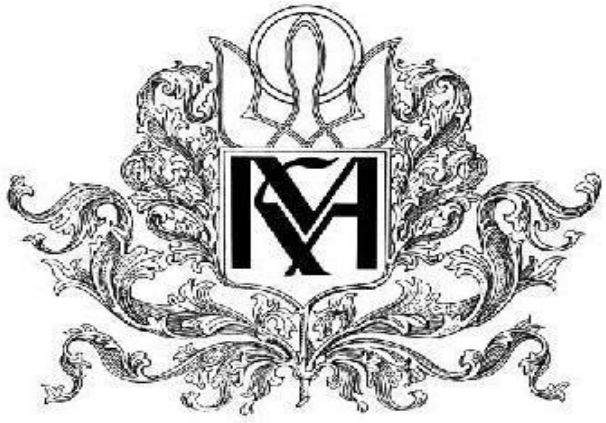
Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики



КУРСОВА РОБОТА

Галузь знань: 12 «Інформаційні технології»

Спеціальність: 122 «Комп’ютерні науки та інформаційні технології»

на тему:

«Реалізація бази знань за допомогою системи Protégé»

Науковий керівник:
к.ф-м.н., доц. Жежерун О. П.
Виконала студентка: Дідко В.С.

Київ 2020

Зміст

Анотація

У даній роботі досліджується проблема неструктурованості наявної у просторах інтернету інформації про породи собак та пропонується рішення у вигляді створення онтології за допомогою системи Protégé. Описується процес проектування та розробки бази знань, виконання SPARQL запитів до розробленої онтології та створений веб-сайт з її використанням.

Вступ

Люди, організації та програмні системи змушені спілкуватися один з одним. Однак спосіб вираження знань про ті чи інші речі на одній і тій самій мові може настільки різнитися, що це призводить до непорозумінь. Це, у свою чергу, веде до труднощів обміну інформацією між людьми, організаціями і програмами, зокрема у формуванні однозначно визначених вимог і специфікацій для складних систем. Незважаючи на достатньо високий рівень розвитку систем моделювання, можливості взаємодії створених з їхньою допомогою програмних моделей, а також повторне використання і поширення цих моделей є доволі обмеженим. Уникнути цього можна за допомогою повного усунення чи хоча б мінімізації концептуальної та термінологічної плутанини та встановлення однозначного розуміння мови, що використовується для формування вимог та специфікацій складних систем.

Так званий «онтологічний підхід» як раз таки забезпечує гнучке моделювання та інтероперабельність даних, стек семантичних технологій, що дозволяє виконувати аналіз неструктурованої інформації та інтелектуальний пошук даних серед безлічі різноманітних джерел, а також машинне навчання, що забезпечує аналіз та класифікацію даних, у тому числі в умовах неповної інформації.

Метою курсової роботи є створення тематичної бази знань з подальшим її використанням онлайн-ресурсами. Для досягнення поставленої мети необхідно вирішити наступні задачі:

1) Спроектувати онтологію;
2) Ознайомитися з фреймворком Protégé ;
3) Дослідити методи інтеграції онтології в онлайн-ресурси;

Об’єктом дослідження є процес створення бази знань порід собак, яка б відповідала поставленим цілями роботи. Предметом дослідження є база знань. Досягнення поставленої мети реалізовано за допомогою використання програмного засобу Protégé. Для проектування бази знань та заповнення її даними використовується мова запитів SPARQL. Для відображення результатів пошуків створено веб-застосунок за допомогою засобів Python-фреймворку Flask.

Робота містить чотири розділи:

Перший розділ присвячений опису основних понять онтології та особливостей розробки баз знань.

У другому розділі описується система Protegé, що використовуються для розробки бази знань, необхідні кроки при підготовці до розробки онтології та сама розробка.

У третьому розділі надається інструкція з виконання запитів SPARQL до розробленої бази знань.

Четвертий розділ присвячено розробці онлайн-ресурсу з використанням створеної онтології.

Розділ 1. Дослідження та аналіз предметної області

1.1 Загальна інформація

Вперше термін «онтологія» був використаний в інформатиці Томасом Грубером. У його роботі розглядалися різні аспекти взаємодії інтелектуальних систем між собою та з людиною. Під інтелектуальними системами маються на увазі програми, які моделюють інтелектуальну діяльність людини. Інтелектуальна програма містить знання про те, що необхідно робити у процесі виконання програми. Ці знання не є фіксованими і мають передаватися системі у вигляді даних. Відповідно виникає необхідність опису цих даних. Цей опис має бути у достатній мірі формальним та зрозумілим як програмі, так і людині. Грубер запропонував описувати знання у двох різних формах. Перша – канонічна, яка представляє собою опис знань на деякій конкретній мові, наприклад мові програмування. Друга – онтологічна, що представляє собою велику кількість класів знань, пов’язаних між собою відношенням узагальнення.

Отже, під онтологією наразі розуміється будь-який опис декларативних знань, створений за допомогою формальної мови та такий, що містить деяку класифікацію знань, що дозволяє людині сприймати ці знання. Необхідно дати визначення деяким термінам для повного розуміння матеріалу:

Онтологія – формалізоване представлення знань з деякої предметної області, що є придатним для автоматизованої обробки.

OWL – мова опису онтологій для семантичної павутини.

Таксономія – ієрархічна організація певного набору об’єктів.

SPARQL – мова запитів до семантичних баз знань.

1.2 Опис понять

1.2.1 Онтологія

Онтологія допомагає представити предметну область за допомогою візуальних та формальних методів. Онтологія використовує знання як окремих індивідуумів, так і груп людей для ефективного повторного використання знань. Основні сфери використання онтологій включають в себе: моделювання бізнес процесів, штучний інтелект, семантична павутина (Semantic Web). Найчастіше онтології використовуються:

1) Для взаємодії між програмними агентами та розробниками;
2) Для ефективного повторного використання знань в предметній області;
3) Для того, щоб зробити припущення в предметній області явними;
4) Для аналізу знань в предметній області;

Базові елементи онтологій включають:

1) індивіди (екземпляри) – є компонентами низького рівня;

2) класи – абстрактні колекції чи набори об’єктів;

3) атрибути – використовуються для опису об’єкті;

4) відношення – залежності між об’єктами.

Онтології можна класифікувати відповідно до двох вимірів: розмір та тип структури концептуалізації, а також предмет концептуалізації. Відповідно до першого виміру онтології поділяються на наступні типи:

Термінологічні онтології, такі як лексикони. Вони встановлюють терміни, які використовуються для представлення знань в області дискурсу. Прикладом подібної онтології є UMLS (Unified Medical Language System).

Інформаційні онтології, що визначають структуру записів баз даних. Схеми баз даних відносяться до інформаційних онтологій.

Онтології моделювання встановлюють концептуалізацію знань. На відміну від інформаційних, онтології моделювання наділяють знання більш детальною та повною структурою.

Прикладні онтології містять всі визначення, які є необхідними для моделювання знань, які вимагаються конкретним додатком.

Предметно-орієнтовані онтології напрямлені на формалізацію частини реального світу чи якоїсь конкретної предметної області

Загальні онтології використовуються для опису понять, що є спільними для багатьох галузей.

Репрезентативні онтології містять концептуалізацію, що стоїть за формалізмом репрезентативних знань. Цей тип онтологій є нейтральним по відношенню до сутностей світу. Це означає забезпечення репрезентативної платформи без вираження аксіом щодо світу.

Практична розробка онтології включає в себе наступні кроки:

1) Визначення класів та таксономій;
2) Визначення слотів та опис їх допустимих значень;
3) Додавання екземплярів;

При розробці онтології важливо розуміти, що не існує єдиного вірного способу моделювання предметної області – завжди є альтернативи. Розробка онтології є ітеративним процесом, адже в процесі структурування знань онтологія розширюється – з’являються все нові класи, слоти та зв’язки.

1.2.2 Класи та таксономії

Необхідно скласти глосарій термінів, які необхідно пояснити користувачеві. В додатку 1 визначається ієрархія понять, що використовуються в онтології. Новий клас описується наступним чином:

<owl: Class rdf:about=”className”/>,

де className – ім’я нового класу. Підклас оголошується так:

<owl: Class rdf:about= “subclassName”>

<rdfs:subClassOf rdf:resource= “className”/>

</owl:Class>,

де subclassName – це ім’я класу, що стоїть нижче в ієрархії, а className – ім’я батьківського класу. Таким чином, перше речення в синтаксисі OWL виглядає так:

<owl: Class rdf:about=“Dog”/>,

<owl: Class rdf:about=“Hound”>

<rdfs:subClassOf rdf:resource= “Dog”/>

</owl:Class>

<owl: Class rdf:about=“Cat”/>

</owl:Class>

Після створення класів Dog і Cat необхідно вказати, що вони є непересічними (disjoint), тобто один індивід не може бути представником одночасно двох класів. Всі класи, які на нашу думку не є підкласами в нашій онтології, в OWL автоматично стають підкласами вбудованого найзагальнішого класу, який називається Thing (Річ). Цей клас є класом усіх індивідів та суперкласом для усіх OWL класів. Також існує вбудований найбільш специфічний клас під назвою Nothing (Ніщо), який не має ніяких представників і являє собою підклас усіх OWL класів. На малюнку 1 представлена ієрархія, що вийшла. Тут і надалі представлені скріншоти з редактору онтологій Protégé.

1.2.3 Властивості та екземпляри

У Protégé є два види властивостей: властивості типів даних (Data Properties) зв’язують об’єкт зі значенням типів даних та властивості об’єкту (Object Properties), що виражають відношення між двома екземплярами. У Protégé існують наступні типи властивостей об’єкту:

1) *Транзитивні* – транзитивна властивість пов’язує екземпляр a з екземпляром *b*, а екземпляр *b* з екземпляром *c*. Тоді можна стверджувати, що індивід *a* пов'язаний з індивідом *c* цією властивістю.
2) *Рефлексивні* - властивість r називається рефлексивною, коли екземпляр a повинен бути пов'язаний сам з собою.
3) *Ірефлексивні* – якщо властивість r є ірефлексивною, то вона пов’язує екземпляр a та екземпляр b, де a та b мають обов’язково бути різні.
4) *Симетричні* – якщо властивість r симетрична, то вона пов’язує екземпляр a з екземпляром b, і тоді індивід b пов'язаний з індивідом a через ту саму властивість.
5) *Асиметричні* – якщо властивість r асиметрична, тоді вона пов’язує екземпляр a з екземпляром b, а індивід b не може бути пов'язаний з індивідом a через ту саму властивість.
6) *Функціональні* – якщо для даного екземпляра може існувати не більше одного екземпляру, який має відношення r до першого екземпляру через цю властивість, то властивість r є функціональною.
7) *Обернено-функціональні* – якщо властивість r є зворотною до функціональної властивості, то вона є обернено-функціональною.

Властивості типів даних стосуються у першу чергу представників класу – екземплярів. Властивості оголошуються за допомогою

owl:DatatypeProperty і згодом використовується як модифікатор для деякого класу чи для встановлення характеристик екземплярів.

Нижче наведено приклад оголошення властивості типів даних «опис_коду» і присвоєння екземпляру PAT(E) значення «Поліетилентерефталат»:

```
<owl:DataProperty rdf:ID="description" />
 <rdfs:domain rdf:resource="#Dog" />
<rdfs:range rdf:resource="&xsd;string"/>
</owl:DataProperty>
<Dog rdf:about="Pug">
<description>Smart and energetic</description>
</Dog>
```

Екземпляри представляють собою об’єкти в предметній області, яку ми описуємо. Важлива різниця між Protégé і OWL полягає в тому, що OWL не використовує припущення унікальне імен (UNA – unique name assumption). Це означає, що два різні імені фактично могли б посилатися на один і той самий екземпляр. Наприклад, «Принц Чарльз» та «Принц Уельський» можуть відноситися до однієї особи. У OWL необхідно явно вказувати, однакові ці екземпляри чи різні. В додатку 2 оголошується екземпляр об’єкту класу «Toy». Синтаксис OWL:

```
<className rdf:ID=“Pug”>
</className>,
```

де ind_name є назвою екземпляру. Тоді в нашому прикладі виходить:

```
<Toy rdf:ID=“Pug”>
</Toy>
```

1.2.4 Аксіоми

Аксіоми використовуються для запису висловлювань, які є завжди істинними. Їх використання може бути обумовлене різними цілями: перевірка коректності описаної в онтології інформації, накладення комплексних обмежень на значення атрибутів чи вивід нової інформації. Аксіоми виражають очевидні твердження, які пов’язують поняття та відношення. Під аксіомою можна розуміти твердження, що вводиться в онтологію в готовому вигляді, з якого можуть бути виведені інші твердження. Вони дозволяють виразити ту інформацію, яку не можна відобразити у онтології, шляхом побудови ієрархії понять і встановлення різних відношень між цими поняттями. Найпростіший для розуміння та найпопулярніший приклад аксіоми, який можна навести: «Якщо x смертний, то коли-небудь x помре». Прикладом числового обмеження є твердження, що для людини кількість біологічних батьків рівна двом. Кількість та ступінь деталізації аксіом найчастіше залежать від типу онтології. Як показано на рисунку 1, аксіоми в OWL можуть бути деклараціями, аксіомами про клас, аксіомами властивостей об’єктів чи даних, визначеннями типу даних, ключами, твердженнями та аксіомами про анотації.

Рис 1

1.2.5 Визначення домену, діапазону та кардинальності

Як і для класів, для властивостей можлива ієрархічна структуризація з виділенням «властивостей-підвластивостей». Вершиною ієрархіє слугує властивість TopObjectProperty. Формування звязків між класами відбувається шляхом вказування властивостей об’єкту. Для кожної об’єктної властивості можна у явному виді задати домен та діапазон. Домен представляє собою клас, екземпляри якого будуть пов’язуватись заданою властивістю, а діапазон - клас, з індивідами якого буде пов’язана задана властивість чи тип даних.

Відношення «частина-ціле»

Для визначення відношення «частина-ціле» необхідно ввести дві властивості об’єкту (ObjectProperty): bePartOf (бути частиною) та consistsOf (складатися з), а також оголосити їх оберненими один до одного:

<owl:ObjectProperty rdf:about=“consistsOf”/>

<owl:ObjectProperty rdf:about=“bePartOf”/>

<owl:inverseOf rdf:resource= “/consistsOf”/>

</owl:ObjectProperty>

Також у синтаксисі OWL передбачена можливість різноманітних обмежень на властивості. Вказання кількості елементів є кардинальністю. Обмеження кардинальності в OWL відомі як локальні обмеження, тому що вони поширюються на властивості кожного конкретного класу. Тобто ці обмеження задають кардинальність заданої властивості для представників даного класу. minCardinality є вбудованою властивістю OWL, яке показує, що будь-який представник заданого класу має бути зв’язаний цією властивістю щонайменше с вказаною кількістю екземплярів, а maxCardinality у свою чергу вказує, що будь-який представник даного класу буде пов'язаний цією властивістю не більше ніж з вказаною кількістю екземплярів. Третій тип кардинальності – qualifiedCardinality задає чітку кількість екземплярів, з якою кожен представник заданого класу має бути зв’язаний.

Для задання описаних обмежень властивостям використовується синтаксис:

```
<owl:Class rdf:about= “className1”>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource= “propertyName”/>
            <owl:onClass rdf:resource= “className2”/>
            <owl: cardinalityType rdf:datatype= “&xsd;
nonNegativeInteger”>num</owl:cardinalityType>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>,
```

де className1 – ім’я класу, на який накладаються обмеження,

propertyName – назва властивості,

num – кількість індивідів класу,

cardinalityType – тип обмеження: minCardinality, maxCardinality, qualifiedCardinality.

Тоді слідуючи заданому синтаксисі отримаємо наступний результат:

```
<owl:Class rdf:about= “Dog”>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource= “eats”/>
            <owl:onClass rdf:resource= “Nutrition”/>
            <owl: minCardinality rdf:datatype= “&xsd;
nonNegativeInteger”>1</owl:cardinalityType>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

Розділ 2. Побудова OWL онтології

2.1 Огляд системи Protégé

2.1.1 Загальний опис

Створення онтології – це трудомісткий та складний процес. Для його полегшення у 90-х роках почали створювати перші середовища для процесів розробки онтологій. На даний момент існує великий спектр інструментів, які окрім загальних функцій редагування та перегляду, виконують підтримку документації онтологій, їх імпорт та експорт онтологій різних мов та форматів, підтримку графічного моделювання та управління бібліотеками. Загалом, для

роботи з мовами онтологій існує декілька видів технологій: редактори онтологій (для створення онтологій), DBMS онтологій (для зберігання й звертання до онтології) і сховища онтологій (для роботи з декількома онтологіями). Основна функція редакторів онтологій – це підтримка процесу формалізації знань та представлення онтології як специфікації. Серед найпопулярніших редакторів можна виділити OntoEdit, WebOnto, DOE, Protégé.

Для розробки онтології побутових відходів було обрано середовище Protégé у зв'язку з тим, що цей редактор відповідає стандартам W3C, має як десктоп- так і веб-версію, підтримує найбільшу кількість форматів та мов представлення знань (XOL, DAML+OIL, RDF/RDFS, OWL) у порівнянні з іншими названими вище редакторами, підтримує великий спектр модулів розширення функціональності (plug-ins), надає зручний у використанні графічний інтерфейс користувача. Більш того, Protégé уможливлює використання механізму міркувань (Reasoner), за допомогою якого можна перевіряти, чи є визначення і твердження в онтології взаємно несуперечливими і розпізнавати, чи відповідають визначення певним поняттям. Такий механізм підтримує коректність ієрархії онтології, що є дуже важливим. Ще однією значущою перевагою редактора є підтримка будь-якої бази даних з драйвером JDBC 1.0, що уможливлює підтримку більшості реляційних баз даних (Access, SQL Server, Oracle, MySQL). Інтерфейс користувача містить головне меню і декілька вкладок для редагування різних частин бази знань та її структури. Інтерфейс є доволі простим та інтуїтивним. Навіть користувачі, у яких немає знань з проектування онтологій, легко зрозуміють структуру редактора. Назви вкладок редактору так їхній набір можна легко змінювати. Онтології, створені за допомогою редактору Protégé можуть бути використані як в простий, так і в складних додатках, що базуються на онтології. Онтології можуть бути експортовані в безліч форматів, наприклад OWL, RDF (RDFS – RDF Schema) чи XML Schema.

Protégé доступний для безкоштовного завантаження з офіційного сайту разом з плагінами та онтологіями. Редактор має підтримку значної спільноти, що включає розробників та науковців, корпоративних та урядових користувачів, які використовують його для вирішення задач, пов'язаних зі знаннями, в таких областях, як збір знань, біомедицина та корпоративне моделювання.

2.1.2 Порівняння з аналогами

Інші відомі системи для роботи з онтологіями – NeOn Toolkit, OntoStudio. NeOn Toolkit підтримує специфікацію OWL 2 та надає NeOn OWL Editor для розробки та підтримку онтологій у форматі OWL. Редактор має три компоненти: Ontology Navigator (навігатор по онтології), Individual Panel (панель екземплярів) та Entity Properties (властивості сутностей). З використанням цих трьох панелей користувач може виконувати основні завдання, такі як визначення та редагування класів, властивостей та їх зв'язків в заданій онтології. Також є можливість перевіряти на узгодженість елементи онтології та візуалізувати онтологію. NeOn Toolkit має у своєму арсеналі 45 розширень функціональності (плагінів) доступних для завантаження. Серед недоліків цього програмного засобу – останній реліз був у 2011 році, в той час як Protégé продовжує розвиватися та розширювати свій функціонал на постійній основі.

OntoStudio – це професійне середовище для роботи з онтологіями, що поєднує інструменти моделювання онтологій з компонентами для інтеграції різноманітних джерел даних. Завдяки своєму модульному дизайну, функціонал OntoStudio можна збагатити самостійно розробленими модулями та налаштувати середовище під свої потреби. OntoStudio підтримує стандарти W3C OWL, RDF і RDFS, і F-Logic для логічної обробки правил. Крім того, OntoStudio надає велику кількість конекторів баз даних,

документів, файлових систем та додатків. OntoStudio доступна для безкоштовного завантаження з офіційного сайту.

Серед трьох найпоширеніших інструментів роботи з онтологіями Protégé виділяється найбільш багатим функціоналом, кількість стандартів, що підтримуються та інтуїтивним інтерфейсом. Ще однією перевагою є наявність двох версій – веб- та десктопна. Саме тому ця система була обрана для роботи з онтологією порід собак.

2.2 Визначення масштабу онтології та підходу до розробки

Існує декілька основних підходів до розробки онтології:

1) *Висхідний підхід* розробки полягає в тому, що спочатку визначаються низькорівневі конкретні класи, а потім відбувається групування цих класів в більш загальні поняття. Наприклад, спочатку визначаємо клас Guard dog, потім визначаємо новий клас Dog та робимо клас Guard Dog його підкласом.
2) *Низхідна* розробка починається з визначення найбільш загальних понять з поступовим переходом до більш конкретних понять. Наприклад, можемо створити спочатку клас Dog, потім визначати підкласти класу Dog: Guard Dog, Toy Dog, Sporting Dog, Terrier Dog.
3) *Комбінований підхід* розробки є поєднанням висхідного та низхідного підходів. Спочатку визначаються найбільш очевидні, легкі для розуміння поняття, а потім виконується їх узагальнення чи конкретизація.

Для розробки онтології було обрано низхідний підхід, тому що він мені здався найбільш інтуїтивним.

Область онтології – породи собак. Онтологія буде використовуватися для пошуку за різними характеристиками порід собак та пропонувати породи, які найбільш підходить до параметрів пошуку користувача. Один зі способів визначення масштабу онтології – визначити ряд питань, на які повинна відповідати онтологія. Необхідно визначити чи містить база знань достатню кількість інформації для відповіді на ці питання чи потрібен якийсь особливий рівень деталізації предметної області.

Було визначено наступний список питань:

1. Які дані має містити онтологія про собак?
2. Які проблеми вирішує онтологія?
3. Що є об’єктом та предметом дослідження?

Було вирішено зупинитися невеликому масштабі онтології і зробити її якомога більш деталізованою.

2.3 Опис основних компонент онтології

Основні класи онтології включать: Dog – підклас класу Thing, класи Herding, Hound, Sporting, Non-Sporting, Terrier, Toy та Working є підкласами класу Dog. Також є клас Nutrition з підкласами Dry та Wet та клас Care з підкласами Vaccination, Training, Grooming.

Об’єктна властивість needs пов’язує екземплярів класу Dog та Care. Доменом цієї властивості є клас Dog, діапазоном є клас Care. Властивість eats у свою чергу пов’язує представників класу Dog та Nutrition.

Була створена велика кількість властивостей даних:

Назва властивості	Опис	Тип даних
description	опис породи собак	string

energy	рівень енергійності породи	string
good_for_apartment	чи є порода підходящою для квартир	boolean
good_for_family	чи є порода сімейною собакою	boolean
good_for_guarding	чи є порода сторожевою	boolean
good_with_children	чи є порода доброзичливою до дітей.	string
good_with_dogs	доброзичливість до інших собак	string
hypoallergenic	гіпоалергенність	boolean
image_url	фото	string
life_expectancy	тривалість життя	integer
shedding	линяння	string
size	розмір	string
intelligence	розум	string
trainability	схильність до дресування	string
weight	вага	string
personality	характер	string

Було створено 183 екземпляри порід собак. Загальна кількість аксіом онтології – 3546.

2.4 Перевірка онтології на узгодженість

Під час проектування будь-якої онтології важливу роль відіграє питання несуперечності даних. Для перевірки OWL-онтології на узгодженість використовується reasoner (логічна машина виводу). Базуючись на описі класу різонер робить заключення, чи може даний клас містити якихось індивідів. У випадку, якщо це не є можливим, клас вважається неузгодженим чи суперечливим. Під схожу перевірку підпадають і екземпляри класів для вказання їх суперечностей по відношенню до опису класу. Більш того, логічна машина виводу використовується і для автоматизації побудови ієрархії класів. Різонер може добудовувати ієрархію, автоматично виявляючи неописані зв’язки типу «клас-підклас». Це називається класифікаційним тестом. В результаті відпрацювання такого тесту отримаємо виведену (inferred) ієрархію класів. Далі наведений список функцій логічної машини виводу:

Проведення класифікації та відображення виведеної ієрархії класів;

Перевірка узгодженості і визначення невідповідностей;

Визначення типів індивідів, їх належність до певного класу;

Рекомендовано використовувати логічну машину виводу регулярно – як при додаванні нових аксіом, так і при заповненні онтології екземплярами. У OWL різонери працюють на базі концепції Open World Reasoning – принцип відкритості світу, на противагу концепції баз даних. Наразі у Protégé підтримуються такі логічні машини виводу: CEL, Chainsaw, ELK, FaCT++, fuzzyDL, HermiT, jcel, MORe, ontop та інші. Усі вони мають відкритий код. Алгоритми цих систем ґрунтуються на різноманітних варіаціях табличного методу, що є дуже ефективним. Даний метод уможливлює перевірку існування моделі виконуваного класу шляхом побудови таблиці, що

представляє собою розмічений граф, що задовольняє набір обмежень, які описують семантику конкретної логіки. Модель існує тільки у разі існування таблиці, які задовольняє обмеження. Для того, щоб запустити вбудовану в Protégé логічну машину HermiT потрібно перейти в меню Reasoner та обрати необхідний різонер HermiT 1.4.3.456 (див. Рис. 2).

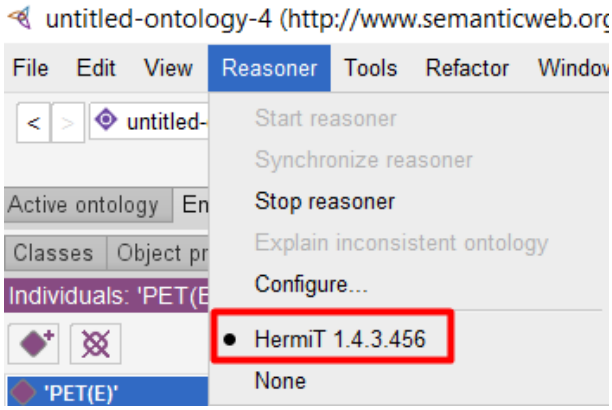


Рис 2

Після цього у тому ж меню слід запустити різонер (Start Reasoner) (див. Рис. 2), це автоматично запустить класифікацію, після якої з'явиться додаткова вкладка з відображенням виведеної ієрархії класів. Суперечливі класи позначені червоним кольором під класом Ніщо (Nothing), а усі інші під їхніми суперкласами (див. Рис.3).

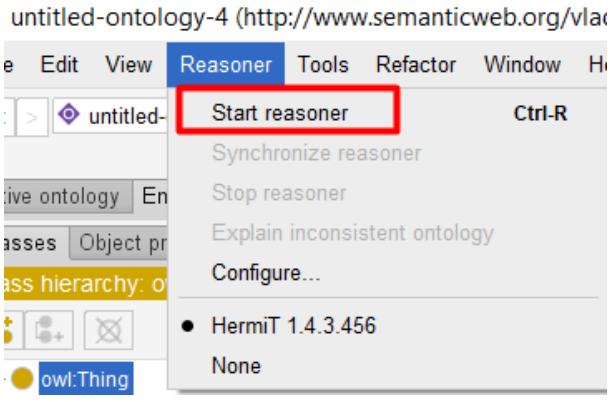


Рис 3

2.5 Побудова графу за допомогою плагіну OntoGraph

Плагін OntoGraph надає підтримку для інтерактивної навігації по OWL онтології. Для автоматичного впорядкування структури онтології підтримуються різні схеми відображення. На графі можна відобразити класи, підкласи, властивості об’єктів з доменом, діапазоном та еквівалентністю. Відносини та типи вузлів можна відфільтровати, що створити необхідний зовнішній вигляд графу. На рисунку 4 зображений граф онтології «Породи собак».

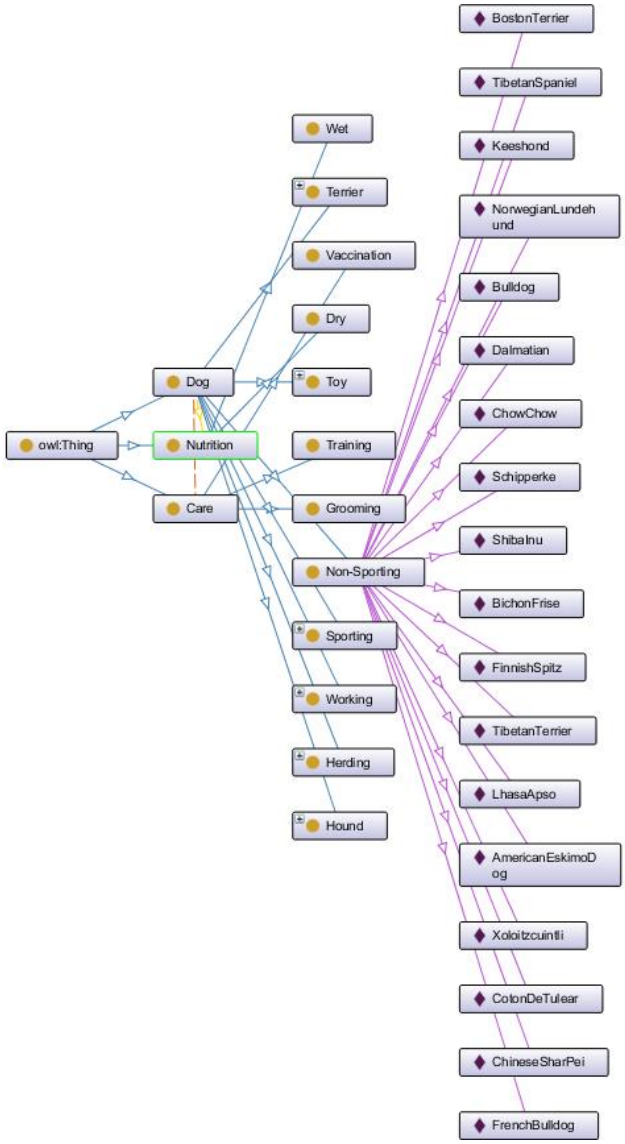


Рис 4

Розділ 3. Інструкція по виконанню SPARQL запитів до бази знань

3.1. Мова запитів SPARQL

SPARQL (SPARQL Protocol and RDF Query Language) – це мова запитів до даних, що є представлені у вигляді моделі RDF, а також протокол для передачі цих запитів та отримання відповідей. Є однією з технологій семантичної павутини та рекомендованим консорціумом W3C. У SPARQL не накладаються обмеження на формат даних, що уможливлює взаємодію між ресурсами різного типу. На рис.1 зображена узагальнена схема SPARQL запитів :

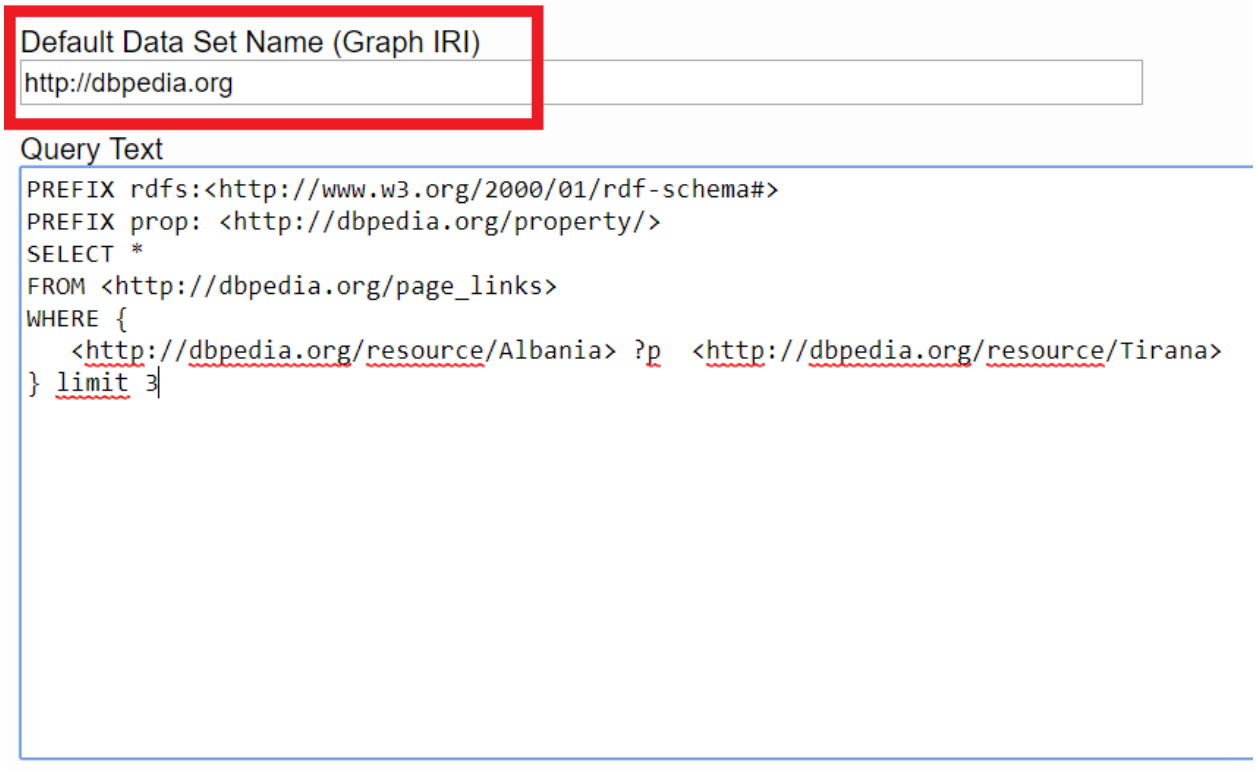


Рис 5

У даному прикладі PREFIX – це префіксні оголошення, які використовуються для того, щоб скоротити URI.

SELECT перераховує змінні, які необхідно повернути. Після службового слова FROM вказується джерело. У даному прикладі - http://dbpedia.org/page_links. LIMIT є модифікатором запиту для обмеження кількості повернених значень запиту. Інші службові зв'язки включать в себе:

OPTIONAL – позначає необов'язковий шаблон.

OFFSET – прибирає з результату перші n значень.

ORDER BY – сортує результат по якомусь параметру у спадаючому (DESC) чи зростаючому (ASC) порядку.
DISTINCT – забезпечує унікальність виведених значень, уникає дублікати.

GRAPH – застосовує шаблон до іменованих графів.

Після WHERE вказуються бажані критерії запиту, переважно у вигляді триплетів. Триплет можна читати як речення з суб'єктом, предикатом і об'єктом. Приклад триплету:

```
SELECT ?x ?y
WHERE
{
?x rdfs:subClassOf ?y.
}
```

Даний простий запит виведе усі класи та підкласи онтології.

3.2. Виконання запитів до розробленої онтології

Для виконання запитів було використано SPARQL сервер Fuseki. Даний сервер забезпечує оновлення REST-стилю SPARQL HTTP, виконання запитів SPARQL та оновлення SPARQL, використовуючи протокол SPARQL через HTTP. Підтримує наступні стандарти:

- SPARQL 1.1 Protocol
- SPARQL 1.1 Graph Store HTTP Protocol
- SPARQL 1.1 Query
- SPARQL 1.1 Update

Для того, щоб запустити сервер та задеплоїти існуючу онтологію, необхідно виконати наступну команду у командній стрічці:

fuseki-server --loc=data --update /your-dataset,

де location (loc) - це місце зберігання RDF-даних, цей параметр може бути налаштований на будь-який інший шлях. Частина “/your-dataset»” - це ім'я набору даних, який можна перейменувати за необхідності. За замовчуванням сервер Fuseki використовує порт 3030. Далі необхідно натиснути на кнопку Query на головній сторінці, щоб перейти до сторінки написання запитів:

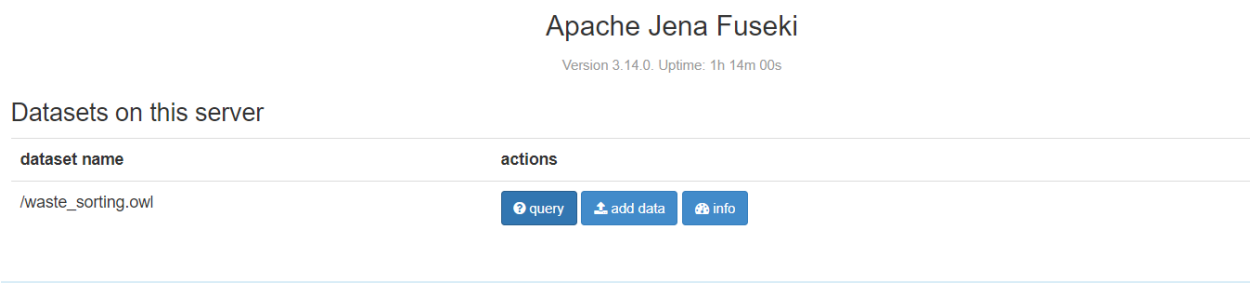


Рис 6

Перший запит виводить всі породи собак, які відносяться до малих:

```
SELECT DISTINCT ?x ?y WHERE {
    ?x dog:size ?y .
  FILTER regex(?y, 'small')

    }
```

Рис 7. Код запиту 1.

	x	y
1	dog:Affenpinscher	"small"
2	dog:AustralianTerrier	"small"
3	dog:Beagle	"small"
4	dog:BichonFrise	"small"
5	dog:BorderTerrier	"small"
6	dog:BostonTerrier	"small"
7	dog:BrusselsGriffon	"small"

Рис 8. Результат виконання запиту 1.

Другий запит виводить перші п’ять порід собак, які є дружніми до дітей та підходящими для утримання у квартирі. Результати відсортовані по назві породи:

```
SELECT DISTINCT ?x ?z WHERE {
    ?x dog:good_with_children ?y .
    ?x dog:good_for_apartment ?z .
  FILTER(?z=true)
    }
    ORDER BY ?x
    LIMIT 5
```

Рис 9. Код запиту 2.

Showing 1 to 5 of 5 entries

	x	z
1	dog:Affenpinscher	"true"^^xsd:boolean
2	dog:AmericanEskimoDog	"true"^^xsd:boolean
3	dog:AustralianTerrier	"true"^^xsd:boolean
4	dog:Azawakh	"true"^^xsd:boolean
5	dog:Basenji	"true"^^xsd:boolean

Рис 10. Результат виконання запиту 2.

Третій запит виводить назву, зріст та вагу собак, які є сторожовими та зріст яких більший за 17 дюймів:

```
SELECT ?x ?y ?z WHERE{
  ?x dog:weight ?y .
  ?x dog:height ?z .
  ?x dog:good_for_guarding ?k .
  FILTER regex(?z, "^17") .
}
```

Рис 11. Код запиту 3.

	x	y	z
1	dog:EntlebucherMountainDog	"50-65 pounds (male), 40-55 pounds (female)"	"17-21 inches (male), 16-20 inches (female)"
2	dog:Basenji	"24 pounds (male), 22 pounds (female)"	"17 inches (male), 16 inches (female)"
3	dog:Brittany	"30-40 pounds"	"17.5-20.5 inches"
4	dog:FinnishSpitz	"25-33 pounds (male), 20-28 pounds (female)"	"17.5-20 inches (male), 15.5-18 inches (female)"
5	dog:GermanPinscher	"25-45 pounds"	"17-20 inches"
6	dog:LabradorRetriever	"28.5-35 pounds (male), 24-31 pounds (female)"	"17-19 inches (male), 16-18 inches (female)"
7	dog:NorwegianBuhund	"31-40 pounds (male), 26-35 pounds (female)"	"17-18.5 inches (male), 16-17.5 inches (female)"
8	dog:SoftCoatedWheatenTerrier	"40-49 pounds (male), 31-40 pounds (female)"	"17.5-19.75 inches (male), 15.75-18 inches (female)"

Showing 1 to 8 of 8 entries

Рис 12. Результат виконання запиту 3.

Четвертий запит виводить породи собак, що є гіпоалергенними та які линяють посезонно.

```
SELECT ?x ?y ?z WHERE{
  ?x dog:hypoallergenic ?y .
  ?x dog:shedding ?z
  FILTER(?y=True) .
  FILTER regex(?z, 'Seasonal') .
}
```

Рис 13. Код запиту 4.

Showing 1 to 5 of 5 entries

	x
1	dog:Affenpinscher
2	dog:CotonDeTulear
3	dog:GiantSchnauzer
4	dog:IrishWaterSpaniel
5	dog:PortugueseWaterDog

Рис 14. Результат виконання запиту 4.

П’ятий запит виводить породи собак, які є або підходящими одночасно і для сім’ї і для утримання у квартирі, або ж непідходящими ні для сім’ї, ні для квартири.

```
SELECT ?x ?y WHERE{
  ?x dog:good_for_apartment ?y .
  ?x dog:good_for_family ?z .
  FILTER(?z=?y) .
}
OFFSET 10
LIMIT 5
```

Рис 15. Код запиту 5.

x	y
dog:BelgianMalinois	"false"^^xsd:boolean
dog:BelgianSheepdog	"false"^^xsd:boolean
dog:BelgianTervuren	"false"^^xsd:boolean
dog:BlackAndTanCoonhound	"false"^^xsd:boolean
dog:BlackRussianTerrier	"false"^^xsd:boolean

Рис 16. Результат виконання коду запиту 5

Розділ 4. Створення веб-сайту з використанням розробленої онтології

4.1 Опис стеку технологій

Для розробки веб-сторінки були використані Python-фреймворк Flask. Flask є мікрофреймворком та надає усі базові інструменти для створення веб-сайтів. Назва «мікрофреймворк» означає збереження ядра простим, але розширюваним. Немає абстрактного рівня бази даних, немає валідації форм. Однак, Flask підтримує усілякі плагіни, що додають необхідну функціональність, а також багато речей є попередньо налаштованими на основі загальної базової конфігурації. Наприклад, шаблони та статичні файли збережені у підкаталогах. Основна ідея проектного рішення є те, що прості завдання повинні залишатися простими та не займати багато коду. Flask використовує локальні треди всередині об’єктів, тому користувачу не потрібно передавати об’єкти у межах одного запиту від функції до функції, залишаючись у безпечному треді. Flask захищає веб-додатки від найпоширеніших способів злому, таких як XSS (cross-site scripting). Для того, щоб встановити Flask необхідно мати Python та pip встановленими та шляхи до папок bin додані у змінні середи Path. У командній стрічці потрібно написати: pip install flask. Для роботи з онтологією було використано модуль owlready2, що використовується для онтологічно-орієнтованого програмування на Python. Цей модуль надає можливість загружати онтології OWL у вигляді об’єктів Python, змінювати їх та зберігати. Owlready 2 забезпечує прозорий доступ до онтологій та включає в себе оптимізоване сховище триплетів, основане на SQLite3. На відміну від першої версії, owlready2 може працювати з великими онтологіями.

4.2 Інтеграція веб-сайту з онтологією

Розроблений веб-сайт призначений для того, щоб допомагати користувачам підібрати породу собак відповідно до різних критеріїв пошуку. Отримані у тесті відповіді обробляються та підставляються у запит до бази знань для знаходження співпадінь.

Варіанти відповідей у шаблоні мають своє ім’я та значення:

```
<p> Are you allergic to dogs? <BR>
    <input type="radio" name="allergic" value="yes">Yes<BR>
    <input type="radio" name="allergic" value="no">No<BR>
</p>
```

У головному класі views завантажуємо створену онтологію порід собак:

```
onto_path.append('')
    onto = get_ontology('file://pupper_ontology_final.owl')
    onto.load()
```

Потім перевіряємо значення кожного обраного критерію та виконуємо пошук по базі знань. Наприклад, якщо обрано варіант «Так» на питання «Чи є у вас алергія на собак», то в базі вибираються екземпляри порід, властивість яких «Гіпоалергенність» має значення «True».

```
if len(allergic) > 0:
        if allergic[0] == 'yes':
            results = onto.search(hypoallergenic = True)
            final_results = list(set(results) & set(final_results))
```

Екземпляри, які підійшли по кожному з заданих критеріїв, заносяться у фінальний масив значень на сторінці результату ми виводимо назву екземпляру, його опис, характер породи та картинку:

```
if name not in final_results_dict:
    final_results_dict[name] = []
    final_results_dict[name].append(breed.description)
    final_results_dict[name].append(breed.personality)
    final_results_dict[name].append(breed.image_url)
```

4.3 Інструкція користувача

Основна сторінка веб-сайту містить тест на підбір породи собаки – список питань, пов'язаний з вподобаннями різних аспектів собак.

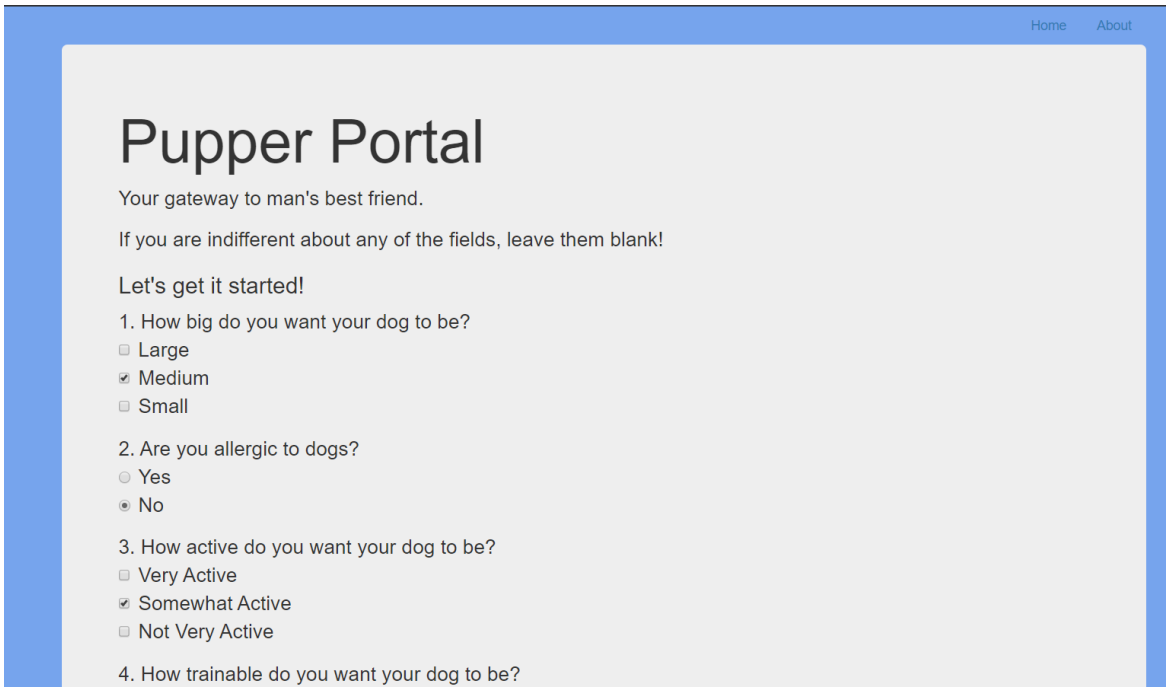


Рис 17

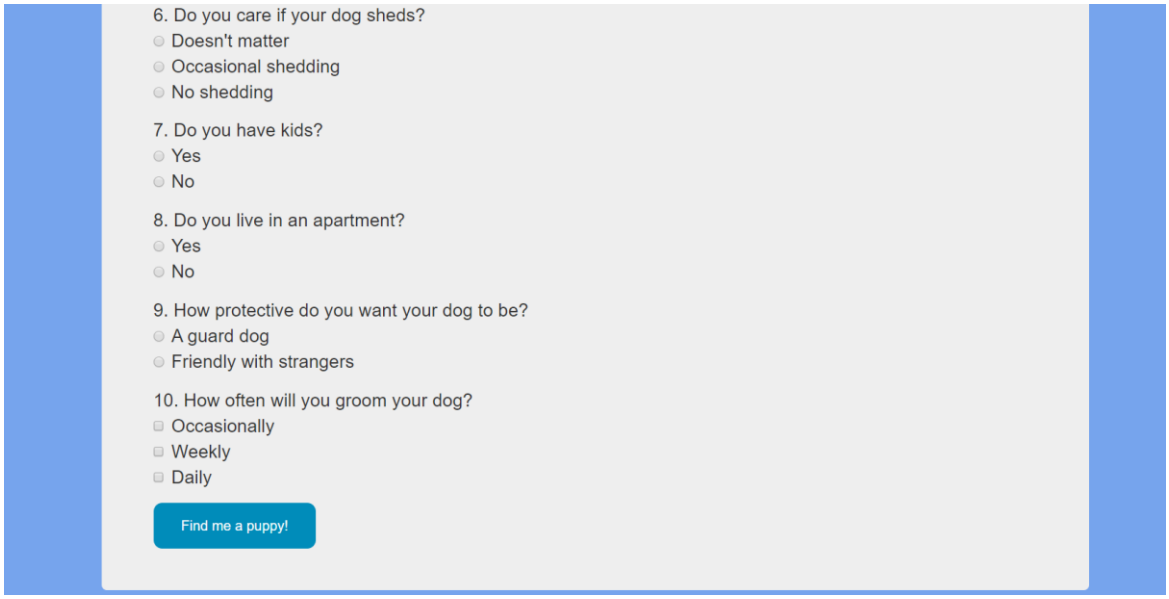


Рис 18

Користувач відповідає на задані питання та натискає кнопку “Find me a puppy!”. Після цього відбувається перехід на сторінку з результатом, на якій відображені найбільш підходящі породи відповідно до наданих відповідей.

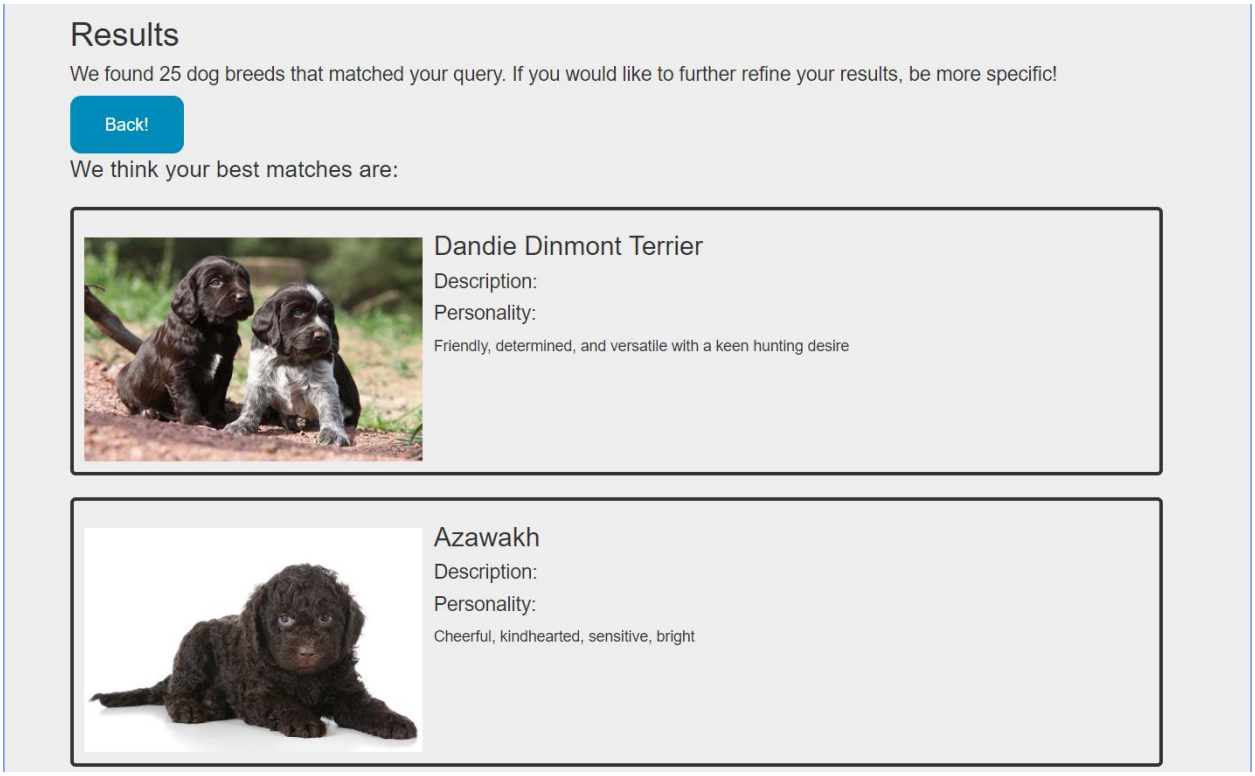


Рис 19

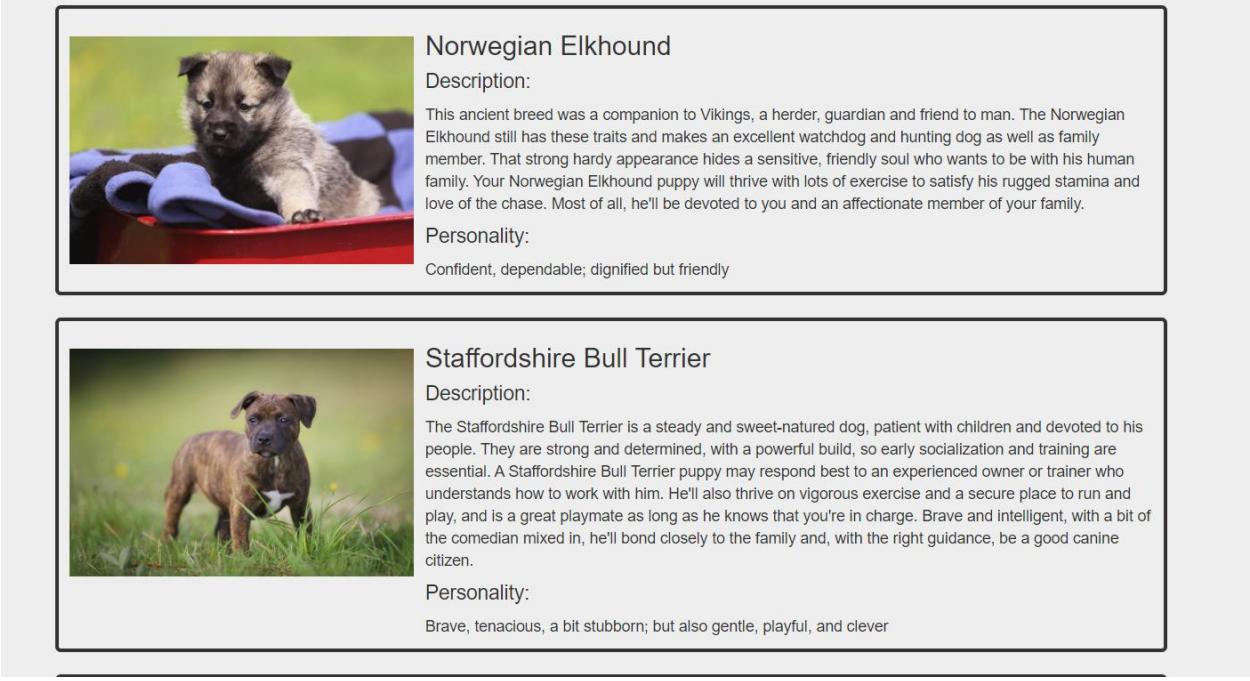


Рис 20

Висновки

У результаті виконання роботи була створена онтологія з предметної області «Породи собак». В неї входить 15 класів та підкласів, 3 понять об’єкту, 17 атрибутів класу та 187 екземплярів. Також написано декілька запитів SPARQL, що показують роботу онтології. Більш того, було створено простий онлайн-ресурс з використанням створеної онтології для підбору порід собак у залежності від критеріїв пошуку.

У ході виконання роботи я поглибила свої навички та знання у проектуванні онтологій за допомогою середовища Protégé, ознайомилась з фреймворком Flask та плагіном owlready2, здобула навички написання SPARQL запитів до онтології на інтегрування онтології у онлайн ресурси.

У перспективі до бази будуть додаватися нові класи тварин, нові зв’язки між об’єктами та екземпляри задля покриття якомога ширшого спектру питань, пов’язаних з темою домашніх тварин.

Список використаних джерел

1. Brickley, D. and Guha, R.V. (1999). Resource Description Framework (RDF) Schema Specification. Proposed Recommendation, World Wide Web Consortium**:** http://www.w3.org/TR/PR-rdf-schema.
2. Gruber T.R. The role of common ontology in achieving sharable, reusable knowledge bases // Principles of Knowledge Representation and Reasoning. Proceedings of the Second International Conference. J.A. Allen, R. Fikes, E. Sandewell – eds. Morgan Kaufmann, 1991.
3. Gruber T.R. A translation approach to portable ontology specifications. Knowledge Acquisition, 1993. – Vol. 5. – P. 199-220
4. Operations on Ontologies. [Електронний ресурс] – Режим доступу: https://www.obitko.com/tutorials/ontologies-semantic-web/operations-on-ontologies.html – Назва з екрана.
5. An Introduction to Ontology Engineering – C. Maria Keet, 2018.
6. SPARQL Query Language for RDF. [Електронний ресурс] – Режим доступу: https://www.w3.org/TR/rdf-sparql-query/ . – Назва з екрана.
7. Protégé Wiki. [Електронний ресурс] – Режим доступу: https://protegewiki.stanford.edu/wiki/Main_Page . – Назва з екрана.
8. The Protege Project. [Електронний ресурс] – Режим доступу: http://protege.stanford.edu. – Назва з екрана.
9. Flask Documentation. [Електронний ресурс] – Режим доступу: https://flask.palletsprojects.com/en/1.1.x/ . – Назва з екрана.
10. Learning Sparql – Bob Ducharme, 2013.