ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка Web-застосунку для парсингу спеціалізованого контенту мовою Python з використанням Javascript React»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Олексій МАРТИНЕНКО
(підпис)

Виконав: здобувач вищої освіти групи ПД-42
Олексій МАРТИНЕНКО

Керівник: Олег ІЛЬЇН
д.т.н., професор

Рецензент: _____

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Мартиненку Олексію Володимировичу

1. Тема кваліфікаційної роботи: «Розробка Web-застосунку для парсингу спеціалізованого контенту мовою Python з використанням Javascript React»
керівник кваліфікаційної роботи д.т.н., професор Олег ІЛЬЇН,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» лютого 2024 р. № 36.

2. Строк подання кваліфікаційної роботи «28» травня 2024 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про методи парсингу, обробки тексту, опис методів візуалізації текстових даних, технічна документація з описом бібліотек для парсингу та обробки тексту.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд та аналіз існуючих методів та технологій парсингу та візуалізації даних.

2. Проектування застосунку для парсингу спеціалізованого контенту мовою Python з використанням Javascript React.

3. Програмна реалізація та опис функціонування застосунку для парсингу спеціалізованого контенту мовою Python з використанням Javascript React ".

4. Тестування застосунку.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до додатку.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. Діаграма діяльності.
6. Діаграма класів.
7. Схема бази даних.
8. Екранні форми.
9. Апробація результатів дослідження

6. Дата видачі завдання «28» лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	28.02.-06.03.2024	
2	Аналіз та дослідження існуючих аналогів	07.03-13.03.2024	
3	Огляд існуючих методів парсингу та візуалізації тексту	14.03-17.03.2024	
4	Проектування додатку	18.03-24.03.2024	
5	Програмна реалізація додатку	24.03-23.04.2024	
6	Тестування додатку	24.03-23.04.2024	
7	Оформлення роботи: вступ, висновки, реферат	29.04-05.05.2024	
8	Розробка демонстраційних матеріалів	06.05-12.05.2024	
9	Попередній захист роботи	13.05-31.05.2024	

Здобувач вищої освіти _____ Олексій МАРТИНЕНКО
(підпис)

Керівник
кваліфікаційної роботи _____ Олег ІЛЬЇН
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 56 стор., 1 табл., 19 рис.,15 джерел.

Мета роботи – спрощення процесу парсингу та аналізу спеціалізованого контенту за допомогою веб-застосунку, створеного мовою Python з використанням JavaScript React для фронтенду.

Об’єкт дослідження – процес парсингу та аналізу текстового контенту.

Предмет дослідження – програмне забезпечення для автоматизованого парсингу та аналізу спеціалізованого контенту.

Методи дослідження: методи моделювання, методи проектування та розробки веб-застосунків, методи об’єктно-орієнтованого програмування.

У ході роботи було проведено аналіз існуючих методів та інструментів для парсингу текстового контенту, таких як Django, Django REST Framework, Selenium, NLTK, spaCy. Проаналізовано наявні рішення для візуалізації даних і обробки текстів. На основі проведеного аналізу було розроблено функціональні та нефункціональні вимоги до програмного забезпечення.

Здійснено проектування моделі веб-застосунку з використанням UML-діаграм. Для розробки було обрано такі технології: Python для бекенду, React для фронтенду, PostgreSQL для зберігання даних, з використанням Docker для контейнеризації бази даних.

Було розроблено веб-застосунок, який дозволяє користувачам вводити URL для парсингу, отримувати та аналізувати текстовий контент, створювати візуалізації, такі як хмари слів та гістограми. Застосунок забезпечує збереження результатів парсингу для подальшого використання та аналізу.

Галузь використання – автоматизований парсинг та аналіз текстового контенту, що може бути корисним для дослідників, аналітиків та інших фахівців, які працюють з великими обсягами текстових даних.

Ключові слова: Python, Django, REST Framework, Selenium, NLTK, spaCy, PostgreSQL, Docker, React, парсинг, аналіз тексту, візуалізація даних.

ЗМІСТ

ВСТУП

Веб-застосунки для парсингу текстового контенту відіграють значну роль у сучасному цифровому світі. Вони забезпечують можливість автоматизованого витягання, аналізу та обробки текстової інформації з веб-сторінок, що є критично важливим для різних галузей, таких як наука, медицина, фінанси та маркетинг. Завдяки таким застосункам користувачі можуть ефективно збирати і аналізувати великі обсяги даних, що допомагає приймати обґрунтовані рішення на основі отриманих результатів.

Об'єктом дослідження є веб-застосунок для парсингу спеціалізованого контенту, розроблений мовою Python з використанням Javascript бібліотеки React.

Предметом дослідження є програмне забезпечення для автоматизації процесу парсингу та обробки текстових даних з веб-сторінок, а також їх візуалізація.

Метою дослідження є створення веб-застосунку, який дозволить користувачам парсити текстовий контент зі спеціалізованих веб-ресурсів, обробляти його і візуалізувати ключові слова та сутності.

Для досягнення мети дослідження були поставлені наступні завдання:

— аналіз методів і засобів для парсингу текстового контенту;
— порівняння аналогічних інструментів для парсингу та візуалізації текстових даних;
— вибір засобів реалізації веб-застосунку для парсингу спеціалізованого контенту;
— розробка вимог до веб-застосунку;
— створення веб-застосунку відповідно до визначених вимог;
— тестування програмного забезпечення.

Ця робота спрямована на розробку ефективного інструменту для автоматизації процесу збору, обробки та візуалізації текстових даних, що сприятиме підвищенню ефективності роботи з інформацією у спеціалізованих галузях.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Аналіз засобів та методів для парсингу спеціалізованого текстового контенту

Парсинг спеціалізованого текстового контенту, такого як наукові статті, є важливим завданням для збору та обробки текстових даних. Сучасні технології пропонують різноманітні засоби та методи для автоматизації цього процесу. Основні інструменти для парсингу можна поділити на три категорії: бібліотеки для роботи з HTML та XML, фреймворки для парсингу та веб-скрейпінгу, а також браузерні автоматизаційні інструменти.

Серед бібліотек для роботи з HTML та XML найпопулярнішим є BeautifulSoup. BeautifulSoup дозволяє легко та ефективно витягувати дані з HTML та XML документів, забезпечуючи простий інтерфейс для навігації по структурі документу.

Scrapy є одним з найвідоміших фреймворків для веб-скрейпінгу. Він надає широкий набір інструментів для створення скрейперів, управління ними та обробки зібраних даних. Scrapy дозволяє налаштовувати обхід веб-сторінок, видобування інформації та зберігання даних в різних форматах.

Одним з найпотужніших інструментів для автоматизації веб-браузера є Selenium. Selenium дозволяє автоматизувати взаємодію з веб-сторінками, виконуючи дії, які зазвичай виконує користувач, такі як клікання по елементах, заповнення форм та навігація між сторінками. Це особливо корисно для парсингу динамічного контенту, що завантажується за допомогою JavaScript, або для роботи з сайтами, що вимагають аутентифікації.

Для парсингу спеціалізованого контенту було обрано Selenium. Це обумовлено декількома ключовими факторами:

— Динамічний контент: Багато сучасних веб-сайтів використовують JavaScript для динамічного завантаження контенту. Selenium дозволяє ефективно

обробляти такі сайти, забезпечуючи коректне відображення та взаємодію з усіма елементами сторінки.

— Взаємодія з формами та навігація: Для доступу до певного контенту, наприклад, для виконання пошукових запитів або навігації по сторінках статей, необхідно автоматизувати взаємодію з веб-формами та кнопками. Selenium забезпечує цю функціональність.

Таким чином, використання Selenium для парсингу спеціалізованого текстового контенту дозволяє забезпечити високу ефективність та точність видобування даних, що є критичним для наукових досліджень та аналізу.

1.2. Аналіз аналогів для парсингу та візуалізації текстових даних

Для ефективного парсингу та візуалізації текстових даних існує ряд інструментів, які пропонують різноманітні можливості та функції. У цьому розділі буде розглянуто три основні інструменти: Orange, Voyant Tools та ParseHub.

Orange — це програмний пакет для аналізу даних та машинного навчання з відкритим кодом. Він надає візуальне програмне середовище для аналізу даних за допомогою блок-схем, що робить його доступним для користувачів без глибоких знань у програмуванні. Orange підтримує роботу з великим обсягом даних та включає модулі для текстового аналізу та візуалізації.

Переваги:

— Інтуїтивний інтерфейс: Orange має простий у використанні інтерфейс, що дозволяє користувачам швидко створювати та налаштовувати аналізи за допомогою блоків, які представляють різні методи та операції.

— Широкі можливості візуалізації: Пакет надає різноманітні інструменти для візуалізації даних, включаючи діаграми, графіки, дерева рішень та кластери.

— Розширюваність: Завдяки своїй модульній архітектурі, Orange легко розширюється за рахунок додавання нових компонентів та модулів.

Недоліки:

— Вимоги до ресурсів: Orange може вимагати значних обчислювальних ресурсів для обробки великих обсягів даних.

— Обмеження у текстовому аналізі: Хоча Orange включає модулі для текстового аналізу, його можливості в цій області можуть бути обмеженими порівняно з спеціалізованими інструментами.

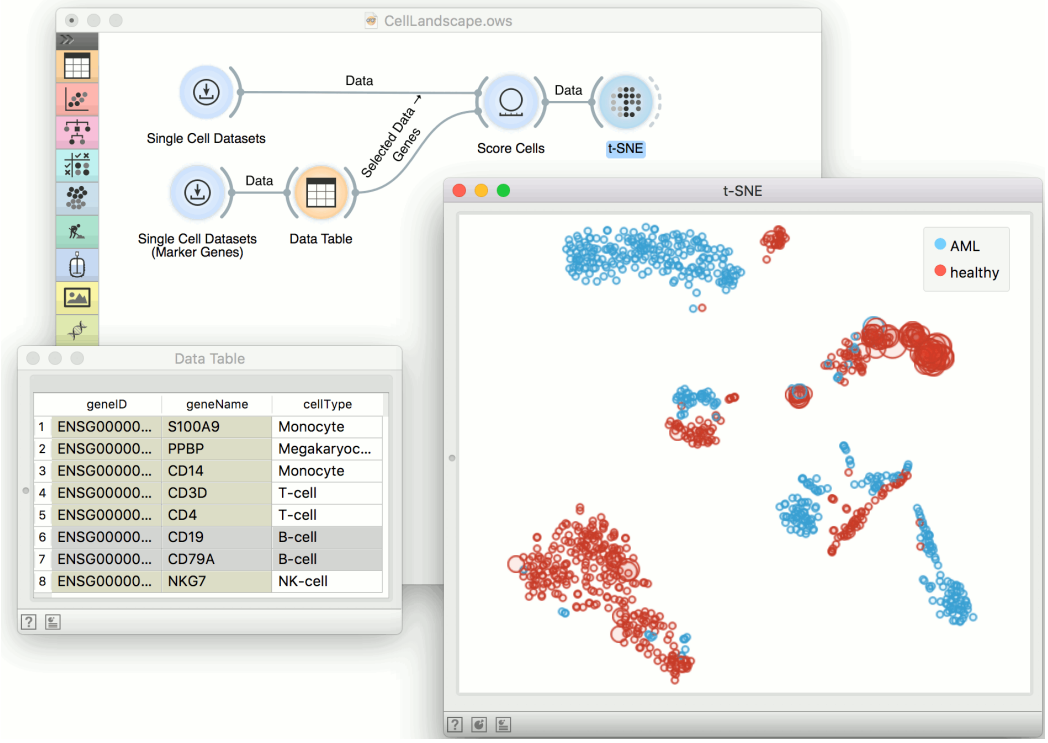


Рис 1.1 Приклад екранних форм додатку Orange

Voyant Tools — це веб-інструмент для аналізу та візуалізації текстових даних, спеціально розроблений для гуманітарних досліджень. Він дозволяє користувачам завантажувати тексти та аналізувати їх за допомогою різноманітних інтерактивних візуалізацій та статистичних методів.

Переваги:

— Доступність: Voyant Tools є веб-застосунком, що робить його доступним з будь-якого місця без необхідності встановлення додаткового програмного забезпечення.

— Інтерактивність: Інструмент пропонує інтерактивні візуалізації, які дозволяють користувачам досліджувати текстові дані у реальному часі.

— Простота використання: Інтуїтивний інтерфейс та готові до використання аналітичні модулі роблять Voyant Tools зручним для користувачів без спеціалізованих знань у програмуванні чи статистиці.

Недоліки:

— Обмеженість у можливостях: Voyant Tools орієнтований переважно на простий текстовий аналіз та візуалізацію, що може бути недостатньо для складних аналітичних задач.

— Вимоги до інтернет-з'єднання: Як веб-застосунок, Voyant Tools потребує стабільного інтернет-з'єднання для роботи.

Рис 1.2 Приклад екранних форм додатку Voyant Tools

ParseHub — це інструмент для веб-скрейпінгу, що дозволяє користувачам витягувати дані з веб-сторінок за допомогою візуального інтерфейсу.

Він підтримує роботу з динамічними сайтами, що використовують JavaScript, та дозволяє автоматизувати процес парсингу.

Переваги:

- Візуальний інтерфейс: ParseHub надає користувачам можливість створювати проекти для скрейпінгу даних за допомогою візуального інтерфейсу, що спрощує процес налаштування навіть для тих, хто не має технічних знань.
- Підтримка динамічного контенту: Інструмент може працювати з сайтами, що використовують JavaScript для динамічного завантаження контенту, забезпечуючи коректне витягування необхідних даних.
- Гнучкість та масштабованість: ParseHub дозволяє налаштовувати складні сценарії парсингу, включаючи логіку переходів між сторінками та фільтрацію даних, а також підтримує роботу з великими обсягами інформації.

Недоліки:

- Обмежені можливості візуалізації: Хоча ParseHub є потужним інструментом для парсингу, він не пропонує розширених можливостей для візуалізації зібраних даних, тому для цього можуть знадобитися додаткові інструменти.
- Вартість: ParseHub має обмежену безкоштовну версію, а для доступу до більш просунутих функцій та обробки великих обсягів даних може знадобитися платна підписка.

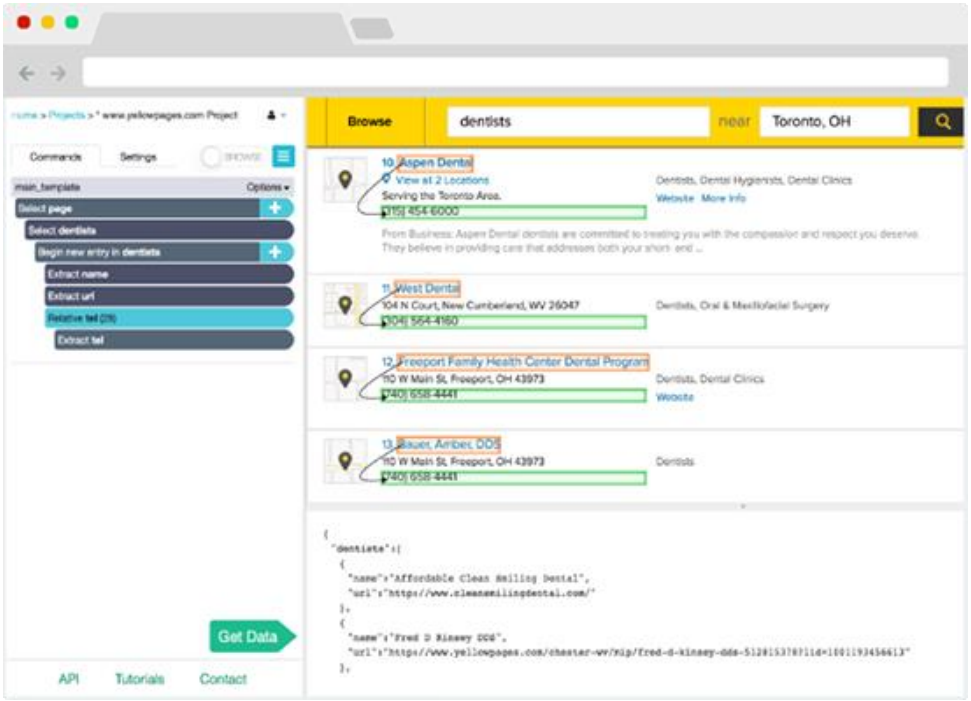


Рис 1.3 Приклад екранних форм додатку ParseHub

Зведені результати аналізу характеристик розглянутих додатків наведено у таблиці 1.1.

Таблиця 1.1

Зведені результати аналізу характеристик додатків для парсингу та візуалізації текстових даних

Показник	Orange	Voyant Tools	ParseHub
Тип інструменту	Пакет для аналізу даних та машинного навчання	Веб-інструмент для текстового аналізу	Інструмент для веб-скрейпінгу
Інтерфейс	Візуальний, блок-схеми	Веб-інтерфейс, інтерактивні візуалізації	Візуальний інтерфейс для налаштування парсингу

Продовження таблиці 1.1

Зведені результати аналізу характеристик додатків для парсингу та візуалізації текстових даних

Показник	Orange	Voyant Tools	ParseHub
Підтримка динамічного контенту	Обмежена	Ні	Так
Можливості візуалізації	Широкі можливості, включаючи графіки, діаграми	Основні інтерактивні візуалізації	Відсутні
Платформа	Десктоп	Веб	Веб
Підтримка великих обсягів даних	Так	Обмежена	Так
Вартість	Безкоштовний	Безкоштовний	Безкоштовна версія з обмеженнями, платна підписка для розширених можливостей

1.3. Визначення функціональних та нефункціональних вимог

Аналіз функціональних та нефункціональних вимог є важливим етапом у процесі розробки веб-застосунку для парсингу спеціалізованого контенту. Вимоги охоплюють різні аспекти, такі як продуктивність, безпека даних, можливість обробки великих обсягів інформації та інтеграція з іншими системами. Глибоке

розуміння потреб користувачів і особливостей предметної області дозволяє визначити критичні функціональні можливості застосунку.

Функціональні вимоги:

1. **Можливість парсити текст за вказаним URL:**
   - Користувач повинен мати можливість вказати URL для парсингу текстового контенту.
2. **Автоматичне визначення та виділення іменних сутностей:**
   - Система повинна автоматично визначати та виділяти іменні сутності, такі як особи, організації, локації тощо, в обробленому тексті.
3. **Відображення хмари слів:**
   - Система повинна створювати хмару слів, яка візуально репрезентує частоту ключових слів у тексті.
4. **Будування гістограм:**
   - Застосунок повинен мати можливість будувати гістограми, які відображають частоту найбільш вживаних слів у тексті.
5. **Збереження результатів парсингу:**
   - Система повинна забезпечувати збереження результатів парсингу, обробки та аналізу тексту для подальшого використання.

Нефункціональні вимоги:

1. **Можливість обробляти текстові дані:**
   - Система повинна бути здатною обробляти великі обсяги текстових даних без втрати продуктивності.
2. **Забезпечення безпеки даних:**
   - Система повинна забезпечувати захист даних користувачів та захист від несанкціонованого доступу.

Ці вимоги допоможуть створити ефективний веб-застосунок для парсингу спеціалізованого контенту, який відповідатиме сучасним стандартам та потребам користувачів.

2. ЗАСОБИ ТА МЕТОДИ РОЗРОБКИ ВЕБ ДОДАТКУ ДЛЯ ПАРСИНГУ СПЕЦІАЛІЗОВАНОГО КОНТЕНТУ

2.1. Мови програмування

Мови програмування є фундаментальною складовою у розробці програмного забезпечення, а правильний вибір мови має велике значення для успішної реалізації проекту. Кожна мова програмування відрізняється своїми особливостями, синтаксисом та парадигмами, що визначають їх придатність для різних типів завдань та проектів.

Для розробки Web-застосунку для парсингу спеціалізованого контенту використовуються різні мови програмування, які відповідають різним аспектам проекту. У випадку з backend-розробкою, мова Python займає чільне місце завдяки своїй гнучкості та широкому набору бібліотек, що полегшують процес парсингу даних. Python забезпечує потужні інструменти для роботи з різними типами даних, що є критично важливим для проектів, які потребують збору, обробки та аналізу великих обсягів інформації.

Frontend-розробка в цьому проекті реалізується за допомогою Javascript, зокрема бібліотеки React. Javascript є мовою програмування, яка широко використовується для створення інтерактивних і динамічних елементів веб-сторінок. React, у свою чергу, забезпечує компонентний підхід до розробки користувацького інтерфейсу, що дозволяє створювати модульні та повторно використовувані компоненти.

Вибір Python та Javascript обґрунтований не лише їхніми технічними можливостями, а й активною підтримкою спільноти та наявністю численних ресурсів для навчання та розвитку. Спільноти цих мов регулярно оновлюють бібліотеки та фреймворки, що забезпечує безперервний розвиток та вдосконалення інструментів, доступних розробникам.

Крім того, важливим аспектом є екосистема кожної мови. Python має розвинену екосистему для роботи з даними, включаючи такі бібліотеки як BeautifulSoup, Scrapy та Pandas, які значно спрощують процес парсингу та обробки даних. Javascript з React забезпечує ефективну взаємодію з користувачем, дозволяючи створювати швидкі та інтерактивні веб-застосунки.

Таким чином, вибір мов програмування Python для backend та Javascript для frontend обумовлений їхньою здатністю ефективно вирішувати поставлені завдання в рамках проекту. Кожна з цих мов надає розробникам необхідні інструменти та можливості для реалізації функціоналу, який відповідає вимогам сучасного веб-розробки.

2.2. Фреймворки до обраних мов програмування

Фреймворки являють собою набір попередньо написаних та готових до використання програмних компонентів, які забезпечують структуру і складні функціональні можливості для розробки програмного забезпечення. Вони надають розробникам необхідні інструменти та ресурси для полегшення процесу розробки, встановлюють стандарти та сприяють підвищенню продуктивності.

Основною метою фреймворків є забезпечення високого рівня повторного використання коду та спрощення процесу розробки програмного забезпечення. Це дозволяє швидше і ефективніше створювати нові програми або розширювати існуючі. Фреймворки пропонують готові рішення для типових завдань, таких як маршрутизація, робота з базами даних, керування сесіями, автентифікація користувачів та багато інших аспектів, що робить процес розробки більш систематичним та керованим.

2.2.1. Django Rest Framework

Django Rest Framework (DRF) є потужним інструментом для розробки RESTful API з використанням мови програмування Python.

Він побудований на основі фреймворку Django, який забезпечує високий рівень інтеграції та надає широкий спектр можливостей для розробки складних веб-додатків. DRF дозволяє створювати ефективні API завдяки своїй архітектурі, яка підтримує модульність та розширюваність.

Однією з ключових особливостей DRF є його можливість автоматично генерувати API на основі моделей Django. Це дозволяє розробникам швидко налаштовувати і впроваджувати RESTful сервіси без необхідності написання великої кількості коду. Використання DRF також сприяє забезпеченню узгодженості та стандартизації API, що важливо для великих проектів з багатьма взаємодіючими компонентами.

DRF надає численні вбудовані інструменти для роботи з автентифікацією, дозволами та обробкою запитів. Це включає підтримку різних методів автентифікації, таких як токени, OAuth, та інші, що дозволяє легко інтегрувати систему безпеки в API. Крім того, система дозволів DRF забезпечує гнучке налаштування доступу до ресурсів, що є важливим для захисту даних та контролю доступу.

Ще однією перевагою DRF є його підтримка серіалізації даних, яка дозволяє легко перетворювати складні типи даних, такі як моделі Django, у формат JSON або інші формати, що використовуються для обміну даними через API. Це значно спрощує процес передачі даних між клієнтом та сервером, забезпечуючи ефективний і зрозумілий спосіб обробки даних.

Активна спільнота та обширна документація DRF роблять його привабливим вибором для розробників. Спільнота забезпечує регулярні оновлення та підтримку, що допомагає швидко вирішувати виникаючі проблеми та впроваджувати нові функціональні можливості. Документація надає детальні інструкції та приклади використання, що полегшує процес навчання та розробки для новачків та досвідчених розробників.

Таким чином, Django Rest Framework є важливим інструментом для розробки RESTful API у проектах, які використовують мову Python. Його функціональність,

інтеграція з Django, підтримка автентифікації, серіалізації та активна спільнота роблять його незамінним для створення ефективних та безпечних веб-сервісів.

2.2.2. React

React є бібліотекою JavaScript, розробленою для створення користувацьких інтерфейсів, особливо односторінкових додатків, де ефективність і швидкість оновлення інтерфейсу є критичними. Заснований на компонентній архітектурі, React дозволяє розробникам створювати багаторазові компоненти, що сприяє більш організованій та структурованій розробці.

Одна з ключових переваг React полягає в його здатності до оновлення тільки тих частин сторінки, які зазнали змін, замість перезавантаження всієї сторінки. Це досягається завдяки використанню віртуального DOM, який мінімізує операції з реальним DOM і таким чином підвищує продуктивність додатків. Віртуальний DOM створює легку копію реального DOM і порівнює її зі змінами, що дозволяє ефективно керувати оновленнями.

React також підтримує односпрямовану передачу даних, що сприяє передбачуваності і спрощує процес відстеження змін стану додатка. Це означає, що дані передаються від батьківських компонентів до дочірніх, що полегшує розуміння і керування станом додатка. Для більш ефективного управління станом складних додатків часто використовуються додаткові бібліотеки, такі як Redux або MobX.

Крім того, React має потужну екосистему та активну спільноту, що забезпечує розробників численними інструментами і бібліотеками для розширення функціональності. Серед них можна виділити React Router для управління маршрутизацією і React Native для розробки мобільних додатків. Це робить React універсальним інструментом для створення як веб-додатків, так і мобільних додатків з єдиною кодовою базою.

React також підтримує JSX - розширення синтаксису JavaScript, яке дозволяє використовувати HTML-подібний код у JavaScript-файлах. Це полегшує розробникам створення компонентів інтерфейсу, об'єднуючи логіку та

представлення в одному місці. JSX дозволяє використовувати повну силу JavaScript в поєднанні з можливістю створення структурованих та чітко зрозумілих шаблонів.

Завдяки своїй модульності, ефективності оновлень, передбачуваності управління станом і підтримці широкої екосистеми, React є важливим інструментом для розробки сучасних веб-додатків. Його можливості дозволяють створювати динамічні та інтерактивні інтерфейси, які відповідають високим стандартам продуктивності та зручності використання.

2.2.3. Інструменти розробки

Для розробки сучасних веб-застосунків важливим є вибір правильного інструментарію, що забезпечує ефективний робочий процес і підтримує всі необхідні функції для продуктивної роботи. Одним з таких інструментів є інтегроване середовище розробки (IDE) PyCharm Professional(рис. 2.1), розроблене компанією JetBrains. Це потужне IDE призначене для розробки на мові Python, але також підтримує безліч інших мов програмування і технологій.

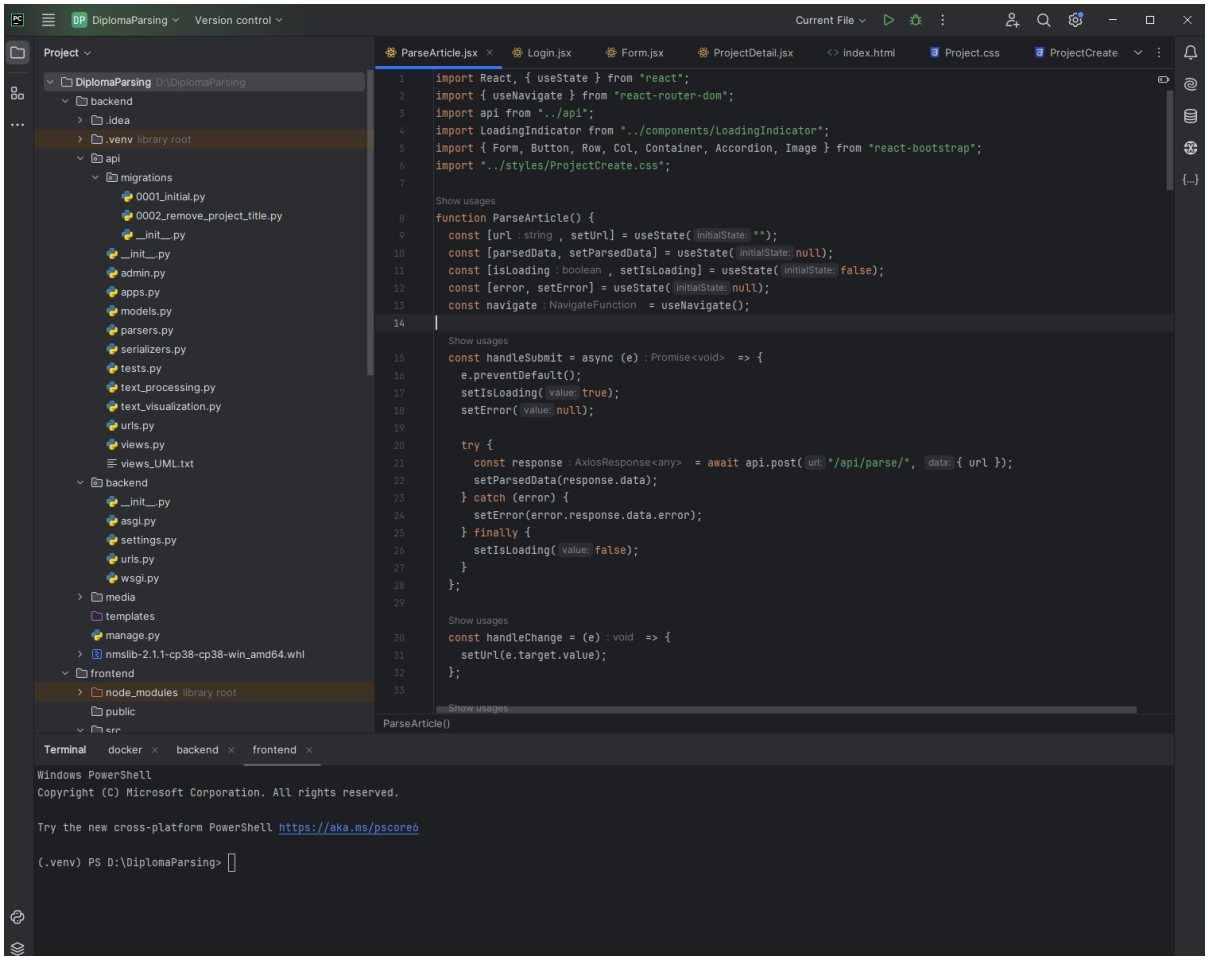


Рис 2.1 Інтерфейс IDE PyCharm Professional

PyCharm Professional надає широкий спектр функцій, які значно полегшують процес розробки. Серед них інтелектуальне завершення коду, що допомагає швидше писати код, а також інструменти для рефакторингу, які дозволяють ефективно перебудовувати і оптимізувати кодову базу. Навігація по проекту реалізована таким чином, що розробники можуть швидко знаходити необхідні файли, класи або методи, що значно прискорює робочий процес.

Це IDE також підтримує інтеграцію з різними системами контролю версій, такими як Git, що забезпечує легке керування версіями коду та спільну роботу над проектами. PyCharm Professional має вбудовані інструменти для налагодження і тестування, що дозволяє розробникам легко виявляти та виправляти помилки в коді. Налагоджувач підтримує роботу з різними типами проектів і забезпечує глибокий аналіз та відстеження виконання програми.

Однією з важливих особливостей PyCharm Professional є його підтримка веб-розробки, включаючи роботу з HTML, CSS, JavaScript і сучасними фреймворками, такими як React. Це робить його універсальним інструментом для повного циклу розробки веб-застосунків, від створення серверної логіки до реалізації клієнтської частини. PyCharm також підтримує численні плагіни, які дозволяють розширювати його функціональність і адаптувати під конкретні потреби проекту.

PyCharm Professional забезпечує розробників всіма необхідними інструментами для роботи з базами даних, включаючи інтеграцію з різними СУБД, можливість виконання SQL-запитів, візуалізацію даних та управління схемами баз даних. Це значно спрощує процес розробки додатків, які взаємодіють з базами даних, і дозволяє ефективно керувати даними без необхідності використовувати окремі інструменти.

Таким чином, PyCharm Professional є комплексним рішенням для розробників, яке забезпечує повний набір інструментів для створення, тестування і налагодження програмного забезпечення. Його функціональні можливості, підтримка різних мов і технологій, а також інтеграція з базами даних роблять його ідеальним вибором для розробки сучасних веб-застосунків.

2.3. База даних та Docker

PostgreSQL є популярною системою керування реляційними базами даних, яка відзначається своєю багатофункціональністю та відповідністю стандартам. Вона надає розробникам потужні інструменти для роботи зі складними типами даних та підтримує транзакції, сумісні з ACID, що забезпечує цілісність і надійність даних. sОднією з ключових переваг PostgreSQL є її здатність ефективно масштабуватися і підтримувати високу продуктивність у багатокористувацьких середовищах.

Використання Docker(рис 2.2) для розгортання PostgreSQL додає додатковий рівень гнучкості та ефективності. Docker є платформою контейнеризації, яка дозволяє розробникам упакувати додаток та його залежності в ізольовані

контейнери. Це забезпечує консистентність середовища розробки, тестування та виробництва, зменшуючи проблеми, пов'язані з різними конфігураціями системи.

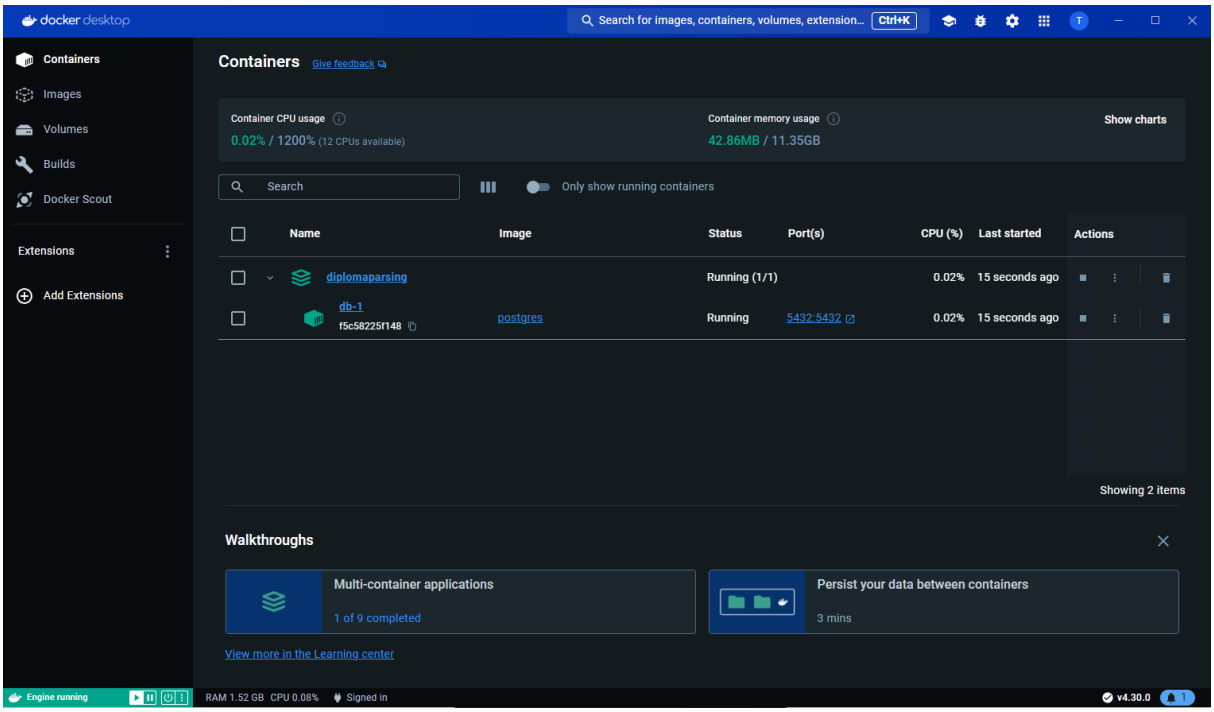


Рис 2.2 Екранні форми додатку Docker

PostgreSQL у контейнері Docker може бути швидко розгорнуто та легко масштабовано. Використання офіційного Docker-образу PostgreSQL дозволяє з легкістю налаштувати і запускати базу даних, знижуючи час налаштування середовища. Контейнери забезпечують ізоляцію процесів, що дозволяє запускати кілька інстансів бази даних на одному хості без ризику конфліктів.

Docker також спрощує процес резервного копіювання та відновлення баз даних PostgreSQL. Контейнери дозволяють легко створювати знімки стану бази даних, що забезпечує швидке і безпечне відновлення у випадку збоїв. Це особливо корисно у середовищах з високими вимогами до доступності та надійності даних.

Таким чином, використання PostgreSQL у поєднанні з Docker забезпечує ефективне, гнучке та надійне рішення для зберігання та управління даними у сучасних веб-застосунках. Контейнеризація дозволяє легко розгортати,

масштабувати та керувати базами даних, знижуючи витрати на інфраструктуру та підвищуючи загальну продуктивність системи.

2.4. Парсинг текстового спеціалізованого контенту

Парсинг текстового спеціалізованого контенту є важливим етапом у процесі автоматизованої обробки даних. Він передбачає витягування, структуризацію та перетворення даних з веб-сторінок або інших джерел у формат, придатний для подальшого аналізу та використання. Основним інструментом, який використовується для парсингу в даній роботі, є Selenium.

Selenium — це популярний інструмент для автоматизації веб-браузерів. Він дозволяє імітувати дії користувача на веб-сторінках, що робить його незамінним для парсингу динамічного контенту, який завантажується за допомогою JavaScript. Використовуючи Selenium, можна автоматизувати процес переходу між сторінками, заповнення форм, натискання кнопок та інші дії, необхідні для отримання потрібних даних.

Процес парсингу за допомогою Selenium включає кілька етапів. Спочатку необхідно визначити веб-сторінки, які містять потрібний контент. Потім налаштовується середовище Selenium, що включає встановлення драйверів для браузерів та налаштування параметрів сесії. Далі створюються сценарії, які виконують необхідні дії на веб-сторінках для витягування даних. Ці дані можуть бути збережені у вигляді текстових файлів або безпосередньо записані у базу даних для подальшого аналізу.

Однією з переваг використання Selenium є його гнучкість та можливість налаштування під конкретні завдання парсингу. З його допомогою можна витягувати дані з веб-сторінок різної складності та структури. Однак, варто враховувати, що Selenium може вимагати значних обчислювальних ресурсів, особливо при роботі з великими обсягами даних або складними веб-сайтами.

Парсинг текстового контенту за допомогою Selenium дозволяє автоматизувати процес збору даних, що значно спрощує та прискорює роботу з

великими обсягами інформації. Це особливо важливо для досліджень та аналізу спеціалізованого контенту, де вручну виконувати такі завдання було б занадто трудомістким та часозатратним. Завдяки можливостям Selenium, можна ефективно витягувати необхідні дані, забезпечуючи високу точність та надійність процесу парсингу.

2.5. Обробка текстового контенту

Обробка текстового контенту є ключовим етапом у процесі аналізу даних, що включає кілька важливих підпроцесів. Вона починається з попередньої обробки тексту, яка готує дані для подальшого аналізу. Цей процес включає різні кроки, такі як токенізація, лематизація, усунення стоп слів та очищення тексту від пунктуації. Кожен з цих кроків відіграє важливу роль у забезпеченні точності та ефективності подальшого аналізу. Ефективна обробка текстового контенту є необхідною для досягнення точних і корисних результатів у різних аналітичних завданнях.

2.5.1. Токенізація

Токенізація є першим та ключовим етапом у процесі обробки тексту.

Її мета – розбити вхідний текст на окремі частини, які називаються токенами. Токени можуть представляти різні одиниці тексту, такі як слова, речення, символи, або навіть більш спеціалізовані одиниці, залежно від потреб аналізу.

Токенізація перетворює текст на більш керовані компоненти, що спрощує подальшу обробку. Наприклад, розбиття тексту на окремі слова допомагає визначати частоту вживання певних слів, виявляти ключові терміни або проводити інші лінгвістичні операції.

Токенізація впливає на текст шляхом його поділу на менші частини, які називаються токенами. Кожен токен може мати власне значення та роль у контексті тексту. Наприклад, поділ речення на окремі слова дозволяє аналізувати семантику та структуру тексту.

Процес поділу тексту на токени може включати видалення розділових знаків, символів пунктуації та спеціальних символів, а також розділення слів за допомогою пробілів або інших правил. Наприклад, речення "Це складне завдання!" може бути розбите на окремі токени, такі як ["Це", "складне", "завдання", "!"].

Токенізація може варіюватися залежно від потреб та специфіки текстових даних. Вона може враховувати особливості мови, контексту або використовувати машинне навчання для кращої адаптації до конкретних завдань обробки тексту.

Після токенізації тексту отримуємо послідовність окремих токенів, яку можна використовувати для подальшого аналізу, лематизації, видалення стоп-слів та інших операцій обробки тексту. Важливо враховувати контекст та мету аналізу тексту при виборі підходу до токенізації, оскільки різні типи токенів можуть бути корисними в різних аспектах обробки тексту.

2.5.2. Лематизація

Лематизація - це процес нормалізації тексту, який зводить слова до їх базової форми, відомої як лема. Лема - це стандартна форма слова, що представляє всі його граматичні варіації.

Наприклад, слова "говорити", "говориш", "говорять" будуть приведені до леми "говорити". Лематизація дозволяє зменшити розмір словника та спрощує аналіз тексту, фокусуючи увагу на важливих аспектах. Цей процес важливий для розуміння семантичних зв'язків між словами, покращення контекстуального розуміння та підвищення якості результатів подальших завдань, таких як класифікація текстів, пошук інформації або аналіз настроїв. Лематизація допомагає створити єдине представлення для різних форм одного і того ж слова, що є критично важливим для точного аналізу.

Процес лематизації зазвичай використовує морфологічні правила, лексичні словники або машинні моделі, які співвідносять слова з їх базовими формами. Це може включати визначення частин мови, зміну закінчень, префіксів та суфіксів, а також інші морфологічні перетворення. Наприклад, речення "Вони читають книгу" буде лематизовано до "Вони читати книга".

Один з популярних підходів до лематизації базується на морфологічному аналізі. Такий підхід враховує граматичні правила та структуру слів для знаходження їх базової форми. Наприклад, для української мови цей підхід може використовувати правила суфіксів, закінчень та інші морфеми для приведення слів до їх основи.

Лематизація є необхідним етапом для подальшого аналізу тексту, такого як класифікація, витягування ключових слів або інформаційний пошук, оскільки дозволяє звести різні граматичні форми до єдиного представлення. Це забезпечує більш точне та ефективне опрацювання тексту, що є особливо важливим для спеціалізованих завдань обробки текстової інформації.

2.5.3. Усунення стоп слів та очищення тексту від пунктуації

Усунення стоп слів та очищення тексту від пунктуації є важливими етапами попередньої обробки тексту. Стоп слова — це поширені слова, такі як "і", "в", "на", які не несуть значного смислового навантаження і можуть бути пропущені без втрати змісту тексту. Видалення стоп слів допомагає зменшити обсяг тексту, зосереджуючи увагу на більш важливих термінах, що полегшує подальший аналіз.

Процес усунення стоп слів починається з визначення списку таких слів, який може бути специфічним для кожної мови або навіть окремої галузі знань. Після цього текст проходить через фільтр, який видаляє всі слова, що входять до цього списку. Це дозволяє зменшити шум у даних і покращити точність таких завдань, як тематичне моделювання, класифікація або пошук інформації.

Очищення тексту від пунктуації також є важливим етапом, який допомагає уніфікувати текстовий контент і зробити його більш придатним для аналізу. Пунктуація, така як крапки, коми, дужки, несе структурну, але не смислову інформацію, тому її видалення допомагає зосередитися на суті тексту. Наприклад, речення "Доброго дня, як ви сьогодні?" після очищення виглядатиме як "Доброго дня як ви сьогодні".

Процес очищення тексту від пунктуації включає в себе ідентифікацію та видалення всіх символів пунктуації з тексту. Це може бути здійснено за допомогою регулярних виразів або спеціальних функцій.

Усунення стоп слів та очищення тексту від пунктуації значно покращують якість даних для подальшого аналізу, знижуючи обсяг та складність тексту. Це допомагає алгоритмам машинного навчання та іншим аналітичним методам працювати ефективніше, зосереджуючись на значущих аспектах текстового контенту.

2.5.4. Вибір сутностей та ключових слів

Вибір сутностей та ключових слів є важливим етапом обробки текстового контенту, що дозволяє виділити найбільш значущі елементи тексту для подальшого аналізу. Сутності можуть бути іменами власними, датами, місцями, організаціями тощо, які мають конкретне значення в контексті тексту. Ключові слова, у свою чергу, є важливими термінами, що відображають основний зміст тексту.

Процес вибору сутностей полягає у виявленні та класифікації іменованих сутностей у тексті. Це завдання може бути виконане за допомогою методів обробки природної мови (NLP), таких як частинно-мовний аналіз та машинне навчання. Наприклад, в реченні "Apple Inc. оголосила про новий продукт у Каліфорнії" сутностями будуть "Apple Inc." та "Каліфорнія". Виділення сутностей дозволяє більш точно ідентифікувати ключові об'єкти та події в тексті, що значно покращує якість подальшого аналізу.

Вибір ключових слів здійснюється шляхом аналізу частоти вживання слів у тексті та їх відносної важливості. Алгоритми, такі як TF-IDF (Term Frequency-Inverse Document Frequency), дозволяють визначити слова, які мають найбільшу вагу в конкретному документі відносно до всього корпусу текстів. Ключові слова допомагають швидко зрозуміти основну тему документа і використовуються для різних завдань, таких як класифікація текстів, тематичне моделювання та пошукова оптимізація.

Для виявлення сутностей та ключових слів можуть бути використані різні інструменти та бібліотеки, такі як NLTK та SpaCy. Ці інструменти забезпечують широкі можливості для аналізу тексту, включаючи токенізацію, лематизацію, частинно-мовний аналіз та інші методи NLP. Наприклад, бібліотека SpaCy забезпечує ефективні алгоритми для розпізнавання іменованих сутностей та виявлення ключових слів, що дозволяє автоматизувати процес аналізу великих обсягів текстових даних.

Вибір сутностей та ключових слів є критичним етапом обробки тексту, оскільки дозволяє виділити найбільш релевантні елементи для подальшого аналізу. Це сприяє покращенню точності та ефективності різних аналітичних завдань, забезпечуючи глибше розуміння змісту текстового контенту.

2.5.5. Візуалізація отриманих результатів

Візуалізація отриманих результатів є важливим етапом аналізу текстового контенту, що дозволяє наочно представити дані та спростити їх інтерпретацію. Після виконання всіх етапів обробки тексту, таких як токенізація, лематизація, усунення стоп слів та вибір ключових слів, отримані дані можуть бути візуалізовані за допомогою різних методів.

Одним з основних методів візуалізації є виділення сутностей у тексті(рис. 2.3). Цей метод дозволяє підкреслити імена, дати, місця та інші значущі елементи тексту, що полегшує їх розпізнавання та аналіз. Виділення сутностей забезпечує зрозуміле представлення ключових компонентів тексту, дозволяючи користувачам швидко ідентифікувати важливу інформацію.

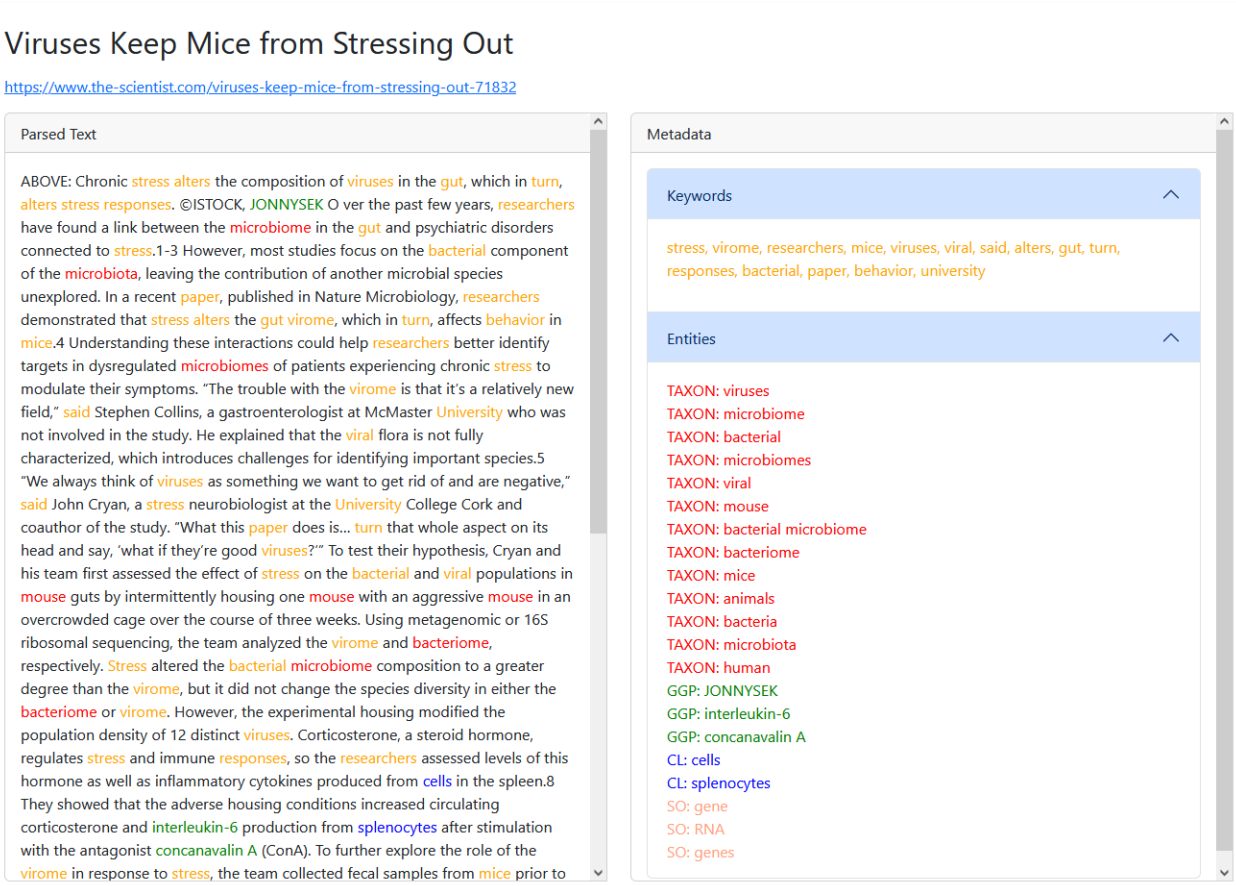


Рис 2.3 Виділення ключових сутностей

Ще одним популярним методом візуалізації є створення хмари слів. Хмара слів є графічним зображенням, в якому найбільш часто вживані слова представлені у вигляді різних за розміром та кольором шрифтів. Чим частіше слово зустрічається у тексті, тим більшим воно буде у хмарі слів. Це дозволяє легко визначити основні теми та ключові слова тексту. Наприклад, якщо аналізується стаття про сучасні технології, у хмарі слів будуть домінувати такі терміни, як "штучний інтелект", "автоматизація", "машинне навчання".

Для створення хмари слів(рис. 2.4) можна використовувати різні інструменти, такі як WordCloud у Python. Цей інструмент дозволяє швидко та ефективно генерувати хмари слів з текстових даних, забезпечуючи наочне представлення інформації.

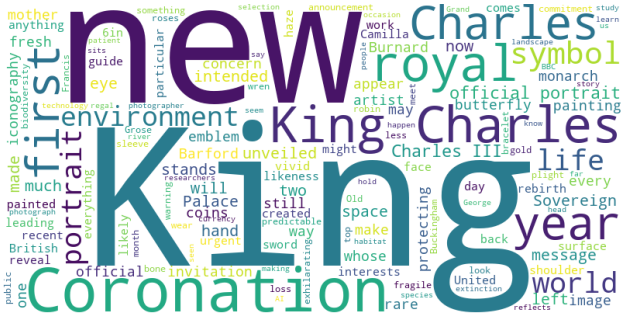


Рис 2.4 Хмара слів

Також популярним методом візуалізації є створення гістограми найбільш вживаних слів. Гістограма є графічним зображенням, яке представляє частоту вживання кожного слова у вигляді стовпчиків. Чим частіше слово зустрічається у тексті, тим вищим буде стовпчик на гістограмі. Це дозволяє легко визначити основні теми та ключові слова тексту.

Наприклад, на гістограмі найбільш вживаних слів у певному тексті (рис. 2.4) можна побачити, що слово "stress" зустрічається найчастіше, за ним йдуть "virome" та "researchers". Така візуалізація допомагає швидко зрозуміти, які терміни домінують у тексті та які теми є центральними.

Для створення гістограми можна використовувати різні інструменти, такі як Matplotlib у Python. Цей інструмент дозволяє швидко та ефективно генерувати гістограми з текстових даних, забезпечуючи наочне представлення інформації. Гістограми можуть бути корисними для аналізу текстів у різних сферах, від академічних досліджень до аналізу відгуків клієнтів.

Таким чином, використання гістограм для візуалізації найбільш вживаних слів є потужним інструментом для аналізу тексту, який допомагає зрозуміти структуру та зміст текстових даних. Це дозволяє більш ефективно працювати з

великими обсягами інформації та приймати обґрунтовані рішення на основі отриманих даних.

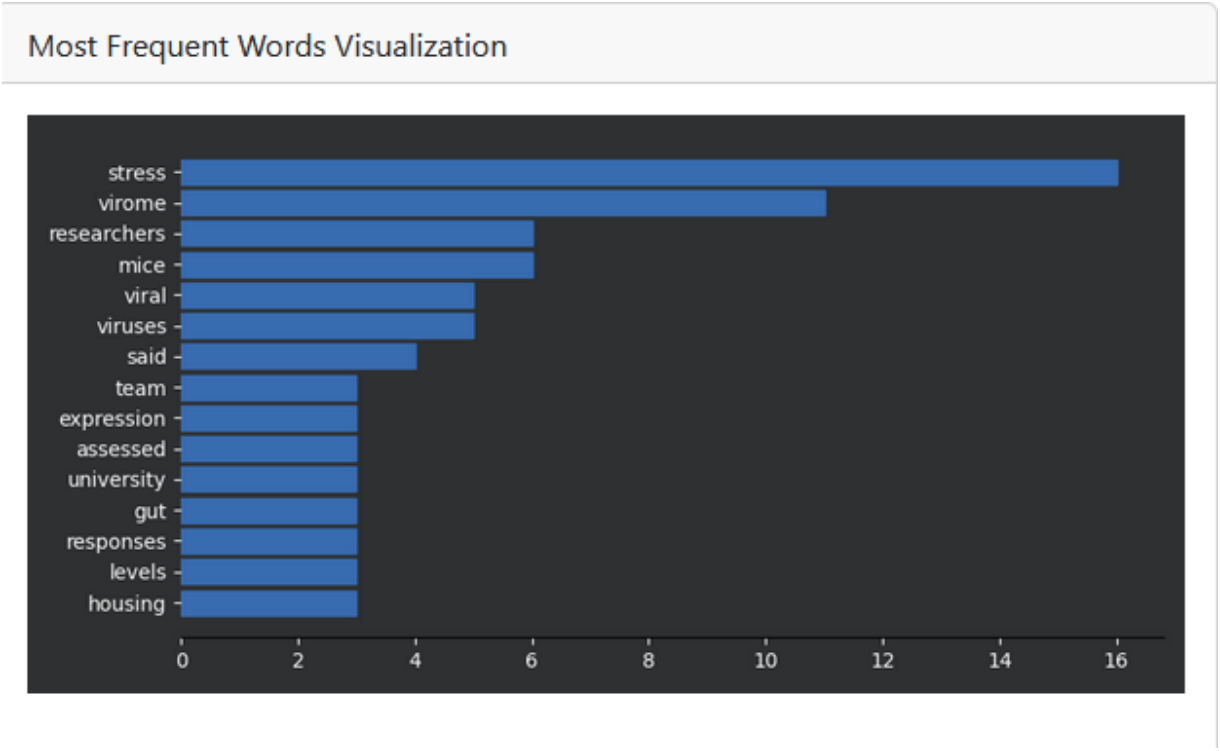


Рис 2.5 Гістограма найбільш вживаних слів

Загалом, візуалізація отриманих результатів є важливим етапом аналізу текстового контенту, який допомагає спростити інтерпретацію даних та представити результати у зрозумілій та наочній формі.

3. ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Етапи розробки системи

Першим етапом розробки системи був збір вимог до веб-застосунку. На цьому етапі проводився детальний аналіз потреб користувачів і визначення основної функціональності системи. Основна мета цього етапу полягала в тому, щоб створити чітке розуміння того, які завдання повинен виконувати веб-застосунок, які особливості та інструменти повинні бути інтегровані для досягнення ефективності та зручності використання.

Наступним кроком стало проектування архітектури системи. Визначалися основні компоненти веб-застосунку та їхня взаємодія між собою. Було створено модульну архітектуру, яка передбачає розподіл функціональності на фронтенд і бекенд частини. Фронтенд реалізовувався за допомогою Javascript бібліотеки React, а бекенд - мовою Python з використанням Django Rest Framework для створення API. Це дозволило забезпечити гнучкість, масштабованість і легкість підтримки веб-застосунку.

На етапі реалізації компонентів проводилася безпосередня розробка кожного модуля системи. Були реалізовані функції парсингу веб-сторінок, обробки тексту, видалення стоп-слів та візуалізації результатів. Кожен компонент мав свою внутрішню логіку обробки даних і був створений з використанням відповідних бібліотек та інструментів, таких як Selenium для парсингу, NLTK та spacy для обробки тексту.

Після завершення розробки окремих компонентів відбувалася їх інтеграція в єдину систему. Це включало налаштування взаємодії між фронтендом і бекендом, перевірку коректності передачі даних і забезпечення безперебійної роботи всіх частин веб-застосунку. Інтеграція була важливим етапом для забезпечення цілісності та узгодженості роботи всіх компонентів.

Завершальним етапом було тестування веб-застосунку. Тестування включало виявлення та виправлення помилок, перевірку роботи системи на різних сценаріях використання, а також оптимізацію продуктивності. Особлива увага приділялася коректності парсингу тексту, точності видалення стоп-слів і якості візуалізації ключових слів та хмари слів. Ретельне тестування забезпечило високу надійність та ефективність роботи веб-застосунку у реальних умовах.

3.2. Моделювання вимог до програмного забезпечення

Для моделювання вимог до веб-застосунку для парсингу спеціалізованого контенту було використано засоби мови UML.

3.2.1. Діаграма варіантів використання

На рис. 3.1 наведено діаграму прецедентів, яка описує основні можливості, доступні користувачеві системи.

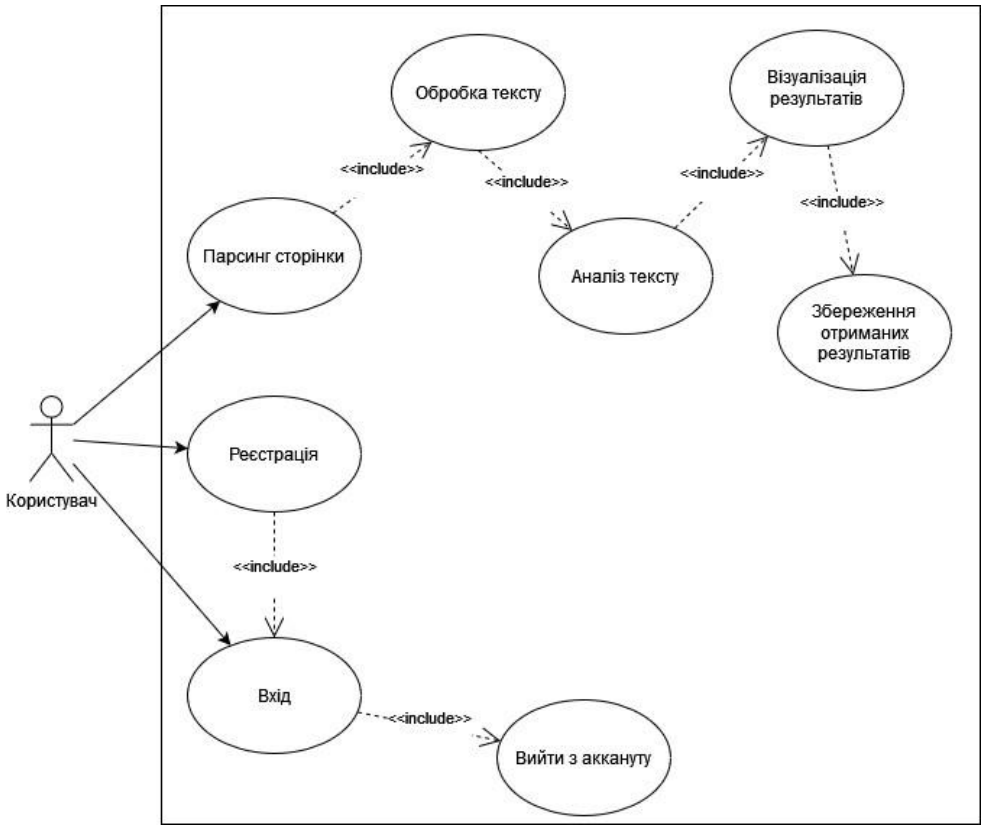


Рис 3.1 Діаграма прецедентів

Основні дії, доступні користувачеві системи, включають нижченаведені:

— **Реєстрація**: користувач має можливість створити обліковий запис для доступу до функціоналу веб-застосунку.

— **Вхід**: авторизація користувача у системі для отримання доступу до основних можливостей.

— **Парсинг сторінки**: користувач вводить посилання на статтю, яку потрібно парсити, і веб-застосунок завантажує та обробляє її.

— **Обробка тексту**: система здійснює обробку тексту, включаючи видалення стоп-слів та очищення від пунктуації.

— **Аналіз тексту**: після обробки тексту проводиться його аналіз, виявляються ключові слова та сутності.

— **Візуалізація результатів**: результати аналізу візуалізуються, зокрема підсвічуються ключові слова та створюється хмара слів для наочного представлення даних.

— **Збереження отриманих результатів**: користувач може зберегти результати аналізу для подальшого використання.

— **Вийти з акаунту**: користувач може вийти зі свого облікового запису, завершивши роботу в системі.

Ця діаграма відображає ключові функціональні можливості веб-застосунку та взаємодію користувача з різними його компонентами. Моделювання вимог на основі UML дозволяє краще зрозуміти структуру та функціональність системи, що сприяє ефективному плануванню та розробці веб-застосунку

3.2.2. Діаграма класів

Для реалізації бекенд-частини веб-застосунку, призначеного для парсингу спеціалізованого контенту, було розроблено набір взаємопов'язаних класів. На Рисунку 3.2 представлена діаграма класів, що відображає структуру та зв'язки між цими класами.

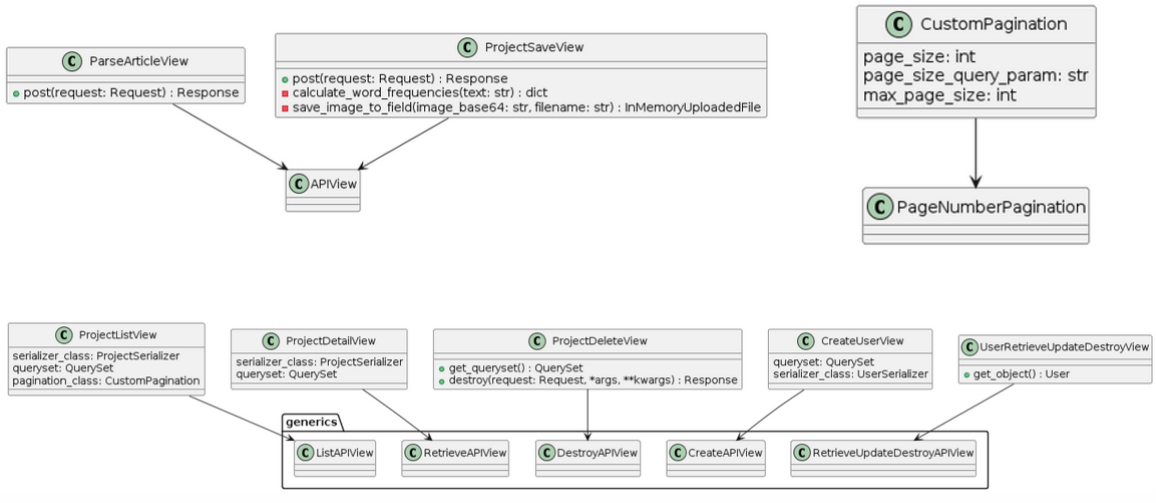


Рис 3.2 Діаграма класів

У верхній частині діаграми знаходяться класи, відповідальні за обробку HTTP-запитів. ParseArticleView обробляє POST-запити, що містять посилання на статті для парсингу. Клас ProjectSaveView зберігає оброблені дані, включаючи розраховані ключові слова та зображення хмари слів.

APIView виступає базовим класом для всіх API-представлень, реалізуючи спільну логіку обробки запитів. Від нього успадковуються класи ProjectListView, ProjectDetailView, ProjectDeleteView, що відповідають за роботу зі списками проектів, детальною інформацією про проекти та видаленням проектів відповідно.

Клас CreateUserView відповідає за реєстрацію нових користувачів, а UserRetrieveUpdateDestroyView - за отримання, редагування та видалення даних користувача.

Для реалізації пагінації списків проектів використовуються класи CustomPagination та PageNumberPagination.

Всі класи, що успадковуються від generics.ListAPIView, generics.RetrieveAPIView, generics.DestroyAPIView та generics.CreateAPIView, використовують фреймворк Django REST Framework для спрощення розробки API.

Дана структура класів забезпечує модульність, розширюваність та зручність підтримки коду бекенд-частини веб-застосунку.

3.2.3. Діаграма діяльності

Діаграма діяльності - це графічний інструмент моделювання, який використовується для візуалізації послідовності дій або процесів в системі, проекті або бізнес-процесі. Вона дозволяє детально описати кроки, дії та умови, які відбуваються під час виконання певного процесу або задачі.

На Рисунку 3.3 представлена діаграма діяльності, що візуалізує процес взаємодії користувача з додатком для парсингу спеціалізованого контенту.

Першим кроком є введення користувачем URL для парсингу. Користувач вводить відповідні дані у поле на головній сторінці та натискає кнопку "Парсити". Це ініціює процес передачі даних до серверної частини системи.

На серверній стороні спочатку відбувається отримання URL. Після цього здійснюється парсинг тексту, де система вилучає необхідний текстовий контент з наданого джерела.

Наступним етапом є обробка та аналіз тексту. Система обробляє зібрану інформацію, аналізує її та готує дані для візуалізації.

Після завершення аналізу система створює візуалізації на основі оброблених даних. Ці візуалізації допомагають користувачеві зрозуміти структуру та зміст тексту більш наочно. Користувач може переглянути отриману хмару слів та гістограму найбільш вживаних слів.

На заключному етапі користувач має можливість переглянути результати. Якщо користувач авторизований, система дозволяє зберегти отримані дані для подальшого використання. У випадку, якщо користувач не авторизований, система повідомляє про необхідність авторизації для збереження результатів.

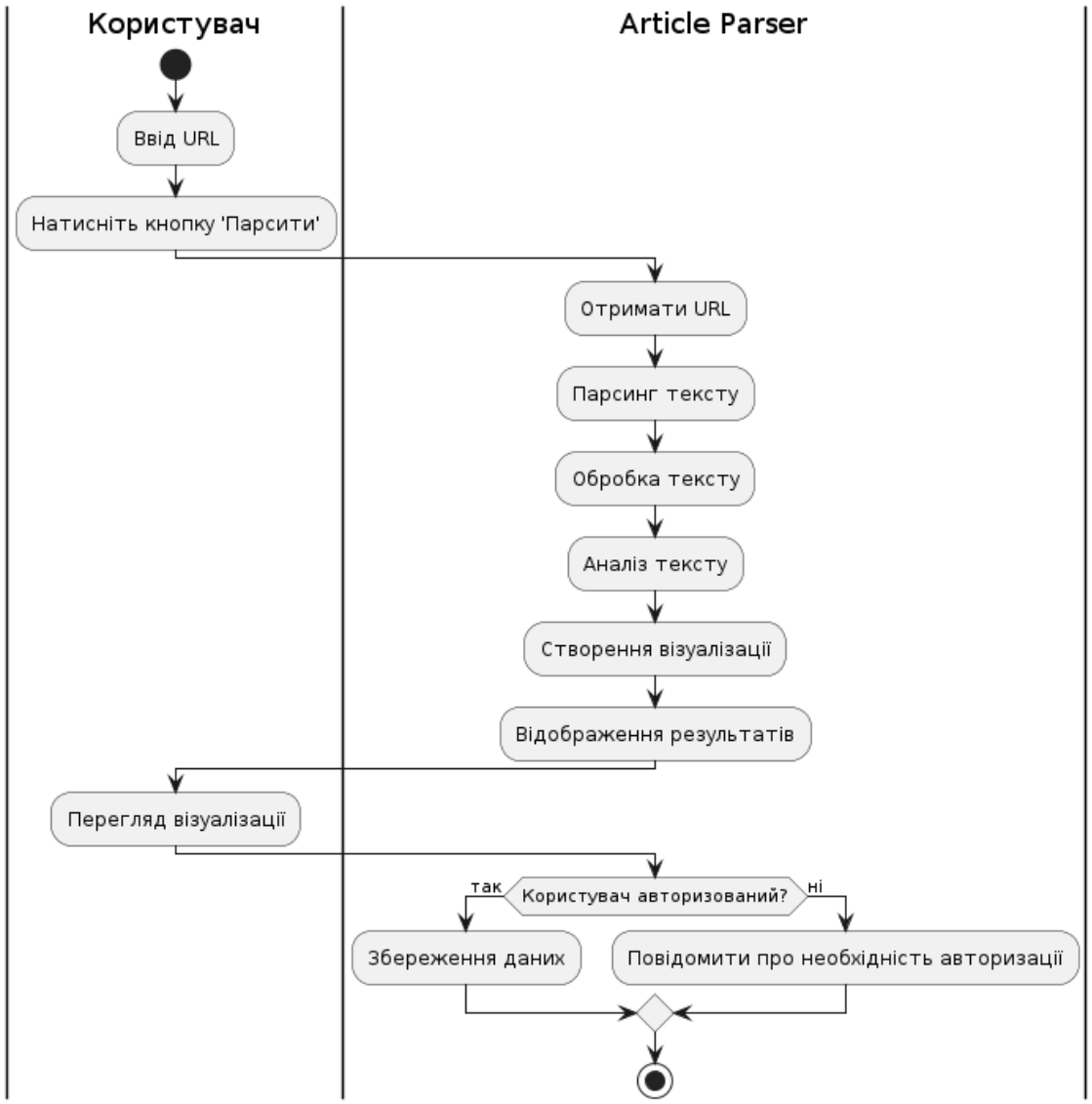


Рис 3.3 Діаграма діяльності процесу парсингу контенту

Таким чином, діаграма діяльності ілюструє весь процес від введення даних до отримання кінцевих результатів, забезпечуючи чітке розуміння послідовності дій та взаємодії між користувачем та системою.

3.2.4. Схема бази даних

Для зберігання даних веб-застосунку для парсингу спеціалізованого контенту було розроблено реляційну базу даних. Її схема, зображена на Рисунку 3.4

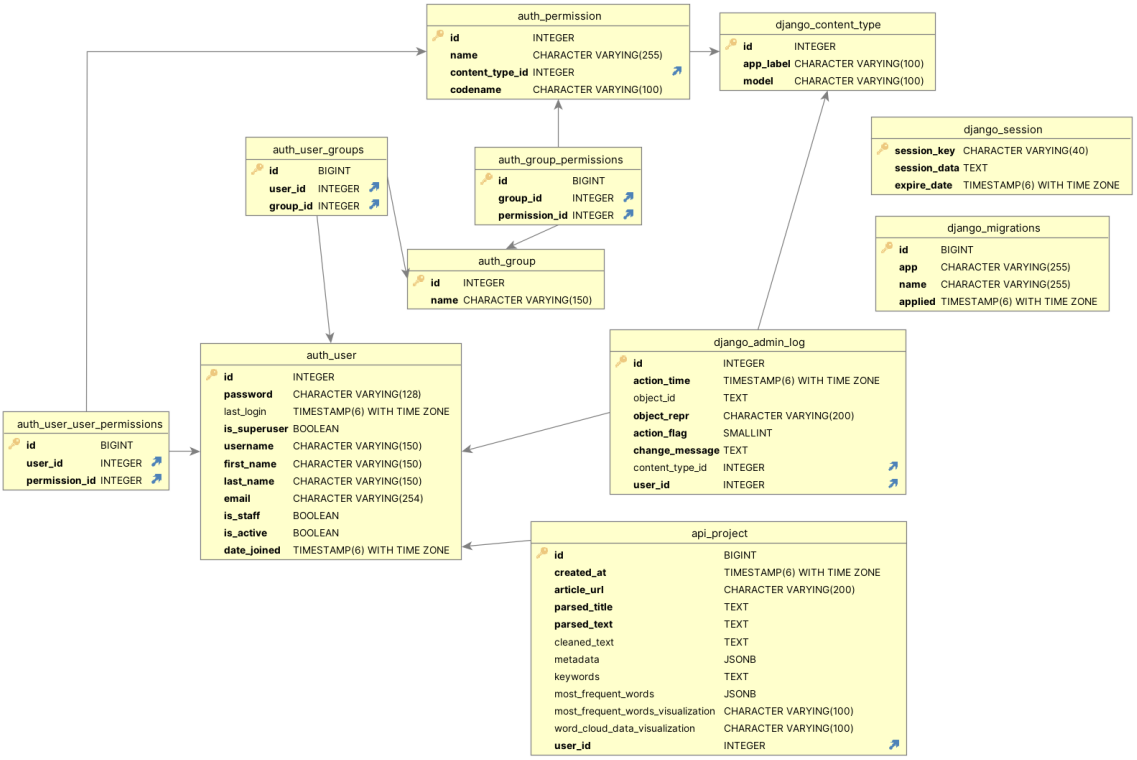


Рис 3.4 Схема бази даних

База даних складається з наступних таблиць:

— **auth_user**: Таблиця для зберігання даних користувачів, включаючи ім'я користувача, пароль, адресу електронної пошти та інші атрибути.

— **auth_user_groups**: Таблиця зв'язку "багато до багатьох" між користувачами та групами.

— **auth_user_permissions**: Таблиця зв'язку "багато до багатьох" між користувачами та дозволами.

— **auth_group**: Таблиця для зберігання даних про групи користувачів.

— **auth_group_permissions**: Таблиця зв'язку "багато до багатьох" між групами та дозволами.

— **auth_permission**: Таблиця для зберігання даних про дозволи, доступні в системі.

— **django_content_type**: Таблиця, що використовується Django для зберігання інформації про типи контенту.

— **django_admin_log**: Таблиця журналу дій адміністратора в Django.

— **django_session**: Таблиця для зберігання даних сесій користувачів.

— **django_migrations**: Таблиця, що використовується Django для відстеження застосованих міграцій.

— **api_project**: Основна таблиця, що містить дані про оброблені проекти, включаючи URL-адресу статті, розпарсений текст, ключові слова, візуалізації хмари слів та іншу релевантну інформацію. Зв'язана з таблицею **auth_user** через поле **user_id**, що вказує на автора проекту.

Така структура бази даних забезпечує ефективне зберігання та доступ до даних, необхідних для функціонування веб-застосунку.

3.3. Проектування інтерфейсу користувача

Проектування інтерфейсу користувача для системи парсингу спеціалізованих текстових даних включає створення різних сторінок, які забезпечують функціональність та зручність взаємодії для користувачів. Головна мета проектування інтерфейсу полягає в наданні користувачам можливості легко реєструватися, входити в систему, здійснювати парсинг даних, аналізувати результати та переглядати збережені результати парсингу.

Процес проектування інтерфейсу базувався на аналізі потреб користувачів. Були проведені дослідження для визначення функціональних можливостей, які має включати програмний продукт, а також відповідних елементів інтерфейсу для зручного використання. Важливими аспектами проектування стали логічна структура, проста навігація та зрозумілі елементи керування.

Перша сторінка інтерфейсу, яку бачить користувач, - це головна сторінка (рис. 3.5). Вона містить привітання та поле для парсингу.

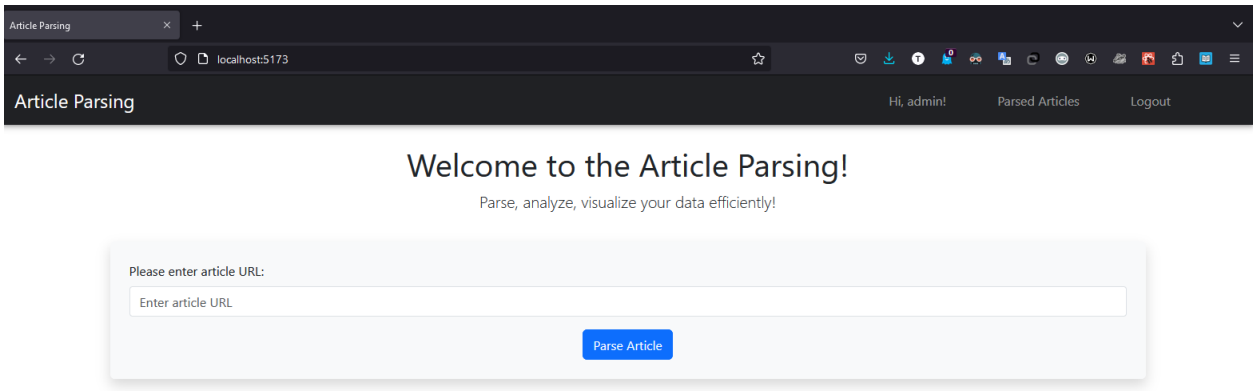


Рис 3.5 Головна сторінка

При переході на сторінку реєстрації користувач бачить форму, де потрібно ввести свої дані для створення нового акаунту (рис. 3.6). Форма містить поля для введення імені, електронної пошти та паролю. Після заповнення форми користувач може зареєструватися, натиснувши відповідну кнопку.

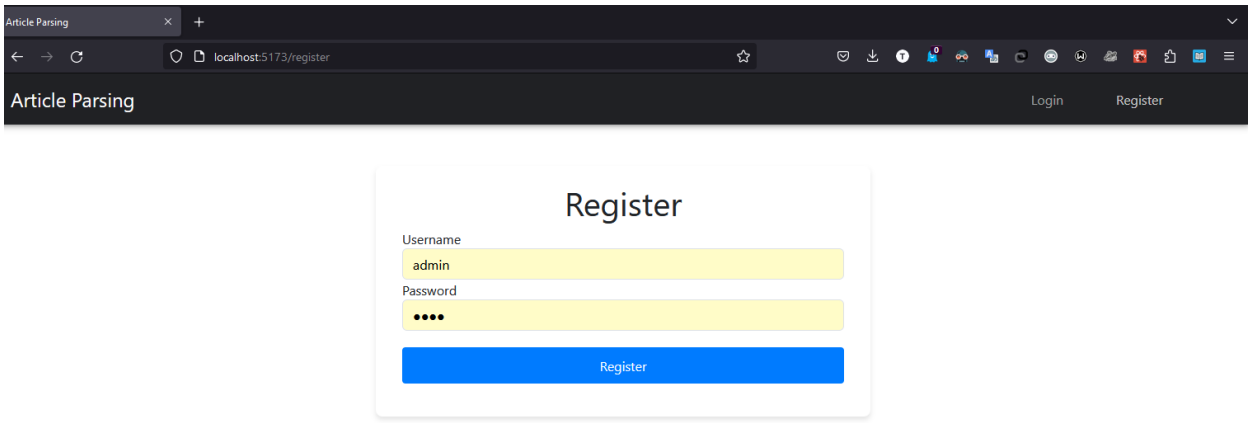


Рис 3.6 Сторінка реєстрації

Сторінка логіну відображає форму для введення електронної пошти та паролю, щоб увійти в систему (рис. 3.7). Після успішної авторизації користувач перенаправляється на головну сторінку або іншу відповідну сторінку залежно від обраної дії.

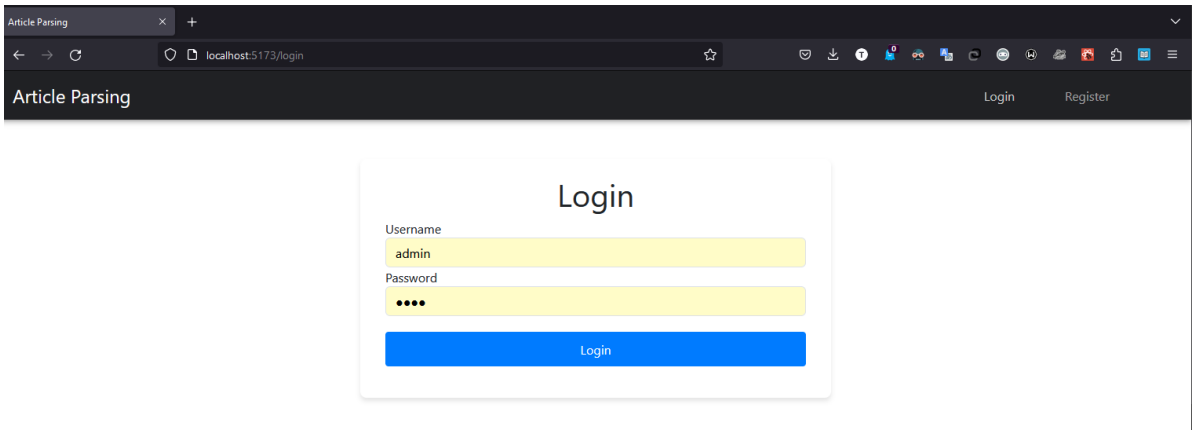


Рис 3.7 Сторінка логіну

Сторінка введення URL відображає форму для введення посилання на статтю, яку необхідно проаналізувати. Користувач вставляє URL статті в спеціальне поле та натискає кнопку "Parse Article" для початку процесу парсингу(рис. 3.8).

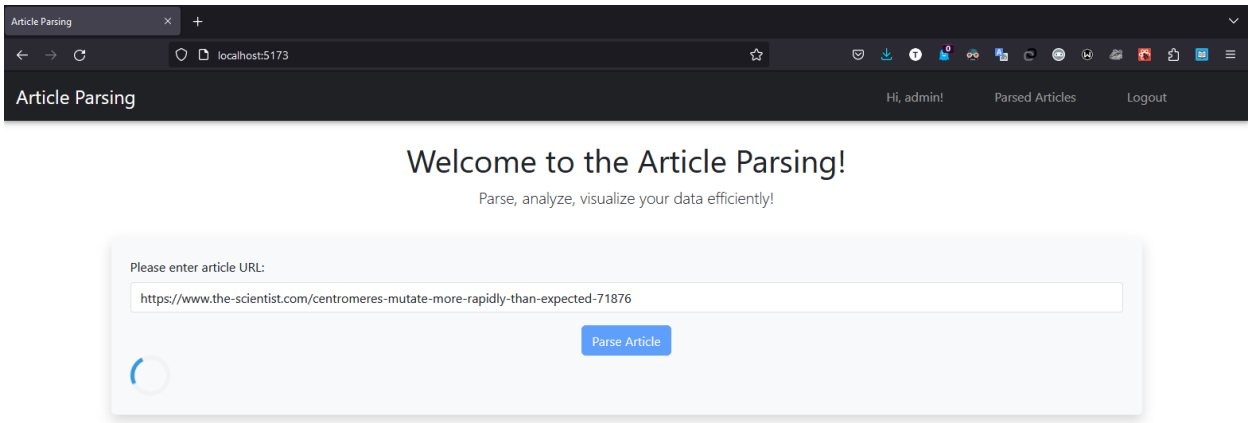


Рис 3.8 Форма введення URL для парсингу статті

Після завершення процесу парсингу користувач перенаправляється на сторінку результатів (3.9 – 3.10). Тут відображаються всі зібрані дані у вигляді таблиць та графіків. Користувач може аналізувати отримані результати, переглядати деталі та зберігати дані в різних форматах.

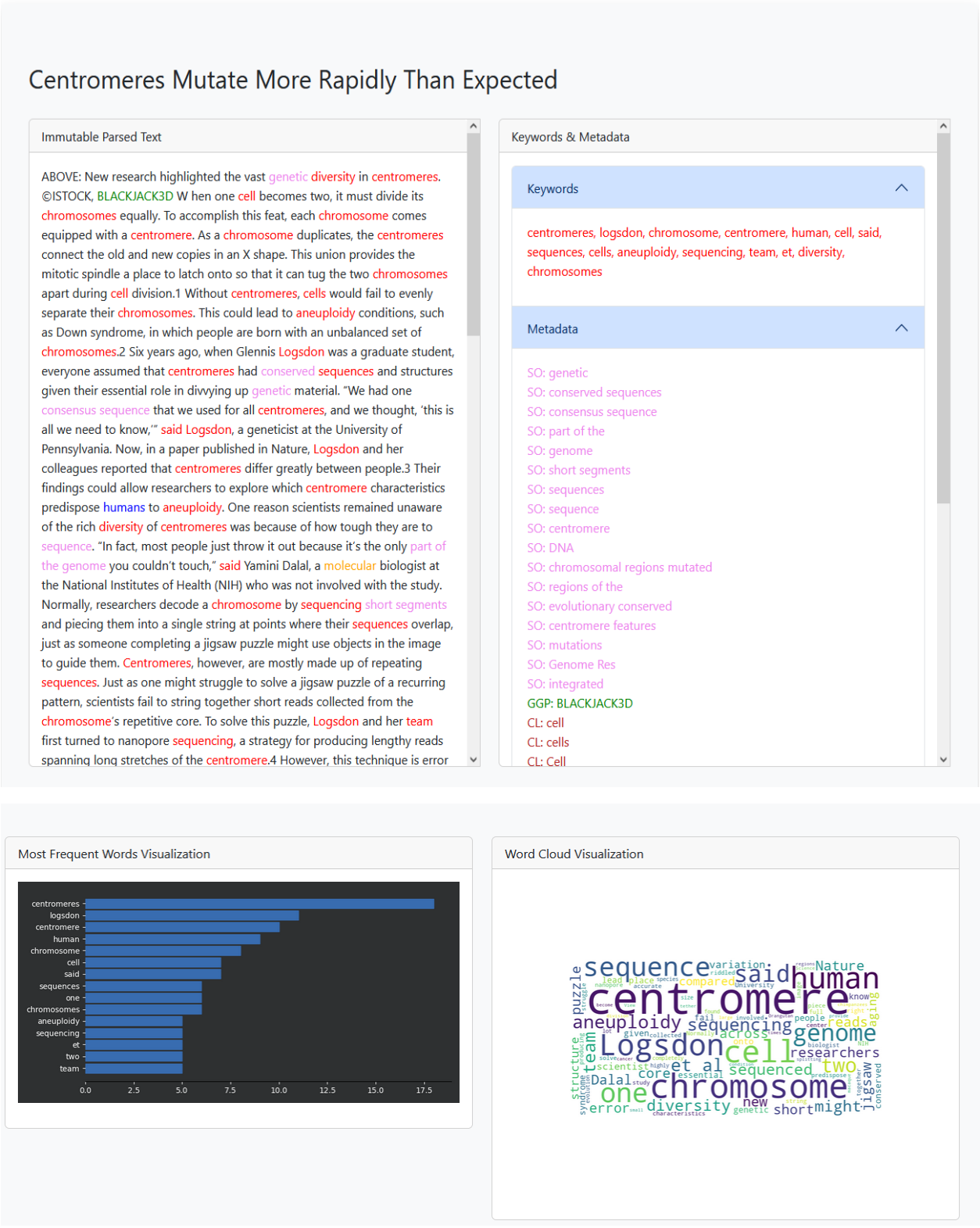


Рис 3.9 – 3.10 Сторінка результатів парсингу та аналізу

На цій сторінці відображаються всі результати парсингу, створені користувачем (рис. 3.11). Користувач може переглядати попередні результати та

видаляти непотрібні. Для перегляду детальної інформації та аналізу треба натисну на назву статті.

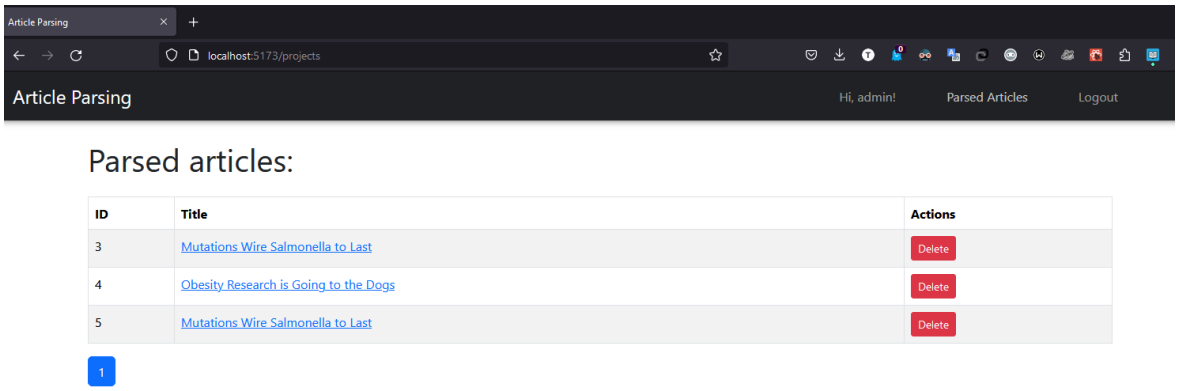


Рис 3.11 Сторінка з усіма результатами парсингу

Результатом проектування є створений інтерфейс користувача, який забезпечує повну функціональність для виконання завдань парсингу, аналізу та управління результатами. Інтерфейс розроблений з урахуванням потреб користувачів, логічної структури та ефективної навігації, що дозволяє користувачам легко взаємодіяти із системою та досягати своїх цілей з мінімальними зусиллями.

3.4. Тестування

Сутність тестування полягає в перевірці веб-застосунку або системи для виявлення помилок, дефектів або неправильного функціонування. Тестування є критично важливим для забезпечення якості програмного забезпечення та задоволення вимог користувачів. Основні цілі тестування включають перевірку правильності роботи програми, виявлення помилок, забезпечення відповідності вимогам та підтримку надійності та стабільності системи.

Для веб-застосунку для парсингу спеціалізованого контенту було вирішено використовувати ручне тестування API за допомогою Postman. Тестування

охоплювало кілька ключових сценаріїв, спрямованих на перевірку функціональності різних ендпоінтів API.

Тестування процесу реєстрації та авторизації

1. Перевірено коректність обробки правильних та неправильних облікових даних для реєстрації нового користувача.
2. Виконано тестування форми реєстрації, перевірено правильність заповнення полів та валідацію введених даних.
3. Тестування процесу авторизації з використанням зареєстрованих даних, перевірено, що система коректно обробляє вхідні запити та видає токени для доступу.

Тестування функціональності парсингу

1. Проведено тестування ендпоінту для початку процесу парсингу. Введено правильні та неправильні URL-адреси для перевірки обробки запитів.
2. Тестування обробки різних обсягів даних для перевірки стабільності та ефективності процесу парсингу.

Тестування результатів парсингу та аналізу

1. Тестування ендпоінту для отримання результатів парсингу. Перевірено, що дані правильно зберігаються та відображаються у відповідному форматі.
2. Перевірено функціональність аналізу даних, включаючи генерацію хмари слів та гістограми на основі зібраних даних.

Тестування управління результатами парсингу

1. Тестування ендпоінту для отримання списку всіх результатів парсингу, створених користувачем. Перевірено, що всі дані коректно відображаються та доступні для подальшого використання.
2. Перевірено функцію видалення результатів парсингу, щоб упевнитися, що система коректно обробляє запити на видалення.
3. Тестування функції перегляду детальних результатів парсингу для кожного запису в списку.

Ручне тестування API за допомогою Postman дозволяє тестувальнику активно взаємодіяти з системою та динамічно перевіряти її реакцію на різні вхідні дані та сценарії. Це забезпечує гнучкий та деталізований підхід до тестування та виявлення потенційних проблем. Кожен етап тестування включав ретельний аналіз відповідей сервера та перевірку правильності обробки даних, що дозволило виявити та усунути недоліки в роботі API.

ВИСНОВКИ

В процесі виконання дипломної роботи було розроблено веб-застосунок для парсингу спеціалізованого контенту з використанням мови програмування Python та бібліотеки React для JavaScript. Мета дипломної роботи виконана в повному обсязі.

1. Проведено аналіз методів та алгоритмів парсингу тексту з різних джерел, включаючи статті та наукові публікації, що підтвердило необхідність розробки власного веб-застосунку для парсингу спеціалізованого контенту.

2. Проаналізовано існуючі інструменти для парсингу контенту, такі як Orange, Voyant Tools та ParseHub, і виявлено їх недоліки, які враховано при розробці власного веб-застосунку.

3. Сформовано функціональні і нефункціональні вимоги. Ключовими функціональними вимогами є: можливість введення URL для парсингу, обробка та аналіз текстових даних, створення візуалізацій результатів парсингу та можливість збереження отриманих даних.

4. Проведено огляд програмних та технічних засобів реалізації веб-застосунку. В розробці використано: Python для бекенду, JavaScript (React) для фронтенду, PostgreSQL для зберігання даних та Docker для контейнеризації і забезпечення стабільної роботи системи.

5. Розроблено та протестовано веб-застосунок, який дозволяє користувачам парсити текстовий контент з різних джерел, аналізувати його та створювати наочні візуалізації, такі як хмари слів та гістограми найбільш вживаних слів.

6. Мартиненко О.В., Ільїн О.Ю., Розробка Web-застосунку для парсингу спеціалізованого контенту мовою Python з використанням Javascript React. Всеукраїнська науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті», 15 травня 2024 р., Київ, Державний університет інформативно-комунікаційних технологій. Збірник тез. К.: ДУІКТ,2024.С.132.

7. Мартиненко О.В., Ільїн О.Ю., Аналіз засобів для парсингу текстового контенту. Всеукраїнська науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті», 15 травня 2024 р., Київ, Державний університет інформативно-комунікаційних технологій. Збірник тез. К.: ДУІКТ,2024.С.241.

ПЕРЕЛІК ПОСИЛАНЬ

1. Django Documentation. URL: https://docs.djangoproject.com/en/stable/ (дата звернення: 03.06.2024).
2. Django REST Framework Documentation. URL: https://www.django-rest-framework.org/ (дата звернення: 03.06.2024).
3. Selenium Documentation. URL: https://www.selenium.dev/documentation/en/ (дата звернення: 03.06.2024).
4. Natural Language Toolkit (NLTK). URL: https://www.nltk.org/ (дата звернення: 03.06.2024).
5. spaCy Documentation. URL: https://spacy.io/usage (дата звернення: 03.06.2024).
6. PostgreSQL 15.1 Documentation. URL: https://www.postgresql.org/docs/15/index.html (дата звернення: 03.06.2024).
7. Docker Documentation. URL: https://docs.docker.com/ (дата звернення: 03.06.2024).
8. React Documentation. URL: https://reactjs.org/docs/getting-started.html (дата звернення: 03.06.2024).
9. Bird S., Klein E., Loper E. NLTK Book: Natural Language Processing with Python. Gravenstain: O'Reilly Media, 2009. URL: https://www.nltk.org/book/ (дата звернення: 03.06.2024).
10. Bird S., Klein E., Loper E. Natural Language Processing with Python. O'Reilly Media, 2009.
11. Docker for Developers. URL: https://www.docker.com/use-cases/developers (дата звернення: 03.06.2024).
12. Obe R., Hsu L. PostgreSQL: Up and Running: A Practical Guide to the Advanced Open Source Database. O'Reilly Media, 2017. URL:

https://www.oreilly.com/library/view/postgresql-up-and/9781491963392/ (дата звернення: 03.06.2024).

13. Маккінні В. Python для аналізу даних. 2019. URL: https://www.oreilly.com/library/view/python-for-data/9781491957650/ (дата звернення: 03.06.2024).
14. Stefanov S. React: Up & Running: Building Web Applications. O'Reilly Media, 2016. URL: https://www.oreilly.com/library/view/react-up/9781491931827/ (дата звернення: 03.06.2024).
15. Raj P., Chelladhurai J. S., Singh V. Learning Docker. Packt Publishing, 2015. URL: https://www.packtpub.com/product/learning-docker/9781784399315 (дата звернення: 03.06.2024).

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка Web-застосунку для парсингу спеціалізованого контенту мовою Python з використанням Javascript React

Виконав студент 4 курсу
групи ПД-42
Мартиненко Олексій Володимирович
Керівник роботи
Доктор технічних наук професор, професор кафедри Ільїн Олег Юрійович

Київ – 2024

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи** – спростити процес виявлення та візуалізації іменних сутностей та ключових слів в спеціалізованому текстовому контенті.
- **Об’єкт дослідження** – процес аналізу спеціалізованого текстового контенту з метою виявлення ключової інформації.
- **Предмет дослідження** – методи парсингу, виявлення іменних сутностей та візуалізації ключових термінів у спеціалізованому текстовому контенті.

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Провести огляд існуючих методів та інструментів парсингу текстового контенту.
2. Дослідити сучасні підходи до виявлення іменних сутностей в тексті.
3. Дослідити методи візуалізації текстових даних, а саме хмари слів та гістограми частоти термінів.
4. Спроектувати архітектуру web-застосунку для парсингу спеціалізованого контенту.
5. Розробити backend частину застосунку з використанням Python та Django REST Framework.
6. Розробити frontend частину застосунку з використанням JavaScript React для відображення результатів парсингу та візуалізації даних.
7. Налаштувати взаємодію між backend API та frontend частиною.

3

АНАЛІЗ АНАЛОГІВ

Показник	Orange	Voyant Tools	ParseHub	ArticleParser
Збереження результатів	Так	Ні	Ні	Так
Інтерфейс	Візуальний, блок-схеми	Веб-інтерфейс, інтерактивні візуалізації	Візуальний інтерфейс для налаштування парсингу	Веб-інтерфейс, візуалізації
Підтримка парсингу динамічного контенту	Обмежена	Ні	Так	Так
Можливості візуалізації отриманих даних	Широкі можливості, включаючи графіки, діаграми	Основні інтерактивні візуалізації	Відсутні	Основні візуалізації
Платформа	Десктоп	Веб	Веб	Веб

4

ВИМОГИ ДО ДОДАТКУ

Функціональні вимоги:

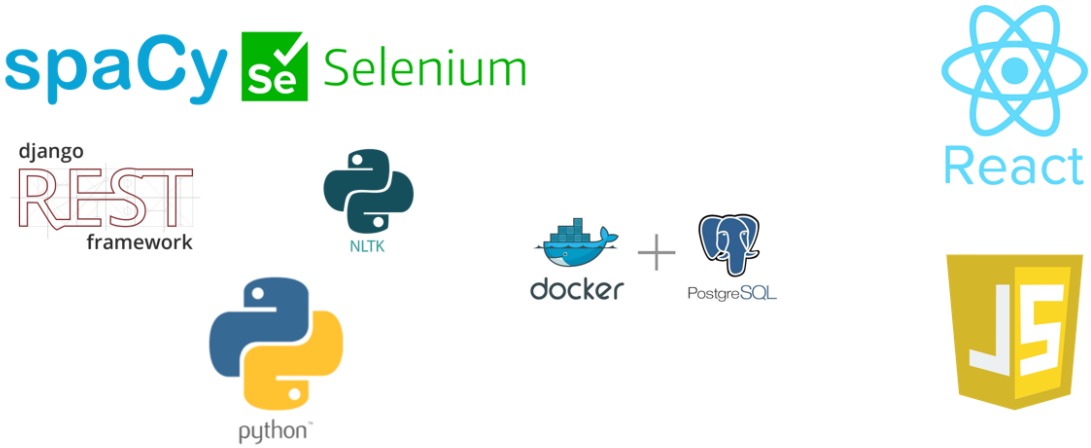
1. Можливість парсити текст за вказаним URL.
2. Автоматичне визначення та виділення іменних сутностей (персони, організації, локації тощо) в обробленому тексті.
3. Відображення хмари слів, що візуально репрезентує частоту ключових слів у тексті.
4. Будувати гістограми, яка відображає частоту найбільш вживаних слів у тексті.
5. Збереження результатів парсингу, обробки та аналізу тексту для подальшого використання.

Нефункціональні вимоги:

1. Можливість обробляти текстові дані.
2. Забезпечення безпеки даних користувачів та захист від несанкціонованого доступу.

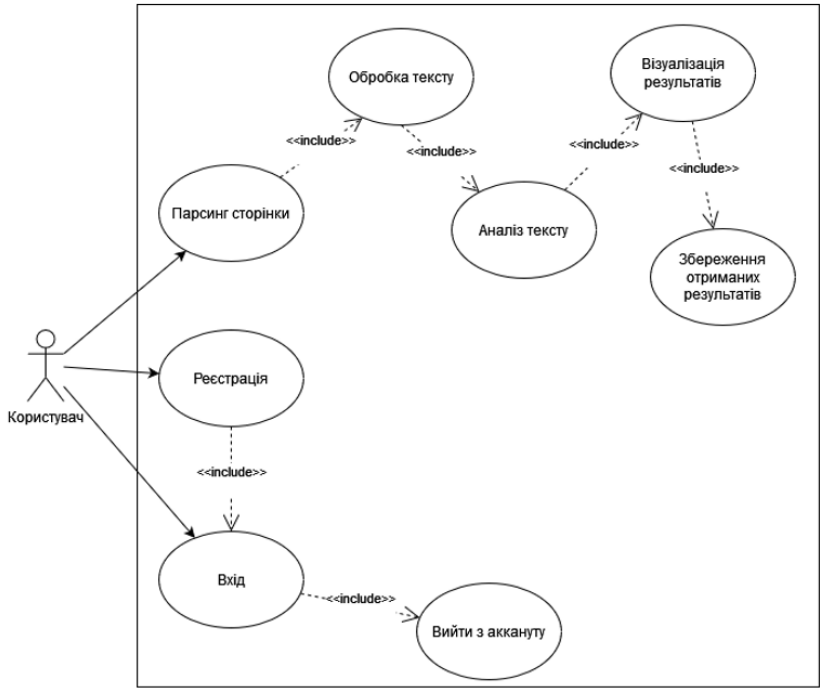
5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ



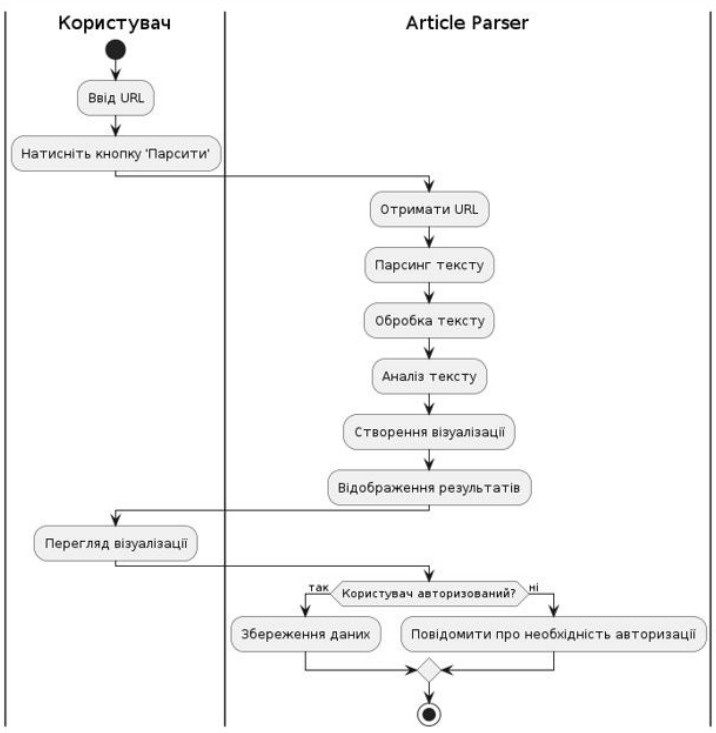
6

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ



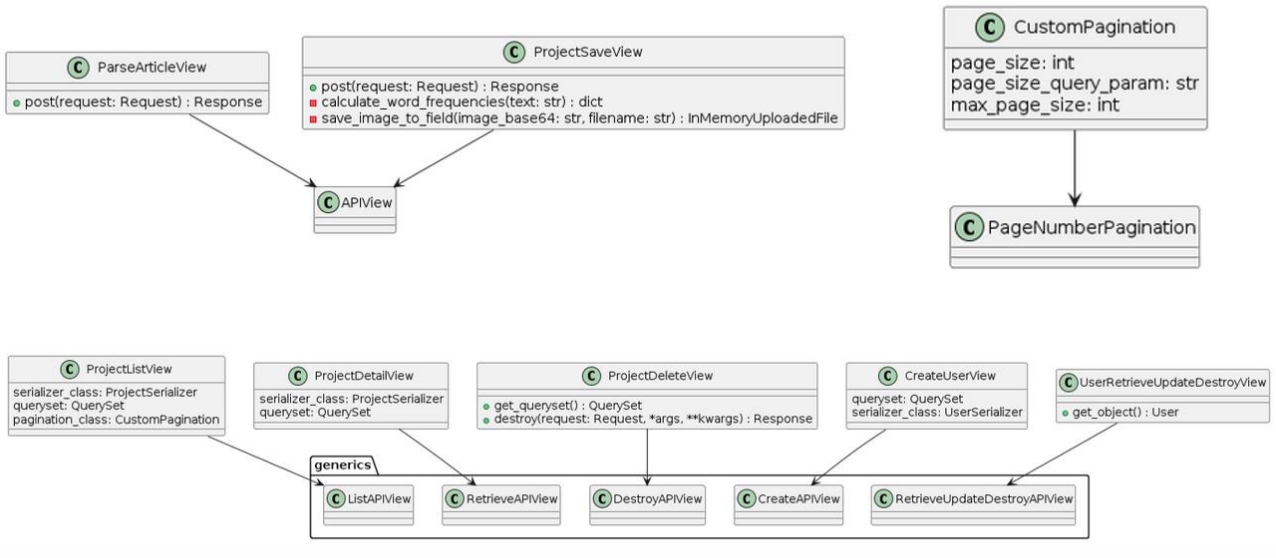
7

ДІАГРАМА ДІЯЛЬНОСТІ



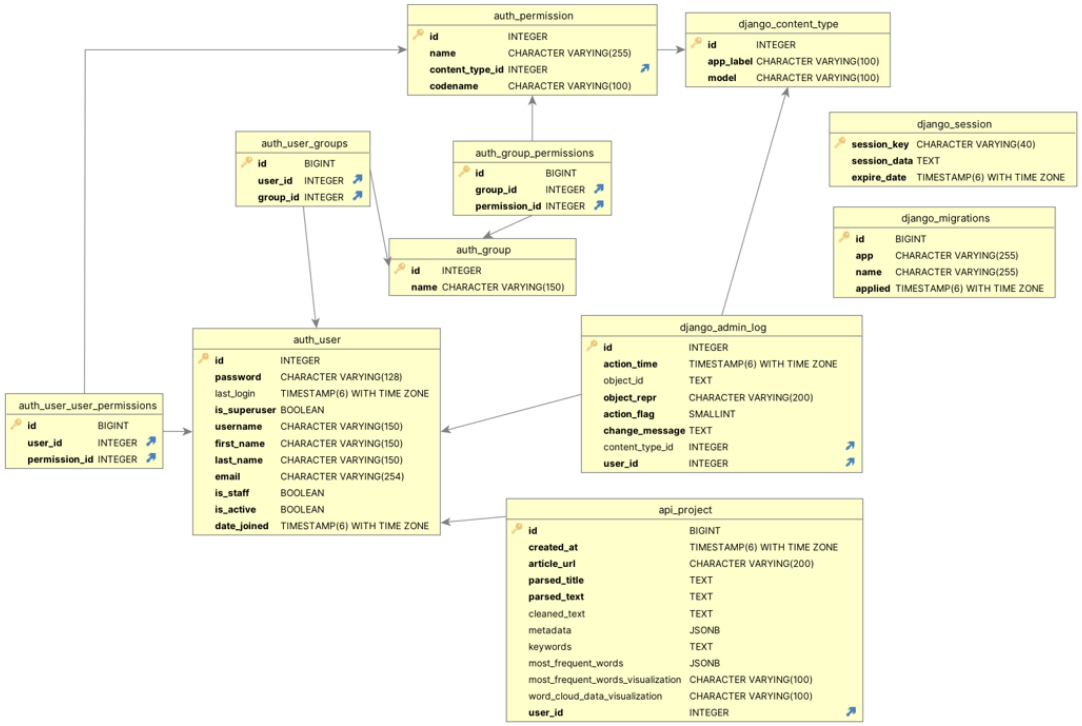
8

ДІАГРАМА КЛАСІВ



9

СХЕМА БАЗИ ДАНИХ



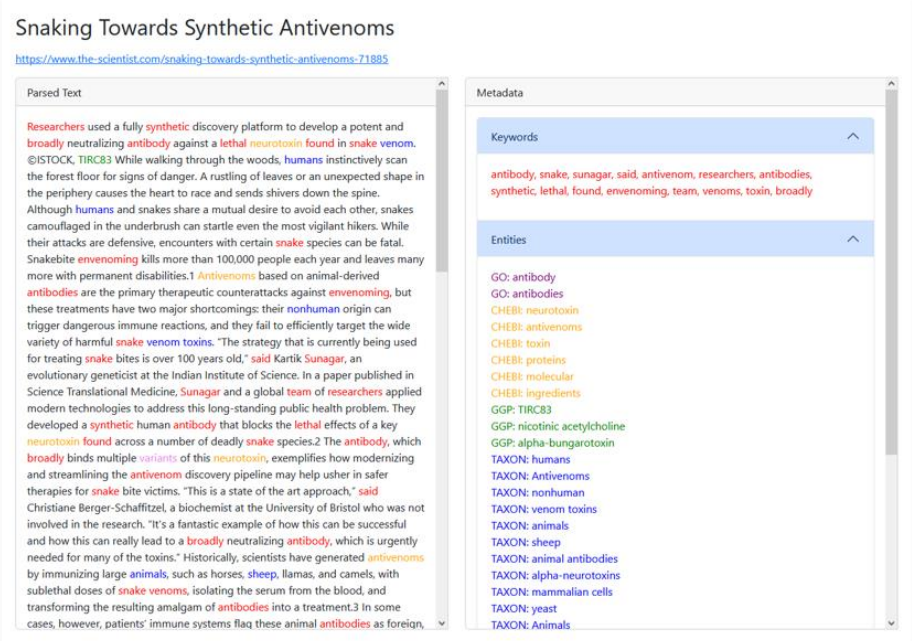
10

ЕКРАННІ ФОРМИ

Головна сторінка та поле для парсингу за URL

11

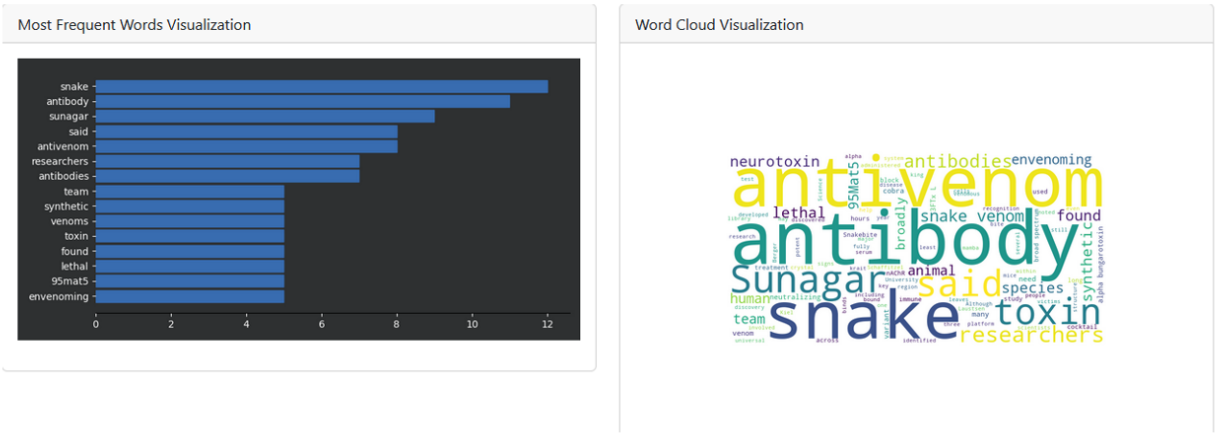
ЕКРАННІ ФОРМИ



Виділення іменних сутностей в тексті

12

ЕКРАННІ ФОРМИ



Хмара слів та гістограма найбліш вживаних слів

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Мартиненко О.В. РОЗРОБКА WEB-ЗАСТОСУНКУ ДЛЯ ПАРСИНГУ СПЕЦІАЛІЗОВАНОГО КОНТЕНТУ: Матеріали IV Всеукраїнської науково-практичної конференції «Сучасні інтелектуальні інформаційні технології в науці та освіті».15.10.2024, ДУІКТ, м. Київ.
2. Мартиненко О.В. АНАЛІЗ ЗАСОБІВ ДЛЯ ПАРСИНГУ ТЕКСТОВОГО КОНТЕНТУ: Матеріали IV Всеукраїнської науково-практичної конференції «Сучасні інтелектуальні інформаційні технології в науці та освіті».15.10.2024, ДУІКТ, м. Київ.

ВИСНОВКИ

1. Було проведено огляд сучасних методів та інструментів парсингу текстового контенту. Це дозволило визначити оптимальні методики для роботи з різними типами даних та форматами, а також вибрати відповідну бібліотеку Python для реалізації функціоналу парсингу в розробленому web-застосунку

2. Досліджено методи виявлення сутностей та візуалізації текстових даних, зокрема хмари слів та гістограми частоти термінів. Це дозволило розробити інтерактивну візуалізацію даних, яка покращує розуміння результатів парсингу та сприяє аналізу спеціалізованого контенту.

3. Розроблено backend частину web-застосунку з використанням Python та Django REST Framework для обробки даних, парсингу контенту та забезпечення API для взаємодії з frontend. Frontend частина реалізована з використанням JavaScript React для відображення результатів парсингу, візуалізації даних та забезпечення інтерактивного інтерфейсу користувача.

ДЯКУЮ ЗА УВАГУ!