazało się wsparcie Spring Framework dla rozwiązań NoSQL.

Pozostaje w zgodności ze standardowymi technologiami platformy JavaEE takimi jak JDBC, JPA, JMS, CORBA. Jednym z założeń Spring Framework jest zmniejszenie czasu, który programista musi poświęcić na obsługę skomplikowanych standardów aby mógł się skupić na logice biznesowej.

Spring Boot

Spring Boot [41] jest to jeden z pakietów dostarczonych w ramach platformy Spring. Jego podstawowym celem jest ułatwienie procesu wdrażania aplikacji na serwerach aplikacyjnych takich jak Tomcat i Jetty. Same serwery również są dostarczone w ramach tego pakietu. Takie rozwiązanie pozwala na automatyczną konfigurację ustawień platformy uruchomieniowej Spring, co zmniejsza szanse na trudne do znalezienia błędy.

Spring Tool Suite

Spring Tool Suite[42] jest to zintegrowane środowisko programistyczne (IDE ang. *integrated development environment*) oparte na pakiecie Eclipse. Nie jest ono niezbędne do uruchomienia aplikacji dostarczonej do tej pracy, jednakże znacząco ułatwiło jej tworzenie. Spring Tool Suite jest zalecany podczas programowania aplikacji wykorzystujących pakiet Spring Framework ponieważ:

- automatycznie parsuje pliki konfiguracyjne oraz standardowe adnotacje Spring Framework
- jest zintegrowany ~~się~~ z systemem kontroli wersji Git, pakietem Apache Maven i językiem programowania aspektowego AspectJ
- dokonuje szybkiej walidacji i podpowiadania kodu Spring Framework

3.2.2. Apache Maven

Apache Maven [43] jest oprogramowaniem automatyzującym budowanie aplikacji opartych na języku Java. **Działanie programu polega na tym, że użytkownik wybiera cel, który ma zostać wykonany.** Warto wspomnieć, że w przeciwieństwie do innych narzędzi takich jak Ant, nie trzeba ograniczać się do predefiniowanych celów i można zdefiniować własne wtyczki (ang. *plugin*) zwiększające możliwości programu Maven.

Autorzy projektu zdefiniowali standardowe cele:

- validate - sprawdzenie, czy kod źródłowy programu jest poprawny składniowo i czy wszystkie potrzebne biblioteki są dostarczone
- compile - wykonanie kompilacji kodu źródłowego projektu
- test-compile - kompiluje kod źródłowy testów
- test - przeprowadza testy jednostkowe
- package - Maven buduje paczkę dystrybucyjną
- install - paczka projektu zostaje umieszczona w repozytorium lokalnym. Dzięki temu można ją wykorzystać w przyszłych projektach jako zależność
- deploy - paczka projektu zostaje umieszczona w repozytorium zdalnym, co oznacza wdrożenie projektu w zdefiniowane miejsce

W momencie, gdy osiągnięcie danego celu wymaga wykonania wcześniejszych kroków, zostaną one automatycznie przeprowadzone.

Zarządzanie zależnościami i plik pom.xml

Bardzo ważną funkcjonalnością programu Apache Maven jest zarządzanie zależnościami (ang. *dependencies*). Sprowadza się ono do automatycznego zarządzania bibliotekami używanymi w projekcie. W chwili, gdy użytkownik umieszcza w specjalnym pliku pom.xml chęć wykorzystania jakiejś biblioteki (np. JUnit, Hibernate, Spring Framework bądź dowolnej innej), poprzez podanie jej nazwy i wersji, biblioteka ta zostanie automatycznie pobrana do lokalnego repozytorium i dołączona do projektu. Automatyczne zarządzanie zależnościami pozwala **na przykład** na szybką aktualizację bibliotek do nowszej wersji bez konieczności ręcznego pobierania odpowiednich archiwów JAR. Kod źródłowy projektu może być dystrybuowany bez, często obszernych, bibliotek - wystarczy że na komputerze docelowym będzie zainstalowany program Maven.

3.2.3. JDBC - połączenie z bazą danych

JDBC (Java DataBase Connectivity) jest uniwersalnym interfejsem programowania (API) należącym do Oracle Corporation. Umożliwia on, niezależnie od platformy aplikacjom napisanym w języku Java, porozumiewać się z bazami danych za pomocą języków zapytań tychże baz. Warto przy tym wspomnieć, że został dołączony do pakietu Java 1.1 JDK, a więc był przeznaczony zwłaszcza dla języka SQL. Wówczas, w 1997 roku SQL był jedynym powszechnie stosowanym językiem zapytań. Inne języki zapytań powstały później np. XQuery (1998), XPath (1999), SPARQL(2007). Jednakże, jeżeli producent bazy danych dostarczy odpowiednie sterowniki, JDBC swobodnie może zostać zastosowany dla baz nierelacyjnych takich jak Neo4j bądź MongoDB.

Wszystkie zapytania wykonane przy użyciu technologii JDBC mają 3 etapy [44]:

- Nawiązanie połączenia z bazą danych lub dostęp do innego źródła danych (np.: plik csv).
- Wysłanie zapytania w wybranym języku
- Przetworzenie wyników do postaci tabelarycznej

Na standard JDBC zdecydowano się ze względu na jego wieloletnią, ugruntowaną obecność na rynku. Ten interfejs pozwala na wykonywanie zapytań bezpośrednio w języku Cypher, co przełożyło się na istotne uproszczenie kodu źródłowego aplikacji.

3.2.4. Technologie wykorzystane przy tworzeniu interfejsu graficznego.

Twitter Bootstrap

Aplikacje są coraz częściej uruchamiane nie tylko na komputerach, ale i urządzeniach mobilnych takich jak tablety i smartfony. Aby sprostać potrzeby użytkowników takich urządzeń konieczne jest pisanie aplikacji, która poprawnie wygląda i działa na wszelkich urządzeniach. Warto skorzystać z gotowych rozwiązań wprowadzających większą przenośność aplikacji. Jedną z nich jest Twitter Bootstrap dostarczający przejrzyste arkusze stylów i funkcje jQuery pozwalające nimi manipulować.

Twitter Bootstrap [25] jest to framework CSS wydany przez twórców popularnej witryny Web 2.0 - Twittera. Jest to zbiór narzędzi opierających na HTML, JavaScript i CSS 3.0 przyspieszających tworzenie interfejsu graficznego dla aplikacji internetowych. Jest wykorzystywany do prostego umieszczania typowych elementów stron internetowych - formularzy, przycisków, zakładek itp. Jego style zostały opracowane w taki sposób, żeby aplikacje pracowały prawidłowo na wielu przeglądarkach (Internet Explorer, Safari, Chrome itd.).

Tym samym aplikacja, która została napisana w ramach tej pracy jest w pełni mobilna i powinna prawidłowo pracować na większości znanych przeglądarkach.

jQuery

Aby aplikacja internetowa działająca w oparciu o przeglądarkę nie sprawiała wrażenia powolnej, konieczne jest ograniczenie do minimum przesyłanych danych. Aktualny standard Web 2.0 przyzwyczaił użytkowników do tego, że witryny internetowe pracują płynnie i nie wymagają całkowitego przeładowania. Oczekuje się wprowadzenia takich funkcjonalności jak animacje i pełna interaktywność przy równoczesnym ograniczeniu do minimum czekania na dane.

Jednym z rozwiązań pozwalających na pisanie nowoczesnych aplikacji internetowych jest, już powszechnie przetestowany i zaakceptowany, framework jQuery [26]. Jest to lekka (jej rozmiar nie przekracza kilkudziesięciu KB) biblioteka jQuery. Efektem korzystania z niej przez programistów jest istotnie zmniejszenie ilości kodu potrzebnego do otrzymania oczekiwanej funkcjonalności, w porównaniu z „czystymi” skryptami JavaScript, pozbawionych bibliotek.

Wprowadza ona dodatkowe funkcjonalności takie jak:

- Uproszczenie wykonania skomplikowanych, a typowych operacji. Przykładem są zapytania asynchroniczne napisane w technologii AJAX. Wprowadzenie ich korzystając z języka JavaScript wymaga na programistach pełną kontrolę nad obiektami XMLHttpRequest - kontrolę ich stanów

gotowości i statustów HTTP. Zazwyczaj nie ma potrzeby niskopoziomowego zarządzania zapytaniami HTTP (łatwo popełnić błąd) i dlatego warto zainteresować się rozwiązaniami zaimplementowanymi w jQuery.

- Pełna kompatybilność z przeglądarkami - zarówno współczesnymi, jak i leciwymi (np. Internet Explorer 7). Jak już zostało wspomniane we wcześniejszych akapitach, Web 3.0 oczekuje od aplikacji jak największej mobilności. Projekt ten nie może zostać wcielony w życie, jeżeli nie będzie zgodny wstecz z Web 1.0/Web 2.0. jQuery wprowadza interfejsy zezwalające na zarządzanie zdarzeniami i operacje drzewem DOM (metody `.attr()`, `.on()`) na dowolnych przeglądarkach. Odciąża to programistę z konieczności pisania kilku wersji aplikacji w zależności od platformy użytkownika.
- Animacje - ich głównym celem jest, jak łatwo zgadnąć, poprawienie estetyki aplikacji. jQuery korzysta z wielowątkowości, co może służyć do lekkiego ukrycia ubocznych efektów ładowania przed użytkownikiem.

Podstawowym minusem jQuery jest przede wszystkim zwiększenie narzutu czasowego na wykonanie skryptów JavaScript. Wsparcie równocześnie dla wszystkich przeglądarek (wysoka generyczność) i używanie pętli powoduje, że skrypty działają wolniej niż dedykowane dla danej przeglądarki.

Poniżej znajduje się jeden z fragmentów kodu jQuery zastosowanych do operacji asynchronicznych:

```
$.post( '/availableRelationshipModelsForSingleClass', {
    classCode1 : $('#selectClass'+code).val()
}, function(data)
{
    $('#selectClass'+c2).find('option').remove();
    $('#selectClass3').find('option').remove();
    $.each( data, function( key, val )
    {
        $('#selectClass'+c2).append("<option_data-attr='"+val.classCode2.attributes +
        "_value='" + val.classCode2.code + "'>" + val.classCode2.name + "</option>" );
    });
    $('#selectClass3').append( "<option_value='" + val.code + "'>" + val.name + "</option>" );
});
    updateAvailableRels();
});
```

Powyższy kod służy do wykonania asynchronicznego zapytania metodą POST z URN=*availableRelationshipModelsForSingleClass* i dynamiczne wygenerowanie formularza zezwalającego na dodanie połączenia ze sobą dwóch klas. W nagłówku jest przesyłany obiekt w formacie JSON zawierający kod identyfikujący. Instrukcje takie jak `remove()` (usunięcie obiektu z drzewa DOM) i `append()` (dodanie elementu) dają efekt dynamicznie zmieniającego się formularza w zależności od tego, która klasa jest aktualnie zaznaczona.

Dynatable

DataTables [36] jest wtyczką do biblioteki jQuery. Jej celem jest rozwinięcie wprowadzenie funkcjonalności dynamicznych tabel do jQuery. W aplikacji dostarczonej do pracy magisterskiej użytkownik może zdefiniować model z klasami o dowolnej liczbie atrybutów. Aby sprostać potrzebie wyświetlania tych dynamicznych danych konieczne było skorzystanie z biblioteki, która to umożliwiła. Dynatable wprowadza funkcjonalność budowania tabeli w oparciu o dane JSON a następnie jej przeszukiwanie.

4. Grafowa baza danych Neo4j

4.1. Bazy NoSQL

Tradycyjny model baz danych został zaproponowany w 1970 roku przez Edgara Codd'a w pracy *A Relational Model of Data for Large Shared Data Banks*. Implementacje modelu baz danych, który został tam opisany aż do ostatnich lat stanowiły ponad 90% rynku. Rozwój sieci Web 2.0 spowodował, że aplikacje bazodanowe przestały służyć głównie do bankowości i rachunkowości. Aktualnie szacuje się, że procent zajmowania rynku przez relacyjne bazy danych stanowi ok. 79%. Implementacje modelu relacyjnego nadal są w ścisłej czołówce, jednakże trend spadkowy skłania do prób zainteresowania się innymi modelami. Model relacyjny nadaje się do większości rozwiązań, lecz potrzeba pracy na coraz większej ilości, coraz bardziej połączonych ze sobą informacji sugeruje aby „przyjrzeć się” także innym rozwiązaniom w tych ekstremalnych przypadkach. Mnogość nowych potrzeb takich przechowywanie danych geolokacyjnych do map, serwisy społecznościowe, oraz szeroko składowanie Big Data wygenerowała zapotrzebowanie na nowe modele baz danych do tych rozwiązań.

4.2. Relacyjny model danych

Opowieść o modelu grafowym należy rozpocząć od krótkiego wprowadzenia na temat relacyjnych baz danych. Aby w pełni zrozumieć zalety wynikające z zastosowania grafowych baz danych, należy dostrzec ograniczenia modelu relacyjnego w pewnych szczególnych przypadkach.

Relacyjny model danych jest oparty na matematycznej teorii mnogości, oferującej zbiór operatorów do manipulacji danymi, co w znacznej mierze wpłynęło na jego popularność. Definiuje on bazę danych jako zbiór tabel (relacji). Każda tabela składa się z pewnej, z góry określonej kolumny, a także z wierszy do nich należących. Każdy wiersz, zwany rekordem (lub krotką) oznacza pojedynczy byt. Każda kolumna ma określoną nazwę i służy do opisu jednej cechy obiektu z danego wiersza. Tabela jest złożona z wierszy o tych samych atrybutach, określonych jeszcze przed uzupełnieniem danych. Dane nie mogą mieć więcej informacji (dodatkowych atrybutów) niż przewiduje to schemat tabeli.

PESEL	Imię	Nazwisko
1234567890	Anna	Cabacka
9103052351	Paweł	Babacki
9878542151	Jan	Zazacki

Rysunek 4.1: Przykładowa relacja

Model relacyjny dostarcza szereg operatorów na relacjach takie jak:

- Selekcja – Jako argument przyjmuje relację, na wyjściu zwraca relację zawierającą tylko te krotki, dla których warunek logiczny był prawdziwy. Przykładowym operatorem selekcji jest $\sigma_{imię='Anna'}(relacja)$
- Projekcja (rzutowanie) – Krotki relacji wynikowej powstają wskutek rzutowania wyjściowych krotek na podany ciąg. Przykładowym operatorem projekcji jest $\pi_{pesel,imię}(relacja)$
- Złączenie – W celu pobrania informacji z kilku tabel wykonuje się operacje złączenia. Jako argumenty przyjmuje dwie relacje $r(X)$ i $s(Y)$. W wyniku powstaje relacja $Z(X \cup Y)$. Dla każdej pary krotek z dwóch relacji, które mają identyczne wartości dla wspólnych atrybutów, powstaje krotka nowej relacji wskutek dołączenia do pierwszej krotki wartości drugiej krotki. Powoduje ona istotne zwiększenie złożoności wyszukiwania, ponieważ relacje w pamięci są przechowywane jako niezależne struktury. Złożoność operacji wynika z tego, że dopasowanie do siebie fragmentów wiedzy z różnych tabel bazuje na porównywaniu pewnych atrybutów o szczególnym znaczeniu. Zostaje wykonane łączenie rekordów, w których wartość atrybutu z pierwszego rekordu jest równa kluczowi głównemu drugiego, w jeden duży rekord. Model relacyjny zakłada, że wszystkie tabele powinny posiadać szczególną (często sztucznie utworzoną) kolumnę, znaną jako klucz główny, której wiersze są wartościami unikalnymi. [45][58]

Istnieje możliwość stworzenia dodatkowych struktur, takich jak indeksy, pozwalających na przyspieszenie operacji złączenia. Problem pojawia się, gdy wykorzystywane tabele liczą miliony wierszy, które są często aktualizowane. Są to przypadki ekstremalne, jednakże takie zdarzają się w serwisach internetowych utworzonych dla tysięcy użytkowników. W RBD wprowadza się optymalizacje zapytań (statyczne lub dynamiczne), jednakże nawet najlepiej dostosowane zapytanie może nie poradzić sobie z zapytaniami rekurencyjnymi, które są standardem w dużych sieciach.

4.3. Bazy NoSQL

Pojęcie NoSQL, zostało po raz pierwszy użyte w 2009 roku przez E. Evansa [46]. Rozwinięciem tego pojęcia jest „nie tylko SQL” (ang. Not Only SQL), które określa podstawowe cechy baz tego typu:

1. Zapis danych i ich odczyt z pominięciem języka SQL,
2. Model danych nie jest oparty na relacjach.

Grafowe bazy danych są częścią dużej grupy technologii oznaczonych jako NoSQL, mających być remedium na problemy baz relacyjnych. Bazy typu NoSQL w pierwszych latach swojego istnienia spowodowały spore zamieszanie na rynku. Mogło wydawać się wręcz, że będą mogły zastąpić model dominujący - zaczęły się nimi interesować największe przedsiębiorstwa związane z technologiami bazodanowymi (patrz Oracle NoSQL). Model relacyjny, mimo dość **wysokiemu poziomowi** nienaturalności, jest modelem powszechnym i idealnie nadającym się do większości typowych zastosowań, w których dane można naturalnie przedstawić w postaci tabeli. Dekady przyzwyczajęń u programistów i administratorów baz danych, a także doskonałe fundamenty matematyczne baz relacyjnych spowodowały spowolnienie ekspansji baz typu NoSQL.

Poniższa ilustracja wskazuje popularność baz NoSQL w porównaniu do baz relacyjnych. **Wynik Neo4j jest zaledwie ułamkiem rzędu procenta w porównaniu do przodujących baz tworzonych przez Oracle (Oracle Database i MySQL).** Metodologia serwisu DB-Engines jest oparta na następujących parametrach dobrze odpowiadających rzeczywistej popularności baz danych:

- Ogólne zainteresowanie (m.in. ilość publikacji i badań) w oparciu o wyniki analiz Google Trends,
- Ilość tematów (zapytań) na forach **tematycznych** takich jak Stack Overflow i DBA Stack Exchange,
- Ilość ofert pracy przy danej technologii,
- Znajomości technologii deklarowanej na profilach użytkowników LinkedIn.

Rank			DBMS	Database Model	Score		
Aug 2015	Jul 2015	Aug 2014			Aug 2015	Jul 2015	Aug 2014
1.	1.	1.	Oracle	Relational DBMS	1453.02	-3.70	-17.83
2.	2.	2.	MySQL	Relational DBMS	1292.03	+8.69	+10.81
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1108.66	+5.60	-133.84
4.	4.	↑ 5.	MongoDB +	Document store	294.65	+7.26	+57.30
5.	5.	↓ 4.	PostgreSQL	Relational DBMS	281.86	+9.04	+32.01
6.	6.	6.	DB2	Relational DBMS	201.23	+3.12	-5.19
7.	7.	7.	Microsoft Access	Relational DBMS	144.20	-0.10	+4.58
8.	8.	↑ 10.	Cassandra +	Wide column store	113.99	+1.28	+32.09
9.	9.	↓ 8.	SQLite	Relational DBMS	105.82	-0.05	+16.95
10.	10.	↑ 11.	Redis +	Key-value store	98.81	+3.73	+28.01
20.	20.	↓ 17.	Informix	Relational DBMS	36.80	+0.91	+3.64
21.	21.	↓ 18.	Memcached	Key-value store	33.38	+0.25	+2.39
22.	22.	22.	Neo4j +	Graph DBMS	33.16	+1.81	+10.25
23.	23.	↓ 21.	CouchDB	Document store	27.28	+0.35	+3.15

Rysunek 4.2: Popularność baz NoSQL wg. serwisu DB-Engines[52]. Pierwsza baza grafowa znajduje się dopiero na 22. miejscu.

Do baz NoSQL zaczęły powstawać specjalne moduły (na przykład moduł Phoenix do bazy HBase), które umożliwiają użycie języka SQL zamiast interfejsów standardowych. Świadczy to o tym, że bazy relacyjne **bez** wiele lat będą w czołówce rynku światowego. Bazy NoSQL **po mniej** więcej 6 latach swojego istnienia **istnieją raczej** jako rozwiązania alternatywne dla pewnych charakterystycznych zastosowań. Celem pracy jest **analiza czy** grafowe bazy danych mają szansę sprawdzić się w budowie Semantic Web, czy obecnie stosowany model oparty na trójkach składowanych w bazach relacyjnych (lub dedykowanych rozwiązaniach) jest rozwiązaniem lepszym.[47]

4.3.1. Cechy NoSQL

Martin Fowler wymienia cechy określające główny nurt baz NoSQL. Listy tej nie należy traktować jako formalną definicję.[48]

Bazy typu NoSQL:

- Zaprojektowane są do pracy w dużych, rozproszonych klastrach. Bazy tego nurtu przewidują możliwość przechowywania i przeszukiwania Big Data. Big Data jest zbiór danych, o rozmiarze rzędu nawet setek terabajtów, powiększający się tak szybko, że wymaga specjalistycznych narzędzi do zarządzania nimi. Baza Neo4j jest w stanie przechować 2^{35} (~34 miliardy) relacji oraz taką samą ilość węzłów na pojedynczym serwerze. [50] Wszystkie rozwiązania cechuje akceptacja redundancji danych, którą, w modelu relacyjnym należałoby usunąć w trakcie procesu normalizacji.
- Mają otwarty kod źródłowy - nie jest to wymóg, ale cecha charakterystyczna baz NoSQL. Stąd bazy obiektowe nie są często traktowane jako bazy NoSQL. Baza Neo4j ma otwarty kod źródłowy pozostając aplikacją komercyjną, [47]
- Są skalowalne i wydajne. W ogólności wspierają ACID. ACID jest skrótem od słów: atomi-city (atomowość), consistency (spójność), isolation (izolacja), durability (trwałość),
- Nie używają modelu relacyjnego do przechowywania danych. Brak w nich narzuconych schematów wymuszających porządek w bazie danych. Autorzy baz NoSQL nie muszą ściśle przestrzegać zaleceń standardów SQL,
- Bazują na potrzebach generowanych przez użytkowników Internetu,
- Fragmentacje w klastrach są przeźroczyste – system zachowuje swoją funkcjonalność przy rozproszeniu danych w sposób transparentny dla użytkownika [50]
- Nie używają języka SQL jako drogi komunikacji z bazą danych. Pierwotnie miały nie stosować go w ogóle, jednakże znajomość, ustandaryzowanie i funkcjonalność języka SQL spowodowała, jak wcześniej wspomniano, powstawanie wtyczek zezwalających na używanie go. Bazy NoSQL mają języki zapytań, których składnia (w szczególności słowa kluczowe) może przypominać SQL. Nazwa NoSQL sugeruje, iż dane będą będą przetwarzane w sposób zupełnie odmienny, niż poprzez język SQL, jednakże ze względu na swoje zakorzenienie w społeczności IT jest on wzorcem dla gramatyk nowych języków. Języki zapytań w bazach NoSQL nie są ustandaryzowane, w przeciwieństwie do SQL nadzorowanego przez komitety ANSI/ISO. W bazie Neo4j stosuje się język zapytań Cypher:

```
MATCH comment
WHERE has (comment.rating)
           AND has (comment.creation_time)
           AND comment.rating IN range(1,5)
SET comment:RatedComment
RETURN comment;
```

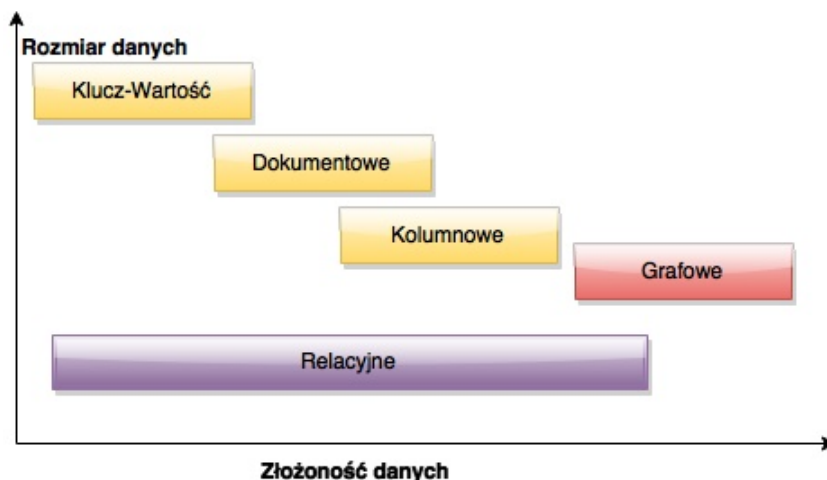
a w bazie Cassandra język CQL (Cassandra Query Language), którego gramatyka jest maksymalnie przybliżona do języka SQL

```
SELECT * FROM users WHERE first_name = 'jane' and last_name='smith';
```

- Mają wysoką dostępność danych – użytkownicy są w stanie uzyskać przynajmniej jedną kopię pożądaney informacji. Nie jest to cecha różniąca bazy NoSQL od baz relacyjnych,

4.3.2. Klasyfikacja baz NoSQL

Wyróżnia się 4 podstawowe kategorie nierelacyjnych baz danych: [50][51]

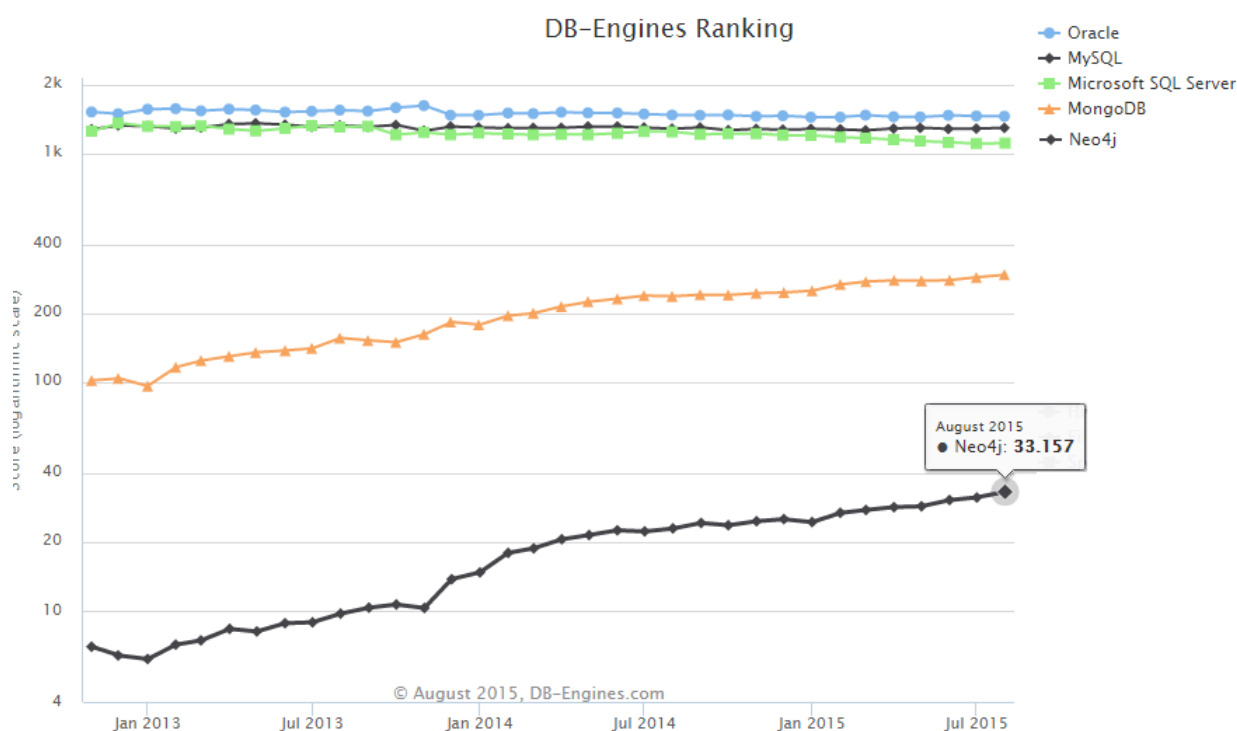


Rysunek 4.3: Porównanie modeli NoSQL oraz relacyjnego. Opracowane w oparciu o [58] i [59].

- **Bazy klucz-wartość** - przeznaczony dla bardzo dużej liczby, nieszczególnie złożonych danych. W obrazowym opisie jest to olbrzymia tablica asocjacyjna (mapa). Każdy rekord w bazie typu klucz-wartość jest identyfikowany przy pomocy klucza i może przechowywać wartość dowolnego typu, najczęściej łańcucha tekstowego. Jest to najprostszy rodzaj bazy NoSQL i udostępnia minimalny zbiór operacji na danych - w ogólności dozwolne są operacje „dodaj”, „zamień”, „usuń”, „znajdź za pomocą klucza”. Bazy te są przeznaczone raczej do składowania i wyszukiwania prostych informacji, niż ich późniejszej analizy. Doskonale nadają się do takich zastosowań jak przechowywanie chmury tagów, linkowanie artykułów i katalogów, a więc tam, gdzie z natury tablica asocjacyjna jest najlepszą strukturą danych. Przykładowe bazy danych tego typu Oracle NoSQL Database, InfinityDB, Amazon DynamoDB, Project Voldemort napisany przez LinkedIn. Istnieją bazy danych przechowujące wszystkie informacje w RAM o minimalnych opóźnieniach, takie jak Redis. Według rankunku DB-Engines jest to najpopularniejsza baza typu klucz-wartość [52]. Z powodzeniem takie same informacje można przechowywać w bazach relacyjnych, jednakże mnogość nieużywanych funkcjonalności z niego wynikających, spowodowałaby spowolnienie pracy SZDB.
- **Bazy dokumentowe** - jest to rozszerzenie idei baz typu klucz-wartość. Z każdym kluczem zintegrowana jest wartość będąca zagnieżdżoną strukturą taką jak JSON (struktura JSON zostanie opisana dokładnie w późniejszym rozdziale), BSON lub XML. Wartość ta jest nazywana dokumentem. Można eksplorować wnętrza tych dokumentów, co różni bazy kolumnowe od klucz-wartość. W poprzednio omówionym typie bazy NoSQL nie było możliwości „wejścia” do wartości na poziomie bazy danych. Dokumenty mają strukturę drzewiastą i mogą z kolei zawierać pary klucz-wartość lub następne zagnieżdżone dokumenty. Reprezentantami tej grupy są MongoDB przechowujący dokumenty typu JSON i BaseX przechowujący dokumenty typu XML i zezwalający na przeszukiwanie ich językami XPath i XQuery.
- **Bazy kolumnowe** – Dane zgrupowane są pionowo czyli kolumnami. Zamiast wierszy występujących w bazach relacyjnych dane zorganizowano tak, aby umożliwić jak najszybsze wyszukiwanie i ich agregację kosztem wolniejszego wstawiania danych. Kolumny natomiast są zgrupowane w jednostki bardziej ogólne zwane rodzinami kolumn. [53] Wiersze wielkości do 64KB (na przykładzie bazy HBase) wchodzi w skład kolumn. Danych szuka się używając głównie kolumn. Układ taki zezwala na zachowanie przypominające indeksowanie poszczególnych atrybutów w bazach tradycyjnych. Ten model bazodanowy jest skuteczny dla bardzo dużych zbiorów danych i

z **pomyślnością** stosowany przez firmę Google dla danych rozproszonych w usługach Google Analytics i Google Earth. Przykładami baz kolumnowych są Bigtable (Google) oraz Apache HBase (stosowany w połączeniu z Hadoop) lub Cassandra używana w CERN przy eksperymencie ATLAS [54].

- **Bazy grafowe** – Dane składowane są w postaci wierzchołków, natomiast zależności między nimi są przedstawione w postaci krawędzi. Szczególnie nadają się do użycia **tam gdzie** dane są natury grafowej tj. przy generowaniu rekomendacji na podstawie odwiedzeń witryn, przy przechodzeniu po sieci społecznej, wyszukiwaniu defraudacji i **to**, co szczególnie nas interesuje, w sieciach semantycznych.[47][50][57] Nie ma konieczności wykonywania explicite złączeń pobrania danych z bazy, ponieważ dane w bazach grafowych są fizycznie połączone jako referencje do siebie. Znalezienie analogii pomiędzy trójką węzeł-relacja-węzeł, a trójką podmiot-predykat-obiekt zainicjowało badania w ramach tej pracy.



Rysunek 4.4: Popularność bazy Neo4j wg serwisu DB-engines w porównaniu do trzech najpopularniejszych baz relacyjnych i MongoDB [52]. Popularność bazy Neo4j stopniowo wzrasta. Wykres jest w skali logarytmicznej.

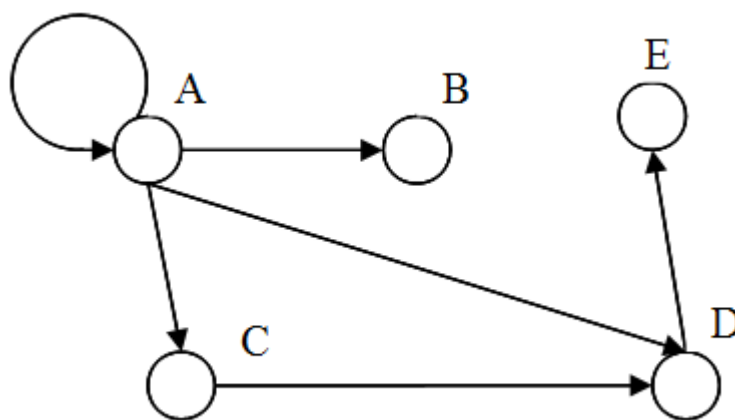
4.4. Graf

Podstawowym obiektem, który wchodzi w skład grafowej bazy danych jest graf **skierowany** etykietowany. W celu zrozumienia mechanizmów, które wykorzystuje grafowa baza danych, warto jest pokrótce przybliżyć fundamentalne pojęcia z teorii grafów.

W matematyce dyskretnej graf $G(V, E)$ definiuje się jako zbiór wierzchołków V opcjonalnie połączonych krawędziami E . Publikacja *The Graph Traversal Pattern* [45] podaje następującą definicję grafu: „Graf jest strukturą złożoną z wierzchołków (np.: węzły, punkty) połączonych ze sobą poprzez zbiór krawędzi (połączenia, linie).” [3]

Przyjmując, że graf $G := (V, E)$ jest złożony ze zbioru wierzchołków V i rodziny krawędzi E , nazywamy go:

- Prostym, jeżeli V jest zbiorem niepustym, a E jest rodziną (zbiorem zbiorów) dwuelementowych podzbiorów zbioru wierzchołków V i $E \subseteq \{\{u, v\} : u, v \in V, u \neq v\}$. Gdy krawędź $\{u, v\}$ łączy wierzchołki u i v , nazywa się je sąsiednimi,
- Grafem ogólnym (a także zwyczajnie grafem lub multigrafem), jeżeli krawędź może łączyć wierzchołek z nim samym (pętla) lub między dwoma węzłami może wystąpić więcej, niż jedna krawędź.
- Grafem etykietowym, jeżeli przez zdefiniowanie funkcji zV lub E można przypisać krawędziom lub wierzchołkom w pewien zbiór X etykiety, służące do przechowywania dodatkowych informacji. Etykiety liczbowe są nazywa się wagami.
- Grafem skierowanym (digrafem), jeżeli jego wierzchołki mają ustalony kierunek tj. $E = \{(u, v) \neq (v, u) : u, v \in V\}$. Kolejność wierzchołków w parze wyznacza kierunek krawędzi – w przypadku pary u, v krawędź biegnie z wierzchołka u do wierzchołka v [60].



Rysunek 4.5: Graf skierowany

Trasą nazywamy „linię, po której przedostajemy się z jednego wierzchołka do innego, składająca się z ciągu kolejno przechodzonych krawędzi”. [60] Trasa, w której każdy wierzchołek pojawia się dokładnie jeden raz, jest nazywana **drogą**. Na powyższym rysunku może być ciąg $A \rightarrow D \rightarrow E$ lub $A \rightarrow B$.

Z grafu, który jest wiedzą w **ontologii wyróżnić** można podgrafy, w których każdą parę wierzchołków można połączyć dokładnie jedną drogą. Grafy tego typu nazywane są **drzewami**. Drzewa są istotnymi elementami wielokrotnie pojawiającymi się przy składowaniu i analizie dokumentów w Web 3.0 (standard DOM, XML, HTML). Najprostszym przykładem drzewa jest trójka ontologiczna.

4.5. Grafowa baza danych Neo4j

4.5.1. Struktura danych w bazie Neo4j

Baza danych Neo4j przechowuje dane w postaci grafu skierowanego etykietowanego, co wstępnie pokrywa się z wymaganiami stawianymi przez Semantic Web. Grafowe bazy danych ze względu na brak standaryzacji i mnogość dedykowanych zastosowań różnią się od siebie wewnętrzną implementacją.

Jak pokazuje historia rozwoju baz, takich jak Cassandra, Amazon Dynamo i FlockDB, bazy NoSQL są tworzone w ramach przedsiębiorstw do konkretnego celu, takiego jak składowanie wiadomości prywatnych lub rekomendowanie produktów. Od grafowych baz danych oczekuje się, aby zezwalały na szybki dostęp do sąsiadów. Szczególnie interesujący jest wewnętrzna implementacja przechowywania danych przez bazę Neo4j, dająca nadzieję na **wykorzystanie przy** przechowywaniu trójek sieci semantycznej. [47]

Wierzchołki grafu mogą być dodatkowo opisane przez etykiety wprowadzające podział wierzchołków na mniejsze kategorie. Etykiety poprawiają czytelność zapytań i zezwalają na budowanie indeksów zwiększających wydajność wyszukiwania. Baza nie narzuca obowiązku nadania etykiet na węzły. Etykiety można traktować jako analogię do klas ontologii, więc zależność ta została wykorzystana przy pisaniu programu. **Baza danych pozwala nadać więcej niż jedną etykietę, jednak z tego nie skorzystano.** Mimo, iż graf jest skierowany, to istnieje możliwość ignorowania kierunku wierzchołka. Część relacji pomiędzy instancjami jest dwukierunkowa i nie ma sensu narzucania sztucznego kierunku. Między dwoma konkretnymi węzłami w bazie danych może istnieć wiele krawędzi. Tak samo dozwolone jest stworzenie pętli.

Zarówno wierzchołki, jak i krawędzie mogą być opisane przez atrybuty, będącymi parą klucz-wartość. Zbiór atrybutów i ich wartości dla wierzchołka (bądź relacji) jest zwracany jako dokument JSON. Nazwa atrybutu musi być wartością tekstową, a wartość jest reprezentowana przez wszystkie typy podstawowe dostarczone w języku Java, przy czym liczby całkowite zostają rozszerzone do typu `long` (64-bitowa liczba ze znakiem), a liczby zmiennoprzecinkowe do typu `double` (64-bitowa liczba w notacji IEEE 754). Należy przy tym zwrócić uwagę, że zapis liczb w bazie Neo4j różni się od specyfikacji JSON, **który ze** względu na genezę (oparty na języku JavaScript), duże wartości całkowite zwraca jako liczby zmiennoprzecinkowe w postaci wykładniczej.

Dane bazy Neo4j **fizycznie** przechowywane **są** w niezależnych plikach o zmiennej długości, odpowiadających węzłom, krawędziom i ich typom, atrybutom i opcjonalnym indeksom.

Węzeł jest komórką wielkości 9 bajtów, zawierającą referencję do fragmentu **pliku z** jego pierwszym atrybutem, adres **do** części pliku z jego pierwszą krawędzią, a także **flagę** logiczną, sprawdzającą, czy węzeł ma jakiegoś sąsiada. Aby uzyskać węzeł na podstawie jego identyfikatora, wystarczy uzyskać dostęp do komórki pamięci przesuniętej w pliku węzłów o $9 * ID$ bajtów od jego początku, utrzymując złożoność wynoszącą $O(1)$.

Atrybuty przechowuje się w rekordach złożonych z zarezerwowanych 4 bloków pamięci dla zmiennej i jej typu, identyfikatora następnego atrybutu i opcjonalnego wskaźnika do pliku indeksującego te atrybuty. Atrybuty reprezentuje się jako listę jednokierunkową.

Krawędź jest komórką o długości 33 bajtów. Złożona jest z identyfikatorów węzłów sąsiadujących, wskaźnika do pliku ze swoim typem i wskaźników do poprzednich i następnych relacji do węzłów sąsiadujących. Wskaźniki te, GBD wykorzystuje w liście dwukierunkowej służącej do przechodzenia pomiędzy węzłami.

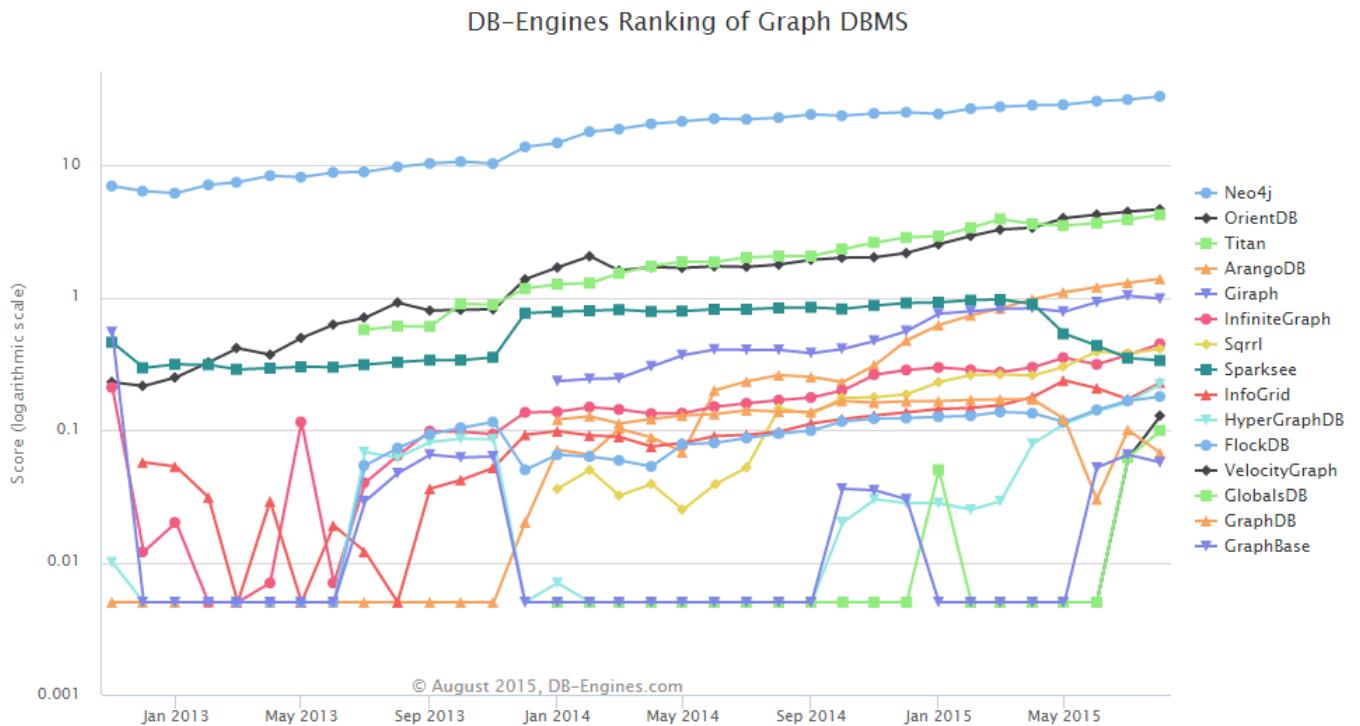
Intencją autora było maksymalne wykorzystanie własności wewnętrznej struktury bazy, celem uproszczenia sieci do absolutnego minimum.

Element sieci semantycznej	Struktura z Neo4j	Uwagi dot. metadanych
Klasa	Etykieta nałożona na węzeł	Klasa dodaje 1 węzeł metadanych
Podmiot lub obiekt	Węzeł	
Relacja (predykat)	1 węzeł + krawędź	Definicja relacji dodaje 1 relację
Atrybut	Atrybut	Nazwa atrybutu dodaje 1 węzeł
Podklasy (dziedziczenie)	Krawędź	
Pytanie	Węzeł + relacja	

Tablica 4.1: Porównanie elementów sieci **semantycznej do** odpowiadających struktur Neo4j

4.5.2. Inne grafowe bazy danych istniejące na rynku

W trakcie pisania tej pracy rozważano także inne grafowe bazy danych. Na rynku istnieje dość duży zbiór alternatyw dla Neo4j, **jednak są one raczej niszowe.** Literatura niestety opisuje głównie Neo4j i **na temat niektórych baz trudno uzyskać jakiejkolwiek informacje, poza ich stroną oficjalną.**



Rysunek 4.6: Popularność bazy Neo4j wg serwisu DB-Engines w porównaniu do innych baz grafowych. Wykres jest w skali logarytmicznej.

Poniżej znajduje się lista kilku GBD, których specyfikacja odpowiada potrzebom sieci semantycznych.

OrientDB

Jedną z istniejących grafowych baz danych jest OrientDB. Jest to hybrydowy (wielomodelowy) **SZDB** zaprogramowany w języku Java. Wspiera pary klucz-wartość, dokumenty oraz grafy, które podobnie jak w **Neo4j** są przechowywane fizycznie w postaci **fizycznych**, bezpośrednich połączeń. Umożliwia przeszukiwanie bazy danych korzystając z języka Gremlin wraz z dialektem **SQL wzbogaconym** o obsługę grafów. Wprowadziła nowy system indeksowania MVRB-Tree oparty na drzewach czerwono-czarnych oraz B+. Zapewnia wsparcie ACID i dostarcza interfejs REST dostępu do danych.

Titan

Nawiązując do opisu na witrynie autorów [61] TitanDB jest skalowalną, rozproszoną bazą zoptymalizowaną pod kątem przechowywania i przeszukiwania grafów zawierających miliardy krawędzi i węzłów rozproszonych na wielomaszynowym klastrze. Udostępnia replikację w celu optymalizacji zapytań i zwiększania niezawodności. Wspiera dane geolokacyjne i przeszukiwanie **pełnotekstowe oparte** o Apache **Lucene zachowując** licencję Open Source.

Jako ciekawostkę wyróżniającą tę bazę **należy wsparcie** dla importu danych z Apache Cassandra, Apache HBase i Oracle BerkeleyDB. W serwisie DB-Engines jej popularność jest monitorowana zaledwie od 2 lat, a przez ten czas utrzymuje **2/3** lokatę w kontekście popularności.

AllegroGraph

W odróżnieniu od innych baz **grafowych dane** przechowuje natywnie w postaci trójek RDF, które można przeszukiwać językiem SPARQL. Każdą utworzoną trójkę rejestruje w 7 **indeksach celem** szybkiego wyszukiwania. Administrator BD może dodatkowo zdefiniować własne indeksy. Zapewnia pełne wsparcie dla transakcyjności i zbioru narzędzi do sporządzania kopii **zapasowych**, oraz ich szybkiego odtwarzania. Specyfikacja AllegroGraph jest zgodna z obecnymi rekomendacjami Semantic Web wyda-

nymi przez W3C, zatem jest to dobry kandydat do przyszłych badań nad zastosowaniem grafowych baz danych w Semantic Web. AllegroGraph jest oprogramowaniem płatnym, co negatywnie przekłada się na możliwości przetestowania jej funkcjonalności.

4.5.3. Możliwości Neo4j

Poniżej znajduje się lista możliwości bazy danych Neo4j, które skłoniły do próby zbudowania na niej sieci semantycznej:

- Pełne wsparcie dla transakcji ACID, co jest niezbędne przy zabezpieczeniu danych,
- W wersji komercyjnej możliwość rozproszenia danych w klastrze,
- Przechowywanie do ok. 34 mld węzłów, 34 mld krawędzi i 68 mld atrybutów związanych ze sobą w sposób natywny, co zwiększa wydajność późniejszego przeszukiwania,
- Szybki dostęp do wybranego węzła. Atrybuty węzła można swobodnie modyfikować,
- Dynamiczna struktura w naturalny sposób odzwierciedlająca wymaganiom stawianym przez Semantic Web,
- Język zapytań Cypher, którego można używać w ramach JDBC oraz inne interfejsy dostępu do danych,
- Indeksowanie tekstu oparte o bibliotekę Apache Lucene.
- Jako subiektywną zaletę autor wysoko ceni bardzo dobrze przygotowaną dokumentację techniczną tej bazy.
- W wersji Community jest ona dostępna na licencji GPL v3. W przypadku, gdy istnieje konieczność skorzystania z zaawansowanych funkcji wersji Enterprise (zaawansowane cache, klasteryzacja i tworzenie kopii zapasowych w czasie rzeczywistym), można zamówić specjalną licencję dla celów edukacyjnych.

4.5.4. Język Cypher

Deklaratywny język Cypher został zaprojektowany w tak, aby zapytania były zrozumiałe dla osób nie związanych bezpośrednio z IT. Cypher wprowadza wzór zapytań w postaci wzorca oczekiwanego podgrafu. Dzięki niemu można pobierać dane powiązane ze sobą na kilku poziomach głębokości. Jako język deklaratywny, bardziej skupia się na tym co użytkownik ma otrzymać, a określa sposób dostarczenia wyników. Słowa kluczowe są wzorowane na językach SQL i SPARQL.

Ogólnie zapytania do bazy danych mają format, przy czym jedynym słowem kluczowym wymagającym do wykonania zapytania jest RETURN lub DELETE.

[MATCH CREATE WHERE]

[OPTIONAL MATCH WHERE]

[WITH [ORDER BY] [SKIP] [LIMIT]]

RETURN/DELETE [ORDER BY] [SKIP] [LIMIT]

4.5.5. CREATE

Klauzula **CREATE** służy do utworzenia węzła w grafie. Tworzenie węzła pozbawionego etykiety z jednym atrybutem wygląda następująco:

```
CREATE (wezel{ atrybut : wartość_atrybutu }) RETURN wezel;
```

`wezel` jest lokalna nazwa utworzonego węzła. Bezpośrednio po niej jest opcjonalny obiekt **JSON** przechowujący atrybuty tego węzła.

Aby nadać etykietę wystarczy dopisać po symbolu „:” jej nazwę. Jeżeli podana etykieta jeszcze nie istnieje, zostanie ona utworzona. Poniżej znajduje się przykład nadania kilku etykiet:

```
CREATE (wezel : EtykietaPierwsza : EtykietaDruga {
    atrybut : wartość_atrybutu }) RETURN wezel;
```

4.5.6. MATCH

Jest to najważniejsze słowo kluczowe, w którym wskazuje się wzorzec oczekiwanego grafu. Ogólnie składnia zapytania wygląda następująco:

```
MATCH (a) – [] – (b) RETURN a, b
```

Zapytanie to wyszuka i zwróci wszystkie pary węzłów $\{a, b\}$ niezależnie od tego, w którą stronę jest skierowana relacja oraz jakiego jest typu. Lokalne nazwy węzłów `a` oraz `b` są dowolnie wybrane i używa się ich jedynie na czas życia zapytania. Nawiasy okrągłe określają węzły, natomiast myślniki i nawiasy kwadratowe symbolizują krawędzie. Można dodatkowo zastosować limity typów relacji:

```
MATCH (a) – [r : OWNER] – (b) RETURN a, b, r
```

Zapytanie to zwróci trójki węzłów połączone krawędziami `r` typu `OWNER`. Jeżeli użytkownika interesuje w wyniku jeden z węzłów można usunąć lokalne nazwy pozostałych. Baza jest grafem skierowanym i można tę cechę, gdy istnieje potrzeba wyszukania podgrafów z uwzględnieniem ich kierunku.

```
MATCH (author) – [:COMMENTED] –>() RETURN author
```

Zwróci to osoby, które nawiązały relację `COMMENTED` z pewnym węzłem.

4.5.7. WHERE

Instrukcja **WHERE** służy do zawężania wyników wyszukiwania, które mogą być uzyskane przy pomocy instrukcji **MATCH**. Jej argumentami mogą być funkcja `has(węzeł.atrybut)`, weryfikująca, czy dany węzeł posiada pewien atrybut, standardowe operatory matematyczne `+`, `-`, `*`, `/` and `%`, `^`, operatory porównujące `=`, `<>`, `<`, `>`, `<=`, `>=`, operatory logiczne **AND**, **OR**, **XOR**, **NOT**, operator porównywania wyrażeń regularnych `=~` oraz operator **IN**, sprawdzający, czy dany obiekt znajduje się w tej kolekcji.

Przykładowe zapytanie do bazy danych serwisu społecznościowego, pobierające wiadomości pomiędzy dwoma znanymi użytkownikami, wysłane od 2013 roku i posiadające załącznik mogłoby wyglądać tak:

```
MATCH (n : User) – [:SENT] –>(message : Message) – [:TO] –>(m : User)
WHERE n.login = "user1"
    AND m.login = "user2"
    AND HAS(message.creation_time)
    AND HAS(message.attachment)
    AND message.creation_time >= 1388534400
RETURN message, n.firstName AS author
ORDER BY message.creation_time DESC;
```

Zapytanie to szuka węzłów z nałożoną etykietą *User*, pomiędzy którymi pośredniczy węzeł z nałożoną etykietą *Message*. Głębokość zapytania można dowolnie rozszerzyć. Etykiety nie są konieczne, jednakże ułatwiają systematyzację danych i szybsze poszukiwanie węzłów. Optymalizator zapytań Neo4j domyślnie przeszukuje indeksy, o ile takowe istnieją.

Poprzez umieszczenie dodatkowej instrukcji na początku zapytania

```
START n=node:User(login = "pawel02"), m=User(login="adrian01")
```

można wymusić od optymalizatora zmianę planu zapytania tak, aby zapytanie rozpoczęło się od znalezienia tych 2 węzłów. Jeżeli etykiety nie zostały nałożone, można wymusić przeszukiwanie tych wygenerowanych automatycznie na atrybutach

```
START n=node:node_auto_index(login = "pawel02")
```

Istnieje możliwość przeszukiwania węzłów na podstawie identyfikatora. Jest to rozwiązanie najszybsze, ponieważ ze względu na opisaną wcześniej strukturę pliku z węzłami, odnalezienie węzła przebiega w czasie stałym.

```
MATCH (node) WHERE id(node) = 65120 return node
```

Lucene. Wyszukiwanie pełnotekstowe.

Istotne w dużych dokumentach tekstowych jest wyszukiwanie pełnotekstowe (ang. full-text search). Baza Neo4j korzysta z wydajnej wyszukiwarki pełnotekstowej Apache Lucene. Biblioteka ta dostarcza algorytmy bardzo szybkiego wyszukiwania tekstu **kosztem**, budowania dużych indeksów pomocniczych. Użycie jest bardzo proste i ogranicza się do zastosowania operatora `~`. Poniżej znajduje się przykładowe zapytanie języka Cypher **wyszukujące** atrybut `description` we wszystkich węzłach w bazie danych, które zawierają w sobie zwrot „Wydział Fizyki i Informatyki Stosowanej” ignorując wielkość znaków.

```
MATCH (n)
WHERE r.name =~ '(?i)Wydział_Fizyki_i_Informatyki_Stosowanej.*'
RETURN DISTINCT r AS result, id(r) AS id,
ORDER BY length(r.description)
LIMIT 10
```

Zostanie zwróconych 10 węzłów o najdłuższych dokumentach z tym zwrotem.

4.5.8. Interfejsy dostępu do bazy danych Neo4j

Neo4j udostępnia trzy podstawowe metody dostępu do danych:

- Natywna biblioteka operująca na abstrakcyjnych obiektach Neo4j
- REST API - metoda ta opcjonalnie używa języka Cypher
- JDBC używający języka Cypher, choć w praktyce metoda ta jest oparta na REST API

Neo4j Embedded

Neo4j jest bazą danych utworzoną w języku Java. Aplikacje utworzone w tym samym języku mają możliwość dostępu do natywnego interfejsu bazy. Aby go uzyskać wystarczy dołączyć do `classpath` projektu plik JAR lub dodać zależność do `pom.xml` projektu Maven (opis technologii Maven zostanie przedstawiony w jednym z następnych rozdziałów).

```
<project>
  <dependencies>
    <dependency>
      <groupId>org.neo4j</groupId>
```

```

        <artifactId>neo4j</artifactId>
        <version>2.2.5</version>
    </dependency>
</dependencies>
</project>

```

Następnie, aby utworzyć bazę danych wystarczy zainicjować instancję obiektu klasy `GraphDatabaseService`. Wykonuje się to poprzez metodę `newEmbeddedDatabase` z fabryki `GraphDatabaseFactory`. `DATABASE_PATH` jest to ścieżka do podstawowego pliku bazy danych.

```
GraphDatabaseService neoGraph =
    new GraphDatabaseFactory().newEmbeddedDatabase( DATABASE_PATH );
```

Jak widać, `autorzPo` zakończeniu operacji na bazie należy zamknąć strumień połączenia poprzez

```
neoGraph.shutdown();
```

Neo4j, jak już wspomniano, zezwala na transakcyjność zgodną z ACID. Transakcje w bazie Neo4j nie są obowiązkowe, jednakże korzystanie z tej funkcjonalności jest bardzo dobrą praktyką, znacznie poprawiającą integrację posiadanych danych. Transakcję należy umieścić w bloku

```

try ( Transaction tx = neoGraph.beginTx() )
{
    // Tutaj należy umieścić instrukcje objęte transakcją.
    tx.success();
}
finally
{
    tx.close(); //zamknięcie transakcji
}

```

Baza danych zezwala na wprowadzenie własnych typów relacji. Wymagane jest, żeby każdy typ relacji został zdefiniowany w ramach typu wyliczeniowego implementującego interfejs `RelationshipType`. Jest to pewne ograniczenie, które przemawiało na korzyść języka Cypher, zamiast połączenia natywnego

```

private static enum Relations implements RelationshipType
{
    KNOWS
}

```

Załóżmy, że istnieje potrzeba utworzenia dwóch węzłów, nadania im jakiegoś atrybutu i powiązania ich między sobą. Da to nam strukturę przypominającą trójkę Semantic Web. Aby to zrobić przy użyciu natywnego dostępu do bazy, należy utworzyć stosowne obiekty klas `Node` (węzeł) i `Relationship` (relacja).

```

Node node1 = neoGraph.createNode();
Node node2 = neoGraph.createNode();
Relationship relationship;

```

```
firstNode.setProperty( "message", "Hello ,_" );
secondNode.setProperty( "message", "World!" );

relationship = node1.createRelationshipTo( node2 , Relations.KNOWS );
```

Metoda `createNode()` klasy `GraphDatabaseService` zwraca obiekt węzła, natomiast `createRelationshipTo` klasy `Node` tworzy połączenie skierowane z innym węzłem. Drugim argumentem tej metody jest jedna z predefiniowanych wartości klasy dziedziczącej po `RelationshipType`.

Operacje są natychmiast wykonywane na poziomie SZBD.

REST API

Jest to domyślna metoda dostępu do bazy danych, gdyż dla systemów zewnętrznych nie wymaga żadnych dodatkowych bibliotek poza możliwością nawiązywania połączeń HTTP.

API to ma domyślnie włączoną transakcyjność i zezwala na wykonanie szeregu instrukcji języka Cypher przesłanych zapytaniami HTTP. Transakcja może być otwarta na czas życia kilku zapytań HTTP tak długo, aż użytkownik nie zdecyduje się wykonać operacji `commit` lub `rollback`.

Transakcję rozpoczyna się poprzez przesłanie pierwszego zapytania HTTP wysłanego pod URI **`http://nazwa_serwera:7474/db/data/transaction`**, którego nagłówek zawiera :

- **Accept:** `application/json; charset=UTF-8`
- **Content-Type:** `application/json`

Żądanie utworzenia węzła ogranicza się do zapytania pod URI **`http://localhost:7474/db/data/node`**. Treść zapytania powinna zawierać dokument typu JSON

```
POST http://localhost:7474/db/data/node
{
    "nazwa" : "wartość_atrybutu",
    "wiek":_4
}
```

W odpowiedzi zostanie zwrócony ID tego węzła, lub błąd w razie niepowodzenia.

Dostęp do węzła, którego ID znamy jest trywialny i ogranicza się do

```
GET http://localhost:7474/db/data/node/10391
```

Usunięcie węzła odbywa się analogicznie, z tą różnicą, że zamiast metody `GET` stosuje się `DELETE`. Instrukcja dodawania relacji z interfejsu Java może przypominać zapytanie REST:

```
POST http://localhost:7474/db/data/node/1/relationships
{
    "to" : "http://localhost:7474/db/data/node/2",
    "type" : "KNOWS"
}
```

To co w bibliotece było argumentami metody `createRelationshipTo` (węzeł docelowy i typ relacji), w REST API jest kolejnymi kluczami dokumentu JSON.

W razie potrzeby wykonania bardziej zaawansowanych operacji, można wykonywać żądania języka Cypher.

```
{
    "statements" : [ {
        "statement" : "CREATE_(n_{ props })_RETURN_n",
        "parameters" : {
```

```

      "props" : {
        "name" : "Węzeł_pierwszy"
      }
    }
  ]
}

```

Statements jest tablicą zawierającą obiekty *statement*, czyli poszczególne instrukcje. *Parameters* jest zbiorem wartości klucz-wartość opisującymi atrybuty poszczególnych obiektów.

W odpowiedzi serwer zwraca dokument JSON zawierający opis utworzonych, zmodyfikowanych i usuniętych danych, oraz ewentualne błędy

```

{
  "commit" : "http://localhost:7474/db/data/transaction/2/commit",
  "results" : [
    {
      "columns" : ["n"],
      "data" :
        [ {
          "row" :
            [ {
              "name" : "Węzeł_pierwszy"
            } ]
        } ]
    },
    "transaction" :
    {
      "expires" : "Thu, 03 Sep 2015 11:20:21 +0001"
    },
    "errors" : [ ]
  ]
}

```

Autoryzacja użytkowników

Interfejs REST zezwala na opcjonalną autoryzację użytkowników. Uwierzytelnianie użytkownika jest oparte na metodzie HTTP Basic Auth opisanej już w dokumencie Hypertext Transfer Protocol RFC 1945. Odbyna się to poprzez przesłanie nazwy użytkownika i hasła w formie:

`nazwa_użytkownika:hasło`

zakodowanej metodą Base64 i dodanej do zapytania jako nagłówek Authorization. Oczywiście brak bezpieczeństwa tej metody mocno zaleca używanie połączeń szyfrowanych.

```

GET http://localhost:7474/user/neo4j
Accept: application/json; charset=UTF-8
Authorization: Basic YWRyaWFuOmhhc8WCbw==

```

Operacje takie jak zmiana hasła użytkownika także opierają się na zapytaniach z dokumentami JSON:

```

POST http://localhost:7474/user/neo4j/password
Accept: application/json; charset=UTF-8
Authorization: Basic YWRyaWFuOmhhc8WCbw==
{
  "password" : "nowe_hasło123"
}

```


4.5.9. JDBC

Połączenie przez JDBC można traktować jako hybrydę dwóch poprzednich metod. Ze względu na potrzebę rozwinięcia opisu węzłów i relacji strukturami bardziej przypominającymi standardy Semantic Web, połączenie natywne nie jest wystarczające. Komunikacja poprzez połączenie bezpośrednio z REST API wprowadziłaby nadmiar zbędnego kodu.

Do aplikacji należy dostarczyć sterownik JDBC (można tego dokonać przy pomocy Maven), a następnie nawiązać standardowe połączenie klasy `Connect`:

```
Class.forName("org.neo4j.jdbc.Driver"); //Dodanie sterownika
//nawiązanie połączenia
Connection con =
    DriverManager.getConnection("jdbc:neo4j://localhost:7474/");

try(Statement statement = con.createStatement())
{
    //Wykonanie zapytania
    ResultSet rs =
        statement.executeQuery("CREATE_(n{'Name':'Piotr'})_RETURN_n");
    while(rs.next())
    {
        //Wyświetlenie wyniku
        System.out.println(rs);
    }
}
```

4.6. Wady Neo4j

Podstawową wadą jest brak standaryzacji baz danych i języków zapytań. Nie ma pewności, że aplikacje powstała teraz, będą działała w przyszłych wersjach bazy danych. Neo4j dynamicznie się rozwija i część funkcjonalności może zostać wyłączona w przyszłości. Przykładowo, autorzy tej bazy danych zapowiedzieli, że dostęp do węzłów bezpośrednio przy pomocy ID zostanie usunięta, ponieważ zostaną wprowadzone mechanizmy kompresji zmieniające wewnętrzną strukturę plików z węzłami. Spowoduje to, że identyfikatory węzłów pomiędzy kolejnymi transakcjami będą mogły ulec zmianie.

Autorzy nie zalecają stosowania GBD w sytuacjach, ilość krawędzi jest niewielka. Struktura danych, w których liczba zależności jest niewielka, bardziej przypomina tabele, a więc lepiej skorzystać z klasycznych rozwiązań relacyjnych. Przez skorzystanie z grafowej bazy danych, należy upewnić się, czy sieci semantycznej nie można prosto zapisać w postaci tabeli. Istnienie baz relacyjnych od ponad 40 lat spowodowało pojawienie się mnóstwa narzędzi (optymalizatory zapytań) znacznie minimalizujących wady baz relacyjnych. Na daną chwilę grafowa baza danych Neo4j nie posiada statystyk znanych z RBD, więc wydajność zapytań w bardzo dużej mierze zależy od umiejętności i doświadczenia osób piszących zapytania.

Ponadto migracja danych do innego systemu bazodanowego jest bardzo utrudniona, ze względu na brak uniwersalnego języka, takiego, jak DDL znanego z SQL. Eksport danych może wymagać szeregu konwersji pomiędzy typami CSV/XML/JSON. Ze względu na wyszukiwanie pełnotekstowe Apache Lucene i szereg indeksów na atrybutach, przestrzeń dyskowa zajmowana przez bazę danych jest 1,5 – 2 razy większa od przestrzeni zajmowanej przez bazy relacyjne o podobnej liczbie dokumentów.

5. Opis aplikacji

Aby rozwiązać problem przedstawiony w temacie pracy magisterskiej, postanowiono utworzyć narzędzie, które umożliwi użytkownikowi utworzenie modelu danych, wypełnienie go oraz przeszukiwanie.

5.1. Funkcjonalność aplikacji

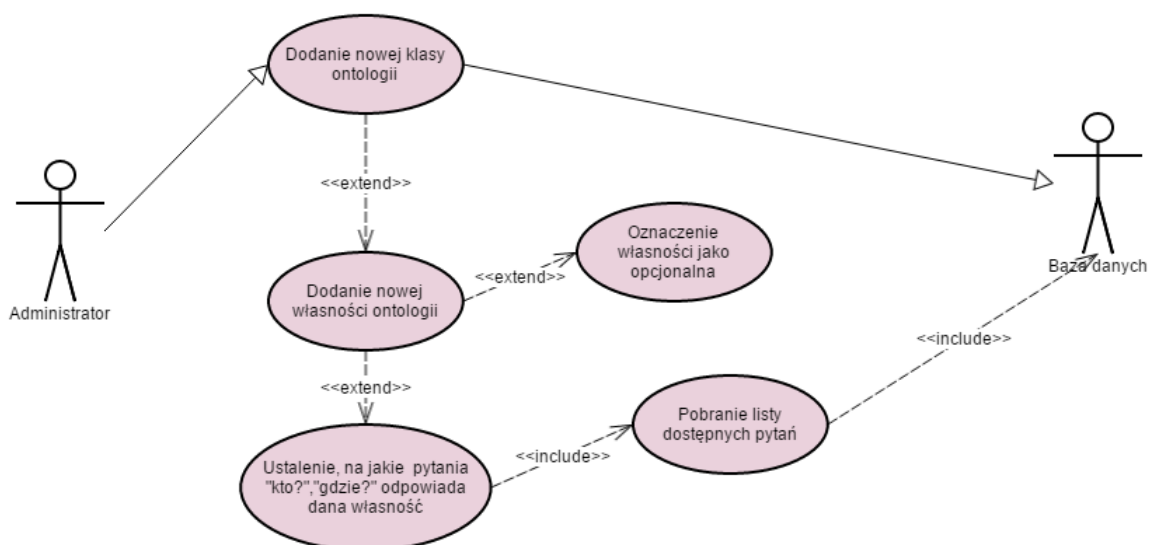
5.1.1. Założenia funkcjonalności

Celem badań przed napisaniem pracy magisterskiej było zaprogramowanie aplikacji, która zezwala użytkownikowi na zbudowanie modelu dowolnej ontologii, wprowadzeniu do niej informacji, a następnie przeszukiwaniu jej.

Budowanie ogólnych ontologii wymagało zaimplementowanie pojęć przez nie definiowanych tj. klas, instancji, relacji i atrybutów. Jako, że zarówno baza danych Neo4j jak i aplikacja są zaimplementowane w języku Java, wszystkie dane są przechowywane przy użyciu kodowania Unicode UTF-8. W rezultacie powstała aplikacja, która może zostać użyta jako narzędzie do zarządzania ontologią. Do wszystkich danych został uruchomiony serwis REST, który umożliwia dostęp do danych dla zewnętrznych systemów.

5.1.2. Klasa

Klasa jest to najprostsza struktura dostępna w programie. Użytkownik ma prawo utworzyć klasę o dowolnej nazwie oznaczającej pewien abstrakcyjny byt. Klasa może być złożeniem kilku słów oddzielonych spacją np. Auto Osobowe. Instancja (węzeł) jest to rzeczywisty reprezentant danej klasy, podobnie jak to zostało opisane w sekcji poświęconej ontologii.



Rysunek 5.1: Proces dodawania nowej klasy.

Użytkownik ma możliwość dodania nowej ontologii, ustalenie domyślnych atrybutów w klasie oraz opcjonalnie dodanie informacji na jakie pytanie odpowiada dany atrybut. Utworzenie klasy powoduje powstanie nowej etykiety i metadanych umożliwiających późniejsze przeszukiwanie.

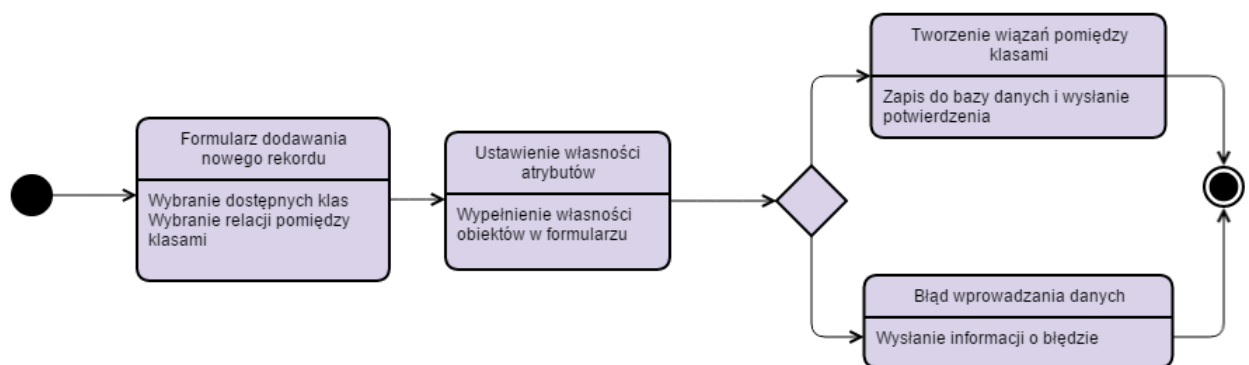
5.1.3. Atrybut

Atrybuty są to cechy określające daną instancję. W momencie definicji klasy użytkownik może proponować domyślnie oczekiwane atrybuty. Na przykład dla klasy Osoba może zdefiniować Imię, Nazwisko i PESEL. W momencie wprowadzania instancji do bazy nie ma obowiązku podawania wszystkich atrybutów oczekiwanych przez klasę tj. nie każda osoba musi mieć nadany PESEL (obcokrajowcy). Można także zdefiniować na jakie pytanie ma ona odpowiadać („kto?”, „co?”, „kiedy?”).

Podczas dodawania instancji można także podać atrybuty dodatkowe charakterystyczne dla danej instancji, których nie przewidziała klasa. Podczas wprowadzania instancji do bazy można od razu nawiązać połączenie z innymi instancjami.

Jeżeli już pewna klasa istnieje, użytkownik może skorzystać z funkcjonalności dziedziczenia. Podanie pewnej klasy jako nadrzędnej spowoduje skopiowanie od niej domyślnie oczekiwanych atrybutów.

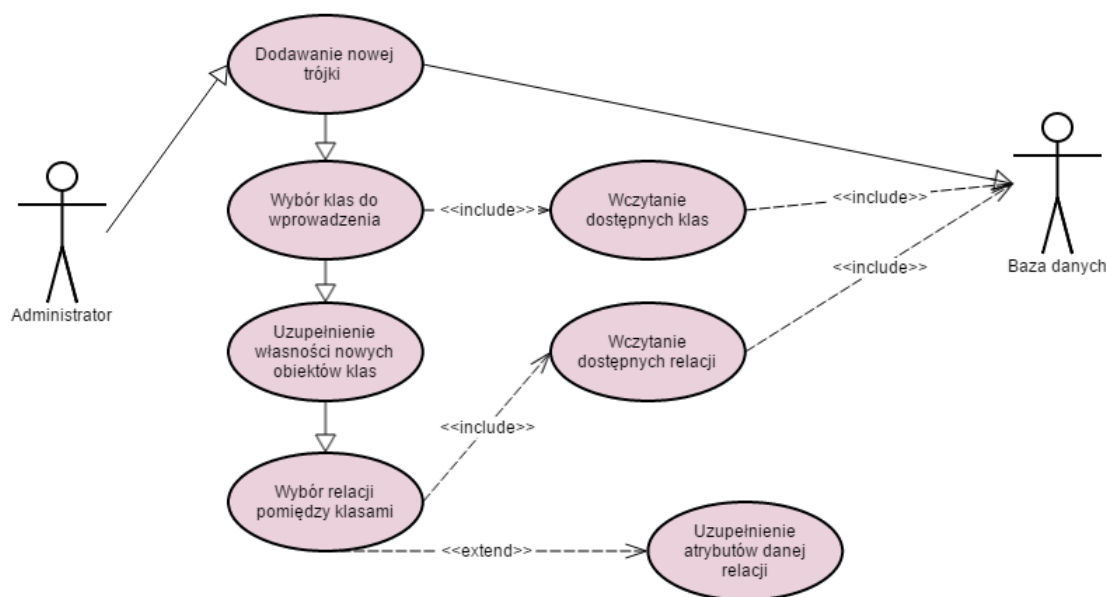
Na poniższym diagramie znajduje się poglądowy schemat opisujący proces tworzenia instancji.



Rysunek 5.2: Proces dodawania instancji klasy do aplikacji.

Dodawanie instancji jest procesem dwuetapowym. Tworzenie instancji oraz wiązań wymaga:

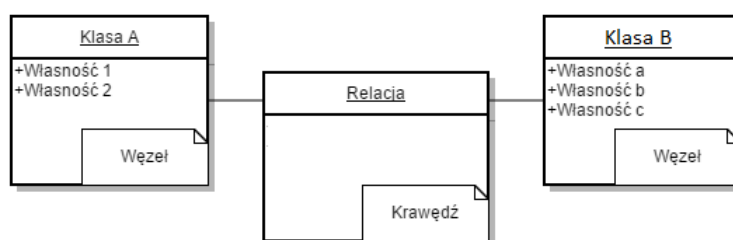
1. wprowadzenia instancji do grafu,
2. wykonanie żądania spajającego pary danych.



Rysunek 5.3: Proces dodawania instancji do aplikacji.

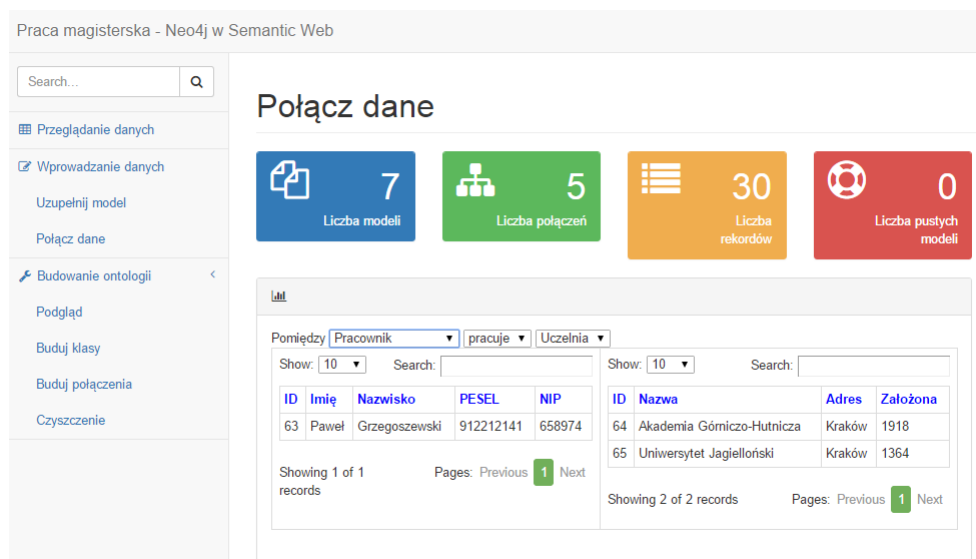
5.1.4. Relacja

Relacja (krawędź) jest to połączenie nałożone pomiędzy dwiema klasami. Połączenie to sprawia, że przeszukiwanie bazy danych za pewną informacją spowoduje także sprawdzenie instancji klas sąsiadujących z interesującą nas informacją. Relacja powinna być pojedynczym słowem. Pomiedzy 2 konkretnymi instancjami może istnieć więcej niż jedna relacja.



Rysunek 5.4: Rysunek przedstawia poglądowo pojęcia ontologii i ich odpowiedniki z grafowej bazy danych.

Poniżej znajduje się zrzut z aplikacji przedstawiający tabelę łączenia dwóch elementów. Aplikacja jest oparta o technologię AJAX - zmiana klas, które użytkownik chciałby połączyć powoduje dynamiczną zmianę rozmiaru tabeli w zależności od liczby atrybutów.



Rysunek 5.5: Interfejs graficzny łączenia instancji

5.1.5. Metadane

Kluczową trudnością w projektowaniu aplikacji było ustalenie jak przechowywać opis modelu ontologii przy zachowaniu maksymalnej prostoty i optymalnej wydajności. W grafowej bazie danych każda informacja jest przechowywana jako węzeł, atrybut, bądź relacja. Zdecydowano pozostanie przy tej konwencji i w związku z tym opis ontologii jest przechowywany w postaci grafu w Neo4j. Metadane tworzą się w trakcie budowania ontologii na poziomie interfejsu użytkownika.

Podejście takie jest dużą różnicą w porównaniu do idei zakładającej przechowywanie modelu w postaci dokumentu RDF. Dzięki temu ontologię można eksportować do pliku w formacie JSON korzystając wyłącznie z interfejsu graficznego dostarczonego przez samą bazę danych.

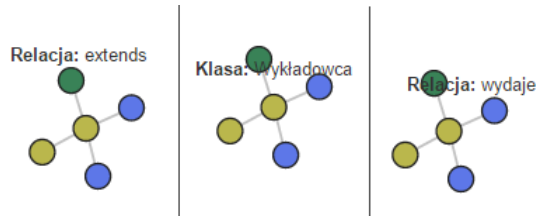
Wprowadzono specjalne etykiety zarezerwowane dla przechowywania metadanych:

- *AvailableClass* - opisuje klasę, której instancje mogą być utworzone w interfejsie użytkownika. Przy tworzeniu tej metadanej na jaw wyszła pewien brak w bazie Neo4j. W ontologiach wielokrotnie stosuje się nazwy wielowyrazowe, ale baza ta nie obsługuje etykiet zawierających spację. W związku z tym wprowadzono specjalny atrybut `code` przechowywany w węzłach typu *AvailableClass*. Atrybut ten przechowuje pełną nazwę klasy. Wpłynęło to niestety negatywnie na poziom złożoności zapytań, gdyż wymagane są konwersje pomiędzy nazwą klasy z UI, a tą w bazie danych.
- *AvailableAttribute* - opisuje atrybut wchodzący w skład klasy. Węzły *AvailableAttribute* powstały w celu indeksowania atrybutów klas, a tym samym przyspieszeniu wyszukiwania.
- *AvailableRelationship* - opisuje relację dozwoloną pomiędzy instancjami. Dla dziedziczenia zostało utworzone zarezerwowane słowo `extends` oznaczające dziedziczenie dwóch klas.
- *AvailableQuestions* - opisuje zaimbek pytający, na który może odpowiedzieć dany atrybut. Jest on połączony z węzłem *AvailableAttribute* relacją `answers` skierowaną od atrybutu. Przykładowo jeżeli istnieje atrybut „rok założenia”, to można go połączyć z *AvailableQuestions* o wartości „kiedy?”. Pytanie można traktować jako szczególny rodzaj klasy.

Dostarczono także metody zliczające liczbę modeli, liczbę połączeń, rekordów i pustych modeli w bazie danych.

Rysowanie modelu ontologii

W celu wizualizacji istniejącego modelu ontologii zaprogramowano dodatkowy panel, który pobiera z Neo4j istniejące węzły metadanych opatrzone etykietami *AvailableClass* oraz *AvailableRelationship* a następnie rysuje graf przedstawiający relacje pomiędzy klasami. Daje to pogląd na strukturę istniejącą w aplikacji.



Rysunek 5.6: Ilustracja przedstawia część interfejsu odpowiedzialną za podgląd kształtu ontologii.

Do pobrania danych grafu zastosowano następujące zapytanie Cypher zawierające trzykrotną agregację:

```
MATCH (m: AvailableClass) <- [reltype: AvailableRelationship] -(n: AvailableClass)
RETURN id(m) as id_class, reltype.name as reltype,
collect(m.name) as rawclass_m,
collect(n.name) as rawclass_n,
collect(id(n)) as ids_col
```

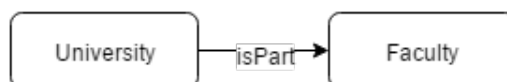
Zapytanie to pobiera wszystkie trójki klasa-relacja-klasa i zwraca je w formie agregatów prostych do narysowania w bibliotece D3.js jako grafy. Do narysowania modelu ontologii wykorzystano bibliotekę Data-Driven Documents (D3.js) [37] do języka JavaScript, która udostępnia szeroki zbiór funkcji pozwalających na wizualizację danych w postaci wykresów, grafów i map.

5.2. Wyszukiwanie

Zaimplementowano mechanizm przeszukiwania sieci semantycznej biorąc pod uwagę kilka możliwości sformułowania zapytań przez użytkownika. Standard Semantic Web oczekuje łatwości znalezienia każdej informacji. W związku z tym założono, że każda wprowadzona informacja może być wyszukana. Autor przy tym chciałby pokreślić, że mechanizm wyszukiwania działa niezależnie od wprowadzonej ontologii.

Większość współczesnych aplikacji już na etapie specyfikacji produktu definiuje, które informacje w bazie mogą być wyszukiwane. W momencie projektowania zakłada się, że będą istniały funkcje wyszukujące po poszczególnych kolumnach tabel. Programiści muszą osobno tworzyć każdy z przewidzianych raportów. Program dostarczony do tej pracy już z założenia przeszukuje całą wiedzę ontologii. Założenie takie powoduje, że model wiedzy jest elastyczny na zmiany. W normalnej sytuacji (tj. przy użyciu RDB) rozszerzenie modelu danych wymagałoby zmiany struktury bazy danych i kodu źródłowego, a w przypadku sieci opartej o graf, taka modyfikacja może odbyć się już z poziomu aplikacji. Pozwala to na tworzenie aplikacji bazodanowej, której przeznaczenie może ulec zmianie bez udziału programisty.

Zaimplementowano kilka możliwych zapytań do ontologii. Aplikacja analizuje słowo kluczowe i dokonuje skanu aby znaleźć informacje związane z nim. Dla wizualizacji założymy, że przeszukujemy ontologię PACSO. Możliwe warianty wyszukiwania dla danych z dla trójki:



Rysunek 5.7: Przykład trójki

- klasa - zwróci wszystkie instancje z danej klasy np. „University”,
- atrybut - „Akademia Górniczo-Hutnicza im. Stanisława Staszica” zwróci węzły, które mają którykolwiek atrybut o takiej wartości,
- relacja-klasa - „University isPart” - zwróci trójki, w których występuje predykat „isPart”,
- klasa-klasa - „University Faculty” - zwróci trójki złożone z takich klas wraz z połączeniami między nimi,
- atrybut-relacja - „Akademia Górniczo-Hutnicza im. Stanisława Staszica isPart” - zwróci trójki na podstawie atrybutu podmiotu i orzeczenia, opisującą które wydziały należą do Akademii Górniczo-Hutniczej,
- relacja - atrybut - „isPart Wydział Fizyki i Informatyki Stosowanej” - zwróci trójkę, odpowiadającą na pytanie, do której uczelni należy Wydział Fizyki i Informatyki Stosowanej. Wszystkie atrybuty instancji są brane pod uwagę, więc zakładając, że adresem Wydział Fizyki i Informatyki Stosowanej jest „Reymonta 19, 30-059 Kraków”, zapytanie do aplikacji „isPart Reymonta 19, 30-059 Kraków” zwróci to samo.
- klasa-relacja-klasa - zapytanie „University isPart Faculty” zwróci instancje wszystkich uczelni wyższych i należących do nich wydziałów
- atrybut-relacja-atrybut - zapytanie najbardziej precyzyjne. Jest to żądanie o trójki, w których odpowiednio podmiot i dopełnienie mają podane atrybuty i są one połączone znaną relacją.

Powstała funkcjonalność daje nam jeden z najważniejszych elementów sieci semantycznej - dostęp do dowolnej informacji przy użyciu ujednoliconego interfejsu.

5.3. REST API

Zaprojektowano interfejs zgodny z RESTful zwracający dokumenty typu JSON, który umożliwia podłączenie się do aplikacji dla zewnętrznych aplikacji. Wszystkie operacje na sieci semantycznej można wykonać przy pomocy otwartego interfejsu. Pozwala to na utworzenie systemu dla systemów zewnętrznych. Pełny opis dostępnych metod znajduje się w Dodatku A.

TUTAJ BEDA DIAGRAMY UML

- wprowadzono pełnego restful API jako załącznik
- struktura katalogów aplikacjis
- wyświetlenie istniejącego grafu jako rysunek
- plusy - prosta technologia łatwiej się zarządza
- minusy -

5.4. Diagram klas

5.5. Opis pełnego interfejsu REST i klas

5.6. Zaimplementowana Ontologia PACSO - opis co to jest

- Informacja o pracach magisterskich, które wykorzystałem
- Zrobić własne rysunki w draw.io

6. Zakończenie

6.1. Napotkane problemy i możliwości rozwinięcia funkcjonalności

Pierwszą z możliwości rozwinięcia aplikacji jest dodanie bardziej rozwiniętej strony, na której znajdują się rezultaty wyszukiwania zapytań. Obecnie istniejące rozwiązanie nie jest niestety dostatecznie czytelne.

Funkcjonalnością, która znacznie zwiększyłaby znaczenie projektu jest większa niż obecnie głębokość przeszukiwania sieci semantycznej. Mimo usilnych prób nie udało znaleźć się algorytmu, który brałby pod uwagę bardziej skomplikowane zapytania o głębokości trójek większej niż jeden. Siła grafowej bazy danych tkwi w algorytmach przeszukiwania, jednakże możliwość kombinacji pytań, które można utworzyć z połączenia kilku kilku klas i relacji, przerosła oczekiwania autora ze względu na duży poziom skomplikowania algorytmu. Problemy, które się pojawiły najprawdopodobniej są podobne do tych, z którymi zmagali się twórcy OWL Full.

Przełożenie mnogości zapytań z nawet bardzo prostego języka naturalnego na język formalny Cypher jest zadaniem wykraczającym poza zakres możliwości jednej osoby. Autorzy bazy danych Neo4j rozpoczęli pracę nad pluginem *Neo4j Sparql*, który umożliwi wykorzystanie zapytań języka SPARQL w tej bazie danych. Projekt ten jest we wczesnej fazie rozwoju i rzadko aktualizowany, ale zastosowanie języka SPARQL znacznie zwiększyłoby szansę na zastosowanie grafowych baz danych w Semantic Web.

6.2. Wyniki pracy

W ramach pracy magisterskiej zbadano możliwości wykorzystania grafowych baz danych przy budowie sieci semantycznych. Dokonano przeglądu literaturowego obecnie najpopularniejszych technologii stosowanych w dynamicznie rozwijającym się standardzie Semantic Web. Planem niniejszej pracy magisterskiej było zbadanie możliwości zastosowania grafowych baz danych w Semantic Web. Jako przykładową bazę danych wybrano Neo4j, ponieważ jest ona bardzo dobrze udokumentowana i najpopularniejsza wśród istniejących publikacji.

Planowano napisanie aplikacji, która umożliwi użytkownikowi przechowywanie dowolnych ontologii i przeszukiwanie ich na kilku poziomach głębokości, czyli szeregowi zależnych od siebie trójek. Cel udało się osiągnąć częściowo. Aplikacja może przechowywać skomplikowane ontologie, jednakże ich przeszukiwanie zostało ograniczone do bardziej elementarnego poziomu czyli pojedynczych trójek oraz ich składowych. Działanie aplikacji z sukcesem sprawdzono na ontologii PACSO, która w ubiegłych latach była jednym z obiektów badań w ramach innych prac magisterskich.

Struktura danych w Neo4j jest wystarczająca do realizacji oczekiwań Semantic Web, a otwarte interfejsy dostępu do danych ułatwiają tworzenie aplikacji otwartych dla innych systemów.

Autor stwierdza, że baza danych Neo4j ma potencjał aby przechowywać w niej dane ontologii, pod warunkiem że już w momencie implementacji projektu, ontologia ta będzie w pełni znana. Duża elastyczność modelu powoduje komplikację algorytmów wyszukiwujących. Znajomość kształtu ontologii w momencie tworzenia systemu pozwoliłaby na wykorzystanie unikalnej funkcjonalności języka Cypher, czyli przeszukiwania wzorcami. W rzeczywistych projektach model danych i oczekiwane przez klienta raporty są znane, zatem nic nie stoi na przeszkodzie tworzenia skomplikowanych zapytań.

7. Bibliografia

- [1] Mark Watson, Practical Semantic Web and Linked Data Applications, 2011
- [2] https://en.wikipedia.org/wiki/Semantic_Web
- [3] <http://www.w3.org/standards/semanticweb/>
- [4] Sareh Aghaei, Mohammad Ali Nematbakhsh, Hadi Khosravi Farsani, *Evolution of the World Wide Web: From Web 1.0 to Web 4.0*, International Journal of Web & Semantic Technology (IJWeST) Vol.3, No.1, 2012
- [5] https://en.wikipedia.org/wiki/Web_2.0
- [6] Scientific American, The Semantic Web, Maj 2001
- [7] Ryszard Tadeusiewicz, *Komputerowe Wspomaganie Decyzji*, XVI Konferencja Automatyków RYTRO 2012
- [8] https://en.wikipedia.org/wiki/Semantic_Web_Stack
- [9] Pierwotna wersja Semantic Web Stack, Timothy Berners-Lee, <http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html>
- [10] Internationalized Resource Identifiers (IRIs), RFC 3987, <https://tools.ietf.org/html/rfc3987>
- [11] Extensible Markup Language (XML) 1.0 W3C Recommendation, <http://www.w3.org/TR/REC-xml/>
- [12] XML Core Working Group Public Page, <http://www.w3.org/XML/Core/>
- [13] UTF-8, a transformation format of ISO 10646, RFC 3629 <https://tools.ietf.org/html/rfc3629>
- [14] Statystyki użycia UTF-8, http://w3techs.com/technologies/overview/character_encoding/all
- [15] Unicode 8.0, <http://unicode.org/versions/Unicode8.0.0/>
- [16] Internet Systems Consortium, obraz z raportu, <http://ftp.isc.org/www/survey/reports/2015/01/hosts.png>
- [17] RDF 1.1 Concepts and Abstract Syntax, <http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/>
- [18] Uniform Resource Identifiers (URI): Generic Syntax, RFC 2396, <https://tools.ietf.org/html/rfc2396>
- [19] C. Gutierrez, C. Hurtadouc, A. O. Mendelzonut, *Foundations of Semantic Web Databases*, 2009
- [20] RDF Schema 1.1 W3C Recommendation, <http://www.w3.org/TR/rdf-schema/>
- [21] Vuk Milićić, *Problems of the RDF syntax*, <http://milicicvuk.com/blog/2011/07/21/problems-of-the-rdf-syntax/>, 2011

-
- [22] SPARQL 1.1 Overview, W3C Recommendation, <http://www.w3.org/TR/sparql11-overview/>
 - [23] <https://en.wikipedia.org/wiki/Ontology>
 - [24] N. F. Noy, D. L. McGuinness, *Ontology Development 101: A Guide to Creating Your First Ontology*, http://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html
 - [25] Framework Twitter Bootstrap, <http://getbootstrap.com/>
 - [26] Biblioteka jQuery, <http://jquery.com/>
 - [27] Michael Jakl, *REST - Representational State Transfer*, <https://blog.interlinked.org/static/files/rest.pdf>
 - [28] Roy Thomas Fielding, *Architectural Styles and the Design of Network-based Software Architectures*, <https://www.ics.uci.edu/~fielding/pubs/dissertation/conclusions.htm>
 - [29] Model–view–controller, <https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller>
 - [30] XMLHttpRequest, W3C Working Draft, <http://www.w3.org/TR/XMLHttpRequest/>
 - [31] MSDN, XMLHttpRequest object, [https://msdn.microsoft.com/en-us/library/ms535874\(VS.85\).aspx](https://msdn.microsoft.com/en-us/library/ms535874(VS.85).aspx)
 - [32] AJAX Tutorial, <http://www.w3schools.com/ajax/>
 - [33] Document Object Model (DOM), <http://www.w3.org/DOM/>
 - [34] The application/json Media Type for JavaScript Object Notation (JSON), RFC 4627, <https://tools.ietf.org/html/rfc4627>
 - [35] OWL Web Ontology Language Semantics and Abstract Syntax, W3C Recommendation, <http://www.w3.org/TR/owl-semantics/>
 - [36] Wtyczka Dynatable <http://www.dynatable.com/>
 - [37] Data-Driven Documents, <http://d3js.org/>
 - [38] Statystyki użycia technologii <http://w3techs.com/>
 - [39] Statystyki użycia technologii związanych z językiem Java <http://zeroturnaround.com/rebellabs/top-4-java-web-frameworks-revealed-real-life-usage-data-of-spring-mvc-vaadin-gwt-and-jsf/>
 - [40] Spring Framework <http://projects.spring.io/spring-framework/>
 - [41] Spring Boot <http://projects.spring.io/spring-boot/>
 - [42] Spring Tool Suite <https://spring.io/tools/sts>
 - [43] Maven <https://maven.apache.org/>
 - [44] JDBC <http://www.oracle.com/technetwork/java/overview-141217.html>
 - [45] M. A. Rodriguez, P. Neubauer *The Graph Traversal Pattern*, 2010 <http://arxiv.org/abs/1004.1001>
 - [46] Blog Erica Evansa, http://blog.sym-link.com/2009/05/12/nosql_2009.html
 - [47] I. Robinson, J. Webber, E. Eifrem, *Graph Databases*, O'Reilly, 2013
 - [48] Martin Fowler – definicja NoSQL, <http://martinfowler.com/bliki/NosqlDefinition.html>
-

- [49] Pojemność bazy Neo4j, <http://neo4j.com/docs/stable/capabilities-capacity.html>
- [50] A. B. M. Moniruzzaman, S. A. Hossain *NoSQL Database: New Era of Databases for Big data Analytics - Classification, Characteristics and Comparison* (International Journal of Database Theory and Application Vol. 6, No. 4. 2013) , 2013
<http://arxiv.org/ftp/arxiv/papers/1307/1307.0191.pdf> [Dostęp Styczeń 2014]
- [51] N. Leavitt, *Will NoSQL Databases Live Up to Their Promise?* IEEE Computer 43(2):12-14 (2010), <http://www.leavcom.com/pdf/NoSQL.pdf> [Dostęp Listopad 2013]
- [52] Ranking baz danych, <http://db-engines.com/en/ranking/graph+dbms>
- [53] Dokumentacja Apache HBase, <http://hbase.apache.org/book/datamodel.html>
- [54] Użycie Cassandra w P-BEAST, <https://cdsweb.cern.ch/record/1432912>
- [55] Ra Soussi M.-A. Aufaure, H. Baazaoui *Graph Database for collaborative Communities*, 2011
- [56] D. De Abreu, A. Flores, G. Palma, V. Pestana, J. Piñero, J. Queipo, J. Sanchez, M. Vidal. *Choosing Between Graph Databases and RDF Engines for Consuming and Mining Linked Data*, 2004
- [57] Justin J. Miller, *Graph Database Applications and Concepts with Neo4j*, Southern Association for Information Systems Conference, Atlanta, GA, USA, 2013
- [58] C. Vicknair, M. Macias, Z. Zhao i inni *A Comparison of a Graph Database and a Relational Database. A Data Provenance Perspective*, 2010
- [59] Dokumentacja Neo4j, <http://docs.neo4j.org/>
- [60] R. J. Wilson, *Wprowadzenie do teorii grafów*, Wydawnictwo Naukowe PWN, Warszawa, 2000
- [61] Baza danych Titan, <http://thinkaurelius.github.io/titan/>

Spis rysunków

2.1	Semantic Web Stack. Został on opracowany w oparciu o ilustrację autorstwa Tima Berners-Lee [9] oraz rozszerzenie ze ten pochodzi ze strony Wikipedia [8].	13
2.2	Przykład Uniform Resource Identifier	14
2.3	Liczba hostów na świecie w skali logarytmicznej. Rysunek pochodzi z witryny Internet Systems Consortium. [16]	17
2.4	Przykład trójki	18
2.5	RDF dla zasobu bluza	18
2.6	Zrzut ekranu aplikacji GeoNames dla wyszukiwania instancji związanych z Krakowem. http://www.geonames.org/search.html?q=Krakow	23
2.7	Zrzut ekranu DBPedia dla węzła dotyczącego Polski: http://dbpedia.org/page/Poland	23
3.1	Architektura REST	25
3.2	Schemat modelu MVC	26

3.3	Przykład użycia JSON w REST. Pod odpowiedni URI zostaje wysłane żądanie metodą POST z żądaniem o uzyskaniem temperatury w Krakowie w stopniach Celsjusza. Serwer dopasowuje żądanie do dostępnych usług i w odpowiedzi zwraca dokument JSON zawierający oczekiwane informacje. Taką funkcjonalność można zaimplementować w ramach technologii AJAX, co spowoduje, że aplikacja (np. widget na urządzeniu mobilnym) będzie miała systematycznie odświeżaną temperaturę adekwatną do aktualnej lokalizacji.	30
4.1	Przykładowa relacja	35
4.2	Popularność baz NoSQL wg. serwisu DB-Engines[52]. Pierwsza baza grafowa znajduje się dopiero na 22. miejscu.	36
4.3	Porównanie modeli NoSQL oraz relacyjnego. Opracowane w oparciu o [58] i [59].	38
4.4	Popularność bazy Neo4j wg serwisu DB-engines w porównaniu do trzech najpopularniejszych baz relacyjnych i MongoDB [52]. Popularność bazy Neo4j stopniowo wzrasta. Wykres jest w skali logarytmicznej.	39
4.5	Graf skierowany	40
4.6	Popularność bazy Neo4j wg serwisu DB-Engines w porównaniu do innych baz grafowych. Wykres jest w skali logarytmicznej.	42
5.1	Proces dodawania nowej klasy.	50
5.2	Proces dodawania instancji klasy do aplikacji.	51
5.3	Proces dodawania instancji do aplikacji.	52
5.4	Rysunek przedstawia poglądowo pojęcia ontologii i ich odpowiedniki z grafowej bazy danych.	52
5.5	Interfejs graficzny łączenia instancji	53
5.6	Ilustracja przedstawia część interfejsu odpowiedzialną za podgląd kształtu ontologii.	54
5.7	Przykład trójki	54

Część I

Dodatek A

7.1. Szczegóły implementacyjne

Jako metodę połączenia z bazą danych Neo4j wybrano JDBC przy użyciu interfejsu `JdbcTemplate` dostępnego w ramach frameworka Spring. Jest to dodatkowa warstwa abstrakcji upraszczająca do minimum ilość niezbędnego kodu.

Poniżej znajduje się przykładowa metoda modelu wykonująca zapytanie Cypher, które zwraca dostępne klasy (z węzłów metadanych) wraz z ich atrybutami, a następnie dokonuje rzutowania wyników do listy obiektów typu `AvailableClass`.

```
private static final String LIST_OF_AVAILABLE_CLASSES =
    "MATCH (class:AvailableClass) RETURN class.name as name,
    class.code as code, class.attr as attributes;";

public List<AvailableClass> getAvailableClasses(){
    JdbcTemplate jdbcTemplate =
        (JdbcTemplate) context.getBean("jdbcTemplate");
    return jdbcTemplate.query(LIST_OF_AVAILABLE_CLASSES,
        new BeanPropertyRowMapper<>(AvailableClass.class));
}
```

7.2. Zawartość dołączonego do pracy nośnika

Nośnik dodany do pracy zawiera:

- Katalog lib - katalog z programami wymaganymi do uruchomien
 - Program Maven 3.3.3 dla systemu Windows
 - Baza danych Neo4j w wersji 2.1.5
- Katalog app:
 - Kod źródłowy programu wraz z plikami projektu Spring Tool Suite
- Katalog example
 - Przykładowa baza danych do programu

7.2.1. Struktura katalogów aplikacji

- src - katalog z projektem aplikacji
- lib - katalog z programami wymaganymi do uruchomien

7.3. Instrukcja wdrożeniowa

Poniżej przedstawiono kroki niezbędne do prawidłowej instalacji aplikacji. Wymanane pliki znajdują się na dołączonym do pracy nośniku:

1. Wymagane jest zainstalowanie następujących aplikacji:
 - (a) środowiska uruchomieniowego JRE dla języka Java 1.7
 - (b) bazy danych Neo4j (wymagana wersja 2.1.x lub wyższa),

- (c) programu Maven w wersji 3.0.x
- 2. Wypakowanie kodu źródłowego aplikacji do miejsca docelowego,
- 3. Uruchomienie pliku wykonywalnego zainstalowej bazy Neo4j,
- 4. (opcjonalnie) Import przykładowej ontologii korzystając z części DATABASE LOCATION w programie Neo4j
- 5. Wciśnięcie „Start” w programie Neo4j
- 6. Wpisanie komendy `mvn spring-boot:run`

7.4. Interfejs REST dla Semantic Web