Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

Реалізація бази знань за допомогою системи PROTÉGÉ

Текстова частина до курсової роботи
за спеціальністю «Комп'ютерні науки» 122

Керівник курсової роботи
к.ф.-м.-н., доц. Жежерун О.П.
_____
(підпис)
"____" _____2024 р.

Виконала студентка
Гусаренко Ю.І.
"____" _____2024 р.

Київ 2024

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ
Зав. кафедри мультимедійних систем,
к.ф.-м.н., доц.
_____Жежерун О.П.
(підпис)
Виконала студентка
Гусаренко Ю.І.
“_____“ _____2024 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студентці Гусаренко Юлії Іванівні

Факультету інформатики 3 р.н. бакалаврської програми

ТЕМА: **Реалізація бази знань за допомогою системи PROTÉGÉ**

Зміст ТЧ до курсової роботи:

1. Індивідуальне завдання
2. Анотація
3. Вступ
4. Розділ 1. Теоретична частина
5. Розділ 2. Практична частина
6. Висновки
7. Список використаної літератури

Дата видачі “_____“ _____2024 р. Керівник _____
(підпис)

Завдання отримав_____
(підпис)

Календарний план виконання курсової роботи

Тема: Реалізація бази знань за допомогою системи PROTÉGÉ

Календарний план виконання роботи

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу.	10.04.2024	
2.	Огляд літератури за темою роботи.	15.04.2024	
3.	Написання першого розділу.	25.04.2024	
4.	Розробка онтології.	01.05.2024	
5.	Розробка веб-застосунку.	04.05.2024	
6.	Написання другого розділу.	10.05.2024	
7.	Створення презентації.	14.05.2024	
8.	Здача курсової роботи на перевірку.	14.05.2024	

Студентка Гусаренко Ю.І.

Керівник Жежерун О.П.

“_____”_____

ЗМІСТ

АНОТАЦІЯ

У цій курсовій роботі представлено методологію створення онтологій з використанням існуючих підходів та систем. Розглянуто та проаналізовано існуючі інструменти для роботи з онтологіями та особливості цих інструментів. Продемонстровано процес створення онтології, на прикладі застосування засобів PROTÉGÉ-OWL. Також описується процес проектування та розробки бази знань та використання SPARQL запитів для взаємодії з онтологією та розробка веб-застосунку.

У результаті даної роботи було створено базу знань, яка спрямована на спрощення пошуку та аналізу інформації про заклади харчування та веб-застосунок на базі мови програмування Python та фреймворку Flask для серверної частини та HTML, CSS, JavaScript для клієнтської частини. Даний застосунок покликаний дати змогу користувачам у зручний спосіб взаємодіяти з базою знань та швидко знаходити заклади громадського харчування за власними вподобаннями.

ВСТУП

В контексті курсової роботи розглядається питання створення онтологій для систематизації великих обсягів даних, специфічних для сфери громадського харчування. Сучасні інтелектуальні системи вимагають високоорганізованих механізмів для ефективного пошуку та аналізу інформації, що зберігається в базах даних. В основу архітектури таких систем закладено онтології, створені для конкретних предметних областей.

Онтологія у даному випадку представляє собою систему, яка інтегрує визначені поняття та взаємозв’язки між ними, дозволяючи структурувати знання про ресторани, кафе та інші заклади харчування. Використання мови запитів SPARQL для доступу до даних в онтології може стати ключовим елементом у розробці засобів для взаємодії з базами знань. Проєкт включає в себе створення бази знань, яка дозволить користувачам здійснювати складні запити для аналізу даних без потреби у глибоких знаннях у сфері програмування. Розробка веб-застосунку на основі цієї бази знань спростить взаємодію з системою, роблячи її доступною для широкого кола користувачів.

Робота реалізована за допомогою програмного редактора PROTÉGÉ, який використовується для проєктування та наповнення онтологічної бази даних. Візуалізація результатів і взаємодія з користувачем забезпечується через веб-інтерфейс, створений з використанням мови розмітки HTML, CSS та JavaScript.
Структура роботи включає кілька розділів:

Перший розділ зосереджений на аналізі задач, вивченні предметної області та основних понять, пов’язаних із закладами харчування.

У другому розділі детально описується процес створення онтології, її структура та методологія розробки в системі PROTÉGÉ.

Третій розділ присвячений особливостям мови запитів SPARQL, представлення розробленої бази знань та її використання для аналізу даних.

Четвертий розділ висвітлює розробку веб-застосунка, який інтегрує створену онтологію для поліпшення користувацького досвіду.

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ЗАДАЧ ТА АНАЛІЗ ОБРАНОЇ ГАЛУЗІ

1.1 Формування задач роботи

Метою цієї курсової роботи є організація, систематизація та управління об’ємними даними у сфері громадського харчування, а також розробка веб-сайту для спрощення обробки інформації про заклади харчування. Використання онтологічного підходу дозволяє чітко визначити та структурувати основні категорії, такі як різні типи ресторанів, кафе та їхні кухні, адреса, середній чек, рейтинг тощо, а також взаємозв’язки між ними та їх атрибути. Це сприяє ефективному пошуку та аналізу даних. Завдяки створенню такої онтології забезпечується швидкий доступ до необхідної інформації та її аналіз. Для зручності користувачів, які не володіють технічними навичками, було розроблено веб-інтерфейс, що дозволяє взаємодіяти з онтологією через простий веб-сайт.

Задачі роботи включають:

1. аналіз предметної області «Заклади харчування»;
2. розгляд наявних методів та інструментів для створення онтологій;
3. реалізація онтології для закладів харчування;
4. розробка SPARQL запитів для маніпуляції даними;
5. створення інтуїтивно зрозумілого інтерфейсу;
6. інтеграція онтології та запитів у фінальну систему.

1.2 Предметна область «Заклади харчування»

Ресторан, кафе, бари це різні форми закладів громадського харчування, кожен з яких пропонує унікальний підхід до подачі їжі та сервісу. Ці установи забезпечують не тільки харчування, але й створюють атмосферу спілкування та відпочинку, залучаючи різноманітні напрямки кухонь та різні формати обслуговування.

Важливу роль в управлінні такими закладами відіграють сучасні технології, які можуть дозволити, зберігати дані про клієнтів, замовлення, меню, персонал та

інші ключові аспекти діяльності. Традиційно ці дані утримуються в реляційних базах даних, що забезпечує можливість швидкого пошуку, сортування за різними параметрами, такими як тип кухні, цінова категорія, розташування та інше. Однак, з огляду на складність і різноманіття взаємозв’язків у сфері громадського харчування, в цій роботі було застосовано онтологічний підхід. За допомогою онтології ефективно моделюється структура та взаємодії всередині таких закладів, що значно спрощує управління даними та підвищує ефективність обробки запитів.

РОЗДІЛ 2. ОСНОВИ ТА РЕАЛІЗАЦІЇ ОНТОЛОГІЙ У PROTÉGÉ

2.1 Теоретичні основи онтологій

Онтологія у контексті інформаційних технологій це формальне представлення знань як набір концептів у певному домені та взаємозв'язків між цими концептами. Використання онтологій дозволяє систематизувати інформацію та сприяє більш ефективному пошуку та обробці даних [1]. Вони забезпечують загальне розуміння домену, що може бути використане для вирішення складних задач у різних галузях, включно з електронною комерцією, управлінням знаннями, інженерією знань та багатьма іншими.

Онтологія зазвичай описує:

1. Класи (поняття) – групи об'єктів з подібними властивостями. Наприклад, у закладів харчування клас може представляти ресторани, кафе, бари тощо;
2. Властивості класів (атрибути) – характеристики, що описують параметри об'єктів класів. Наприклад, локація, тип кухні, середній чек, ретинг;
3. Відносини – способи взаємодії між класами.
4. Обмеження – умови, які застосовуються до властивостей чи відносин, щоб уточнити модель.

Використання онтологій дозволяє комп'ютерним системам ефективніше розуміти та обробляти великі обсяги даних. Завдяки формалізації даних і взаємозв'язків, системи можуть автоматизувати багато процесів, як-от розширений пошук, логічні висновки, інтеграція інформації з різних джерел, та багато іншого.

2.2 Огляд та аналіз методів створення онтологій

Створення онтології є ключовим етапом у розвитку інтелектуальних систем, які здатні аналізувати та обробляти великі обсяги інформації з високим ступенем автоматизації та розумінням контексту. Процес створення онтології охоплює кілька основних етапів та використовує спеціалізовані інструменти для досягнення

оптимальних результатів. Загальний процес створення онтологій включає наступні етапи:

1. Визначення інформації та обсягу: На цьому етапі визначаються цілі онтології, область знань, яку вона має покривати, та рівень деталізації, до якого слід перейти;
2. Збір інформації: Збираються відомості, які допоможуть визначити ключові концепти та взаємозв'язки;
3. Визначення концептів та ієрархій: Класифікація основних понять та встановлення ієрархічних відносин між ними;
4. Створення взаємозв'язків та властивостей: Визначення способів взаємодії між класами та їхніми атрибутами;
5. Формалізація: Переклад концепцій, ієрархій і взаємозв'язків у формальну мову, придатну для машинного оброблення;
6. Імплементація та тестування: Використання онтологічних редакторів для створення онтології та її тестування для забезпечення відповідності цілям і вимогам.

Після проходження основних етапів створення онтології, важливим моментом є вибір ефективного інструментарію, який допоможе реалізувати та втілити заплановане. Розробники мають до свого розпорядження різноманітні засоби, які спрощують створення, редагування, і тестування онтологій, найпопулярніші засоби для створення онтологій [2-6]:

- PROTÉGÉ: Найпопулярніший і універсальний інструмент для розробки онтологій. PROTÉGÉ підтримує різноманітні формати онтологій та має гнучкі налаштування для специфікації властивостей, класів та взаємозв'язків.
- OWL (Web Ontology Language): Стандартна мова опису онтологій, яка використовується для створення комплексних онтологій, сумісних з веб-технологіями;
- SWOOP: Інструмент редагування, розроблений для ефективного створення онтологій з використанням OWL. Відрізняється швидкістю та зручністю в обробці великих онтологій;

- TopBraid Composer: Комерційне програмне забезпечення для моделювання онтологій, що підтримує мови RDF, OWL, і SPARQL. Пропонує візуальне редагування та засоби об’єднання даних.

Ці інструменти дозволяють розробникам ефективно створювати, редагувати та управляти онтологіями, забезпечуючи точність та сумісність з існуючими стандартами. Вибір конкретного інструмента залежить від специфіки проекту, рівня складності онтології та доступних ресурсів.

2.3 Створення онтології «Заклади харчування» у PROTÉGÉ

Створення онтології для предметної області «Заклади харчування» є ключовим етапом в організації знань у цій сфері. Онтологія дозволяє структурувати інформацію таким чином, що комп’ютери можуть її обробляти, роблячи можливими розумні пошуки, автоматичне оновлення тощо. Для заданої теми було обрано наступні класи (рис. 2.1):

- FoodEstablishment (Заклад харчування) основний клас для всіх закладів харчування;
- EstablishmentType (Тип закладу) цей клас визначає тип закладу харчування (наприклад, ресторан, кафе);
- Cuisine (Кухня) цей клас представляє тип кухні, який пропонує заклад (наприклад, італійська, японська, українська);
- Service (Сервіс) представляє різні види послуг, які може пропонувати заклад (доставка, WiFi тощо);
- City (Місто) визначає місто, в якому розташований заклад
- Review (Відгук) цей клас представляє відгук, який користувач може залишити на сторінці із закладом.

Для правильної та структурованої роботи з онтологією між класами потрібно встановити деякі зв’язки або властивості, *hasCuisine* об’єктна властивість, яка пов’язує заклад харчування з типом кухні, яка в ньому представлена. *hasEstablishmentType* об’єктна властивість, яка пов’язує заклад харчування з його

типом. *hasService* об’єктна властивість, яка асоціює заклад з послугами, які він пропонує. *hasCity* об’єктна властивість, яка пов’язує заклад харчування з

його типом. *hasService* об’єктна властивість, яка асоціює заклад з містом, в якому він розташований. *hasRewiew* об’єктна властивість, яка пов’язує заклад харчування з відгуком, який йому залишив користувач.

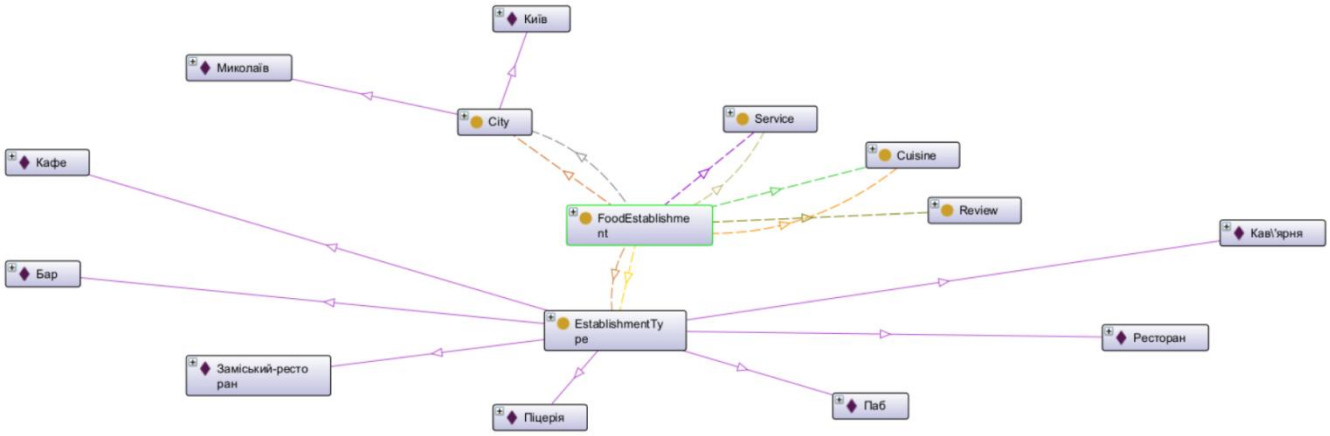

Рисунок 2.1 — Граф створених класів

Далі необхідно створити властивості для типів даних які будуть зберігатись у онтології:

- *hasAddress* – властивість, що вказує адресу закладу харчування;
- *hasAverageBill* – властивість, що вказує середню вартість обіду в закладі;
- *hasName* – властивість для назви закладу;
- *hasPhone* – номер телефону закладу;
- *hasPhoto* – властивість для зберігання посилання на фотографію закладу;
- *hasMenuPhoto* – властивість для зберігання посилання на зображення меню закладу;
- *hasRating* – рейтинг закладу;
- *hasWebsite* – вебсайт закладу;
- *hasWorkingHours* – властивість, яка описує графік роботи закладу.

- *hasUserName* – властивість відкугу, яка зберігає ім’я, введене користувачем
- *hasRewiewText* – властивість, що зберігає сам текст відгуку
- *hasRewiewRating* – оцінка поставлена закладу користувачем
- *hasRewieDate* – дата створення відгуку

У контексті онтологій та програми PROTÉGÉ, індивіди є конкретними інстанціями або екземплярами класів, які представляють певні об’єкти або концепції з реального світу. Простіше кажучи, якщо класи це загальні категорії, то індивіди це специфічні представники цих категорій.

На прикладі онтології «Заклади харчування», якщо є клас «Кухня», індивідами можуть бути конкретні типи кухонь, які існують в реальних закладах харчування (рис. 2.2).

Рисунок 2.2 — Приклад индивідів класів

Використання індивідів в онтології дозволяє не тільки класифікувати реальні об’єкти та події, але й створювати взаємозв’язки між ними. Для онтології закладів харчування було створено екземпляри для класів типу закладу, кухні яку пропонує заклад, послуги які заклад надає та самого закладу.

З таким набором даних можна проводити різноманітні операції над онтологією та даними в ній за допомогою мови запитів SPARQL.

РОЗДІЛ 3. МОВА ЗАПИТІВ SPARQL ТА ЇЇ ЗАСТОСУВАННЯ

3.1 Загальний огляд мови запитів SPARQL

SPARQL (SPARQL Protocol and RDF Query Language) мова запитів, розроблена для роботи з даними у форматі RDF (Resource Description Framework). RDF це модель даних, що використовується для представлення інформації у вигляді графа, де вершинами є ресурси, а ребрами, їх відношення [7]. Мова SPARQL призначена для виконання запитів до таких графових даних. Вона дозволяє вибирати, фільтрувати та обробляти дані, що представлені у вигляді RDF. Основна мета SPARQL це отримання інформації з RDF-джерел за допомогою різноманітних запитів.

Особливості та можливості мови SPARQL включають:

1. Гнучкість: SPARQL дозволяє формулювати складні запити, включаючи умови фільтрування, об’єднання та групування даних;
2. Можливість з’єднання з різними джерелами даних: SPARQL може використовуватися для пошуку даних у різних RDF-джерелах, що дозволяє отримати комплексну інформацію;
3. Підтримка RDF-запитів: SPARQL розуміє структуру даних у форматі RDF, що дозволяє ефективно виконувати запити до графових даних;
4. Можливість використання в різних сферах: SPARQL широко використовується у семантичному вебі, а також в областях, пов’язаних з обробкою даних у форматі RDF, таких як біоінформатика, культурна спадщина та інше;
5. Підтримка різноманітних запитів: SPARQL підтримує різні типи запитів, такі як запити на вибірку даних (SELECT), запити на оновлення даних (INSERT, DELETE), агрегаційні запити тощо;
6. Структура запитів: Запити SPARQL мають структуровану форму, яка складається з таких елементів, як SELECT (для вибірки даних), WHERE (для визначення умов відбору даних), FILTER (для фільтрування результатів),

GROUP BY (для групування даних), ORDER BY (для сортування результатів) та інші.

7. Використання у семантичному вебі: SPARQL є ключовою складовою семантичного вебу, оскільки дозволяє взаємодіяти з даними, що представлені за семантичними правилами та містять відносини між об'єктами;

8. Підтримка анотацій: SPARQL може бути використаний для вилучення анотованих даних, що містять різні метадані та описи ресурсів, що дозволяє вирішувати завдання з пошуку та аналізу інформації з різних джерел;

9. Можливості розширення: Існують різні розширення мови SPARQL, які дозволяють розширити її можливості, наприклад, додавання нових функцій або операцій;

10. Інтеграція з іншими інструментами: SPARQL може бути легко інтегрований з іншими інструментами і технологіями, такими як бази даних, програми для обробки даних та веб-сервіси;

Загалом, мова запитів SPARQL є потужним інструментом для роботи з графовими даними у форматі RDF. Вона надає широкий спектр можливостей для виконання різноманітних запитів та обробки даних, що робить її корисною у багатьох областях, де використовується модель RDF.

3.2 Розробка та виконання запитів SPARQL

Розробка та виконання запитів SPARQL у PROTÉGÉ, інструменті для роботи з онтологіями, дозволяє взаємодіяти з даними у форматі RDF та виконувати різноманітні запити до графових даних. Основними етапами роботи з SPARQL у PROTÉGÉ є створення запиту, його виконання та аналіз результатів. Процес розробки та виконання запиту у PROTÉGÉ можна розглянути на прикладі простого запиту до графових даних. Розглянемо приклад такого запиту:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?subject ?predicate ?object
WHERE {
```

```
  ?subject rdf:type ?object .
}
```

PREFIX: Визначається префікс для простору імен RDF. У цьому прикладі використовується префікс rdf, який вказує на стандартний простір імен RDF. SELECT: Вказується список змінних, які повинні бути повернуті в результаті запиту. У цьому прикладі використовується ?subject, ?predicate та ?object, які відповідають суб'єкту, предикату та об'єкту в графі RDF. WHERE: Визначається умова, за якою обираються трипли з графа. У цьому прикладі використовується умова ?subject rdf:type ?object, яка вказує, що обираються трипли, де суб'єкт має тип, визначений об'єктом. Для прикладу можна розглянути складніший запит:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX : <http://example.org/onto.owl#>

SELECT ?name ?phone ?rating ?workingHours (GROUP_CONCAT(DISTINCT ?cuisineLabel;
separator=", ") AS ?cuisines) (GROUP_CONCAT(DISTINCT ?serviceLabel; separator=", ")
AS ?services)
WHERE {
  ?restaurant rdf:type :FoodEstablishment .
  ?restaurant :hasName ?name .
  ?restaurant :hasPhone ?phone .
  ?restaurant :hasRating ?rating .
  ?restaurant :hasWorkingHours ?workingHours .

  OPTIONAL {
    ?restaurant :hasCuisine ?cuisine .
    ?cuisine rdfs:label ?cuisineLabel .
  }

  OPTIONAL {
    ?restaurant :hasService ?service .
    ?service rdfs:label ?serviceLabel .
  }
}
GROUP BY ?name ?phone ?rating ?workingHours
ORDER BY DESC(?rating)
```

Цей SPARQL-запит призначений для отримання інформації про ресторани з графових даних, які представлені в онтології з використанням відповідних префіксів для простору імен. Розглянемо кожну частину запиту детально. PREFIX: Визначається набір префіксів для використання в запиті. У цьому випадку визначені

префікси для просторів імен RDF, OWL, XSD, RDFS та для конкретного простору імен, що представлений як <http://example.org/onto.owl#>.

SELECT: Вказується список змінних, які повинні бути повернуті в результаті запиту. У цьому запиті відбираються змінні ?name, ?phone, ?workingHours, ?rating, ?services та ?cuisines, які відповідають різним атрибутам ресторану. WHERE: Визначається умова, за якою обираються трипли з графа. У цьому випадку запит шукає всі екземпляри класу :FoodEstablishment, а потім відбираються їх атрибути за допомогою відповідних властивостей (:hasName, :hasPhone, :hasRating і т.д.). Крім того, використовуються опціональні блоки для отримання інформації про послуги та кухні, які можуть пропонувати ресторани. GROUP BY: Ця частина групує результати за визначеними змінними. У цьому випадку, результати будуть згруповані за назвою, телефоном, рейтингом та робочими годинами. ORDER BY: Вказує, як сортувати результати. У цьому випадку, результати будуть відсортовані за рейтингом закладу в порядку спадання. Отже, цей запит дозволяє отримати інформацію про заклади з графових даних, включаючи їх назву, телефон, робочі години , рейтинг, а також можливі послуги та кухні (рис. 3.1). Він використовує концепції RDF/OWL для структурування даних та здійснення запитів до даних у графовому форматі.

?name	?phone	?rating	?workingHours	?cuisines	?services
Loft	+38 066 620 4704	5.0	пн-вс 10:00-24:00	Європейська, Європейська	WiFi, Більярд
Phi-Phi	+38 096 460 9360	5	07:30-23:00	Азіатська, Азіатська, Азіатська, Азіатська, ...	WiFi, Їжа на виніс, Відкритий майданчик, К...
Perets	+38 044 569 2575	4.8	12:00-24:00	Італійська, Італійська, Італійська, Італійськ...	Їжа на виніс, Їжа на виніс, Їжа на виніс, Їжа...
Казка	+38 067 907 8111	4.8	пн-вс 11:00-23:30	Європейська, Європейська, Європейська,...	WiFi, WiFi, WiFi, Жива музика, Жива музик...
Якіторія	+38 068 123 4567	4.8	пн-вс 10:00-24:00	Європейська, Європейська, Європейська,...	VIP зала, VIP зала, WiFi, WiFi, Доставка, До...
Шоті	+38 044 339 9399	4.8	12:00-23:00	Грузинська, Грузинська, Грузинська, Груз...	WiFi, Їжа на виніс, Банкетна зала, Відкрит...
Uno Cafe	+38 093 568 6009	4.6	08:00-21:00	Європейська, Європейська, Європейська,...	WiFi, Їжа на виніс, Відкритий майданчик, М...
Балдіно	+38 067 980 3099	4.5	пн-вс 10:00-24:00	Європейська, Європейська, Європейська,...	WiFi, WiFi, Банкетна зала, Банкетна зала, ...
Spezzo	+38 050 388 3801	4.5	11:00-23:00	Італійська, Італійська, Італійська	WiFi, Їжа на виніс, Приймаються картки
Флібустьєр	+38 050 493 1012	4.4	09:00-23:00	Європейська, Європейська, Європейська,...	WiFi, WiFi, Бранч, Бранч, Бізнес ланч, Бізн...
Асторія	+38 093 635 0593	4.4	11:00-23:00	Європейська, Європейська, Європейська,...	VIP зала, WiFi, Їжа на виніс, Банкетна зала...
Daily Sport	+38 050 747 4884	4.4	11:00-23:00	Європейська, Європейська, Європейська,...	WiFi, Їжа на виніс, Банкетна зала, Відкрит...
Грифель	+38 073 021 6818	4.0	10:00-23:00	Європейська, Європейська, Європейська,...	WiFi, Їжа на виніс, Відкритий майданчик, ...
Sobi Club	+38067 892 2422	3.7	пн–нд 08:00 - 23:00	Європейська, Європейська, Європейська,...	WiFi, WiFi, Банкетна зала, Банкетна зала, ...
Піца Челентано	+38 051 271 8770	3.3	10:00-22:00	Італійська, Італійська, Італійська, Італійськ...	WiFi, WiFi, Їжа на виніс, Їжа на виніс, Доста...

Рисунок 3.1 — Результат виконання запиту

Розглянемо ще один запит:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
```

```
PREFIX : <http://example.org/onto.owl#>

SELECT ?establishmentType (COUNT(?restaurant) AS ?numRestaurants)
      (AVG(?rating) AS ?avgRating)
WHERE {
  ?restaurant a :FoodEstablishment ;
            :hasRating ?rating ;
            :hasType ?type .
  ?type rdfs:label ?establishmentType .
}
GROUP BY ?establishmentType
ORDER BY DESC(?numRestaurants)
```

Цей SPARQL запит виконує обчислення на основі даних, представлених в онтології. Він групує ресторани за їхнім типом, обчислює кількість закладів кожного типу та середній рейтинг закладів для кожного з типів, а потім сортує результати за кількістю ресторанів у кожному типі у порядку спадання (рис. 3.2). Окрім раніше вже згаданих елементів запитів, таких як GROUP BY, ORDER BY та ін., в цьому запиті використаються COUNT(): Функція, яка рахує кількість елементів у групі та AVG(): Функція, яка обчислює середнє значення у групі.

?establishmentType	?numRestaurants	?avgRating
Ресторан	9	4.622222222222223
Кафе	2	4.8
Паб	1	4.8
Піцерія	1	3.3
Кав'ярня	1	4.0
Бар	1	4.4

Рисунок 3.2 — Результат виконання запиту

РОЗДІЛ 4. РОЗРОБКА ВЕБ-ЗАСТОСУНКУ З ІНТЕГРАЦІЄЮ СТВОРЕНОЇ ОНТОЛОГІЇ

4.1 Проектування архітектури застосунку

Проектування архітектури додатку передбачає ретельне врахування різних факторів, включаючи мови програмування, фреймворки та середовища. Коли мова йде про створення веб-додатків, вибір технологій може суттєво вплинути на ефективність, масштабованість та зручність обслуговування системи. Python універсальна і широко використовувана мова програмування, відома своєю простотою, читабельністю та великою екосистемою бібліотек і фреймворків. Легкість у вивченні та чистий синтаксис роблять її чудовим вибором для швидкої розробки та створення прототипів. Flask легкий та гнучкий веб-фреймворк для Python, що ідеально підходить для створення веб-додатків та API. Він дотримується мінімалістичної філософії, надаючи розробникам необхідні інструменти для початку роботи, але залишаючи простір для налаштування відповідно до вимог проекту. Простота Flask у поєднанні з надійністю та масштабованістю робить його гарним варіантом для розробки широкого спектру веб-додатків, від невеликих проектів до масштабних корпоративних рішень [8].

HTML (мова розмітки гіпертексту), JavaScript і CSS (каскадні таблиці стилів) складають основу фронтенд-веб-розробки. HTML забезпечує структуру та зміст веб-сторінок, тоді як CSS покращує їхнє візуальне представлення та макет. JavaScript, з іншого боку, забезпечує динамічну інтерактивність і функціональність, роблячи веб-додатки більш привабливими та адаптивними. Разом ці технології дозволяють розробникам створювати сучасні, зручні інтерфейси, які відповідають потребам та очікуванням користувачів.

Flask слідує архітектурному шаблону Model-View-Controller (MVC) (рис. 4.1), хоча його точніше описати як мікрофреймворк, а не повноцінний MVC-фреймворк.

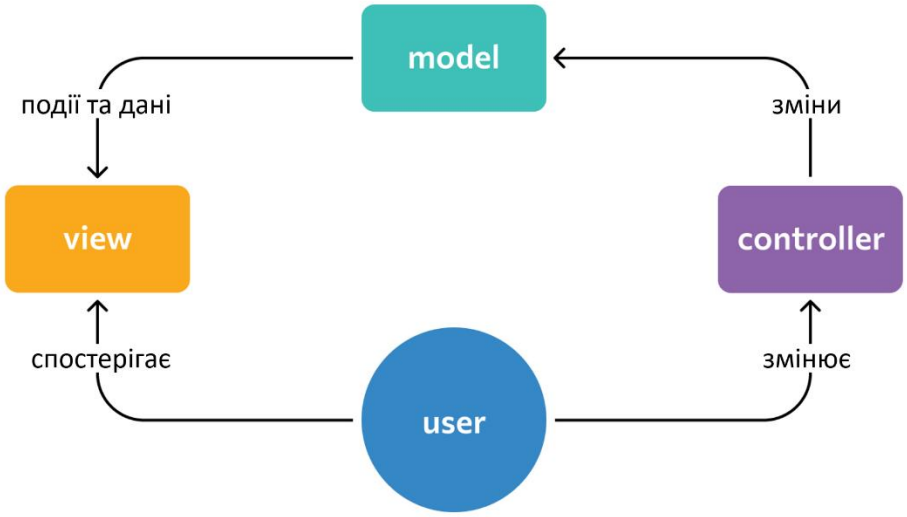

Рисунок 4.1 — Діаграма взаємодії між компонентами шаблону MVC

У архітектурі MVC, Model відповідає за дані та бізнес-логіку програми, керуючи станом та поведінкою. View відповідає за відображення даних та інтерфейс, який користувач бачить, забезпечуючи відображення інформації та подання даних. Controller є посередником між View та Model, обробляючи вхідні дані від користувача, зміни стану та подає команди моделі на зміну даних або стану. Користувач ініціює дії через View, відправляючи запити, на які Controller реагує, вносячи зміни в Model, яка потім може впливати на те, що View показує. У відповідь, View може оновлюватися, щоб відображати зміни, що відбулися у Model [9]. Легкість і розширюваність Flask у поєднанні з гнучкістю роблять його ідеальним рішенням для задач, пов'язаних з онтологіями. Онтології, які визначають взаємозв'язки і семантику в межах предметної області, часто вимагають нестандартного підходу до обробки даних. Мінімалістичний дизайн Flask дозволяє розробникам легко інтегрувати спеціалізовані бібліотеки та інструменти для роботи з онтологіями. Крім того, підтримка Flask RESTful API забезпечує легку інтеграцію з бібліотеками обробки онтологій та зовнішніми сервісами, що сприяє ефективному обміну даними та взаємодії всередині програми.

Отже, вибір Python з Flask, разом з HTML, JavaScript та CSS, забезпечує надійну основу для розробки веб-додатків з онтологічними завданнями. Простота, гнучкість та архітектурна узгодженість Flask роблять її привабливим варіантом для

створення масштабованих та підтримуваних додатків, які використовують можливості онтологій.

4.2 Інтеграція з онтологією «Заклади харчування»

Для використання створеної онтології в мові програмування Python існують декілька важливих бібліотек:

```
from flask import Flask, jsonify, request
from flask_cors import CORS
from owlready2 import get_ontology
import codecs
app = Flask(__name__)
CORS(app)
onto = get_ontology('my_ontology.owl').load()
```

Ця частина коду відповідає за імпорт необхідних бібліотек та налаштування основних параметрів для запуску веб-додатка. Імпортується клас Flask з бібліотеки Flask, а також клас CORS для обробки запитів з різних джерел. Далі завантажується онтологія з файлу my_ontology.owl, використовуючи get_ontology з бібліотеки owlready2 [10]. Потім створюється екземпляр додатка Flask та активація CORS для дозволу хрестового походження запитів. Після чого нам потрібно отримати дані про екземпляр класу закладу харчування:

```
def convert_to_serializable(establishment, detail_level='basic'):
    restaurant_photos = establishment.hasPhoto.split(',') if establishment.hasPhoto
else []
    first_photo_url = restaurant_photos[0].strip() if restaurant_photos else None

    result = {
        'name': serialize_thing(establishment.hasName),
        'types': [serialize_thing(t) for t in establishment.hasType] if
hasattr(establishment, 'hasType') else [],
        'cuisines': [serialize_thing(cuisine) for cuisine in
establishment.hasCuisine],
        'services': [serialize_thing(service) for service in
establishment.hasService],
        'address': serialize_thing(establishment.hasAddress),
        'city': serialize_thing(establishment.hasCity),
        'photo': first_photo_url,
        'working_hours': serialize_thing(establishment.hasWorkingHours),
        'rating': serialize_thing(establishment.hasRating),
        'average_check': serialize_thing(establishment.hasAverageBill),
    }
```

Ця функція використовується для конвертації екземплярів закладів харчування з онтології у серіалізований формат для подальшої передачі через

мережу. Вона отримує екземпляр закладу харчування і створює словник з його характеристиками, такими як ім’я, адреса, телефон, веб-сайт тощо.

Наступним реалізовується основна функція на серверній стороні системи:

```
@app.route('/establishments', methods=['GET'])
def get_establishments():
    print("Завантаження закладів...")
    establishments = list(onto.FoodEstablishment.instances())
    print(f"Знайдено {len(establishments)} закладів.")

    search_query = request.args.get('search', '').lower()
    types = request.args.getlist('type')
    cuisines = request.args.getlist('cuisine')
    services = request.args.getlist('services')
    city = unquote(request.args.get('city', '')).lower()

    sort_by_average_check = request.args.get('sort_by_average_check')
    sort_by_rating = request.args.get('sort_by_rating')

    # Filtering logic
    filtered_establishments = []
    for est in establishments:
        establishment_data = convert_to_serializable(est)
        if (not search_query or search_query in \
        establishment_data['name'].lower()) and \
                (not city or city in establishment_data['city'].lower()):
            if all(type in establishment_data['types'] for type in types) and \
                    all(cuisine in establishment_data['cuisines'] for cuisine \
                    in  cuisines) and \
                    all(service in establishment_data['services'] for service \
                    in services):
                filtered_establishments.append(establishment_data)

    # Sorting logic
    if sort_by_average_check:
        filtered_establishments.sort(key=lambda x: x['average_check'],
reverse=(sort_by_average_check == 'desc'))
    if sort_by_rating:
        filtered_establishments.sort(key=lambda x: x['rating'],
reverse=(sort_by_rating == 'desc'))

    print(f"Після фільтрації залишилося {len(filtered_establishments)} закладів.")
    return jsonify(filtered_establishments)
```

Ця функція визначає маршрут за адресою /establishments для отримання списку закладів харчування з онтології. Вона обробляє параметри запиту, такі як пошуковий запит, фільтри за типом, кухнею, послугами тощо, та повертає список закладів, що відповідають вказаним критеріям у форматі JSON.

Наступним іде реалізація клієнтської частини системи, починаючи з ініціалізації:

```
        document.addEventListener('DOMContentLoaded', function() {
            fetchAndDisplayData();
    });
```

Ця частина коду виконується, коли весь HTML-документ завантажився і готовий для маніпуляцій за допомогою JavaScript.

```
function fetchAndDisplayFilteredData() {
    const searchInput = document.getElementById('searchInput').value.toLowerCase();
    const types = getSelectedValues('type');
    const cuisines = getSelectedValues('cuisine');
    const services = getSelectedValues('services');
    const cityFilter = document.getElementById('cityFilter').value;

    const sortByAverageCheck = document.getElementById('sortByAverageCheck').value;
    const sortByRating = document.getElementById('sortByRating').value;

    const params = new URLSearchParams();
    params.append('search', encodeURIComponent(searchInput));
    types.forEach(type => params.append('type', type));
    cuisines.forEach(cuisine => params.append('cuisine', cuisine));
    services.forEach(service => params.append('services', service));
    if (cityFilter) params.append('city', encodeURIComponent(cityFilter));

    if (sortByAverageCheck) params.append('sort_by_average_check',
sortByAverageCheck);
    if (sortByRating) params.append('sort_by_rating', sortByRating);

    fetch(`/establishments?${params.toString()}`)
        .then(response => response.json())
        .then(data => {
            updateUI(data);
        })
        .catch(error => console.error('Error:', error));
}
```

Ця функція виконує пошук закладів харчування на основі введених користувачем даних, вибраних параметрів фільтрації та різних типів сортування. Вона зчитує значення з поля введення пошукового запиту та вибрані параметри кухні, послуг, типу закладів, міста розташування та сортування за рейтингом або середнім чеком. Потім формує запит з цими параметрами та відправляє його на сервер. Після отримання результатів вона викликає функцію updateUI, яка відображає результати пошуку на сторінці. Функція updateUI реалізована наступним чином:

```
function updateUI(data) {
    resultsContainer.innerHTML = '';
    data.forEach(est => {
        const estDiv = document.createElement('div');
        const photoURL = est['photo'] ? est['photo'] : 'placeholder.jpg';
        estDiv.classList.add('restaurant-container');
        estDiv.innerHTML = `
        <div class="restaurant-info">
            <h3>${est['name']}</h3>
            <div class="photos">
                <img src="${photoURL}" alt="Image of ${est['name']}">
            </div>
            <p>Адреса: ${est['address']}</p>
```

```
            <p>Типи: ${formatWithPrefixRemoval(est['types'])}</p>
            <p>Години роботи: ${est['working_hours']}</p>
            <p>Рейтинг: <span>${est['rating']}</span></p>
            <p>Кухня: ${formatWithPrefixRemoval(est['cuisines'])}</p>
            <p>Середній чек: <span>${est['average_check']}</span></p>
            <p>Сервіси: ${formatWithPrefixRemoval(est['services'])}</p>
            <button id="more_est" onclick="showDetails('${est['name']}')" data-
name="${est['name']}"Детальніше</button>
        </div>`;
        resultsContainer.appendChild(estDiv);
    });
}
```

Ця функція відповідає за оновлення інтерфейсу користувача після отримання результатів пошуку з сервера. Вона очищає контейнер результатів, а потім для кожного отриманого закладу створює новий блок HTML з інформацією про заклад та вставляє його в контейнер. Кожне поле інформації (назва, тип, адреса тощо) виводиться відповідно до вмісту з результатів пошуку.

Перейшовши на сторінку користувач побачить «картки» з закладами харчування (рис. 4.2). Картка закладу харчування, містить організовану інформацію, розділену на розділи для зручності користувача. Вона має наступну інформацію:

- Назва;
- Фото закладу;
- Тип закладу;
- Фізична адреса;
- Години роботи;
- Середній чек;
- Рейтинг;
- Послуги, які пропонує заклад;
- Типи представлених кухонь.

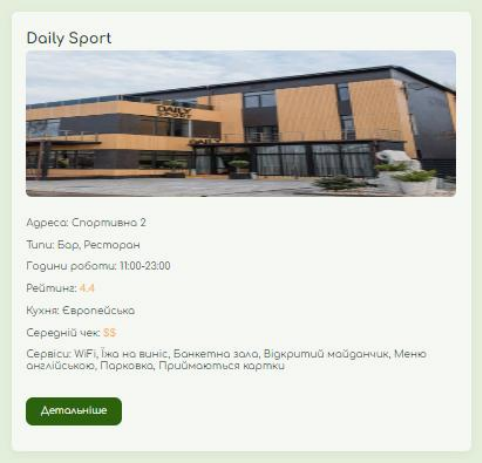

Рисунок 4.1 — Картка закладу зі сторінки

Такий дизайн картки ефективно передає інформацію, вона інформативна, але не перевантажена. Загальний вигляд сторінки зображено на рисунку 4.2. У верхній частині сторінки знаходиться поле пошуку, де користувач може здійснити пошук закладу за його назвою. Це поле пошуку динамічно відфільтровує та відображає результати в режимі реального часу, видаючи списки закладів, які відповідають запиту користувача.

У лівій частині сторінки розташовані фільтри, категоризовані за різними параметрами. Користувач може обирати з різних категорій фільтрів, як-от:

- Місто: Дозволяє відобразити заклади, які розташовані в одному із запропонованих міст;
- Тип закладу: Дозволяє вибрати тип закладу, як от ресторан, заміський-ресторан, кафе тощо;
- Кухня: Дозволяє відфільтрувати заклади згідно з видом кухні, таким як Європейська, Індійська, Італійська тощо;
- Послуги: Дозволяє відшукати заклади з певними послугами, такими як доставка, VIP зала, дитяче меню, караоке та інші.

Також користувач може застосувати сортування для відображеного переліку карток:

- За середнім чеком: у порядку зростання або спадання;
- За рейтингом: аналогічно, у порядку зростання або спадання.

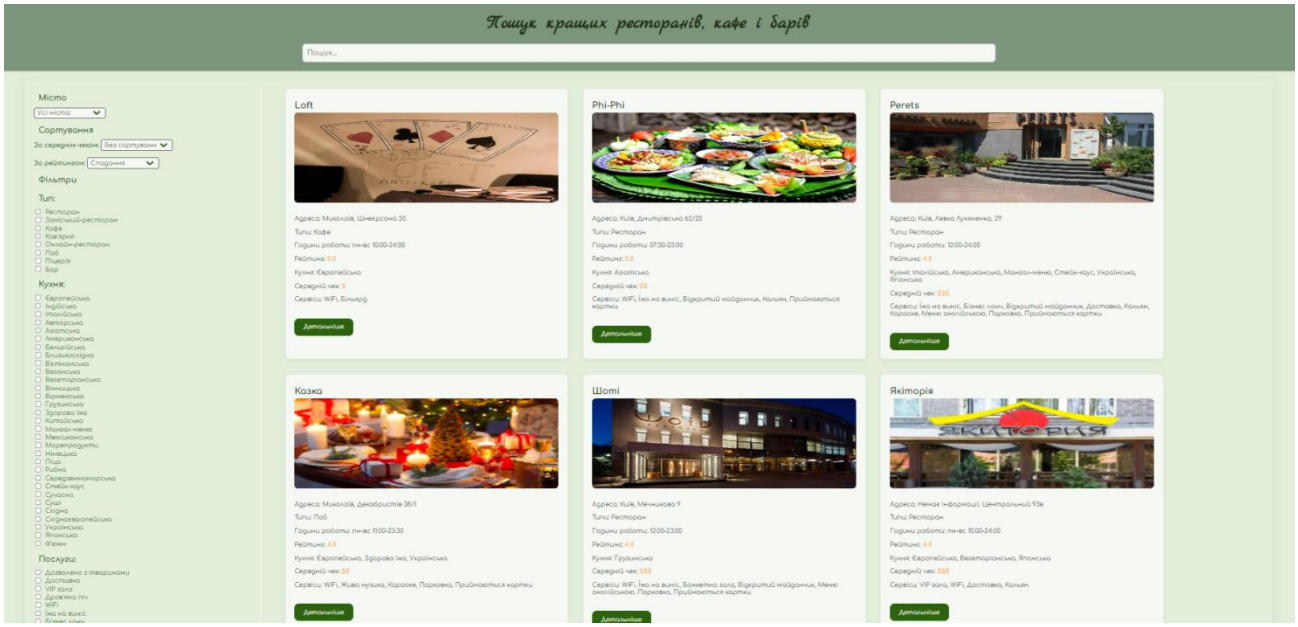

Рисунок 4.2 — Основна сторінка системи

Коли користувач обирає один або декілька фільтрів, система автоматично оновлює список доступних закладів, щоб відобразити лише ті, що відповідають обраним параметрам (рис. 4.3).

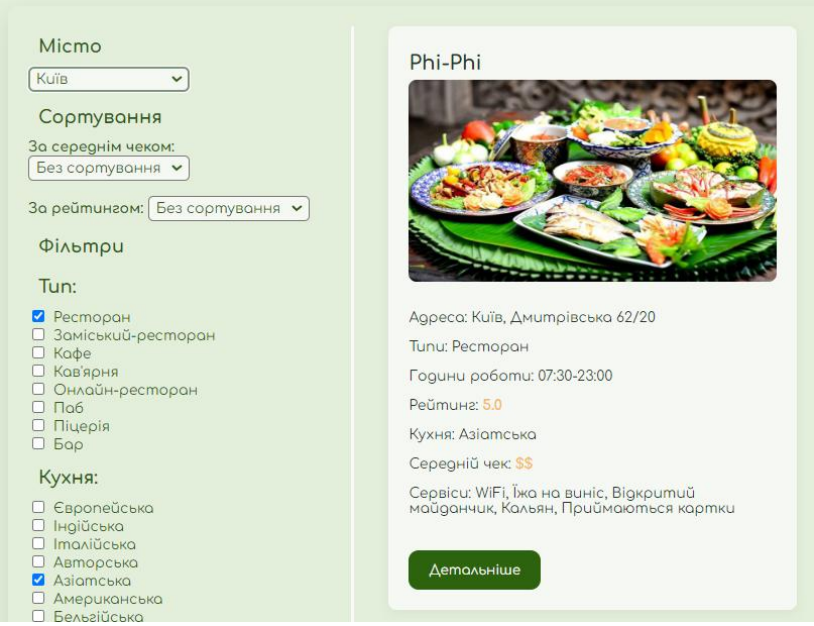

Рисунок 4.3 — Демонстрація фільтрування на сторінці

Комбінування кількох фільтрів дозволяє користувачеві знайти саме ті заклади, які відповідають його побажанням. Фільтри чітко розділені на різні секції, що

сприяє легкому знаходженню необхідного фільтру. Чекбокси дозволяють вибирати кілька опцій одночасно без необхідності натискання додаткових кнопок для підтвердження вибору, там, де не передбачено вибір кількох опцій одночасно, (наприклад, сортування) реалізовано випадні списки для економії простору на сторінці.

Окрім перегляду та взаємодії із переліком закладів харчування, користувач має змогу переглянути більш детальну інформацію про обраний заклад натиснувши на кнопку “Детальніше”, що розташована внизу кожної картки із закладом. Після натискання цієї кнопки користувач опиняється на сторінці, де може ознайомитись з додатковою інформацією (наприклад, контактні дані, опис тощо), переглянути фотогалерею закладу та фото меню, якщо такі надані закладом (рис. 4.4). Якщо прогортати цю сторінку нижче (рис. 4.5), можна переглянути або додати власний відгук. Щоб повернутись на попередню сторінку, варто просто натиснути на надпис “Повернутися на головну” внизу екрана.

Рисунок 4.4 — Сторінка з детальною інформацією про заклад

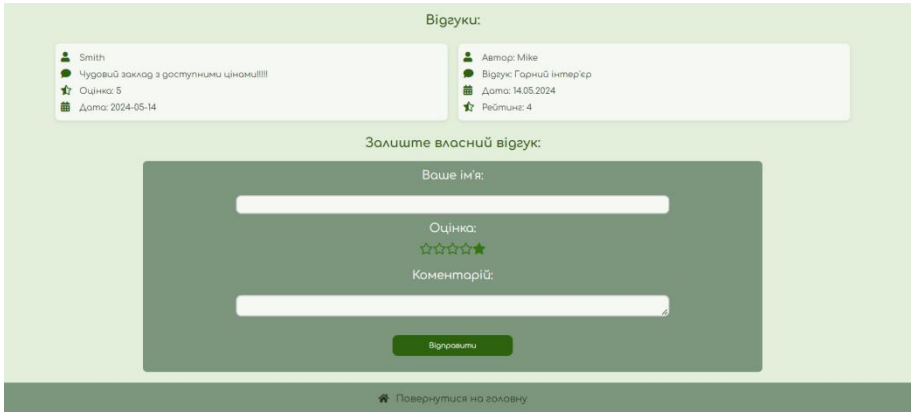

Рисунок 4.5 — Секція з відгуками

Нижче наведено функцію, яка реалізовує можливість додавання відгуку та його збереження до онтології, в якій окрім тексту самого відгуку також зберігається ім’я користувача, яке він вводить самостійно, кількість зірок, поставлена користувачем цьому закладу, та дата створення відкугу.

```
@app.route('/submit_review', methods=['POST'])
def submit_review():
    name = request.form['name']
    rating = int(request.form['rating'])
    comment = request.form['comment']
    establishment_name = request.form['establishment_name']

    establishment = next((est for est in list(onto.FoodEstablishment.instances()) if
serialize_thing(est.hasName) == establishment_name), None)

    if establishment:
        new_review = onto.Review()
        new_review.hasUsername.append(name)
        new_review.hasReviewText.append(comment)
        new_review.hasReviewDate.append(datetime.now().strftime("%Y-%m-%d"))
        new_review.hasReviewRating.append(rating)

        establishment.hasReview.append(new_review)
        onto.save(file="my_ontology.owl", format="rdfxml")

        return jsonify({"status": "success", "name": name, "rating": rating, \
        "comment": comment})
    else:
        return jsonify({"status": "error", "message": "Establishment \
        not found"}), 404
```

Ця функія із запиту отримує дані введені користувачем, знаходить відповідний заклад в базі знань за його назвою, створює новий екземпляр класу Review та заповнює його даними із запиту, додає створений відгук до відповідного закладу та зберігає його у файл з онтологією за допомогою метода onto.save().

Результатом роботи усього вище описаного коду є система яка допомагає користувачу переглядати інформацію стосовно закладів харчування та залишати відгуки.

ВИСНОВКИ

У ході виконання даної курсової роботи було проведено аналіз предметної області та сформовано відповідні задачі. Розглянуто і систематизовано основні поняття та характеристики закладів харчування, що дало змогу ефективно структурувати дані в рамках розробленої онтології.

Було проаналізовано та описано теоретичні основи онтологій, що включають ключові принципи і підходи до їхнього створення та застосування. З використанням середовища PROTÉGÉ розроблено специфічну онтологію для сфери закладів харчування, яка дозволяє репрезентувати знання предметної області в структурованому та машинно-зрозумілому форматі. Мова запитів SPARQL була розглянута як засіб для взаємодії з онтологіями, і були розроблені та виконані запити для екстракції необхідної інформації з онтології. Це надало можливість ефективно керувати даними, що лежать в основі інформаційної системи.

Проектування та розробка веб-застосунку, який інтегрує створену онтологію, стало ключовим етапом практичної частини дослідження. Архітектура застосунку була спроектована з урахуванням вимог до сучасних веб-систем, забезпечуючи інтуїтивно зрозумілий інтерфейс користувача та високу швидкість відгуку системи. Інтеграція з онтологією "Заклади харчування" дозволила реалізувати гнучкі можливості пошуку, фільтрування та сортування, поліпшуючи користувацький досвід.

Завдяки виконаній роботі, веб-застосунок надає зручний інструмент для відвідувачів різних міст у виборі закладів харчування, що відповідають їхнім вподобанням.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Онтології у контексті інтеграції інформації: представлення, методи та інструменти побудови [Електронний ресурс] // О.М. Овдій, Г.Ю. Проскудіна. – 2020. – Режим доступу до ресурсу: http://dspace.nbuv.gov.ua/bitstream/handle/123456789/1683/48%20-%20Ovdiy.pdf.

2. СТВОРЕННЯ ОНТОЛОГІЇ ПРЕДМЕТНОЇ ГАЛУЗІ МАЙБУТНІМ ІНЖЕНЕРОМ-ПЕДАГОГОМ КОМП'ЮТЕРНОГО ПРОФІЛЮ [Електронний ресурс] // Serhii Kozibroda. – 2016. – Режим доступу до ресурсу: https://www.researchgate.net/publication/331469914_STVORENNA_ONTOLOGII_PREDMETNOI_GALUZI_MAJBUTNIM_INZENEROM-PEDAGOGOM_KOMP\'UTERNOGO_PROFILU.

3. A free, open-source ontology editor and framework for building intelligent systems [Електронний ресурс] // Protege. – 2020. – Режим доступу до ресурсу: https://protege.stanford.edu/.

4. Web Ontology Language (OWL) [Електронний ресурс] // OWL. – 2019. – Режим доступу до ресурсу: https://www.w3.org/OWL/.

5. Онтологічний підхід до когнітивного аналізу та моделювання у системах типу e-commerce [Електронний ресурс] // OpenArchive. – 2019. – Режим доступу до ресурсу: https://openarchive.nure.ua/entities/publication/955fe336-b07e-4017-a386-3f777e17fc4b.

6. TopBraid's SPIN Support in AllegroGraph [Електронний ресурс] // AllegroGraph. – 2020. – Режим доступу до ресурсу: https://allegrograph.com/topbraid-composer/.

7. Вступ до зв\'язаних даних (Linked Data) [Електронний ресурс] // DIIA. – 2024. – Режим доступу до ресурсу: https://diia.data.gov.ua/info-center/linkeddata.

8. Welcome to Flask's documentation [Електронний ресурс] // Flask. – 2024. – Режим доступу до ресурсу: https://flask.palletsprojects.com/en/3.0.x/.

9. Ознайомлення з патерном MVC (Model-View-Controller) [Електронний ресурс] // JavaRush. – 2023. – Режим доступу до ресурсу: https://javarush.com/ua/groups/posts/uk.2536.chastina-7-oznayomlennja-z-paternom-mvc-model-view-controller.

10. Welcome to Owlready2's documentation! [Електронний ресурс] // Owlready2. – 2020. – Режим доступу до ресурсу: https://owlready2.readthedocs.io/en/latest/.