МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

ТРИКОНЕНКО СЕРГІЙ ВОЛОДИМИРОВИЧ

Допускається до захисту:
завідувач кафедри
інформаційних технологій
д.т.н., доцент
_____ Нескородєва Т.В.
«___» _____ 20__ р.

РОЗРОБКА І ДОСЛІДЖЕННЯ МЕТОДІВ СИНТАКСИЧНОГО АНАЛІЗУ БІБЛІОГРАФІЧНИХ ДАНИХ

Спеціальність 122 Комп'ютерні науки

Кваліфікаційна (магістерська) робота

Науковий керівник:
Бабаков Р. М., доцент кафедри
інформаційних технологій
д.т.н., доцент
_____
(підпис)

Оцінка _____ / _____ / _____
(бали/за шкалою ЄКТС/за національною шкалою)
Голова ЕК: _____
(підпис)

Вінниця 2022

АНОТАЦІЯ

Триконенко С.В. Розробка і дослідження методів синтаксичного аналізу бібліографічних даних. Спеціальність 122 «Комп’ютерні науки», Освітня програма «Комп'ютерні технології обробки даних (Data Science)». Донецький національний університет імені Василя Стуса, Вінниця, 2022.

В даній кваліфікаційній роботі розглянуті основні принципи та характеристики бібліографічної інформації. Показано, що розробка інтелектуальної системи для таких задач є непростим завданням, це пов'язано з даними цих систем, які постійно оновлюються, а їх кількість надзвичайно велика, для реалізації та обробки даних з цих систем необхідно використовувати відповідні інструментальні засоби та архітектуру.

Продемонстровано основні алгоритми прогнозування послідовностей даних, а саме лінійна регресія та її вдосконалення, рекурсивні нейронні мережі, а також метод експоненціального спуску, який хоч і простий в реалізації, але його вдосконалення дають ефективні результати. Хоча кожен з цих методів має свої переваги та недоліки, їх робота продемонстрована на реальному застосуванні та вивчені їх основних принципів.

Для роботи з великими масивами даних обрано фреймворк Spark, який здатен обробляти великі обсяги даних на кластерах комп'ютерів. Продемонстровано ряд альтернатив використання архітектури веб-додатків, а саме монолітний та мікросервісний, обрано мікросервісний підхід, який хоч і має ряд недоліків, але добре підходить в даному випадку, а можливість поділу системи на невеликі компоненти з подальшим розширенням його складових є чудовою альтернативою в даному випадку.

Продемонстровані основні компоненти розробленої інтелектуальної системи. Сервіс прогнозування використовує описані підходи для прогнозування даних послідовності в майбутньому, а сервіс прийому даних, використовуючи паралельну обробку, отримує та аналізує бібліографічні дані,

після чого передає їх до відповідного сервісу обробки, де, в залежності від розміру даних та складності завдання, дані можуть бути оброблені одразу.

Ключові слова: парсинг, інтелектуальна система, бібліографічні дані, мікросервісна архітектура.

SUMMARY

Trikonenko S.V. Development and research of methods of bibliographic data parsing. Specialty 122 «Computer Science», Educational Program «Computer data processing technologies (Data Science)». Vasyl Stus Donetsk National University, Vinnytsia, 2022.

In this qualification work the basic principles and characteristics of bibliographic information are considered. It is shown that the development of an intelligent system for such tasks is not an easy task, this is due to the data of these systems, which are constantly updated, and their number is extremely large, for the implementation and processing of data from these systems it is necessary to use appropriate tools and architecture.

The main algorithms for predicting data sequences are demonstrated, namely linear regression and its improvements, recursive neural networks, as well as the exponential descent method, which, although simple to implement, but its improvements give effective results. Although each of these methods has its advantages and disadvantages, their work is demonstrated on real applications and their basic principles are studied.

To work with large data sets, the Spark framework was chosen, which is able to process large amounts of data on clusters of computers. A number of alternatives to the use of web application architecture are demonstrated, namely monolithic and microservice, the microservice approach is chosen, which, although it has a number of disadvantages, is well suited in this case, and the possibility of dividing the system into small components with further expansion of its components is an excellent alternative in this case.

The main components of the developed intelligent system are demonstrated. The prediction service uses the described approaches to predict the sequence data in the future, and the data reception service, using parallel processing, receives and analyzes the bibliographic data, and then passes them to the appropriate processing service, where, depending on the size of the data and the complexity of the task, the data can be processed immediately.

Keywords: parsing, intelligent system, bibliographic data, microservice architecture.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API – Application programming interface

PIP – package installer for Python

SJR – Scientific Journal Rankings

SQL – Structured Query Language

UML – Unified Modeling Language

XML – Extensible Markup Language

БД – База даних

ІС – Інформаційна система

ПЗ – Програмне забезпечення

СУБД – Система управління базами даних

ВСТУП

Актуальність теми дослідження.

В Україні та в інших країнах світу, функціонують бібліотеки, які становлять основу культурної, наукової, інформаційної та освітньої інфраструктури. Серед них можна виділити публічні бібліотеки, які є центрами культурного життя суспільства. Головних завданням бібліотек є усунення інформаційної нерівномірності шляхом гарантування рівних можливостей доступу до інформації для різних верств населення. Важливий зв’язок між бібліотеками та їх потенційними користувачами створюють інформаційні ресурси. Сучасна автоматизована бібліотечна інформація системи містять модуль «Електронний каталог», який замінює бібліотечні карткові каталоги та доступний через мережу Інтернет.

Для найкращого задоволення потреб клієнтів у різних населених пунктах, які мають розгалужену мережу бібліотек, необхідні об'єднуючі ресурси про різні відомості. Створення таких інформаційних ресурсів потребує інтеграції бібліографічних даних з різних джерел. Інтеграція в єдиний каталог бібліографічної інформації на документи, що зберігаються у фондах бібліотек, архівів та музеїв – спрямована державна програма в Україні. Також на необхідність створення зведених регіональних каталогів та єдиного джерела бібліографічних записів, висловлені професійною спільнотою. Кінцевим результатом застосування методів парсингу даних на основі різнорідних даних має бути створення цілісного інформаційного ресурсу.

У науці залишається давно актуальна проблема інтеграції даних і розвивається одночасно з інформаційними технологіями та підвищенням вимог до якості даних. Дослідницька організація Online Computer Library Center для розширення доступу до інформації в усьому світі та скорочення витрати на інформацію займається розробкою методів парсингу бібліографічних даних.

Крім того, розроблено основи для глобальної інтеграції даних, зокрема: розробка моделі реляційної обробки неелементарних даних [1, 2, 3]; методи

інтеграції різнорідних баз даних [4, 5, 6, 7]; методи створення зведеної інформації ресурсів спільного доступу [8, 9, 10]; концепція інтеграції інформаційних потоків [11]; методи інтеграції даних у відкритих системах [12]; аналіз та узагальнення методів і засобів парсингу даних [13, 14, 15].

Крім загальних досліджень парсингу даних, існують спеціальні дослідження, які розглядають можливість отримання методів, які матимуть більшу ефективність у конкретному застосуванні. Найбільший дослідницький центр OCLC, який керує глобальною бібліотекою WorldCAT з парсингу різнорідної інформації даних застосовуються до бібліографічних даних наступні дослідження: обробка метаданих у пов'язані дані для покращення цифрового відкриття колекції [26]; бібліографічні методи дослідження даних [28]; актуальні проблеми інтеграції різних бібліографічних даних з онлайн-каталогів [29].

Тому виникає проблема парсингу неоднорідних даних в цілому та зокрема бібліографічних даних. Загальні та спеціальні методи парсингу постійно розвиваються та вдосконалюються, що робить дану тему актуальною.

Об'єкт дослідження – парсинг бібліографічних даних та методи його реалізації, інтеграції та підвищення якості обробки інформації.

Предмет дослідження – комплекс методів автоматизованої стандартизації бібліографічних даних отриманих з різних джерел, які відрізняються підходами до реєстрації інформації.

Мета роботи полягає в дослідженні особливостей парсингу бібліографічних даних шляхом створення інтелектуальної системи для узгодження та якісної інтеграції інформації.

Для досягнення поставленої мети необхідним вирішити наступні **завдання**:

- обробка та аналіз великого масиву бібліографічних даних, які є характерними для публічних бібліотек України;
- дослідження масиву даних з метою пошуку особливостей та закономірностей;

- аналіз головних проблем парсингу бібліографічних даних;
- розробка ефективних методів парсингу даних на основі використання особливостей та закономірностей;
- реалізації методів з метою їх використання для створення з електронного каталогу публічних бібліотек.

Методи дослідження.

У дослідженні передбачається використання методів теорії ймовірності, методів математичної статистики і статистичного аналізу, методів теорії графів, методів аналітичного моделювання, методів теорії прийняття рішень, системного аналізу для аналізу структури інформаційної системи, теорії інформації, теорії класифікацій, теорії комп'ютерного моделювання, теорії порівняння.

Наукова новизна.

У процесі вирішення поставлених завдань отримані наступні результати:

- оцінено вплив на можливість порівняння та очищення інформації при пошуку бібліографічних даних;
- знайдено особливості та закономірності бібліографічних даних, які можливо використати в розроблених методах;
- розроблено методи для парсингу бібліографічних даних на основі різних підходів та способів порівняння інформації;
- реалізовано інтелектуальну систему для парсингу бібліографічних даних з використанням нових методів обробки інформації.

Практичне значення отриманих результатів.

Практична цінність роботи полягає в наступному:

- спроектовано інтелектуальну систему, яка пропонує можливість парсингу бібліографічної інформації з оперативним оновленням;
- розроблено алгоритм: для оцінки якості та пошук закономірностей у бібліографічних даних; очищення бібліографічних даних; інтеграція бібліографічних даних з різних джерел.

Структура роботи

Робота складається з вступу, трьох розділів, висновків та списку використаних джерел. Загальний обсяг роботи складає 89 сторінок, 21 рисунків. Список використаних джерел та літератури містить 50 найменування.

РОЗДІЛ 1
СУЧАСНИЙ СТАН ТА ПРОБЛЕМИ ПАРСИНГУ БІБЛІОГРАФІЧНИХ ДАНИХ

1.1 Типи інформаційних джерел та їх роль у науковому прогресі

Джерело інформації - це особа, річ або місце, звідки інформація походить, або отримана. Джерела інформації можна назвати первинними або вторинними. Це джерело може повідомити людину про щось або надати інформацію про це. Джерела інформації поділяються на різні категорії: первинні, вторинні, третинні тощо.

Тип джерела інформації. Різні епістемології дотримуються різних поглядів на важливість різних типів джерел інформації. Емпіризм розглядає сенсорні дані як остаточне джерело інформації, тоді як інші епістемології дотримуються іншої точки зору.

Різні типи джерел інформації можна розділити на дві великі категорії:

– Літературні джерела;

– Нелітературні джерела.

Першоджерела — це оригінальні документи про події чи відкриття, такі як результати досліджень, експериментів чи опитувань, інтерв'ю, листи, щоденники, юридичні документи та статті в наукових журналах. Першоджерело також є першим описом події. Це можуть бути відеозаписи очевидців, аудіозаписи чи повідомлення новин.

Первинними джерелами інформації є спочатку опубліковані записи оригінальних досліджень і розробок або описи нових процедур або нові інтерпретації старих тем чи ідей. Є необроблені файли, які представляють необроблені невідфільтровані ідеї.

Вони являють собою останню доступну інформацію. Дослідники, які створюють нову інформацію, можуть зробити її доступною для певних спільнот, використовуючи першоджерела. Часто це може бути єдиним доступним джерелом інформації. Першоджерела – це невпорядковані джерела,

якими дуже важко користуватися, вторинні джерела допомагають ними користуватися. Це важливі джерела інформації. Ця тема стала окремою дисципліною, оскільки в цій галузі почали з’являтися незалежні першоджерела. Швидкість, з якою розвивається дисципліна, значною мірою визначається кількістю літератури, створеної у формі першоджерел, що повідомляють про розвиток у цій галузі.

Первинне джерело — це термін, який використовується в багатьох дисциплінах для опису вихідного матеріалу, який є найближчим до особи, інформації, періоду чи ідеї, що вивчається. В історіографії першоджерела (також звані першоджерелами) — це артефакти, документи, записи або інші джерела інформації, створені під час дослідження. Якщо створено людиною, це джерело має прямі особисті знання про описані події. Першоджерело – першоджерело інформації з теми. Подібні визначення використовуються в бібліотекознавстві та інших галузях наукової діяльності. У журналістиці першоджерелом може бути особа, яка безпосередньо знає ситуацію, або документ, створений такою особою. Первинні джерела відрізняються від вторинних джерел, які цитують, коментують або спираються на первинні джерела. Первинні та вторинні — це відносні терміни, які класифікують джерела як первинні чи вторинні залежно від конкретного історичного контексту та того, що вивчається.

Вторинні джерела надають аналіз, огляд подій або знахідок, описаних у первинних джерелах. Вони тлумачать, інтерпретують або узагальнюють першоджерела. Деякі вторинні джерела використовуються, щоб переконати читачів. Вторинні дані можна вважати менш об'єктивними. Приклади вторинних джерел включають: словники, енциклопедії, підручники, статті та редакційні статті, що пояснюють або оглядають дослідницькі статті.

Вторинні джерела інформації – це джерела інформації, які складаються з основного джерела інформації або на які посилається воно. Вихідну інформацію випадково змінювали, вибирали або реорганізовували, щоб служити певній меті для групи користувачів. Ці джерела містять інформацію,

упорядковану та організовану за певними планами. Вони містять упорядковані перепаковані знання, а не нові знання. Інформація в першоджерелах представлена в більш зручному вигляді. За своєю природою вторинні дані легше доступні, ніж первинні.Вони не лише надають перевірену інформацію, але й служать бібліографічним ключем до первинних джерел інформації. Першоджерела з'являються першими, за ними йдуть вторинні. Важко знайти інформацію з первинних джерел безпосередньо. Тому в першу чергу слід звернутися до вторинних джерел, що приведе до конкретних первинних джерел.

Відповідно до уподобань джерел, встановлено, що людина, ЗМІ та Інтернет – це три найпопулярніші жанри інформаційних джерел. Тому спочатку уважно подивилися на розподіл вихідних даних у рамках основної схеми трьох жанрів. Тоді вказали три жанри на основі обсягу джерел у контексті соціальних запитань та розробили нову схему класифікації джерел.

Високий рівень розподілу джерел за жанрами був класифікований на чотири категорії: людина, книги, засоби масової інформації та Інтернет. Хоча книги не вважалися джерелом, що цитується в попередніх дослідженнях, ця категорія була знову додана до жанрового аналізу. В дослідженні завдяки пізнаваній кількості джерел у нашому контексті соціальних запитань та питань. Жанрові категорії були визначені наступним чином.

Людські джерела вказують на знання, досвід, професію чи роботу постачальників джерел або відповідних третіх сторін. У джерелах книг вказуються переважно друковані рукописи чи публікації, але посилання або URL-адреси онлайн-книг також були включені до цієї категорії. Заголовок або автори книг були показані як джерела. Засоби масової інформації вказують джерела з інформацією про передачі, новини, журнали чи телебачення. Наприклад, ім'я широкомовної програми або веб-сайт тієї самої програми може бути вказано як джерело інформації. Таким чином, вона включає як Інтернет, так і офлайн інформацію про ЗМІ. Решта Інтернет-джерел, які не входять до

жодного жанру людини, книг чи засобів масової інформації, були згруповані в Інтернет-джерела.

1.2 Розгляд теоретичної основи терміну «парсинг»

У сучасному інформаційному світі ми часто стикаємося з такими проблемами, коли виникає необхідність швидко завантажити список товарів з інтернет-магазину або зібрати інформацію з сайту, зберігаючи певні необхідні дані. Наприклад, назва, ціна, зображення, посилання на продукт тощо. [1].

Великі обсяги даних неможливо обробити вручну в Інтернеті. У цьому випадку використовується спеціально розроблене програмне забезпечення - парсер. Ці програми створюються для різних завдань, в основному пов'язаних з маркетингом і просуванням сайтів.

Синтаксичний аналізатор — це програма, яка забезпечує розбір, синтаксичний аналіз вмісту в Інтернеті, до певної математичної моделі, створеної мовою програмування (зазвичай Python, PHP або Java).

Алгоритм роботи програми-парсера в кожному випадку приблизно однаковий:

- Програма може отримати доступ до ресурсів в Інтернеті;
- Завантажте код сторінки, який необхідно розібрати;
- читання та обробка інформації;
- Надає результати у зручному форматі – .html, .xml, .sql тощо. [1].

Програма порівнює заданий набір слів або значень із тими, що знаходяться в Інтернеті, враховуючи задані обмеження. Алгоритми подальших дій щодо знайденого контенту також заздалегідь задані – це може бути надання інформації або певні дії, наприклад, розміщення ставки під час аналізу спортивної події.

Parser — імпортер даних. Копіювати інформацію можна не тільки в інтернет-магазинах, а й на синдикованих торгових сайтах, групах у соціальних мережах тощо. Головне підібрати правильні інструменти та алгоритми.

За допомогою парсера можна завантажити найрізноманітнішу інформацію. Це може бути не тільки назва, але й опис, відеоконтент, зображення, основні функції тощо.

Парсер робить все сам. Розбір — трудомістка і виснажлива робота. Довіритися цій автоматиці набагато простіше. Це найкращий варіант, коли стоїть завдання зробити ресурс успішним у довгостроковій перспективі. [2].

Parser надає набір інструментів, який може допомогти отримати потрібне значення з будь-якого формату даних. Витягнуті дані можна зберігати в окремих файлах на вашому локальному комп'ютері, у хмарі, на хост-машині або безпосередньо в базі даних. Це автоматичний процес.

Програмне забезпечення допомагає аналізувати зібрану інформацію. Резолвер завдань надсилає запит GET на сайт-донор, який повинен повернути дані. В результаті цього запиту буде створено HTML-документ, який буде проаналізовано програмою. Потім аналізатор шукає в ньому необхідні дані та перетворює їх у потрібний формат [2].

Існує два способи аналізу мережі:

– отримати доступ до веб-сайту через протокол HTTP, HTTPS або веб-браузер;

– Використовуйте доступ до бота.

Розбір іноді плутають із вибіркою. Це відбувається тому, що процеси схожі. Обидва служать одній меті. Різниця в голі. Обидві операції «заточені» під обробку даних із сайту. Процеси автоматизовані, оскільки використовують ботів-парсерів. Але це лише роботи для обробки інформації чи контенту [3].

Синтаксичний аналіз і сканування використовують програмні інструменти для аналізу веб-сторінок і даних пошуку. Вони не змінюють інформацію, яка там відображається, і використовують її безпосередньо.

Parser збирає дані та сортує їх для виведення відповідно до заданих критеріїв. І це не обов'язково відбувається на веб-сторінках аналізу в Інтернеті. Важливі дані, а не те, де вони зберігаються. Наприклад, робота в аналізі цін. Для цього запускається створений парсер товару та його ціни для збору

інформації з будь-якого інтернет-магазину. Таким же чином ви можете аналізувати дані фондового ринку, рекламу нерухомості тощо.

Сканування або веб-сканування є прерогативою пошукових роботів або павуків. Сканування передбачає перегляд сторінок для пошуку інформації та її індексування, включаючи останню літеру та крапку. Але дані не витягуються одночасно. Інтернет-боти, також відомі як аналізатори пошукових систем, також систематично переглядають Всесвітню павутину, щоб знайти сайти та описати їх вміст.

Найважливіша відмінність від сканера полягає в тому, що він збирає дані та впорядковує їх. Як працює Google або Yahoo – приклад веб-збирання. Це теж аналіз. Коротше кажучи, це коли пошукові системи сканують ваш сайт і використовують отриману інформацію для індексації.

Парсер — це програмне рішення, тоді як парсинг — це процес. Тобто програма, яка використовується для розбору, є парсером. Він використовується для автоматичної обробки та вилучення даних.

Програмне забезпечення зазвичай використовує окремий лексичний аналізатор для аналізу даного тексту. Його називають токенізером або лексичним аналізатором [3]. Токенізатор розбиває всі вхідні дані на токени — окремі символи, наприклад слова. Отримані таким чином токени служать вхідними символами для аналізатора (рис. 1.1).

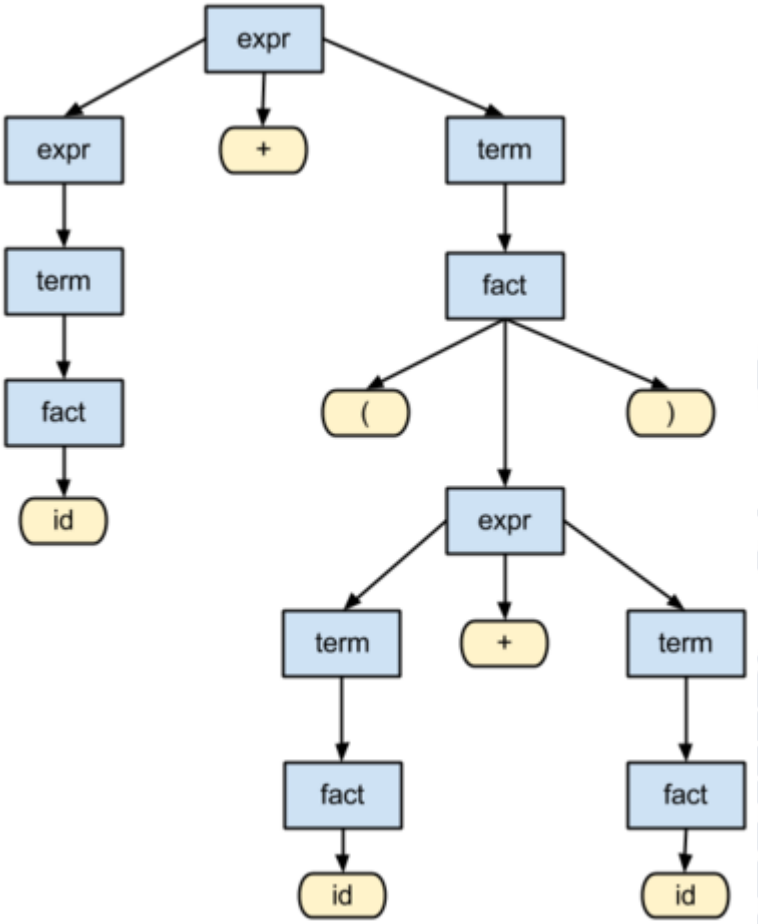

Рисунок 1.1 – Приклад дерева синтаксичного аналізу:
вираз – expr, термін – term, факт – fact, ідентифікатор – ID

Потім програма обробляє синтаксис вхідних даних, аналізуючи їх і створюючи синтаксичне дерево. Виходячи з цього, парсер переходить до обробки інформації – генерації або відбору коду за певними критеріями [4].

Розбір використовується для перетворення тексту в нову структуру, коли:

- Читання програмного коду, наприклад, Java, SQL та інших мов програмування. Ось що роблять аналізатори баз даних. Ось як це працює в цьому випадку: аналізатор надає програмному компілятору структури даних, які можна використовувати для генерації машинного коду.
- Використовуйте аналізатор вихідного коду сторінки, щоб прочитати HTML-код. Для комп'ютера HTML-код — це рядок символів, який аналізатор браузера має проаналізувати. Синтаксичний аналізатор надає опис веб-сторінки як структуру даних, яка потім компілюється в потрібному порядку та відображається на екрані комп'ютера.

- аналіз розмітки XML. Спеціальні аналізатори XML відповідають за аналіз таких документів і включають інформацію для подальшого використання.
- Читання URL-адрес та Інтернет-протоколів HTTP та HTTPS. Синтаксичний аналізатор розбиває складні URL-адреси та схеми протоколів на ієрархії.
- Пошукова система. За допомогою парсерів пошукових систем роботи вибирають релевантний для них текст із вмісту веб-сайту. Після початкового відбору зразків дані обробляються, а результати аналізу доступні для перегляду [5].

У цифровому маркетингу аналіз використовується для збору та аналізу певної інформації з вмісту потрібного сайту. Аналіз сторінки є основним методом збору інформації з веб-вмісту сайту, який використовується в різних сферах: продажі, маркетинг, фінанси, електронна комерція, збір інформації про конкурентів тощо. Він широко використовується в різних галузях промисловості (рисунок 1.2).

У роздрібній торгівлі є багато можливостей для використання аналітики. Наприклад, моніторинг цін конкурентів або аналіз ринку, коли аналітика використовується для обробки даних і отримання цінної інформації для маркетологів. Так, для електронної комерції може знадобитися незліченна кількість фотографій і описів продуктів.

Їх неможливо створити за кілька днів, оскільки навіть копіювання та вставлення кожного займає деякий час. Простіше і швидше створювати парси і швидко «вибирати» все, що потрібно. Або візьміть аналіз цін на ринку - регулярний аналіз веб-сторінок конкурентів допоможе вчасно помітити і врахувати всі зміни на ринку [6].

Отже, парсинг – це процес аналізу вмісту сайту та вилучення необхідної інформації. Використовуючи наведену вище інформацію, ви можете автоматично наповнити свій веб-сайт великою кількістю вмісту. Це дає можливість виграти час для створення великого обсягу необхідної інформації та надати її в найкоротші терміни.

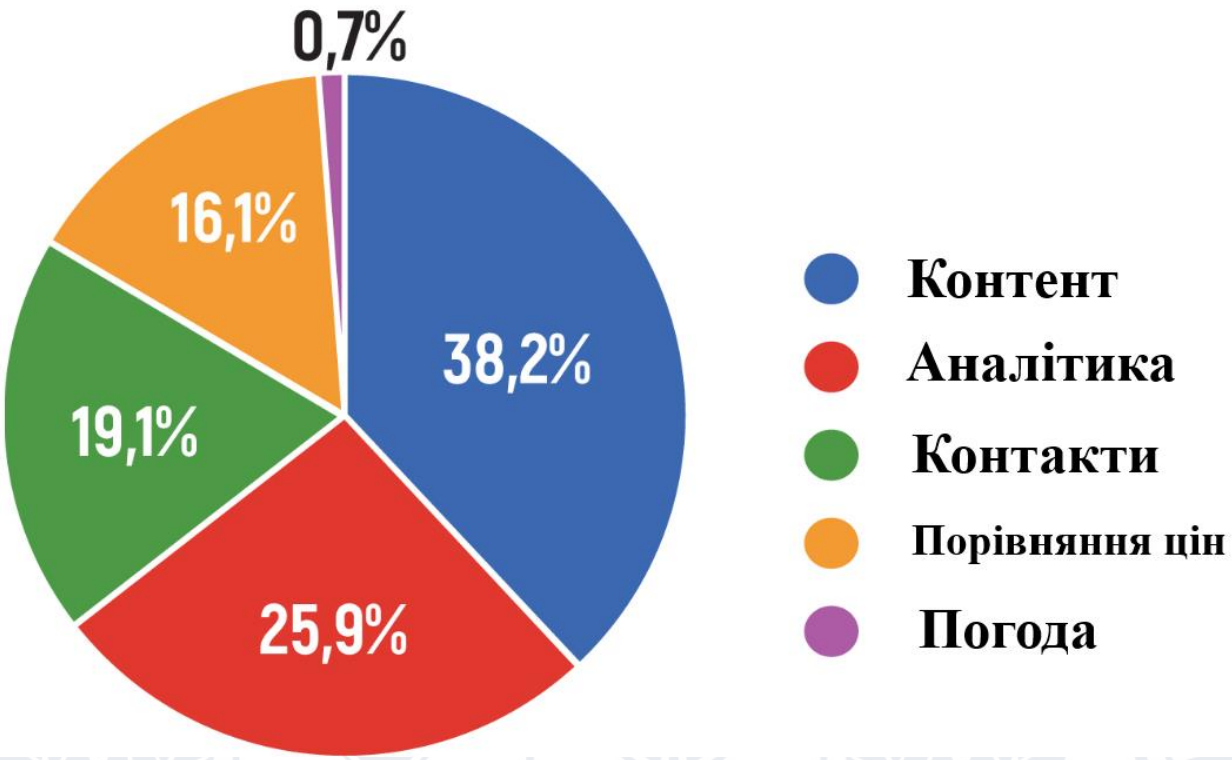

Рисунок 1.2 – Активне застосування по частоті, дослідження HubSpot роздрібного маркетингу

Розберемо розбір прикладу. Уявімо, що ми приходимо в бібліотеку і хочемо швидко переписати всі статті у велику енциклопедію та розмістити їх на сайті. Кожен том, кожна стаття вивчається багато днів.

Або знайдіть цікавий сайт рецептів і захочете швидко й повністю скопіювати їх на інший сайт. При аналізі сайту чи блогу кількість сторінок не має значення. Однак при розборі форуму, каталогу чи інтернет-магазину це стає серйозною проблемою. Користувачі також можуть використовувати Інтернет для виконання цього завдання. Перед розбором важливо зрозуміти вихідний матеріал. Є деякі винятки, наприклад, коли інформацію вбудовано у відеофайл Flash.

Розбір може створювати будь-який формат файлу на вимогу. Це може включати HTML, зображення або відео в каталозі, файл Excel, CSV або текстовий файл.

Розуміння того, що всі процеси, які вимагають синтаксичного аналізу, існують, відкриває, що парсери задіяні в багатьох автоматизованих перекладах.

Це включає автоматичні транслятори мов програмування, які створюють програмний код на машинно-орієнтовану мову. Це також стосується автоматизованих запитів SQL через мови програмування та подібні процеси.

Більшість людей використовують програми, щоб допомогти їм розібрати текст, оскільки вручну або візуально розібрати текст для більшості людей важко.[7]

Люди часто запитують про спеціалізовані послуги аналізу даних на популярних ринках фрілансерів або форумах SEO. Деякі з найпопулярніших варіантів – це дані від конкретного автора або певної теми.

Вони часто використовують ці інструменти для своїх досліджень:

- створення пошукових систем.
- Для інформації, пов’язаної з пошуковими запитами, ключовими є слова та значення.
- До даних включено інформацію про веб-сайти конкурентів.
- Інформація, зібрана з медіаресурсів, дискусійних форумів тощо, часто є основою для написання.
- Дані з онлайн-магазинів, наприклад ціни та товари, збираються через API.

Іншим прикладом є те, що такі каталоги сайтів можна проаналізувати за допомогою аналізатора. Завдяки цьому процесу отримані дані можуть бути таблицею або звітом про спроби SEO. Його також можна використовувати для пошуку веб-сайтів, таких як електронна пошта, блоги та форуми; пошук ресурсів; репост контенту; і більше. Під час аналізу сайтів на предмет зовнішніх посилань люди можуть аналізувати сайти конкурентів і визначати сайти довіри, які зараз популярні.

Визначення синтаксичного аналізу в інформатиці полягає в створенні математичної моделі для порівняння токенів із формальною граматикою. Зазвичай це робиться за допомогою аналізу синтаксису. Однією з поширених мов програмування, яка використовується для створення граматики, є Python, PHP, Ruby або Perl.[8]

Щоб зрозуміти літературний твір через належну філологію, потрібно проаналізувати його структуру, порівнюючи слова, які він бачить на сторінці (лексеми), зі своїм особистим словниковим запасом.

Багато програм використовують однаковий базовий алгоритм. Наприклад, парсер може обробляти кілька сторінок сайту одночасно. Якби алгоритм виконувався послідовно для кожної нової сторінки, розбір великих проектів зайняв би багато часу. Використовуючи мультипарсер, він може обробляти багато сторінок одночасно, мінімізуючи ресурси, які вони використовують. Це дозволить синтаксичному аналізатору використовувати ресурси комп'ютера раціонально і в той же час не виснажувати ресурси комп'ютера. Таким чином, час розбору скорочується в десять разів.

Розбирати сайти можна за допомогою різних парсерів - багатопотокових і універсальних, і вузькоспеціалізованих, орієнтованих на конкретні завдання. Резолвери сайтів чимось схожі на пошукових роботів, які використовуються відомими пошуковими системами. Однак зазвичай вони аналізують веб-сайт за заданими параметрами та збирають сам контент і можливість його використання, а не надають інформацію про нього. Синтаксичний аналізатор вмісту сайту може зібрати його з будь-якого джерела в Інтернеті, відкритого для людей і пошукових роботів. Це можуть бути довідники, інтернет-форуми, рекламні сайти, магазини, сайти-візитки, блоги, портали компаній, маркетплейси тощо. Наприклад, власники інтернет-магазинів активно використовують парсери сайтів для автоматизації процесу збору характеристик і фотографій товарів на офіційних сайтах виробників і дистриб'юторів. Тобто вони автоматизують роботу, яка займала б у людини в десятки разів більше часу і потребувала б більше грошей [10].

1.3 Основні та принцип роботи парсингу

Парсер оптимізує цей процес. З їх допомогою ви зможете досягти бажаного ефекту всього за «кілька кліків». Це дуже зручно, і алгоритм особливо підходить для тих інтернет-магазинів, які продають індивідуально брендовані товари на офіційному сайті, де зосереджені тисячі найменувань товарів. [10].

Використовувати резольвер в інтернет-магазині чи іншій сфері діяльності дійсно зручно. Адже це дозволяє:

- Збирати та швидко копіювати інформацію з інших сайтів;
- Дублюйте дані та зберігайте інформацію постійно актуальною;
- Збирайте та обробляйте великі обсяги інформації, а потім розміщуйте її в особистих ресурсах.

Незалежно від формальної мови програмування, на якій написаний парсер, алгоритм його дій однаковий:

- Доступ до Інтернету, отримання та завантаження кодів мережевих ресурсів;
- Читати, витягувати та обробляти дані;
- Представляти витягнуті дані в доступній формі – файли .txt, .sql, .xml, .html та інші формати.

Багато сайтів стверджують, що пошукові роботи або боти постійно працюють. Однак більшість з них насправді працюють на персональному комп'ютері.

Парсер принципово відрізняється від комп'ютерного вірусу, який є автономною програмою, здатною до відтворення. По суті, парсер працює подібно до трояна, оскільки отримує дані без дозволу власника. Синтаксичний аналізатор також отримує дані, які можуть бути приватними та навіть нефункціональними.

Збір інформації в Інтернеті займає багато часу, трудомістко і рутинно. Кожен, хто вміє користуватися парсером, може знайти необхідну інформацію за один день, переглянувши більшість веб-ресурсів[11].

Дані приватної пошукової системи збирає її активна всесвітня мережа. Програми використовують аналіз для автоматичної перевірки унікального вмісту сотень веб-сторінок. Це одна з причин, чому існують академічні програми, такі як дисертації та програми, зосереджені на аналізі.

Парсинг є важливим інструментом для організаторів спаму або каналів мобільного зв'язку. Для цього їм потрібно запустити «бота» для збору «номерів телефонів, адрес». Деякі власники веб-сайтів, особливо нещодавно організованих веб-ресурсів, люблять наповнювати свої сайти чужим контентом.

Щоправда, вони ризикували, бо пошукові системи їх швидко знаходили та банили.

Звичайно, парсери не читають текст, вони просто порівнюють запропоновані фрази з фразами, які вони знайшли в Інтернеті, і діють відповідно до заданої програми. Як пошукова система має туди потрапити Те, що вона знаходить, записується в командному рядку, який складається з набору літер, слів, виразів і символів синтаксису програмування. Такі командні рядки називаються «регулярними виразами».

Щоб синтаксичний аналізатор розумів регулярні вирази, вони повинні бути написані мовою, яка підтримує обробку рядків. Ця можливість доступна в PHP і Perl. Стиль написання регулярних виразів Unix вважається застарілим, але він надзвичайно популярний завдяки своїй сумісності зі старими файлами.

Налаштувавши поведінку аналізу синтаксису Unix, ви можете зробити його «лінивим», «жадібним» або навіть «наджадібним». Довжина копій текстового аналізатора з веб-сторінки залежить від параметра аналізу синтаксису. Супер скупий синтаксичний аналіз є надзвичайно скромним і отримує повну таблицю HTML і CSS із зовнішнього сайту таблиць CSS як єдиний вміст.

Програми аналізу отримують інформацію з документів, аналізуючи їх синтаксис і вибір слів. Це автоматичні перекладачі, які перетворюють текст з однієї мови на іншу або мови програмування на іншу мову програмування, на якій можуть писати програми. Так працює більшість автоматизованих перекладачів — вони аналізують синтаксис і вибір слів у документах, щоб створити програмний код або запити SQL. Зараз усі процеси, які використовують синтаксичний аналіз, використовують парсери.

Для створення парсерів використовуються різні мови програмування. Це включає широкий спектр мов[14].

Загалом процес аналізу можна розділити на три етапи:

- Аналіз необроблених даних, вибір файлів для подальшої обробки.
- Розберіть обраний файл на складові частини.

- Виберіть необхідну інформацію та збережіть дані в потрібному форматі. [19].

Розбір HTML-сторінки — це процес, який можна розділити на три етапи:

1. Отримати вихідний код веб-сторінки - завантажити програмний код сторінки сайту, яка потребує вилучення інформації. У різних мовах для цього існують різні методи, наприклад, в PHP найчастіше використовується бібліотека cURL або вбудована функція file_get_contents.

2. Витягніть необхідні дані з html-коду. Після отримання сторінки її необхідно обробити - відокремити звичайний текст від гіпертекстових тегів, побудувати ієрархічне дерево елементів документа, правильно реагувати на недійсні коди, витягти зі сторінки саме ту інформацію, на яку спрямований весь процес.

3. Фіксація результатів. Після успішної обробки даних на сторінці їх необхідно зберегти в необхідному вигляді для подальшої обробки.

Розріджені дані зазвичай зберігаються в базі даних, але є й інші варіанти. Іноді вам потрібно записати у файл CSV або створити ієрархічну структуру JSON, іноді вам потрібно перетворити на таблицю Excel, можливо, навіть створити динамічний потік RSS. Часто потрібно розгортати не одну сторінку свого донорського сайту, а багато. У цьому випадку після виконання кроків 1-3 алгоритм парсера повинен включити перехід на наступну сторінку сайту, щоб отримати з неї весь необхідний матеріал. [20].

Переконайтеся, що всі необхідні сторінки веб-сайту обходять різними способами:

- По-перше, при обробці наступної сторінки парсер можна навчити не тільки отримувати необхідні дані, але й вводити всі внутрішні посилання, які він знаходить у своїй базі даних. Звертаючись до свого сховища посилань, програма послідовно відвідує сторінки сайту, поки не обійде їх усі;
- По-друге, під час початкового аналізу веб-сайту часто можна відстежити логіку формування URL-адреси сторінки. Потім на основі ідентифікованого шаблону генерується адреса;

- По-третє, як не дивно, деякі резолвери розраховані на проходження мережевих ресурсів «вручну». Натискаючи на посилання, користувач сам вирішує, які сторінки йому відвідувати, а які – ні. Програма запам'ятовує необхідні дані у фоновому режимі.

Звичайно, ніщо не заважає поєднувати різні підходи. Але перш ніж аналізувати необхідні дані, потрібно знати, чи визначено їх законність [21].

Концепція синтаксичного аналізу не має юридичного визначення, як і пов’язані поняття сканування (обхід обмежень сайту) і сканування (поєднання сканування та аналізу).

Оскільки парсинг є одним із методів збору інформації, на цей процес поширюються правила, встановлені для обробки інформації, зокрема український «Закон про інформацію».

Інформація поділяється на дві групи: за тематикою та за способом доступу. Ці фактори визначають спосіб доступу, за винятком вмісту, який вважається обмеженим. Інформація вважається необмеженою, якщо інше не оголошено обмеженою. Законом України визначено три категорії інформації, доступ до якої обмежено певною групою осіб. Це службова інформація, секретна інформація та конфіденційна інформація. Будь-яка інша інформація є загальнодоступною та може бути використана з будь-якою метою. Важливо визначити конфіденційну інформацію, оскільки багато судових суперечок щодо законності аналізу закінчуються питанням про те, чи була інформація проаналізована. Будь-яка публічна інформація не буде вважатися конфіденційною чи офіційною. [22]

Для цього випуску ми надаємо приклади та тематичні дослідження, пов’язані з цим питанням. Одним із прикладів є люди, які збирають бази даних користувачів для послуг. Наприклад, збір передплатників спільноти в соціальній мережі. Почнемо з того, що практично вся інформація про фізичну особу є конфіденційною і може бути використана лише за її згодою. Користувачі дають свою згоду під час реєстрації на сайті. Цей сайт стає розпорядником особистої інформації. Такими відомостями є відомості або сукупність відомостей, за якими можна ідентифікувати особу. Частковим

винятком є інформація про державних службовців та інших публічних осіб (музикантів, акторів, спортсменів).

Контролер персональної інформації також повинен дати згоду на використання персональних даних. Коротше кажучи, щоб аналіз був законним, знеособлені дані повинні бути проаналізовані або отримана згода від розпорядника інформації [23].

Якщо ми говоримо про неособисту інформацію, то вона визначається як така лише в тому випадку, якщо нею володіє її власник. Як правило, веб-сайти пишуть політику конфіденційності, з якою користувачі знайомляться та погоджуються дотримуватися під час реєстрації. Інформація, до якої можна отримати доступ лише через процес авторизації, майже напевно є конфіденційною та не може використовуватися без згоди власника.

Однак, навіть якщо інформація доступна несанкціоновано, це не означає, що інформація є вільною. Перед синтаксичним аналізом ви повинні переконатися, що правила використання сайту (якщо такі є) забороняють використання даних сайту [24].

Крім того, цей сайт може висловити свої заперечення щодо використання його інформації в інші способи. Наприклад, шляхом блокування IP-адреси резолвера або шифрування даних. Було б незаконно обійти ці блоки, навіть якщо вони були встановлені після початку аналізу.

авторське право.

Також варто згадати аналіз інформації, яка містить дані, захищені авторським правом. Вам потрібно дуже ретельно проаналізувати таку інформацію, оскільки використання такої інформації у вашому проекті цілком може (залежно від типу ліцензії) порушити авторські права.

Збій сайту. Навіть якщо парсинг законний, його виконання не повинно порушувати нормальну роботу аналізованого сайту [25].

Після такої втрати з парсером неминучим питанням є те, як законно проводити синтаксичний аналіз і як запобігти такому збиранню інформації. Наприклад, неможливо повністю уникнути судових процесів, викликаних

розбором, оскільки ця область не визначена. Однак їх вірогідність можна звести до мінімуму.

Досить дотримуватися простих правил:

1. Не аналізуйте обмежену або захищену авторським правом інформацію.

2. Перевірте, чи правила користування сайтом забороняють парсинг.

3. Якщо можна отримати згоду на аналіз, найкраще це зробити. До речі, ця згода може бути API сайту.

4. Якщо цей сайт вживає заходів для обмеження або припинення аналізу після його початку, не слід ігнорувати або обходити його.

5. Не порушувати роботу сайту.

Якщо дотримуватися всіх цих правил, ці позиції можуть бути дуже переконливими навіть у суді. Якщо сайт уже є об’єктом вирішення, і ми не хочемо цього, ми повинні зробити наступне:

1. Заблокуйте відповідні IP-адреси.

2. Надішліть запит автору парсера (якщо його можна встановити) для припинення парсингу.

3. Звернутися до суду про припинення порушення.

Але найголовніше, рекомендується написати детальні умови використання та політику конфіденційності для веб-сайтів [26].

РОЗДІЛ 2
МОДЕЛЬ ОБРОБКИ ДАНИХ З БІБЛІОГРАФІЧНОЮ ІНФОРМАЦІЄЮ.

2.1 Сучасні популярні бібліографічні бази даних

Існує багато рішень, розроблених спеціально для пошуку бібліографічної інформації. Вони широко використовуються вченими по всьому світу для зберігання своїх робіт та розробок.

Найбільш відомими рішеннями на сьогоднішній день є:

- Google Scholar;
- Scopus;
- Web of Science (Clarivate);
- ScienceDirect;
- ArXiv (Корнельський університет);
- Вольфрам Альфа.

Кожна бібліографічна база даних має свою структуру та різну кількість статей за різними напрямками, тому не виключено, що деякі статті є ексклюзивними для певної платформи, або навпаки бути відсутніми на ній, а отже і для аналізу необхідно вибрати з великої кількості бібліографічних баз даних хоча б дві діаметрально протилежні, але водночас доступні широкому колу науковців та об'єднати їхні дані в єдину систему.

Google Scholar

Академія Google – це наукометрична база даних з відкритим доступом, яка створена найпотужнішою пошуковою системою Google. Розробником цього сервісу є вчений індійського походження Анураг Ачарья. Метою створення бази даних було допомогти аналітичній та науковій спільноті.

Цей інструмент дозволяє аналітикам здійснювати пошук широкого спектру наукової літератури, яка доступна в мережі Інтернет, а саме:

- рецензовані статті;
- книги;
- дисертації;

– технічні звіти від професійних товариств;
– технічні звіти від університетів;
– наукові журнали;
– тези доповідей;
– технічні звіти академічних установ;
– технічні звіти дослідницьких груп.

Зараз найбільший обсяг літератури має Google Академія і тому охоплює найбільшу кількість дослідницьких напрямків. Станом на 2016 року її база даних охоплювала 160 млн ексклюзивних документів. Цей показник у кілька разів перевищує аналогічний показник баз даних-конкурентів Scopus та Web of Science.

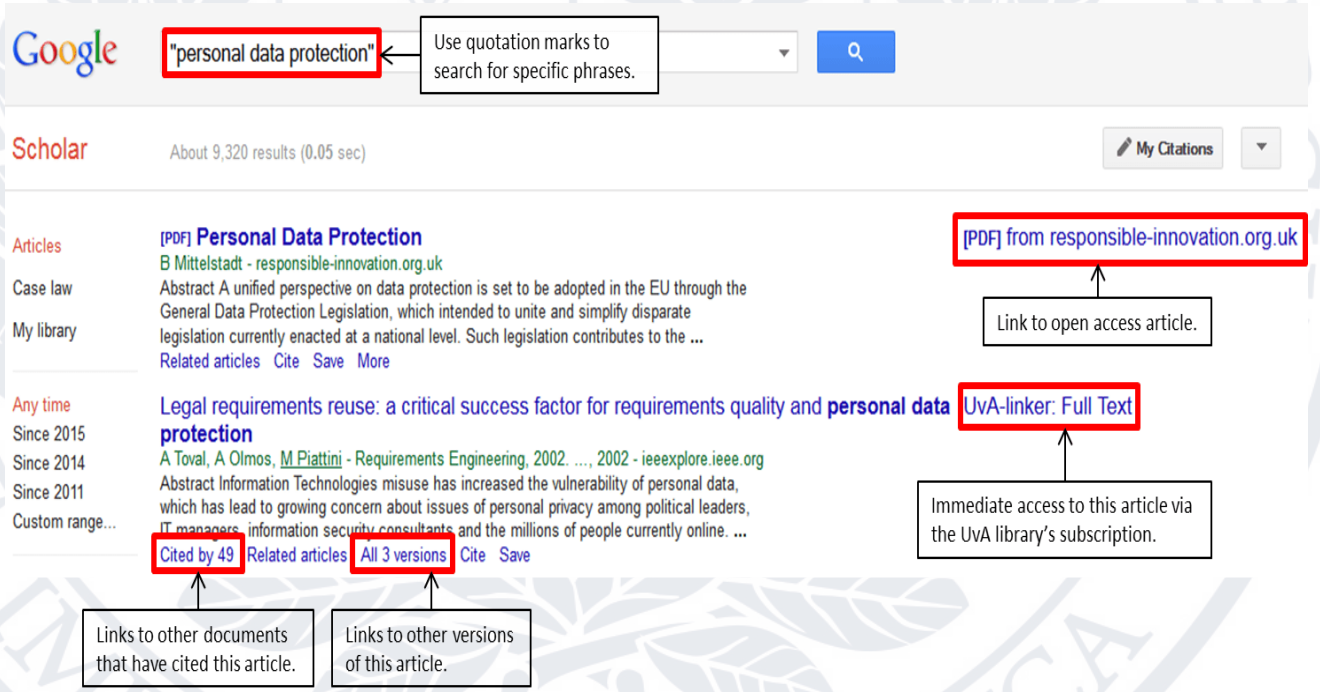

Рисунок 2.1 – Вигляд пошукової системи Google Академії

Академія Google включає як інформацію про документи, для яких доступна лише бібліографія або реферат, а також повнотекстові документи, якщо документи повністю доступні читачеві безкоштовно. Зокрема, послуги передбачають користувачеві деякі додаткові дані:

– індекс цитування документа;

– посилання на авторів статті;

– зовнішні посилання на документи, використані у статті.

На основі профілів науковців, що знаходяться у вільному доступі в Google Академії – Google постійно складає рейтинги наукових груп. Рейтинг складено за такими показниками критеріїв цитованості науковців, які належать до наукових установ або інших групи. Пошукова система представлена на рис. 2.1.

Платформа Scopus

Scopus – це база даних, яка містить велику кількість анотацій та інформації про цитування в науковій літературі. База даних призначена для відстеження цитувань у науковій літературі та має системи, які відстежують дії користувачів для подальшого аналізу популярності статей з візуалізацією статистики.

База даних включає статті з понад двадцяти трьох тисяч міжнародних видань та понад п'яти тисяч видавництв, які досягли авторитетного статусу в усьому світі. Вона широко визнана міжнародними університетами за якість зібраних даних, що високо цінується науковою спільнотою.

Дана бібліографічна база даних охоплює такі галузі, як

– природничі науки;

– гуманітарні та соціальні науки;

– медицина;

– мистецтво;

– інжиніринг.

Заснована у 2004 році власником видавничого дому Elsevier, заснованого у 1880 році в Амстердамі. Базою даних широко користуються рейтингові агентства для створення глобальних рейтингів серед університетів світу. До публікації на ресурсі допускаються лише визнані науковою спільнотою видавці. Для збору інформації необхідно бути членом наукової спільноти та пройти процес реєстрації на платформі.

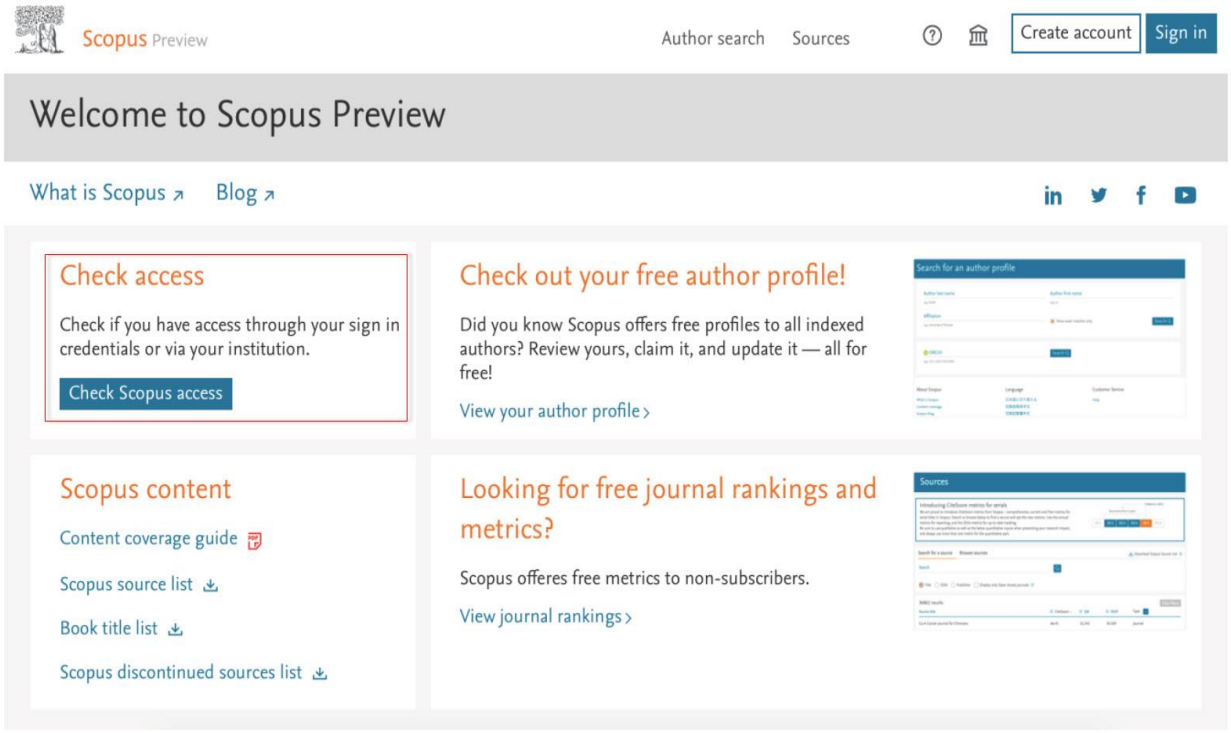

Рисунок 2.2 – Пошукова система Scopus (Elsevier)

Для того, щоб збирати дані – користувач повинен фактично бути частиною наукового закладу, яка має контракт з Elsevier. Платформа має простий інтерфейс та уніфіковану систему пошуку. Пошукова система проілюстрована на рис. 2.2.

Платформа Web of Science

Web of Science – це бібліографічна база даних, яка містить статті з понад 12 000 наукових журналів та 150 000 веб-сайтів. Дана база даних охоплює галузі природничих, гуманітарних, суспільних наук та мистецтв і дозволяє користувачеві обирати найбільш релевантні статті. Зокрема, платформа автоматично пов'язує опубліковані результати з іншими роботами.

Платформа здатна надати користувачеві:

- графічне представлення інформації про цитованість обраної статті серед інших;
- велику кількість сценаріїв пошуку;
- автоматично надає користувачеві інформацію про авторів та їх організації.

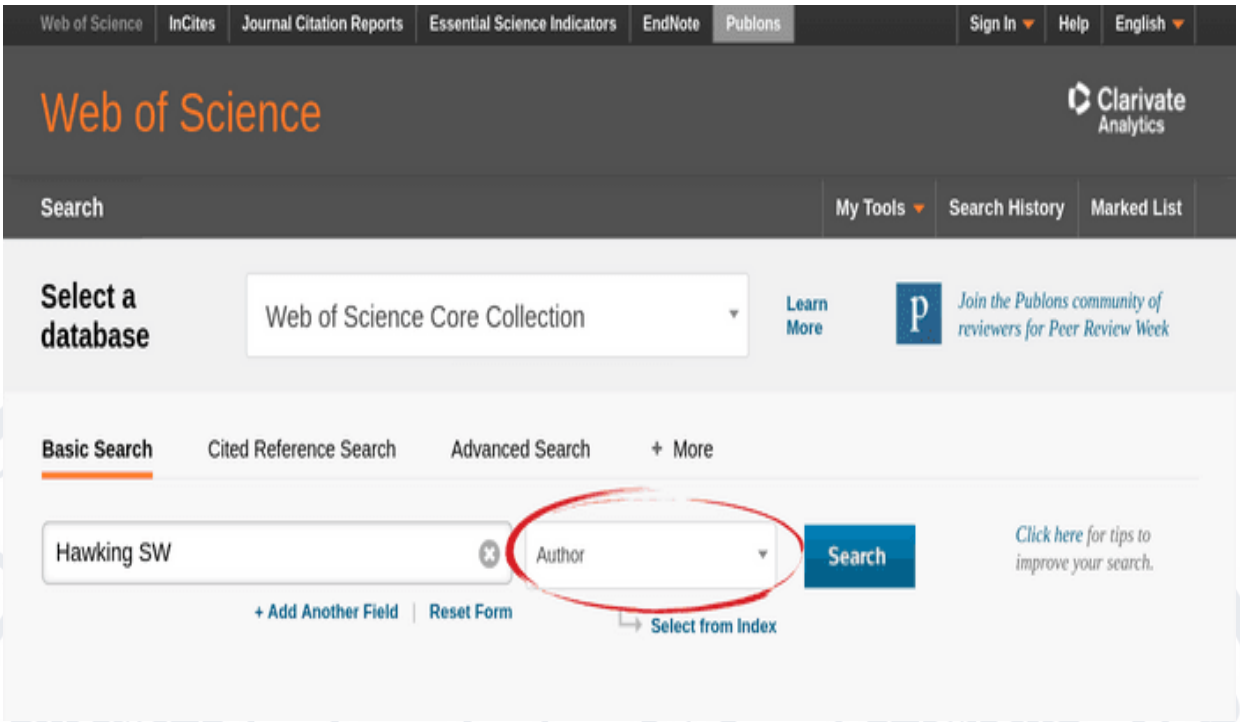

Рисунок 2.3 – Вигляд пошукової системи Web of Science

Пошукова система представлена на рис. 2.3 Платформа доступна користувачеві на умовах платної передплати.

Платформа ScienceDirect

ScienceDirect – це джерело медичної, наукової та технічної інформації, яке зберігає близько 14 мільйонів публікацій з провідних наукових журналів та книг, опублікований видавництвом Elsevier.

ScienceDirect використовується студентами, викладачами та аналітиками по всьому світу для проведення досліджень у різних галузях. Ресурс містить більшість фундаментальних праць, які були написані провідними вченими свого часу з 1823 року. Зокрема, він містить понад 400 публікацій 74 нобелівських лауреатів у різних галузях.

Платформа містить численні інструменти пошуку, які дозволяють користувачеві знайти необхідну інформацію, а саме:

– зручний інтерфейс;
– віддалений доступ до досліджень;
– можливість роботи на мобільних пристроях;

– рекомендації, які надають користувачеві посилання на статті, пов'язані з дослідженням;
– спеціальний пошук за зображеннями або відео;
– візуалізація даних статті в режимі реального часу.

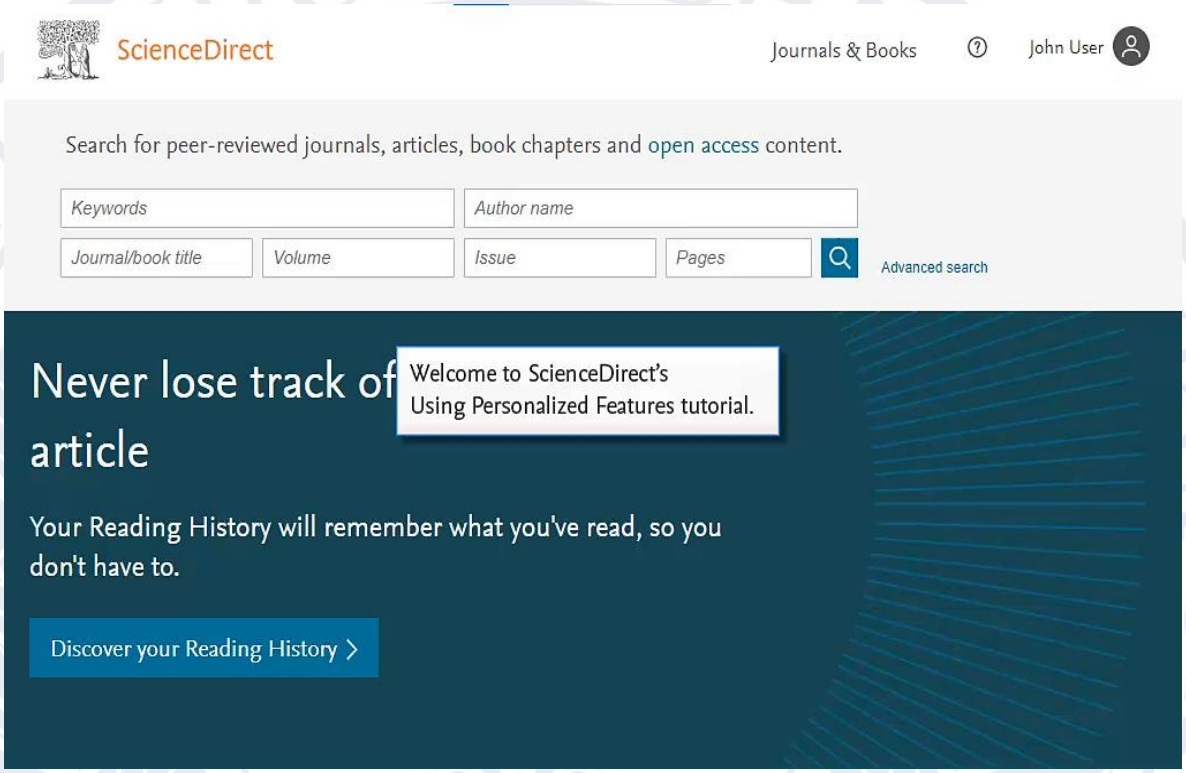

Рисунок 2.4 – Вигляд пошукової системи ScienceDirect

Пошукова система проілюстрована на рис. 2.4. Платформа доступна користувачеві на умовах платної підписки.

Платформа ArXiv

ArXiv (Архів) – база даних, створена на базі бібліотеки Корнельського університету. Корнельський університет (Cornell University), де знаходиться велика кількість публікації: наукові статті та їх публікації, тези доповідей та дисертації. Архів був створений на прохання американського астрофізика Джона Кона. До серпня 1991 року Джон отримав на свою особисту поштову скриньку наукові статті від 180 вчених в галузі астрофізики. Лише в галузі фізики до нього надійшли статті від 180 вчених. Звідси і виникла ідея створення архіву наукових статей, який буде класифіковано за науковими напрямами.

Архів є абсолютно безкоштовним як для всіх бажаючих розмістити статті, так і для видавця – не буде проблем з логістикою, порівнянням текстів та плагіатом. Також не виникатиме питань щодо авторитетності видавця в науковому середовищі, тому кожен бажаючий може опублікувати свою роботу для ознайомлення та оцінки іншими вченими про статтю. У 2018 році до архіву надійшло близько два мільйони статей у різних категоріях. В архіві можливо знайти роботи з наступних галузей:

– Фізика.
– Математика.
– Комп'ютерні науки.
– Статистика.
– Електротехніка.
– Економіка.

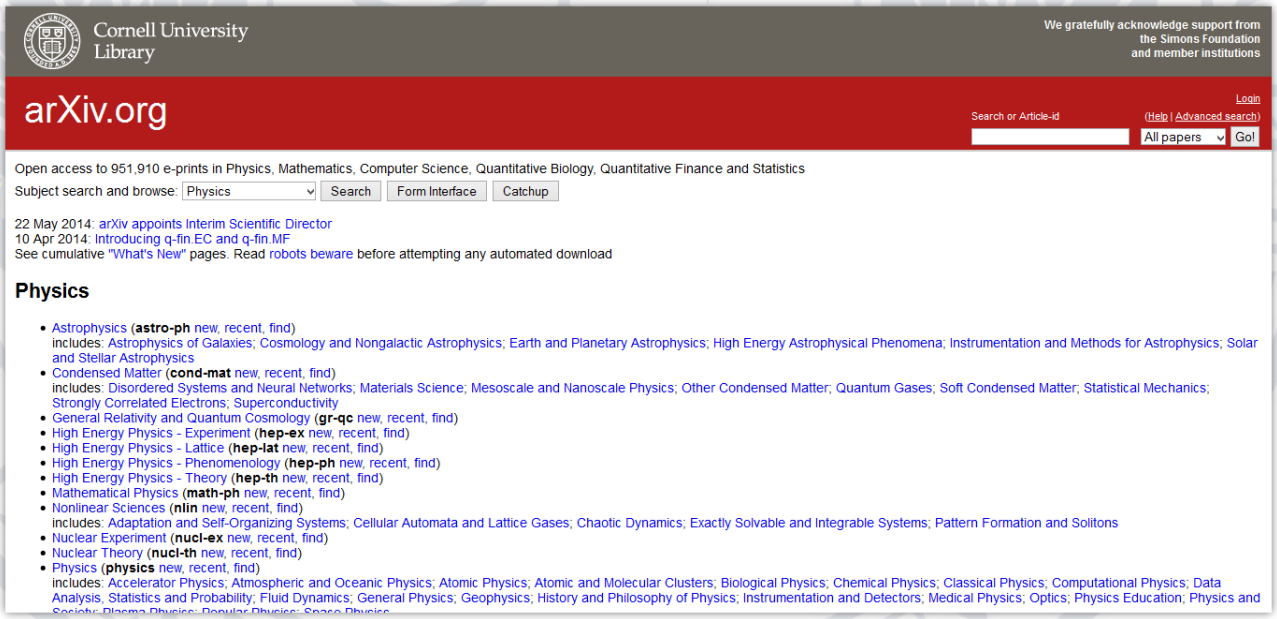

Рисунок 2.5 – Вигляд пошукової системи ArXiv

Незважаючи на відкритість платформи, архів є більш достовірним, ніж багато наукових журналів. Він є більш надійним, ніж багато наукових журналів, оскільки відповідність поданих статей обраній категорії перевіряється науковою спільнотою. Якщо стаття не відповідає обраній

категорії або не є актуальною для наукового співтовариства, категорія перевизначається до відповідної статті або поміщається в категорію з позначкою «мотлох». Вид пошукової системи наведена на рис. 2.5.

Платформа Wolfram Alpha

Wolfram Alpha – це об'єднана пошукова система, відома як «Wolfram Альфа», що є «обчислювальною машиною знань». Поєднуючи в ній елементи пошуку, менеджери результатів пошуку користувачів та алгоритми на основі обробки природної мови, Wolfram Alpha є прямим конкурентом найпопулярнішої пошукової системи Google. Даний ресурс не надає користувачеві список посилань на літературу, який є результатом запиту користувача, оскільки повністю покладається на власну базу даних.

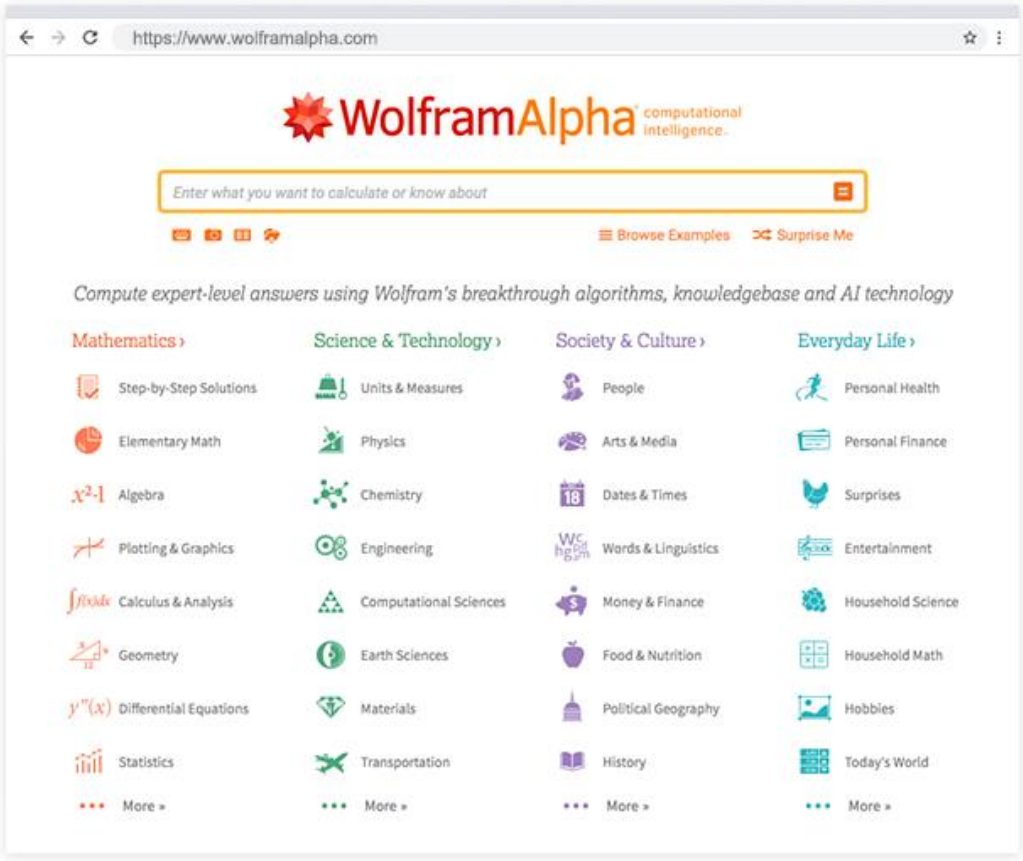

Рисунок 2.6 – Вигляд пошукової системи Wolfram Alpha

Пошукові сервери Wolfram мають близько 10 000 процесорів і постійно підтримуються в робочому стані, що дозволяє користувачам системи мати

безперебійний доступ до інформації. Користувачі, які бажають збирати власні дані для подальшого аналізу необхідна платна підписка. Підписка включає доступ до хмарних сховищ та дослідницької статистики, на основі якої користувач отримує рекомендації щодо подальшої роботи наукових досліджень. Пошукова система представлена на рис. 2.6.

2.2 Математична модель парсингу бібліографічної інформації

Найпростішим алгоритмом зі сфери статистики є метод лінійної регресії. За допомогою лінійної регресії можна прогнозувати значення однієї змінної, яка залежить від значень іншої. Значення, яке дослідник хоче отримати, називається критеріальною змінною і часто позначається як «*Y*». Змінна, на основі якої проводиться аналіз, позначена символом «*X*». У лінійній регресії зв'язок між даними моделюється за допомогою так званих лінійних функцій, а невідомі параметри моделі оцінюються на основі вхідних даних. Як і інші методи регресійного аналізу – лінійна регресія повертає розподіл ймовірностей змінної *Y*, як функцію значення змінної *X*, але не розподіл їх спільних ймовірностей. При розрахунку таких моделей для корекції похибки зазвичай використовується так званий метод найменших квадратів, але можуть застосовуватися й інші методи [11].

У загальному випадку лінійну регресію можна позначити наступним чином:

$$y = \beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n + \varepsilon$$

де y – залежна змінна та множина x – незалежні змінні відповідно, ε – похибка моделі. При цьому математичне очікування цієї змінної також є лінійною функцією і розраховується за формулою:

$$E(y) = \beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n + \varepsilon$$

Вектор параметрів β невідомий, і завдання лінійної регресії полягає в тому, щоб оцінити ці параметри на основі наявних наборів даних для y та x. Тобто, на основі ряду n експериментів визначаються значення $\{y_i, x_1 \dots x_n\}_{i=1}$ незалежних змінних та відповідне значення залежної змінної.

Таким чином, отримано матричне рівняння для змінної y з урахуванням кількості експериментів наступне:

$$y = X\beta + \varepsilon$$

За цими даними необхідно оцінити значення параметрів β, а також розподіл похибок ε. Для мінімізації похибки часто використовується метод найменших квадратів. У цьому методі в якості параметрів оцінок беруться значення, які мінімізують суму квадратів залишків для всіх спостережень:

$$\beta = argmin \sum_{i=1} |y_i - \beta_0 - \sum_{j=1} x_{ij}\beta_j|^2 = argmin||y - X\beta||^2$$

Іншим методом регресії, який може бути використаний для прогнозування, є так звана квантильна регресія, яка має деякі переваги перед звичайною лінійною регресією, але це не означає, що звичайна лінійна регресія не повинна використовуватися, в деяких випадках простота є вищим пріоритетом і явна лінійна характеристика даних може дати хороші результати при використанні лінійної регресії. У загальному випадку лінійна регресія узагальнює зв'язок між сукупністю регресорів і значенням вихідної змінної Y на основі значення функції $E(y|x)$. Однак такий підхід дає нам лише часткове уявлення про залежність, а хотілося б отримати залежності від різних точок Y. Це те, що дозволяє зробити квантильна регресія.

Як і у випадку з функцією, що використовується в лінійній регресії можемо отримати значення, використовуючи, наприклад, функцію медіани $Q_\tau(y|x)$, де медіана дорівнює 50% або квантилю τ множини.

Отримаємо:

$$\beta(\tau) = argmin \sum_{i=1} |\rho_\tau(y_i - x'\beta)|$$

Принцип цього методу такий самий, як і у звичайної лінійної регресії, але цей алгоритм дозволяє розбивати дані на квантилі різних порядків і знаходити залежності в різних квантилях. Медіанна регресія є підвидом квантильної регресії, де медіаною є квантиль $\tau = 0{,}5$. Однак, хоча ці методи дають хороші результати і в контексті даної роботи є ідеальними кандидатами, оскільки розподіл використання фреймворків, мов програмування, засобів розробки в

часі, необхідно відзначити й інші методи прогнозування, які можуть бути використані в даній роботі.

Ще одним методом прогнозування є використання нейронних мереж. Дослідження в галузі нейронних мереж є результатом вивчення спеціалізованих частин, які називаються нейронами. Нейрон – це клітина, яка має кілька входів, що можуть бути активовані зовнішнім процесом. Залежно від кількості активацій і ваги активацій, нейрон виробляє власну вагу активації і посилає її на свої виходи. Крім того, певні входи або виходи можуть бути посиленими, тобто мати більшу вагу, ніж інші входи або виходи даного нейрона. Припущення про те, що мозок людини є нічим іншим, як мережею нейронів, дозволяє нам емалювати мозок, моделюючи нейрон, і включати ці нейрони в мережу за допомогою графа з різними вагами на ребрах [12].

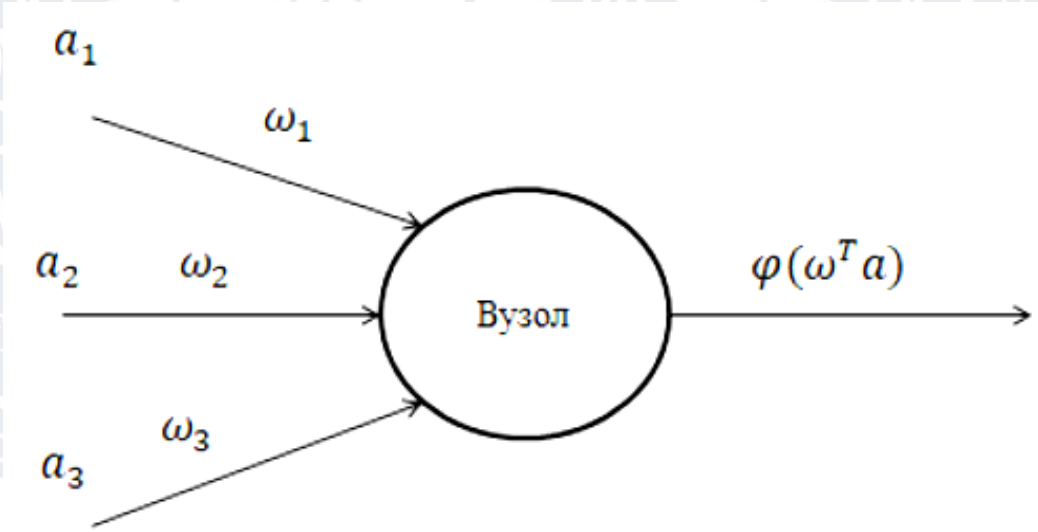

Рисунок 2.7 – Вузол нейронної мережі

Штучним еквівалентом нейрона є так званий вузол (також називається нейрон, але в даній роботі для уникнення плутанини будемо використовувати термін вузол), який отримує набір входів з певною вагою, цей вузол обчислює їх суму за допомогою функції активації φ і передає результати цієї функції наступним вузлам в графі. У загальному вигляді це можна представити формулою:

$$\varphi(\sum_i \omega_i a_i) = \varphi(\omega^T a)$$

Візуально цей вираз можна представити так, як показано на рис. 2.7.

І основна функція активації – лінійна:

$$\varphi(\omega^T a) = \omega^T a$$

Гіперболічний тангенс також є широко використовуваною функцією:

$$\varphi(\omega^T a) = \tanh(\omega^T a)$$

Тому, з'єднавши ці вузли між собою, можна утворити мережу.

Зауважимо, що при використанні мереж для виконання прогнозних операцій простого створення такої мережі може бути недостатньо. Така мережа може бути використана для виконання операції регресії, але для виконання операції прогнозування необхідно використовувати так звані рекурентні мережі, де попередній вихід може бути пов'язаний з наступним входом, як показано на рис. 2.8.

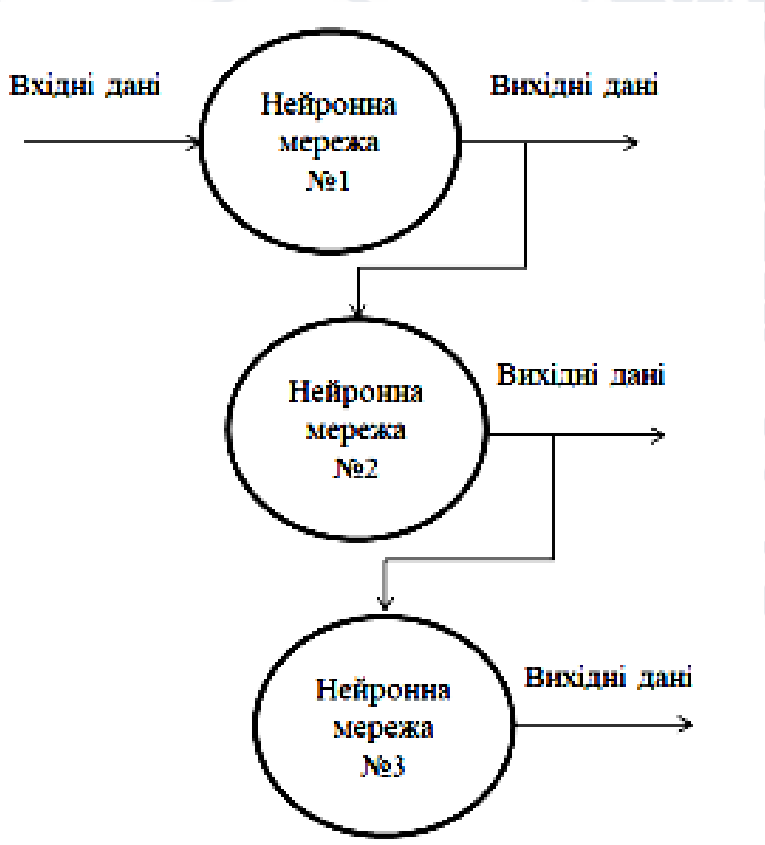

Рисунок 2.8 – Проста рекурентна нейронна мережа

Хоча на рисунку 2.8 показано лише декілька нейронних мереж, об'єднаних в одну, а саме три, слід зазначити, що в реальній задачі таких мереж може бути більше, і на рисунку 2.8 не вказано точну кількість внутрішніх шарів, необхідних в такій мережі. Тому можна уявити, як має працювати така мережа при впливі попередніх значень.

Якщо кількість відомих інтервалів невідома, то отримаємо:

$$\varphi_t(\omega^T a) = \tanh(\varphi_{t\text{-}1}(\omega^T a) + u^T x_t)$$

де змінна u позначає значення ваги на вході мережі в зазначеній рекурентній мережі, а коефіцієнт x – відповідне фіксоване значення на вході. Зауважимо, що за допомогою такої мережі можна робити прогнози на короткі проміжки часу [20].

Для виконання прогнозів на більш тривалі періоди часу можна використовувати нейронну мережу LSTM (Long Short-Term), яка є покращеною рекурентною мережею, що може складатися з декількох нейронних мереж, кожна з яких використовує внутрішні зміни стану, які передаються між вузлами і відповідно модифікуються [20].

Основним завданням є модифікація внутрішнього стану мережі. Такий затвор представляється у вигляді функції, він може приймати фіксовані значення входів мережі та інші необхідні значення, з якими потім використовується операція множення значень внутрішнього стану. При виході з затвора 0 внутрішній стан стає рівним 0, тому його називають затвором забування. Формула такої трансформації має вигляд [20]:

$$f_t = \sigma(\omega_f[h_{t\text{-}1}, x_t] + b_f)$$

На вхід вентилів повинні подаватися такі значення, щоб вихід мережі використовував раніше задані вхідні значення, створюючи на виході число з інтервалу до 1, яке використовується для перерахунку стану [20]:

$$i_t = \sigma(\omega_i[h_{t\text{-}1}, x_t] + b_i)$$

Вихідний затвор використовується для керування відповідними вихідними величинами і коефіцієнтом внутрішнього стану, відповідно, ця частина внутрішнього стану передається на вихід [20].

Така нейромережева модель може бути використана для прогнозування за тривалими періодами даних, на відміну від раніше описаної моделі з декількома мережами, об'єднаними в одну рекурсивну [20].

Найпростішим методом прогнозування є так званий метод експоненціального згладжування. Цей метод дозволяє робити прогнози на короткі періоди, наприклад, на наступний тиждень, день або рік, але його можна використовувати і на більш тривалі періоди, але результат буде ще менш точним [8, 20].

Можна зробити висновок, що основні значення прогнозів на наступних кроках залежать від попереднього значення та попереднього прогнозу. Також прогнози для вже відомих значень приймаються рівними цим значенням, а якщо прогноз не потрібен на більш тривалі інтервали, то передбачається, що отримане прогнозне значення є реальним і очікуваним, стандартне значення α часто обирається рівним 0,5 [20].

Цей алгоритм можна назвати експоненціальним, оскільки він має експоненціальний характер при записі формули в повному вигляді [20]:

$$s_t = \alpha[x_t + (1 - \alpha)x_{t-1} + (1 - \alpha)^2 x_{t-2} + \cdots + (1 - \alpha)^{t-1} x_1] + (1 - \alpha)^t x_0$$

Останнім способом прогнозування є використання ARIMA-моделей. Теоретично ці моделі широко використовуються в прогнозуванні в поєднанні з так званими нелінійними кількісними перетвореннями. Випадкова величина, яка є часовим рядом – стаціонарна, якщо її відповідні властивості змінюються, такі ряди не мають певних тенденцій, але змінюються навколо свого середнього значення. ARIMA (Auto-Regressive Integrated Moving Average) використовує три основні компоненти, описані нижче [13, 20]. Авторегресійний компонент використовує попередні значення в рівнянні регресії ряду Y. У визначенні $ARIMA(p,\ d,\ q)$ параметр p вказує на кількість попередніх результатів, які будуть використані в рівнянні, а сама авторегресійна модель позначається $AR(p)$. Наприклад, модель $AR(2)$ або $ARIMA(2,\ 0,\ 0)$ матиме наступний вигляд [10]:

$$Y_t = c + \varphi_1 y_{t-1} + \varphi_2 y_{t-2} + e_t$$

Іншою складовою цієї моделі є так звана інтегральна складова або інтегрована $I(d)$ складова, параметр d цієї складової відповідає за ступінь різниці в даній моделі. Різниця між рядами передбачає просту операцію віднімання поточного та попереднього d разів [20].

ARIMA має свої переваги та недоліки, наприклад, ARIMA є лінійним, але для його використання необхідний набір попередніх значень, і хоча, на відміну від лінійної регресії, значення, що прогнозуються, не обов'язково повинні бути лінійними, однак, хоча основною метою є прогнозування значень в моделі, використання ARIMA не дасть причини, чому значення та взаємозв'язок параметрів моделі будуть такими.

2.3 Метод збору та аналізу бібліографічної інформації

Структура веб-сайтів

Для того, щоб збирати бібліографічну інформацію зі спеціалізованих сайтів в автоматичному режимі, необхідно розуміти загальну структуру сайтів, які у всесвітній мережі Інтернет та базові знання Інтернет-протоколів.

Структура сайту – це схема розташування сторінок сайту, його категорій, підкатегорії та інші об'єкти. Іншими словами, це схема плану, за яким знаходиться логічний зв'язок між сторінками сайту. Існує 3 основних типи структури сайту: лінійна, довільна та ієрархічна.

Рисунок 2.9 – Приклад лінійної структури сайту

Лінійна структура – використовується, якщо сайт надає користувачеві однакову поетапну інформацію користувачеві. Зазвичай, інформація на таких сайтах переглядається від першої до останньої сторінки, а кожна сторінка

містить лише посилання на попередню та наступну сторінки. Орієнтовна схема лінійного веб-сайту наведена на рисунку 2.9.

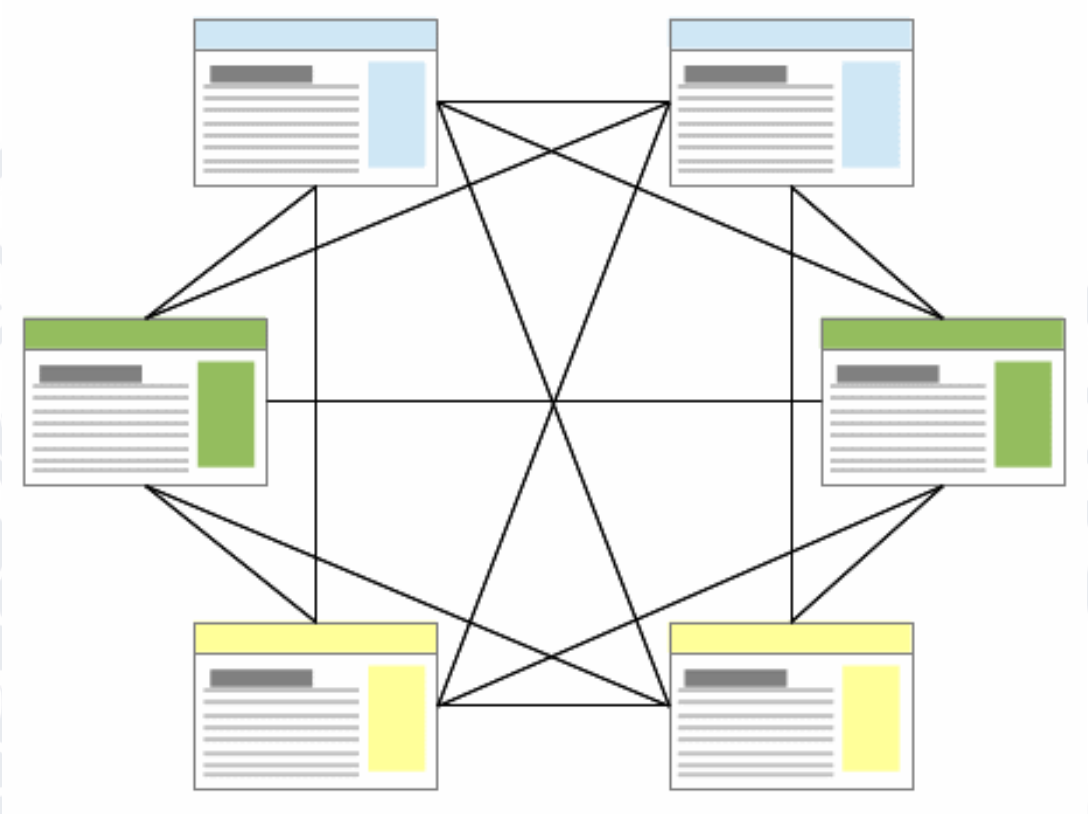

Рисунок 2.10 – Приклад довільної структури веб-сайту

Довільна структура – використовується для сайтів, які не передбачають відсутність обмежень між посиланнями в середині сайтів. Подекуди окремі частини цих веб-сайтів можуть містити елементи з лінійною та ієрархічною структурою. На рисунку 2.10 наведено приблизну схему веб-сайту довільної структури.

Ієрархічна структура вважається доцільною, коли використовуються сторінки з різними рівнями доступу. Кожна сторінка вищого рівня має посилання на сторінку нижчого рівня. Саме така структура використовується для каталогів та бібліографічних баз даних. На рисунку 2.11 показано приблизну схему ієрархічно структурованого веб-сайту.

Найскладніші сайти складаються з доменів :

– розмітка для даних і об'єктів, які несе в собі сайт;

– каскадна таблиця стилів, згідно з якою оформлено сайт;
– скрипти, які виконуються при взаємодії користувача з веб-сайтом;
– скрипти, які передають дані з сервера на сайт;
– база даних, яка структурує та зберігає інформацію, необхідну для належного функціонування веб-сайту та користувачів.

Рисунок 2.11 – Приклад ієрархічної структури веб-сайту

База даних використовується для зберігання інформації, необхідної для належного функціонування веб-сайту, а також для зберігання даних користувачів. Мова розмітки HTML відповідає за розмітку веб-сайтів. Це вказує на розташування блоків, шрифтів, кольорів блоків і тексту, посилань.

Каскадна таблиця – це мова розмітки CSS. Застосовується для дизайну і реакція сайту на дії користувача. Скрипти, які використовує сайт, можуть бути реалізовані на будь-якій мові програмування, але найчастіше використовуються мови, які були розроблені під конкретні завдання. Для взаємодії користувача з сайтом найчастіше використовується мова програмування JavaScript. Скрипти для взаємодії користувачів з сайтом виконуються на стороні користувача. Для виконання процедур взаємодії між сервером і сайтом дуже популярна серверна

мова програмування PHP. Саме через цю мову сервер отримує команди для виконання.

База даних – структура, організована для зберігання даних відповідно до певної концепції, яка описує характеристики цих даних та їхні властивості. У сучасних сайтах різні дані можуть зберігатися на різних серверах, щоб зменшити навантаження на сервер.

Аналіз та зберігання даних

Для автоматичного збору бібліографічної інформації використовують програму або скрипт, який повинен вміти аналізувати сайт і зберігати всі необхідні для користувача дані. Такі скрипти або програми називаються парсерами, а сам процес автоматичного збору інформації з інтернет-ресурсів – парсинг.

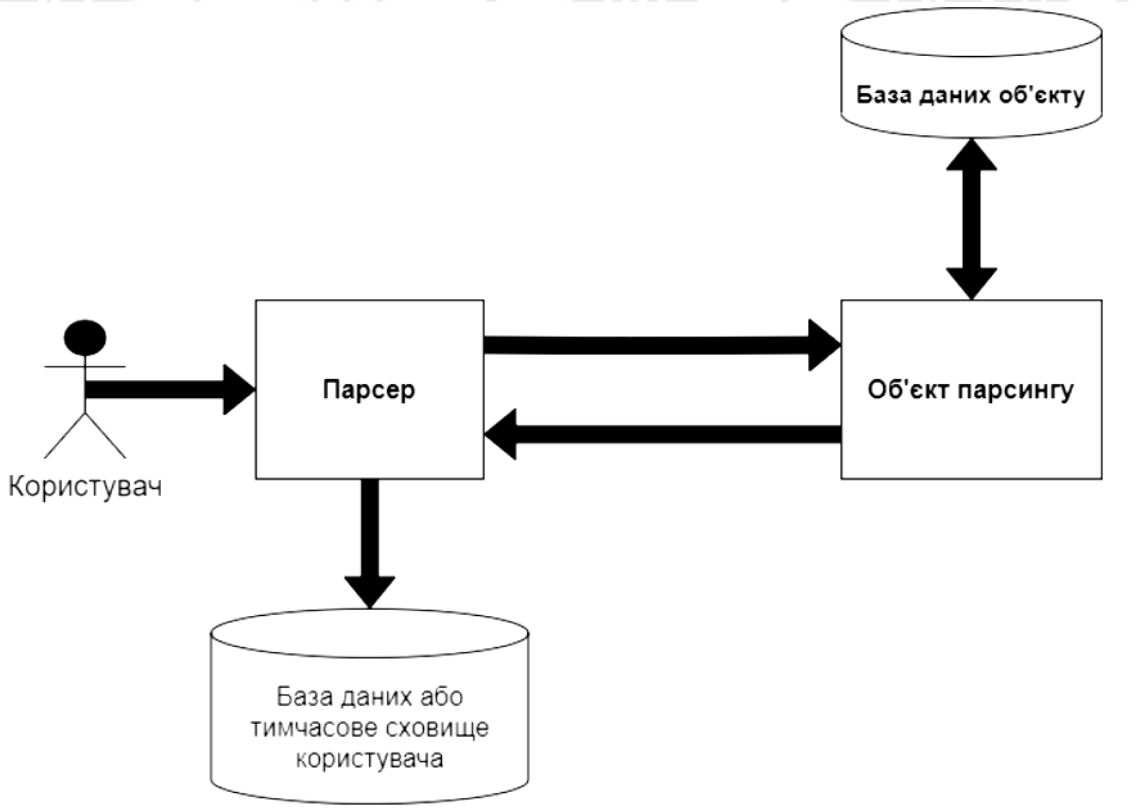

Рисунок 2.12 – Блок-схема роботи скриптів синтаксичного аналізу

Принцип роботи парсеру: створити скрипт пошуку обраного об'єкта збору даних, зв'язати цей скрипт з базою даних сервера, зібрати і обробити

отримані дані, занести дані в базу даних. На рисунку 2.12 наведено приблизну схему роботи синтаксичного аналізатора.

Об'єктом парсинга може бути будь-який інтернет-ресурс, дані якого можна використовувати для подальшої переробки. Результати успішного розбору безпосередньо залежать від захищеності ресурсу, який є об'єктом парсингу. Використання парсерів є суперечливим питанням у всесвітній павутині, оскільки існує можливість збору небажаних для власника даних, без зазначення власника, з порушенням прав інтелектуальної власності, з корисливою метою або навіть шантажу. Парсер став потужним інструментом для аналізу даних у різних сферах життя.

Основними перевагами використання парсерів є:

- Здатність збирати великі обсяги інформації та можливість розміщувати її за певними алгоритмічними сценаріями в необхідних сховищах даних.
- Автоматичний моніторинг та адаптація до змін даних у джерелі, з якого збирається інформація.
- Швидкість у зборі необхідної інформації.
- Точність і безпомилковість результатів завдяки ідеально написаному скрипту.
- Гнучка конвертація інформації в будь-який формат даних.

Незважаючи на велику популярність використання парсерів, вони мають недоліки:

- Залежить від структури кожного сайту окремо, а тому в цілому парсер може виконувати свої завдання на єдиному ресурсі.
- Модифікація скриптів або їх доповнення вимагає втручання програміста.
- При зміні структури сайту, з якого збирається інформація, спостерігаються наступні помилки у виконанні завдання.
- Для отримання бажаних результатів необхідні тривалі та ретельні випробування.

– Вони не можуть повністю замінити людські ресурси.

Ще одним важливим недоліком цих скриптів є неможливість розрізнення діяльності парсеру від дій звичайного користувача ресурсу. В результаті синтаксичний аналізатор може заважати аналізатору збирати необхідну інформацію.

Спосіб локального зберігання бібліографічної інформації.

Однією з найпоширеніших практик зберігання інформації, яка включає в себе елементи різних типів даних в локальному сховищі є створення локальної бази даних.

База даних – це фіксована модель, в якій всі необхідні дані зберігаються в певній формі. У разі зберігання інформації в загальній локальній базі даних доцільно використовувати реляційну модель бази даних. Реляційна модель бази даних – представлення даних у вигляді таблиць з рядками і стовпчиками, де рядки – це записи, а стовпці – поля, де зберігаються самі дані. Сама база даних може зберігати інформацію, але ніяк не може з нею взаємодіяти. Саме для цього була винайдена система управління базами даних (СУБД).

Керування збереженими даними

Для взаємодії програми з інформацією, що зберігається в базі даних, широко використовується технологія ORM.

ORM (Object Relational Mapping) – технологія програмування, метою якої є реалізація створення віртуальної бази даних на об'єктно-орієнтованих можливостях мов програмування.

Об'єктно-орієнтований підхід розроблено з урахуванням основних принципів програмної інженерії (наприклад, комунікація, агрегація, інкапсуляція) та реляційна база даних розробляється з урахуванням математичних теорій. Між цими двома наборами теорій існують суттєві відмінності.

Для усунення цієї невідповідності з'явилася технологія об'єктно-реляційного картографування. Об'єктно-реляційне відображення дозволяє розробникам використовувати бажану мову програмування для доступу до баз

даних без написання SQL (мова структурованих запитів) операторів або запитів вручну.

Використання технологій об'єктно-реляційного відображення забезпечує більш гнучкий спосіб програмування взаємозв'язку між програмою та базою даних, оскільки абстрагується від системи баз даних.

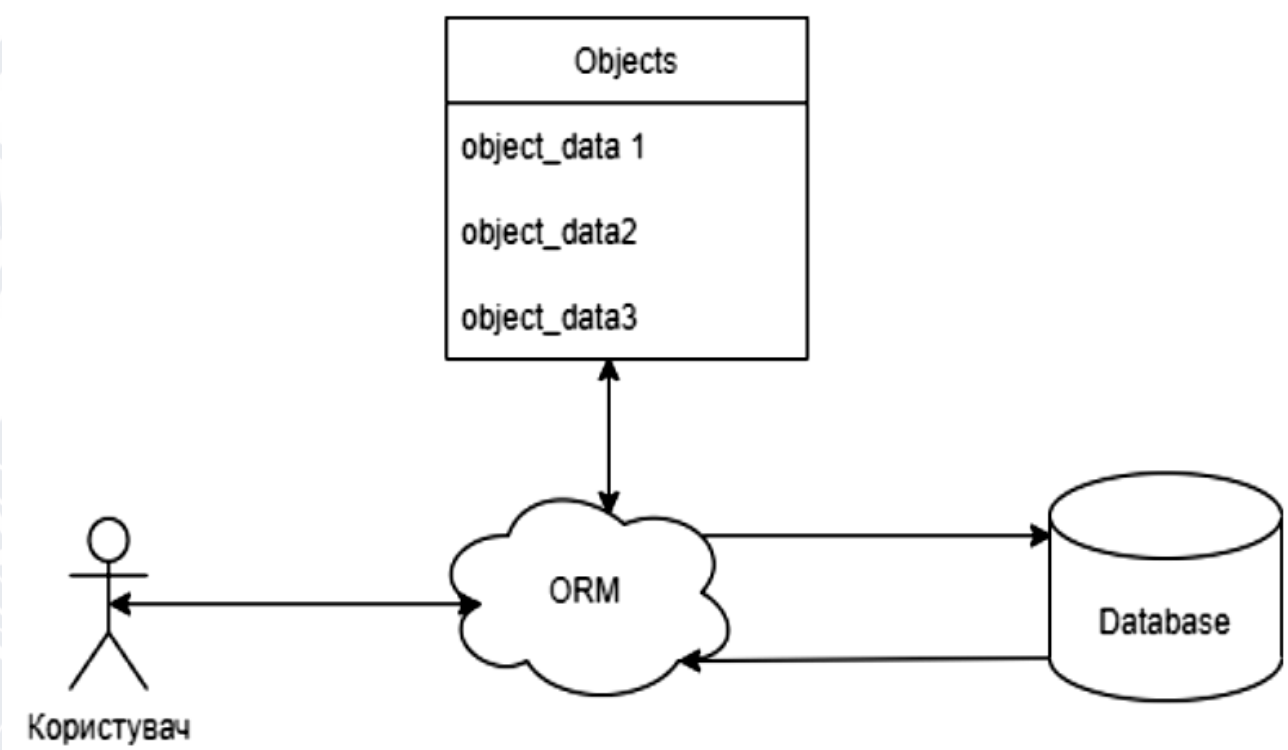

Рисунок 2.13 – Діаграма взаємодії користувача з базою даних через ORM

Система бази даних не пов'язана з програмою, тому розробник може змінювати її в будь-який момент. Вона пов'язана з рештою програми, тому розробник може використовувати його в інших проектах.

Використання технології об'єктно-реляційного картографування економить розробнику значну кількість часу, оскільки:

– Багато деталей робиться автоматично, починаючи від обробки бази даних і закінчуючи взаємодією елементів.
– Розробник створює свою модель даних в одному місці, і йому простіше оновлювати, підтримувати та повторно використовувати код.

– Немає необхідності формулювати складні для виконання SQL-запити.
– Виклик вже готових виписок або транзакцій стає таким же простим, як і виклик звичайних функцій.

Недоліками використання цієї технології є те, що застосування цих засобів вимагає ретельного вивчення їх документації та зниження продуктивності програм при створенні складних запитів за допомогою цих інструментів. Схема взаємодії ORM з базою даних наведена на рис. 2.13.

РОЗДІЛ 3
ПРОЕКТУВАННЯ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ ПАРСИНГУ БІБЛІОГРАФІЧНИХ ДАНИХ

3.1 Модель для роботи з великими масивами даних

Оскільки дані з мереж є неоднорідними за своєю природою та оновлюються з надзвичайною швидкістю, виникає потреба у використанні спеціальних перевірених інструментів, які зарекомендували себе як швидкі та надійні для обробки таких великих масивів даних, однією з таких технологій є використання підходу MapReduce. Слід зазначити, що в даному контексті MapReduce розглядається не як конкретна реалізація, наприклад, Hadoop MapReduce, а як підхід для обробки великих масивів даних. Розглянемо основні принципи такого підходу.

На вхід функції Map необхідно передавати різні типи даних, а саме таблиці даних, різні типи документів, де кожна з цих частин є набором елементарних частинок даних. Об'єднані виходи операції Map і входи операції Reduce в результаті обробки будуть представлені у вигляді пар ключ-значення, таке представлення входів і виходів є спеціальним для обробки за допомогою MapReduce. Наприклад, описана вище функція Map може приймати на вхід певне значення, наприклад, частину даних, раніше оброблених сервісом, яка була вказана, видає на виході, в залежності від ситуації, список пар ключ-значення, або у випадку порожнього набору – порожню пару ключ-значення. Важливим аспектом MapReduce є те, що ключі повинні бути однаковими, це розумний момент, так як без наявності різних ключів операція не може бути виконана, так як групування буде не точним і не буде відповідати реальним даним, самі дані в групі можуть мати дублікати значень. Слід зазначити, що кожен з вузлів в операціях Map та Reduce може бути використаний для потокової передачі вхідних даних, тобто після обробки пакету даних та формування потокового виводу такий вузол може приймати та обробляти необхідні дані [20].

Після такої обробки головний контролер системи ініціює необхідні завдання на виконання, в результаті чого отримуються наступні пари ключ-значення [30]:

$$\ll K1, V1 >, < K2, V2 >, < K1, V3 >, \dots , < KN, VN \gg$$

Трансформовані в кластерні пари форми:

$$< K1, < V1, V3, \dots \gg, \dots , < KN, < VN, \dots \gg$$

Після цього перетворення використані дані повинні бути передані в зчитувач, що обробляє операцію Reduce. На виході операції Reduce, як і при використанні Map, отримуємо пари ключ-значення, причому ключі і значення можуть мати різний тип, в залежності від завдання, але в більшості випадків вони будуть мати однаковий тип. Всі ці пари з усіх вузлів будуть згруповані у файл, який є результатом цієї операції. Зверніть увагу, що для виконання завдання може бути недостатньо виконання однієї команди MapReduce, тому вихідні значення однієї з операцій MapReduce можуть бути вхідними параметрами для інших операцій MapReduce в черзі.

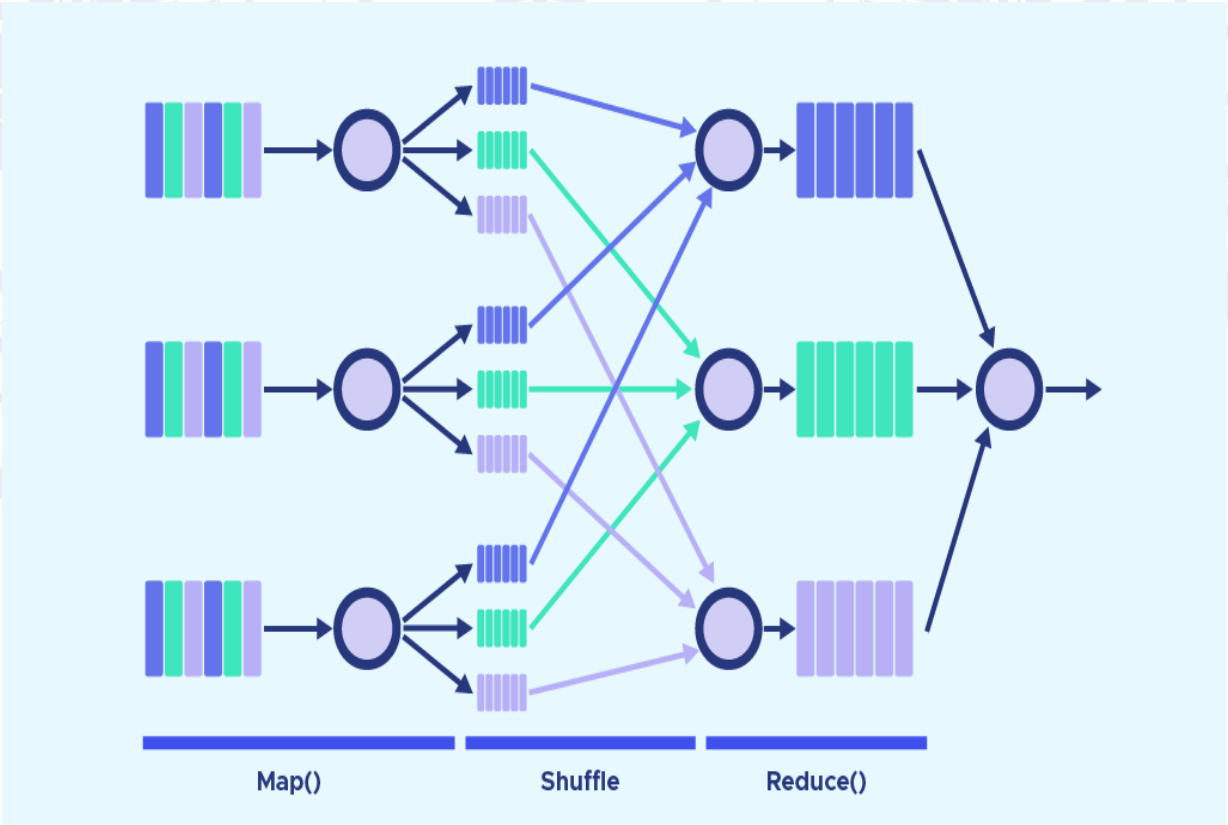

Рисунок 3.1 – Принцип роботи MapReduce

На рисунку 3.1 зображено описаний принцип роботи MapReduce [20]. Хоча основний підхід для обробки великих масивів даних обрано, необхідно обґрунтувати та пояснити основні недоліки порівняно з іншими підходами та можливості, які вони надають.

Іншим рішенням є прискорення роботи реляційної бази даних для зберігання даних, що підлягають обробці, та серверів для виконання обчислень з реляційними даними. У більшості випадків можна розглядати принципи MapReduce як доповнення до СУБД. Перевагу слід надавати СУБД при роботі з одиночними запитами, коли дані видаються з низькою частотою, але зчитуються часто. За таким принципом працює більшість веб-додатків, в яких операції додавання даних відбуваються значно рідше, ніж їх зчитування, наприклад, інтернет-магазин, в якому більшість клієнтів спочатку переглядають каталог товарів, і обравши необхідний, починають відправляти необхідні дані.

Наприклад, у випадку з інтернет-магазином, де більшість покупців спочатку переглядають каталог товарів, а при виборі необхідного товару починають надсилати необхідні дані про бажану покупку на сервер інтернет-магазину. Однак для розрахунків з великими масивами даних такий підхід не є оптимальним, оскільки виникає велика кількість запитів на зберігання цих даних, а саме зчитування відбувається лише для обробки. З іншого боку, бінарні дерева пошуку використовуються для оновлення даних в реляційних базах даних, але було показано, що принципи MapReduce з використанням Sort-Merge є більш ефективним для обробки наборів даних з різних гетерогенних і постійно обновлюваних джерел. Ще однією проблемою використання реляційних баз даних є їх транзакційний характер, що може суттєво сповільнювати обробку великих наборів даних при частому записі цих даних до бази даних. Однак, деякі бази даних, такі як Greenplum, почали використовувати ідеї MapReduce у своїх реалізаціях. Враховуючи зазначені недоліки реляційних баз даних, їх використання в практичній частині даної роботи не рекомендується. Альтернативним підходом для обробки цих даних є використання технології Message Passing Interface (MPI), що базується на грід-

обчисленнях. Такий підхід добре зарекомендував себе при виконанні паралельних обчислень, але має недоліки при доступі до великих обсягів даних з кожного вузла, коли пропускна здатність мережі низька і обчислювальний вузол переходить в стан простою.

В сучасних реалізаціях підходу MapReduce дані знаходяться або на жорсткому диску вузла, або в його оперативній пам'яті, тобто всі реалізації MapReduce враховують, що пропускна здатність мережі є цінним ресурсом для виконання обчислень. Ще одним недоліком MPI є його складність, при використанні реалізацій MapReduce програмісту достатньо реалізувати відповідні методи Map та Reduce, а все інше фреймворк бере на себе. При використанні MPI програмісту доводиться контролювати процеси обробки потоку даних, доступність вузлів, розподіл даних між вузлами, звичайно, такий підхід надає більше свободи під конкретну задачу, але відволікає увагу від написання логіки обробки даних під задачу. Хоча MPI і виграє в швидкості при обробці даних на окремому вузлу, але швидкість читання, складність запису і необхідність розгортання таких вузлів на власному кластері, все ж спонукає використовувати підхід MapReduce.

Основною реалізацією підходу MapReduce є фреймворк Apache Hadoop. Hadoop – це програмна інфраструктура з відкритим вихідним кодом, яка може бути встановлена на кластері машин для спільного зберігання великих обсягів даних та їх подальшої обробки на кластері. Hadoop включає в себе цілу програмну інфраструктуру для виконання таких обчислень з використанням наборів виконання Map та Reduce. Як згадувалося раніше, Hadoop бере на себе всі завдання розподілу, зберігання даних під час обчислень, а все, що потрібно від розробника – це реалізація Map and Reduce.

Основними перевагами Hadoop MapReduce перед попередніми підходами є широкий спектр підтримки мов програмування, тобто Java, C++ або Python, на відміну від MPI, для Hadoop реалізовані спеціальні трекери завдань і балансувальники завдань, які дозволяють перезапустити завдання в разі виникнення помилок, не впливаючи на інші процеси в системі або інші

завдання, а також Hadoop дозволяє розставляти пріоритети завдань і запускати тільки ті завдання, які обробляють основний потік даних першими. Гнучкість Hadoop дозволяє обробляти дані різних типів і структур, а також створювати копії оброблюваних даних на випадок помилок або втрати даних.

Spark – фреймворк, призначений для розподіленої обробки даних з використанням спеціальних функцій і примітивів для обробки в пам'яті, що дозволяє в 100 разів збільшити швидкість пакетної обробки в порівнянні з Hadoop. Це потужний, простий у використанні набір інструментів з відкритим вихідним кодом з API для Java, Scala, Python, R і навіть SQL.

Він може бути використаний для створення додатків, які обробляють дані, як бібліотека або просто як інструмент аналізу даних. Spark включає в себе стек бібліотек, серед яких бібліотеки для роботи з SQL, DataFrames, MLlib, що використовуються для машинного навчання, та Spark Streaming для аналізу даних на льоту. Spark може працювати на Hadoop, на окремому сервері або в хмарному середовищі. Spark має спеціальні модулі для доступу до таких джерел даних, як HDFS, Apache Cassandra, Apache Hbase та S3. Spark широко використовується такими відомими компаніями, як Netflix, Yahoo і Tencent.

В основі Apache Spark лежить концепція абстракції даних, розподіленої колекції об'єктів, яка називається Resilient Distributed Dataset (RDD), що дозволяє писати програми для роботи на цих розподілених наборах даних. Колекції RDD є незмінними, тобто програміст не може змінити елементи колекції, розширити колекцію або видалити елемент такої колекції, і представляють собою дані, які можуть зберігатися в пам'яті або на диску серед кластера машин. Дані розподіляються між машинами в кластері, що дозволяє програмісту обробляти їх паралельно в один і той же час за допомогою низькорівневого API.

Колекції RDD також є стійкими до помилок, і якщо з якихось причин відбувається збій або дані втрачаються, Spark автоматично повертається до копії втрачених даних. З пакетом Spark розробник має спеціальний набір інструментів та алгоритмів машинного навчання в бібліотеці MLlib. MLlib був

розроблений, щоб бути простим, розширюваним і легко інтегруватися з іншими інструментами, що використовуються розробником. Завдяки такій розширюваності, підтримці різних мов та швидкості роботи Spark, розробники можуть вирішувати свої завдання, пов'язані з великими масивами даних, набагато швидше, ніж за допомогою перерахованих аналогів. Основна перевага MLlib полягає в тому, що розробникам і вченим не потрібно зосереджуватися на проблемах, пов'язаних зі складністю розподілу ресурсів на вузлах, визначенням пріоритетів завдань, вирішенням проблем помилок під час роботи, а також розподілом даних і адаптацією алгоритмів машинного навчання до роботи в розподілених середовищах, – все це реалізовано командою Spark. Однак слід зазначити, що Mllib реалізує лише базові алгоритми, придатні для широкого класу задач, але розробник має можливість розширити реалізацію цих алгоритмів під свою конкретну задачу [14].

Пакет Spark включає в себе багато інших цікавих функцій та інструментів, таких як Catalyst Optimizer, Spark Streaming, Dataset, GraphX, ML Pipeline та інші, але вони не будуть розглянуті в цьому документі. Однак навіть при такій кількості переваг пакет Spark має свої недоліки, як вже було сказано, всі операції в Spark використовують оперативну пам'ять, що дозволяє обробляти дані в 100 разів швидше, ніж при використанні Hadoop, але при операціях, які використовують багато пам'яті, це може стати проблемою, ще одним недоліком є те, що Spark не надає можливості зберігати дані і обробляти їх на жорсткому диску як Hadoop, що іноді змушує використовувати Hadoop замість Spark [14].

Серед наведених вище методів обробки великих масивів даних немає однозначного переможця, кожна з технологій має свої переваги та недоліки, пов'язані з часом обробки, споживанням пам'яті, використанням ресурсів та вартістю апаратного забезпечення, яке буде використовуватися, наприклад, використання бази даних SQL є найдешевшою альтернативою, але в той же час це найповільніший спосіб. В даній роботі MPI не буде розглядатися як одна з альтернатив, оскільки цей метод є досить складним у налаштуванні та обробці

даних, а його підтримка хмарними технологіями є нульовою. Два найкращих варіанти – це використання Hadoop та Spark, адже Google Cloud Engine та Azure надають підписку на використання як Hadoop, так і Spark на своїх серверах, що полегшує процес розробки. Однак, оскільки Spark має в своєму пакеті бібліотеку машинного навчання, яка може допомогти в операціях прогнозування та текстового аналізу, вибір стає очевидним, а саме використання Spark як засобу обробки великих масивів даних.

3.2 Вибір архітектури системи для підтримки всіх процесів

Вибір архітектури є важливим етапом у розробці будь-якого додатку. При розробці такого типу додатків очевидним вибором є клієнт-серверна архітектура, яка добре зарекомендувала себе протягом останніх десятиліть. Практична частина цієї роботи не стане винятком і полягатиме в реалізації цього підходу при написанні заявок. Розглянемо основні принципи клієнт-серверної архітектури. Архітектура клієнт-сервер набула популярності завдяки стрімкому розвитку мережі Інтернет та значному впливу і використанню баз даних для зберігання інформації. Цю архітектуру можна визначити як мережеву концепцію, в якій всі ресурси розгорнуті на серверах, які використовуються для задоволення потреб своїх клієнтів, а основними компонентами такої системи можуть бути компоненти з переліку:

1. Сервери, які надають інформацію та інші види даних додатків, що мають до них доступ.

2. Клієнтів, які отримують доступ до відповідних послуг.

3. Мережа, що забезпечує взаємодію між серверами та клієнтами.

Правила взаємодії між клієнтом і сервером називаються протоколами взаємодії між користувачами мережі. Кожен з цих клієнтів може бути як сервером, так і клієнтом такого сервера і має свої обов'язки. Можна виділити основні рівні такого типу: рівень представлення даних, який називається інтерфейсом користувача з різними компонентами для взаємодії користувача з системою. Прикладний рівень, який реалізує основну логіку роботи програми

управління даними, який дозволяє використовувати дані та отримувати до них доступ.

У дворівневій клієнт-серверній архітектурі реалізується взаємодія двох основних компонентів клієнта і сервера. Слід зазначити, що використання та розподіл конкретних функцій між цими клієнтами та серверами дозволяє розрізняти моделі товстих та тонких клієнтів, де товстими клієнтами можуть бути такі пристрої, як кишенькові комп'ютери, мобільні телефони тощо [16].

Трирівнева клієнт-серверна архітектура базується на розділенні прикладного рівня та рівня управління даними. Створюється спеціальний програмний шар, в якому зосереджена логіка роботи додатків, програми середнього рівня працюють під управлінням спеціальних додатків, але запуск цих додатків має здійснюватися спеціальними веб-серверами. Управління даними здійснюється спеціальним сервером даних. [16].

Дворівнева архітектура простіша, всі запити обробляються одним сервером, але в результаті вона менш надійна і вимагає більш високих вимог до продуктивності сервера. Трирівнева архітектура є більш складною, але оскільки функції розподілені між серверами другого та третього рівнів, ця архітектура має [16]:

1. Гнучкість, а також масштабованість.
2. Високий рівень безпеки.
3. Високу продуктивність.

Хоча триланкова архітектура і дозволяє створювати досить гнучкі додатки, але для даного типу додатків вона не підходить з ряду причин, по-перше, обробка тексту повинна відбуватися в два етапи, а саме, вилучення необхідної інформації з даних, отриманих від мережі, а потім його обробка для складання прогнозу та відображення результатів користувачу, причому ці операції потрібно виконувати окремо, і бажано використовувати різні сервери, оскільки у випадку виходу з ладу основного текстового сервера, наприклад, бажано було б використовувати інший сервер. При використанні триланкової архітектури з одним сервером, при виході з ладу сервера або виникненні

помилки при передачі запиту, користувач взагалі не зможе працювати з системою, не кажучи вже про отримання результатів статистики. Звичайно, можна використовувати ще один сервер, повністю скопійований з першого, або навіть кілька, підключених до балансувальника, але при зміні коду будь-якого з модулів системи доведеться міняти весь проект і перекомпілювати його для кожного з цих серверів окремо.

В контексті даного проекту такий результат є неприйнятним, кожна підсистема не повинна залежати від інших, і, наприклад, система прогнозування не повинна залежати від системи обробки тексту, оскільки у випадку виходу з ладу системи обробки тексту або помилки обробки, результат прогнозу може бути отриманий на основі раніше оброблених даних, оскільки динаміка не повинна суттєво змінюватися і при цьому змінювати результати прогнозу.

По-друге, через складність роботи та залежність від засобів обробки даних, які доводиться розгортати на окремому сервері або в хмарному середовищі, залежність сервера від цих модулів може суттєво сповільнювати роботу додатку в цілому, а оскільки основним робочим критерієм є швидкість, використання такого типу архітектури не є доцільним, крім того, якщо всі прикладні модулі розташовані в одному місці, то загальна можливість системи, з точки зору майбутньої модернізації, може бути значно знижена, а додавання нового функціонального модуля може бути значно сповільнено. Такі проблеми виникають при використанні трирівневої архітектури в багатьох великих системах, такі системи ще називають монолітними. Втім, вирішення всіх цих проблем очевидне: можна розділити кожен з модулів системи на менші, слабко пов'язані між собою модулі. Ці рішення забезпечуються використанням архітектури мікросервісів, яка стала популярною кілька років тому.

Офіційного визначення мікросервісів не існує, але суть полягає в тому, що мікросервіси являють собою архітектурний стиль, в якому складні додатки створюються як набір невеликих, легких, самодостатніх, незалежних і слабко пов'язаних сервісів, кожен з яких відповідає за певний процес. Цей стиль

протиставляється монолітному стилю, в якому додатки будуються як єдине ціле. Мікросервіси співпрацюють між собою в залежності від необхідності виконання певної дії. Вони "спілкуються" через API, для яких мова програмування не має значення. Такий підхід схожий на команду розробників, де кожен член зосереджений на виконанні конкретного завдання. Вони мають певну відповідальність, свободу і впевненість, щоб виконувати свою частину роботи найкращим чином. Можна використовувати архітектуру мікросервісів для побудови додатку з нуля і, звичайно, можна розбити монолітний додаток на сервіси. Розглянемо основні переваги такого підходу до розвитку [18]:

1. Швидкі зміни та розгортання. Кожен мікросервіс розгортається окремо, а це означає, що якщо розробник змінює щось в одному з них, він може розгорнути ці зміни, не впливаючи на інші мікросервіси, які можуть продовжувати функціонувати як і раніше. Крім того, зміни можна вносити так часто, як це необхідно для того, щоб певна частина програми відповідала новим потребам бізнесу і не впливала на всю програму, на відміну від монолітної архітектури, описаної вище.

2. Повністю модернізовані програми. Будь-який мікросервіс може бути легко замінений або оновлений. Він може бути переписаний в розумні терміни без необхідності переписування всієї системи. Таку систему легко розширювати і переробляти, і така система завжди може змінити свою "форму" на більш нову і досконалу.

3. Система, яку потенційно легше зрозуміти, підтримувати і тестувати. Мікросервіси, як правило, невеликі за обсягом коду. В результаті команді розробників легше зрозуміти процеси роботи системи, а також таку систему набагато простіше підтримувати і модифікувати, оскільки розробники зосереджуються на конкретних компонентах і впевнені, що зміна їх коду не порушить роботу інших частин програми. Крім того, таку систему набагато легше охопити автоматичними перевірками. Крім того, тестувальникам також необхідно перевіряти невеликі частини системи, а не всю систему, без необхідності розгортання всієї системи на тестових серверах.

4. Помилка в одному мікросервісі не скомпрометує систему. Збої в одному мікросервісі не повинні підривати систему. У більшості випадків помилка в одному сервісі не призведе до поломки системи.

5. Мікросервіси, написані різними мовами програмування, можуть працювати разом. Як і всі члени команди, мікросервіси можуть бути різними, наприклад, в системі можуть бути мікросервіси, написані на мовах Java і .NET, все, що потрібно знати сервісам – це як спілкуватися один з одним, і в основному вони використовують веб-API на основі HTTP-запитів, а також сервісну шину.

Хоча системи мікросервісів мають значні переваги перед монолітними системами, ідеального рішення не існує, додатки мікросервісів мають ряд недоліків, які слід описати. Основними недоліками є складність розробки, оскільки система складається з менших, незалежних компонентів, підтримувати її роботу та обробляти запити набагато складніше, ніж при використанні монолітного підходу. Один з мікросервісів може не відповідати на запити, що може вимагати від розробників написання додаткового коду для обробки такої ситуації. Іншим недоліком є підтримка транзакцій для баз даних, які розподілені між декількома мікросервісами в системі. Хоча розгортання мікросервісів відбувається швидше, проблеми можуть залишатися, наприклад, написання окремої логіки для координації. Підтримувати безпеку для такого типу додатків набагато складніше, оскільки кожен запит повинен бути захищений загальним токеном, або використовувати один і той же механізм безпеки, в монолітній архітектурі підтримка безпеки є набагато простішим процесом, оскільки додатку потрібно перевіряти облікові дані користувача тільки один раз на запит, оскільки всі підсистеми знаходяться в одному місці. В мікросервісній архітектурі додаток не є розподіленим, тому при використанні декількох сервісів для обробки запиту, кожен сервіс повинен передавати маркери того, що є корисним. Хоча мікросервісна архітектура має багато недоліків у порівнянні з монолітним підходом, переваги, які вона надає, є набагато кращими для даної системи, і оскільки хмарні середовища значною

мірою підтримують принципи контейнеризації, розгортання системи з використанням мікросервісної архітектури є гарним варіантом.

3.3 Прототип парсеру для обробки бібліографічних даних

Як зазначалося для реалізації синтаксичного аналізатора була обрана мікросервісна архітектура, що означає, що додаток повинен складатися з невеликих сервісних елементів, кожен з яких виконує певну, покладену на нього задачу. Вирішено, що додаток буде складатися з семи основних частин, кожна з яких виконує свою роботу – дозволяє користувачеві взаємодіяти з іншими сервісами, відповідає за авторизацію тощо. Загальний перелік компонентів системи :

1. Служба інтерфейсу користувача – відповідає за всі види взаємодії користувача з системою, об'єднана практично з усіма системними службами, з якої запускаються всі системні процеси.

2. Служба авторизації та аутентифікації – відповідає за процес створення нового користувача, оновлення профілю та основних даних користувача, генерацію токенів для доступу користувача до інших систем, а також є центральним компонентом для перевірки прав користувача на виконання різного роду дій.

3. Сервіс отримання результатів – відповідає за отримання та аналіз даних з мереж різного типу, оскільки мережі не мають стандартного набору команд та інтерфейсів для взаємодії, даний сервіс впроваджує провайдерів та адаптерів для збору даних для системи.

4. Сервіс обробки вихідних даних – використовується сервісом для отримання вихідних даних для відповідної оперативної обробки даних.

5. Сервіс прогнозування – відповідає за прогнозування майбутніх значень послідовності, яка була отримана та збережена сервісом отримання результатів, для виконання прогнозування використовуються алгоритми, описані в другому розділі.

6. Служба моніторингу та діагностики системи – використовується для фіксації помилок і збоїв в роботі системних служб і програми в цілому.

Функціональні можливості кожного з сервісів та взаємодію сервісів між собою проілюстровано на рисунку 3.2.

Рисунок 3.1 – Схема інтелектуальної системи та основні компоненти

Для реалізації кожного з сервісів використовується ряд фреймворків та інструментів, для реалізації серверної частини використовується фреймворк ASP.NET Core, для компонентів, що мають користувацький інтерфейс вирішено використати бібліотеку React. Для забезпечення взаємодії між кожним сервісом використовується REST API з використанням протоколу HTTP, дані передаються у форматі JSON.

Деякі сервіси мають джерело зберігання даних, а саме бази даних SQL, наприклад, сервіс автентифікації та авторизації, де дані користувачів зберігаються у вигляді таблиць. Докер використовується для об'єднання всіх послуг в контейнерах. Сам додаток розміщується в хмарному середовищі у

вигляді контейнера, з певних причин Google Application Engine був найкращим вибором. Деякі з сервісів використовують для роботи на сторонні API, наприклад, сервіс пошуку результатів, але всі ці сервіси використовують ті ж засоби зв'язку, що і реалізований додаток.

Сервіс взаємодії користувача з системою

Сервіс взаємодії з користувачами є чи не найважливішим сервісом в системі. Даний сервіс містить основні функції передачі та отримання даних від інших сервісів через API та передачі JSON даних на стороні клієнта за допомогою JavaScript фреймворку ReactJS. Основними етапами взаємодії користувача з системою є наступні:

1. Користувач підключається до сайту сервісу.

2. Введення пароля та електронної пошти користувача у відповідну форму для входу в систему.

3. У вікні, що відкрилося, користувач має можливість модифікувати свій профіль та виконувати завдання з прогнозування та аналізу.

Увійшовши у власний обліковий запис, користувач може змінювати свої дані, такі як пароль, e-mail, логін тощо, тобто основні стандартні поля будь-якої веб-системи. Також користувач має можливість робити прогнози, використовуючи дані отримані з попередніх даних аналітичного аналізу, на жаль, результати будуть неточними через брак даних. Під час проведення операцій аналізу та прогнозування користувач буде бачити графік з часовими інтервалами та кількістю збігів за цей період часу.

У загальному вигляді весь процес взаємодії користувача з системою можна представити у вигляді UML-діаграми послідовності, як показано на рис. 3.3. Зауважимо, що на рисунку сервіс інтерфейсу користувача позначено «UIService», сервіс авторизації – «AuthService», сервіс аналізу – «AnalyticsService» та «ForecastingService» відповідає сервісу прогнозування, відповідно.

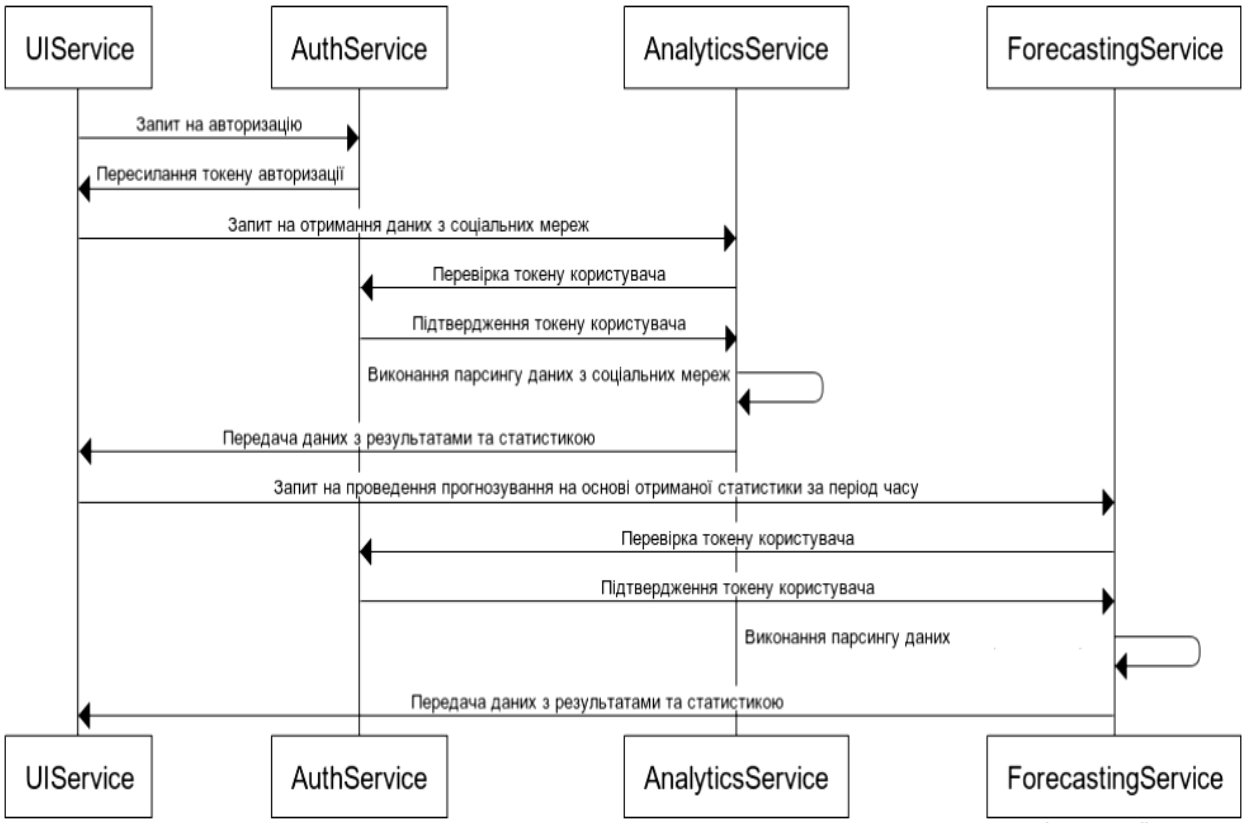

Рисунок 3.3 – Діаграма послідовності UML для інтерфейсу користувача сервісу

Авторизації та автентифікації

Оскільки для даного додатку необхідно зберігати раніше отримані дані, а також дані для додатку, прийнято рішення додати в проект сервіс авторизації та аутентифікації, який зберігає дані користувачів, а також видає ключі для використання інших сервісів. Для підтримки процесу авторизації обрано механізм JSON Web Tokens (JWT). JWT не є новим механізмом, але він дуже добре підходить для мікросервісної архітектури. Для користування прогнозно-аналітичними сервісами користувачу необхідно ввести свій пароль та адресу електронної пошти у відповідній формі. Потім служба інтерфейсу користувача надсилає відповідний POST-запит, щоб згенерувати токен JWT для користувача. За допомогою цього токену користувач авторизується в системі, що дозволяє йому користуватися всіма функціональними можливостями системи. На рисунку 3.4 показано структуру типового токену JWT.

Кожен токен складається з трьох елементів, розділених крапкою, кожен з яких має свої функції при розпізнаванні сервером. Наступний компонент –

основна частина токена, де зберігаються дані користувача, виділена на малюнку фіолетовим кольором. Ця частина може зберігати ролі користувачів в системі, їх пошту, ім'я і т.д. Ця частина токена також форматується як JSON-об'єкт і має вигляд рядка base64, тобто її можна розшифрувати на клієнті при необхідності і отримати необхідні дані користувача.

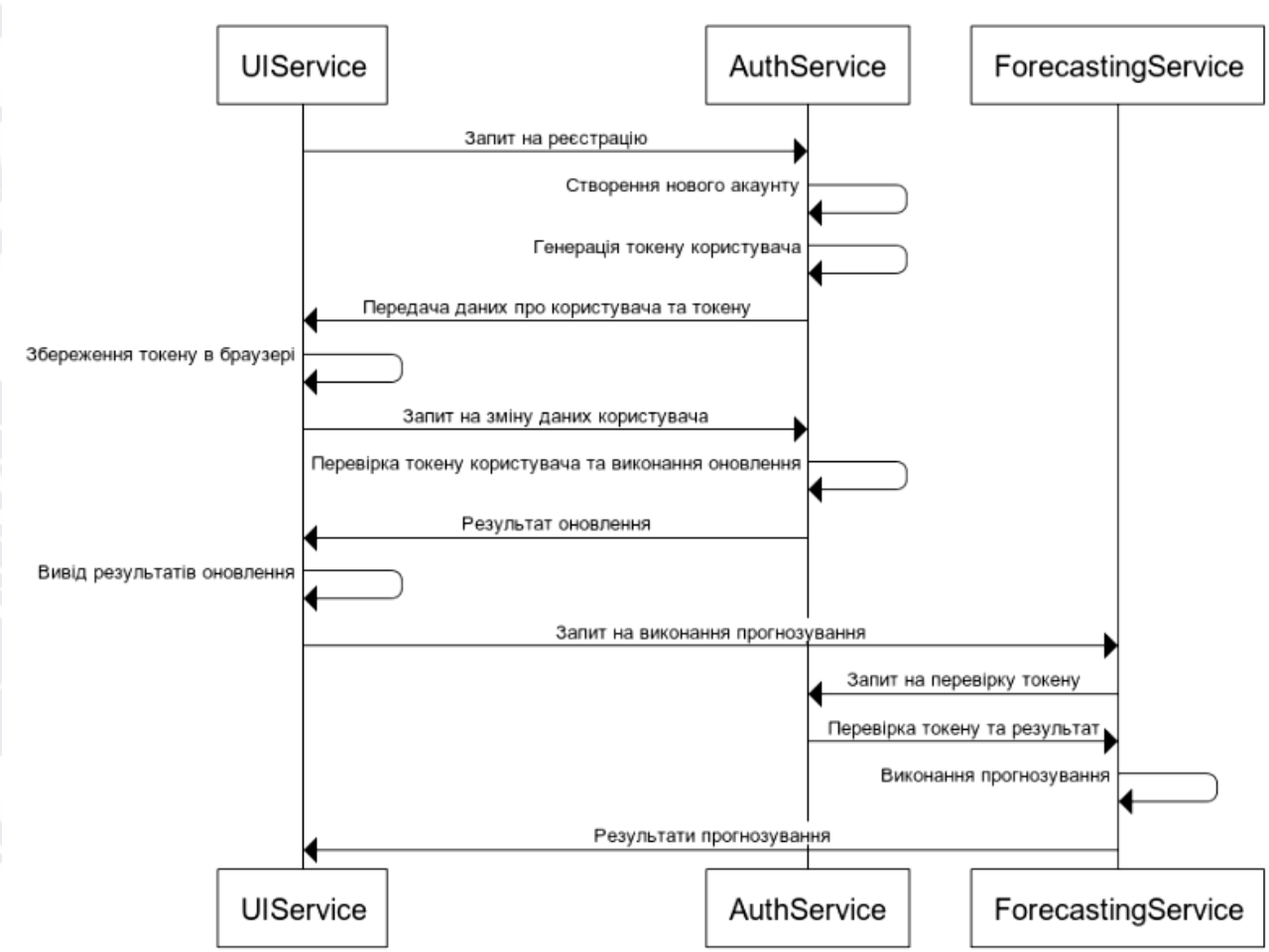

Рисунок 3.4 – Структура типового токену JWT

Найважливіша частина токена виділена синім кольором, ця частина є гарантією того, що сервер отримав дійсний токен і саме з нею працює сервер авторизації для перевірки даних доступу користувача до компонентів системи. Він формується за алгоритмом, зазначеним у заголовку, і розраховується наступним чином:

$$T = Algo(base64(header) + "." + base64(signature), secretKey)$$

Звичайно, як видно з формули, для генерації цієї частини токена сервер повинен мати секретний ключ, за допомогою якого він зможе розшифрувати секретну частину за допомогою алгоритму заголовка токена і перевірити правильність отриманого заголовка і основної частини токена.

Однак це не єдиний функціонал даного сервісу, оскільки він відповідає не тільки за генерацію та верифікацію токену користувача, а й за зберігання та оновлення даних користувача. Користувач має можливість змінити пароль в системі, інші дані, а також видалити свій обліковий запис з системи, оновити основну інформацію про себе, а також переглянути профілі інших користувачів, у разі необхідності, звичайно, якщо користувач має відповідні права, для цього в даному сервісі реалізовані такі кінцеві точки для використання іншими компонентами.

Сервіс прийому та обробки даних є ключовим в системі, оскільки саме в ньому відбувається збір інформації та видача результату відповідно до запиту користувача. Цей сервіс є найважливішим і найскладнішим для реалізації, оскільки мережі не мають єдиного стандарту доступу до даних, а деякі не надають API для роботи з даними для отримання пропозицій роботи за різними фреймами та компонентами.

Сервіс вилучення даних дозволяє користувачеві обирати тип запиту. Слід зазначити, що при обробці даних на стороні сервера використовується обробка відразу в пам'яті, і результат повертається за долі секунди, але для обробки даних потрібен час, тому користувач отримує негайне повідомлення про те, що запит прийнято. Ці дані обробляються службою обробки даних, яка надсилає запит до SAG, де знаходиться Spark, яка обробляє необхідні дані на високій швидкості. Сервіс обробки сам перевіряє результати обробки даних із зазначеним інтервалом на сервері та збирає отримані дані, оскільки Spark не повідомляє про те, що дані були оброблені. Далі отримані дані зберігаються в базі даних у вигляді звіту по конкретному запиту, пов'язаному з користувачем. У подальшому при вході в систему користувач зможе бачити результати обробки даних, що стосуються його запиту.

Для забезпечення такого типу взаємодії між користувачем та відповідними сервісами реалізовано методи API для отримання даних за конкретним статистичним запитом, для отримання всіх статистичних запитів тощо. З метою обробки результатів роботи сервісу вилучення даних реалізовано API сервісу обробки, який приймає необхідні дані та обробляє їх відповідним чином, слід зазначити, що запити до сервісу обробки може робити лише сервіс аналізу.

В цілому весь процес взаємодії інших сервісів з сервісами обробки та аналізу можна представити у вигляді UML діаграми послідовності, де «AuthService» – сервіс авторизації та автентифікації, «UIService» – сервіс інтерфейсу користувача, «AnalyticsSerivce» – сервіс аналізу, «ProcessingService» – сервіс обробки даних.

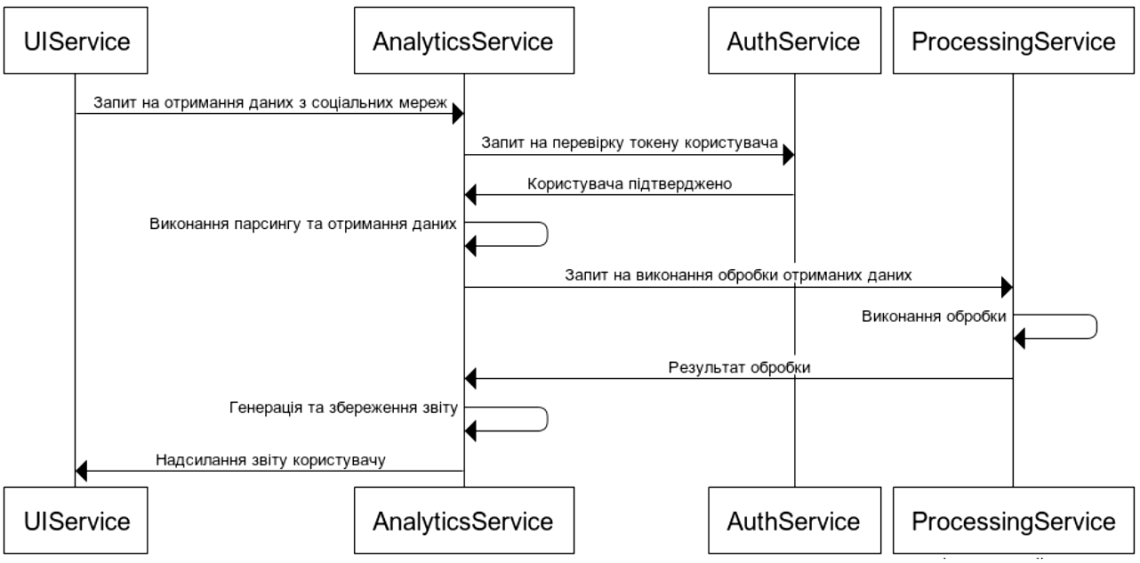

Рисунок 3.5 – Процес взаємодії сервісів аналітики та обробки

Сервіс моніторингу та стану додатку

Послуги статусу та моніторингу програми не використовуються безпосередньо користувачами програми, натомість самі служби надсилають інформацію про помилки та винятки на програмному рівні. Типова помилка програми може виникати через збій, через який певна служба стає

недоступною, або через повільне мережеве з'єднання, яке заважає службі виконувати покладені на неї завдання. Ця послуга корисна для діагностики всієї програми, виявлення основних проблем або нових проблем. Наприклад, припустимо, що один із мережевих API було змінено та оновлено, але старий API не можна використовувати, і в результаті служба мережевих даних не зможе обробляти дані з мережі. Цю ситуацію потрібно вирішити як якнайшвидше, тому дуже важливо мати швидкий спосіб пошуку помилок. Іншим прикладом ситуації, яка може вимагати використання таких служб, є продуктивність часу обробки даних, виміряна певним алгоритмом або службою, тому можна виявити неоптимізовані частини програми та застосувати необхідні засоби для покращення робочого часу та рефакторингу.

По суті, служба виконує запит на реєстрацію свого моніторингу стану за допомогою методу POST. Користувачі з відповідними дозволами можуть використовувати службу інтерфейсу користувача для перегляду журналу стану.

3.4 Практична реалізація парсингу бібліотечної інформації

Після запуску інформаційної системи користувач перенаправляється на веб-сайт.

Рисунок 3.6 – Головна сторінка системи

З самого початку користувач зможе переглядати наступні сторінки:

Статті – це сторінка, на якій користувач може переглянути вже зібрані статті, відсортувати їх, відфільтрувати вибірку (рис. 3.6). Якщо будь-яка зі статей зацікавив користувача, потрібно натиснути на назву статті та вона відкриється у вікні з повною інформацією, також можна перейти за посиланням на оригінальну сторінку, де зберігається елемент.

Наукові напрямки – це сторінка, де користувач може переглядати наукові галузі, з якими можливо працювати та переходити на сторінки окремих науковців. Також можливо скористуватися пошуком, щоб знайти науковців певного напрямку.

Для коректної роботи з розробленою системою парсингу наукометричних даних, необхідно перейти на сторінку «Вхід» (рис. 3.7) та увійти.

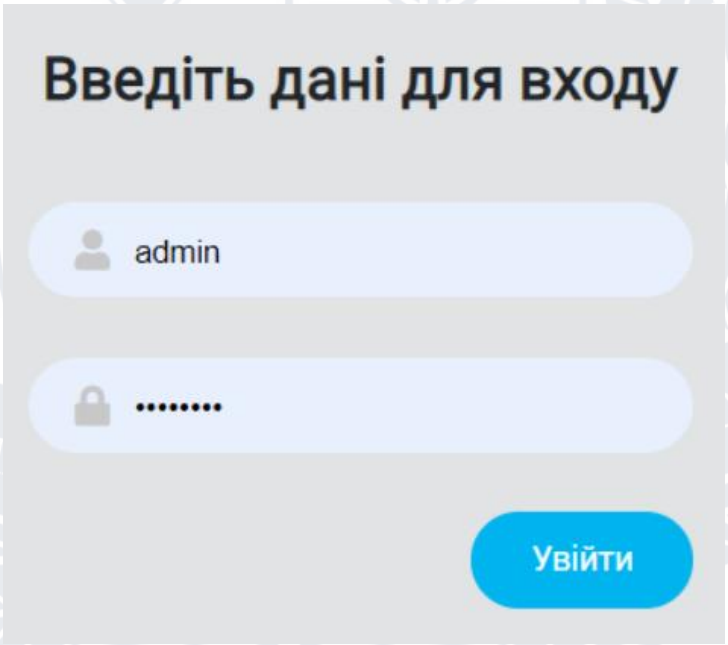

Рисунок 3.7 – Сторінка авторизації

Якщо користувач введе невірні дані – він отримає про це сповіщення. Після цього він знову може повторити спробу підключення до системи. В іншому випадку користувач успішно авторизується. Після цього він повинен перейти до панелі адміністрування (рис. 3.8).

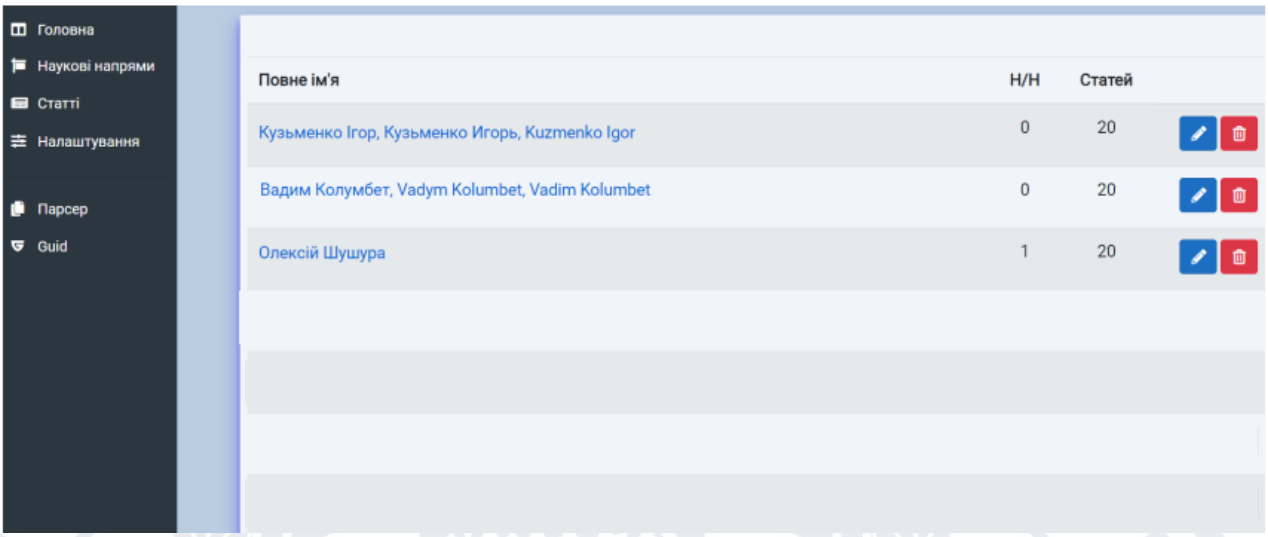

Рисунок 3.8 – Головна сторінка панелі адміністратора

Коли користувач отримує доступ до панелі адміністратора, перед ним є опція змінювати вже додану інформацію, завантажувати статті або змінювати їх, налаштовувати їм наукові напрями. Якщо користувач вперше входить до панелі адміністратора, він може отримати доступ на веб-сторінці Guid. Там описані правила користування панеллю і всі можливі функції взаємодії з ним.

Для прямого доступу до системи, яка збирає наукометричні дані, необхідно натиснути на посилання під назвою «Парсер», який розташований у лівій частині сторінки (рис. 3.9).

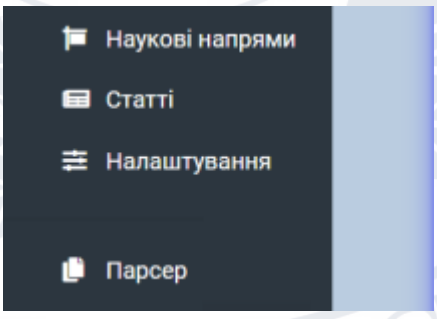

Рисунок 3.9 – Посилання на сторінку аналізатора наукометричних даних

Коли користувач переходить на сторінку парсера, він повинен у полі (рис 3.10) ввести ідентифікатор науковця. Буде зроблено запит до науковометричних баз, і після цього науковця буде додано до основи. На

сторінці, що відкриється, можна додати статті, клацнувши на кожному з них (рис. 3.11). Після натискання на статтю вона завантажується з Інтернет-ресурсу, відображається на екрані у спливаючому вікні. Якщо користувач вважає, що не обов'язково додавати статтю в базу, він може просто видалити її, натиснувши червону кнопку зі значком кошика. Також після додавання елементу, користувач може редагувати поля та зберегти зміни.

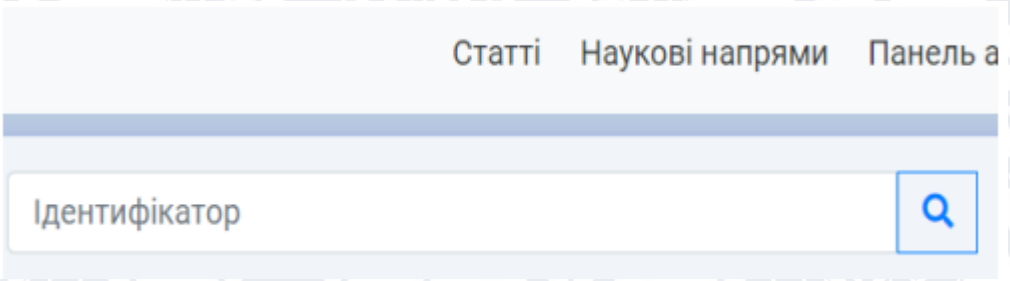

Рисунок 3.10 – Поле ідентифікатора науковця

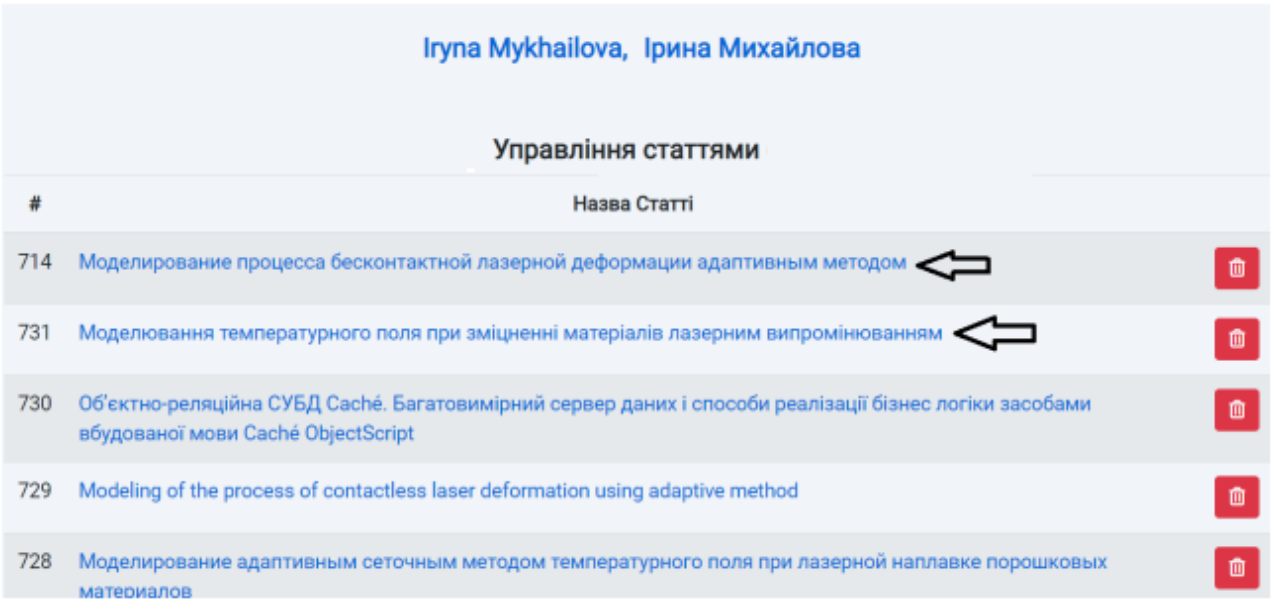

Рисунок 3.11 – Сторінка науковця, завантажена з необхідними елементами

Існує ще один спосіб збору наукометричних даних, для цього необхідно повернутися до сторінки аналізатора. На сторінці. крім поля для введення ідентифікатор, також буде таблиця з науковцями з бази даних. Натиснувши кнопку «Завантажити», відкриється сторінка, де можна знову додати, редагувати та видаляти елементи.

Таким чином користувач повинен використуваати автоматизовану систему збору наукометричних даних. Великою перевагою такого способу додавання науковців є те, що база даних розрахована не тільки на локальних науковців. Даний парсер може додавати будь-яких науковців з усього світу, потрібно лише знати його особистий ідентифікатор.

ЗАГАЛЬНІ ВИСНОВКИ

В даній кваліфікаційній роботі розглянуті основні принципи та характеристики бібліографічної інформації. Показано, що розробка інтелектуальної системи для таких задач є непростим завданням, це пов'язано з даними цих систем, які постійно оновлюються, а їх кількість надзвичайно велика, для реалізації та обробки даних з цих систем необхідно використовувати відповідні інструментальні засоби та архітектуру.

Продемонстровано основні алгоритми прогнозування послідовностей даних, а саме лінійна регресія та її вдосконалення, рекурсивні нейронні мережі, а також метод експоненціального спуску, який хоч і простий в реалізації, але його вдосконалення дають ефективні результати. Хоча кожен з цих методів має свої переваги та недоліки, їх робота продемонстрована на реальному застосуванні та вивчені їх основних принципів.

Для роботи з великими масивами даних обрано фреймворк Spark, який здатен обробляти великі обсяги даних на кластерах комп'ютерів, цей вибір мав найбільше переваг для використання в практичній частині, оминаючи Apache MapReduce, MPI та можливість створення власного аналогу з використанням декількох серверів з базами даних SQL, такий підхід відкинуто через велику кількість часу, який довелося б витратити на розробку та використання баз даних SQL.

Продемонстровано ряд альтернатив використання архітектури веб-додатків, а саме монолітний та мікросервісний, обрано мікросервісний підхід, який хоч і має ряд недоліків, але добре підходить в даному випадку, а можливість поділу системи на невеликі компоненти з подальшим розширенням його складових є чудовою альтернативою в даному випадку.

Продемонстровані основні компоненти розробленої інтелектуальної системи. Розроблений парсинговий додаток складається з сервісів, які мають власний функціонал та відокремлені один від одного. Сервіс прогнозування використовує описані підходи для прогнозування даних послідовності в

майбутньому, а сервіс прийому даних, використовуючи паралельну обробку, отримує та аналізує бібліографічні дані, після чого передає їх до відповідного сервісу обробки, де, в залежності від розміру даних та складності завдання, дані можуть бути оброблені одразу або передані до Spark. Послуга взаємодії з користувачем є чи не найважливішою з точки зору користувача. Вона пов'язує всі сервіси в єдину екосистему, щоб користувач відчував, що він працює з одним додатком, а не з кількома ізольованими підсистемами сервісів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Blei D. Modeling annotated data // Proceedings of the 26th annual international acm sigir conference on research and development in informaion retrieval. – New York, NY, USA: ACM, 2020. – P. 127-134.
2. Brin S. The anatomy of a large-scale hypertextual Web search engine // Computer Networks and ISDN Systems, 2019. – P. 107-117.
3. Cahyono S.C. Comparison of document similarity measurements in scientific writing using Jaro-Winkler Distance method and Paragraph Vector method [Електронний ресурс] / S. C. Cahyono. – 2019. [Електронний ресурс] – Режим доступу до ресурсу: https://www.researchgate.net/publication/337401647_Comparison_of_document_similarity_measurements_in_scientific_writing_using_JaroWinkler_Distance_method_and_Paragraph_Vector_method (дата звернення: 11.09.2022).
4. Cao Y. Real-time traffic information collecting and monitoring system based on the internet of things. // 6th International Conference on Pervasive Computing and Applications. – 2019. P. 45-49.
5. Conceptual Modeling for Data Integration / D. Calvaneseet al. Conceptual Modeling: Foundations and Applications. Lecture Notes in Computer Science, V. 5600. Berlin, 2019. – P. 173-197.
6. Connaway L. Data Mining, Advanced Collection Analysis and Publisher Profiles: An Update on the OCLC Publisher Name Authority File. XXVIII Annual Charleston Conference. [Електронний ресурс] – Режим доступу до ресурсу: http://www.oclc.org/research/presentations/charleston2018.ppt. (дата звернення: 05.10.2022)
7. Connaway L. Radford M. Research Methods in Library and Information Science. 7th ed. Santa Barbara, 2021. – 478 p.
8. Connaway L., Dickey. T. Publisher Names in Bibliographic Data: An Experimental Authority File and a Prototype Application, Library Resources and Technical Services. OCLC. [Електронний ресурс] – Режим доступу до

ресурсу: https://www.oclc.org/content/dam/research/publications/connaway-lrts.pdf. (дата звернення: 05.09.2022)

9. DBLP [Електронний ресурс] – Режим доступу до ресурсу: https://dblp.org/ – (дата звернення: 15.09.2022).
10. Frost R. Parser Combinators for Ambiguous Left-Recursive Grammars. // 10th International Symposium on Practical Aspects of Declarative Languages (PADL), ACM-SIGPLAN, Volume 4902/2018, Pages:, January 2018, San Francisco. – P. 167-181.
11. Grune D. Parsing Techniques – A Practical Guide / D. Grune, C. Jacobs. – Chichester: Originally published by Ellis Horwood, 2020. – 320 p.
12. Gundecha U. Selenium WebDriver 3 Practical Guide: End-to-end automation testing for web and mobile browsers with Selenium WebDriver / U. Gundecha, S. Avasarala. – 2020. – 280 p.
13. Hahn J. Publisher References in Bibliographic Entity Descriptions. Association for Information Science and Technology, №58. 2021. – P.461-465.
14. Heydt M. Python Web Scraping Cookbook / Michael Heydt. – 2018. – 364 p.
15. Hriaziuk V.O. Use of statistical methods for the analysis of international activity // Матеріали XIX Міжнародної науково-практичної конференції молодих вчених і студентів «Сучасні проблеми наукового забезпечення енергетики» – м. Київ.: «КПІ ім. Ігоря Сікорського», 20-23 квітня 2021 р. – С. 249-250.
16. Jihyun K. Exposing Standardization and Consistency Issues in Repository Metadata Requirements for Data Deposition. College & Research Libraries Journal, V. 80: №6. 2019. PP. 843-875.
17. Kogalovsky M.R. Refinement of the Synthesis language specification by the ODMG semantics. INTAS-94-1817 Project Report, IPI RAS, 2018.
18. Natural Language Toolkit (NLTK) [Електронний ресурс] – Режим доступу до ресурсу: https://www.nltk.org/book/ch00.html. (дата звернення: 20.09.2022).
19. Petrenko M. Detection and research of system anomalies in the ISBN field of bibliographic databases and methods of minimizing their negative impact. «Danish Scientific Journal» (DSJ) Kobenhavn. Denmark, 2021. V. 53, P. 78- 80.

20. Rahma A. Text steganography based on unicode of characters in multilingual. International Journal of Engineering Research and Applications. 2019. – P. 1153-1165.
21. Ringlstetter C. The same is not the same-postcorrection of alphabet confusion errors in mixed-alphabet OCR recognition. Eighth International Conference on Document Analysis and Recognition (ICDAR'05). 2020. P. 406–410.
22. Schwartz B. High Performance MySQL: Optimization, Backups, and Replication / B. Schwartz, P. Zaitsev, V. Tkachenko. – 2019. – 826 p.
23. Si Alhir S. Learning UML / Sinan Si Alhir. – 2018. – 304 p.
24. Smith-Yoshimura K. Transitioning to the Next Generation of Metadata. OCLC Research. 2020. – 54 p.
25. Transforming Metadata into Linked Data to Improve Digital Collection Discoverability / G. Bahnemann et al. OCLC Research. 2021. – 74 p.
26. Zhou S. Text categorization based on topic model // International journal of computational intelligence systems. – 2021. – Vol. 2, no. 4. – P. 398–409.
27. Ziegler P. Data Integration – Problems, Approaches and Perspectives. Conceptual Modelling in Information Systems Engineering. Berlin, 2017. P. 39-58.
28. Zuo Y. Word network topic model: A simple but general solution for short and imbalanced texts // Knowledge and information systems. – 2020. – Vol. 48, no. 2. – P. 379-398.
29. Аналіз тональності тексту: веб-сайт. [Електронний ресурс] – Режим доступу до ресурсу:https://uk.wikipedia.org/wiki/ Аналіз_тональності_тексту (дата звернення: 18.09.2022).
30. Берко А.Ю. Методи інтеграції синтаксису різнорідних даних у системах електронного контент-бізнесу. Інформаційні системи та мережі: Вісник Націон. ун-ту «Львівська політехніка». 2018. Т. 621. С. 19–28.
31. Близнюк Б.О. Современные методы обработки естественного языка. Вісник Харківського національного університету імені В.Н. Каразіна. – 2019. Вип. 36. – С. 14-26.

32. Висоцька В.А. Особливості моделювання синтаксису речення слов'янських мов за допомогою породжуваних контекстновільних граматик. Вісник Національного університету «Львівська політехніка». Інформаційні системи та мережі. – 2021. – С. 246-276.
33. Говорущенко Т.О. Сучасні проблеми семантичного аналізу специфікацій вимог до програмного забезпечення. Вчені записки ТНУ імені В.І. Вернандського. Серія: Технічні науки. [Електронний ресурс] – Режим доступу до ресурсу: http://tech.vernadskyjournals.in.ua/journals/2019 (дата звернення: 08.10.2022)
34. Знаменский С.В. Модель и аксиомы метрик сходства [Електронний ресурс] – Режим доступу до ресурсу: https://www.researchgate.net/publication/322552424_Model_i_aksiomy_metrik_shodstva (дата звернення: 19.09.2022).
35. Зоріна Н. Авторитетний файл предметних заголовків як засіб організації тематичного пошуку в електронному каталозі. Вісник Книжкової палати. 2019. № 9 – С. 45-52.
36. Костенко П.П. Веб-сервіс уточнення релевантності веб-документів пошукової видачі Google на основі поведінки користувача // Інженерні та освітні технології. Щоквартальний науково-практичний журнал – Кременчук: КрНУ, 2019. – Випуск 4(8). С. 49-62.
37. Митчелл Р. Скрапинг веб-сайтов с помощью Python / Райан Митчелл., 2019. – 280 с.
38. Небезпеки парсингу. Чим може обернутися парсинг даних із сайту. [Електронний ресурс] – Режим доступу до ресурсу: https://weblana.com/a403615-parsing-sajtov-plyusy.html (дата звернення: 01.10.2022)
39. Парсинг html-сайтів для споживача. Парсинг даних для використання підприємством. [Електронний ресурс] – Режим доступу до ресурсу: parsing.valemak.com/ru/what-whyhow/stages-of-parsing (дата звернення: 01.10.2022).

40. Парсинг. Парсери. Використання парсерів. Алгоритми роботи парсерів. Основи роботи парсерів. Етичні і технічні складнощі роботи з парсерами. [Електронний ресурс] – Режим доступу до ресурсу: https://ipipe.ua/info/parsing. (дата звернення: 16.09.2022)
41. Парсинг. Різновиди парсингу. Етапи парсингу. [Електронний ресурс] – Режим доступу до ресурсу: https://www.seonews.ua/glossary/parsing. (дата звернення: 15.09.2022)
42. Петренко М.В. Дослідження особливостей поля «ISBN» та їх впливу на процес обробки бібліографічних даних. Вчені записки ТНУ імені В.І. Вернадського. Серія: Технічні науки, Київ, 2021. №32(4) – С. 130–136.
43. Петренко М.В. Проблема доступності інформації про асортимент публічних бібліотек. XX міжнародна науково-практична конференція «Інклюзивне освітнє середовище: проблеми, перспективи та кращі практики», Україна, Київ, 18-19 листопада 2020. Ч. 3. С. 62-64.
44. Про інформацію: Закон України від 02.10.1992. №48. 2021. – С. 20-50.
45. Семантика. Парсинг сайту. Етапи парсингу даних. [Електронний ресурс] – Режим доступу до ресурсу: semantica.in/blog/chto-takoe-parsing. (дата звернення: 30.09.2022).
46. Синтаксичний аналіз. Мови програмування. Парсери. Програми-парсери. [Електронний ресурс] – Режим доступу до ресурсу: http://xn--r1a3b.xn--b1amgblet.xn--j1amh/index.ph. (дата звернення: 25.09.2022).
47. Таланчук П.М. Метод виявлення подібності назв видавництв, стійкий до різних видів скорочень. // Інфокомунікаційні та комп'ютерні технології, Університет «Україна», Київ, 2021. №2, С.34-42.
48. Терещенко В.В. Перспективи вдосконалення інформаційного пошуку. [Електронний ресурс] – Режим доступу до ресурсу: http://molodyvcheny.in.ua/files/journal/2017/4.4/25.pdf. (дата звернення: 30.09.2022)
49. Щербина О.С. Впровадження та перспективи використання програм-парсерів у закладах вищої освіти. Бібліотекознавство. Документознавство.

Інформологія: наук. журн. / М-во культури та інформ. політики України, Нац. акад. керів. кадрів культури і мистецтв. Київ: НАКККіМ, 2021. №2. С. 88-95.

50. Що таке парсинг і коли його використовувати? [Електронний ресурс] – Режим доступу до ресурсу: https://alexsmokinof.lviv.ua (дата звернення: 05.09.2022).

ДОДАТОК

__init__.py

```
__all__ = ['Client', 'utils']

from .client import Client
from . import utils
```

```
__all__ = ['Client']
import suds as _suds
import functools as _functools
from base64 import b64encode as _b64encode
from collections import OrderedDict as _OrderedDict
from sys import version_info as _version_info
from limit import limit as _limit

class Client():
    """Query the Web of Science.
       You must provide user and password only to user premium WWS service.
    def __init__(self, user=None, password=None, SID=None, close_on_exit=True,
                 lite=False, proxy=None, timeout=600, throttle=(2, 1)):
        """Create the SOAP clients. user and password for premium access."""

        self._SID = SID
        self._lite = lite
        self._close_on_exit = close_on_exit
        proxy = {'http': proxy} if proxy else None
        options = {'proxy': proxy, 'timeout': timeout}
        search_wsdl = self.searchlite_url if lite else self.search_url
        self._auth = _suds.client.Client(self.auth_url, **options)
        self._search = _suds.client.Client(search_wsdl, **options)
        self._throttle_wait = _limit(*throttle)(lambda: True)
        if user and password:
            auth = '%s:%s' % (user, password)
            auth = _b64encode(auth.encode('utf-8')).decode('utf-8')
            headers = {'Authorization': ('Basic %s' % auth).strip()}
            self._auth.set_options(headers=headers)
    def __enter__(self):
        self.connect()
        return self

    def __exit__(self, exc_type, exc_value, traceback):
        if self._close_on_exit:
            self.close()

    def __del__(self):
```

```
    if self._close_on_exit:
        self.close()

def is_lite(self):
    """Returns True if the client is for lite"""
    return self._lite

def _api(fn):
    limitation (calls per second)."""
    @_functools.wraps(fn)
    def _fn(self, *args, **kwargs):
        self._throttle_wait()
        if not self._SID:
            raise RuntimeError('Session closed. Invoke connect() before.')
        resp = fn(self, *args, **kwargs)
        return (self._search.last_received().str() if self.is_lite()
                else resp)
    return _fn

def _premium(fn):
    @_functools.wraps(fn)
    def _fn(self, *args, **kwargs):
        if self.is_lite():
            raise RuntimeError('Premium API not available in lite access.')
        return fn(self, *args, **kwargs)
    return _fn
@staticmethod
def make_retrieveParameters(offset=1, count=100, name='RS', sort='D'):
    :name: Name of the field to order by. Use a two-character abbreviation
           to specify the field ('AU': Author, 'CF': Conference Title,
           'CG': Page, 'CW': Source, 'CV': Volume, 'LC': Local Times Cited,
           'LD': Load Date, 'PG': Page, 'PY': Publication Year, 'RS':
           Relevance, 'SO': Source, 'TC': Times Cited, 'VL': Volume)
    :sort: Must be A (ascending) or D (descending). The sort parameter can
           only be D for Relevance and TimesCited.
    return _OrderedDict([
        ('firstRecord', offset),
        ('count', count),
        ('sortField', _OrderedDict([('name', name), ('sort', sort)]))
    ])
def connect(self):
    """Authenticate to WOS and set the SID cookie."""
    if not self._SID:
        self._SID = self._auth.service.authenticate()
        print(('Authenticated (SID: %s)' % self._SID).encode('utf-8'))
    self._search.set_options(headers={'Cookie': 'SID="%s"' % self._SID})
    self._auth.options.headers.update({'Cookie': 'SID="%s"' % self._SID})
    return self._SID
```

```
    def close(self):
        if self._SID:
            self._auth.service.closeSession()
            self._SID = None
    @_api
    def search(self, query, count=5, offset=1, editions=None,
               symbolicTimeSpan=None, timeSpan=None, retrieveParameters=None):

        :query: User query for requesting data. The query parser will return
                errors for invalid queries
        :count: Number of records to display in the result. Cannot be less than
                0 and cannot be greater than 100. If count is 0 then only the
                summary information will be returned.
        :offset: First record in results to return. Must be greater than zero
        :editions: List of editions to be searched. If None, user permissions
                   will be substituted.
                   Fields:
                   collection - Name of the collection
                   edition - Name of the edition
                       Valid values:
                       '1week' - Specifies to use the end date as today and
                       the begin date as 1 week prior to today.
                       '2week' - Specifies to use the end date as today and
                       the begin date as 2 week prior to today.
                       '4week' - Specifies to use the end date as today and
                       the begin date as 4 week prior to today.
                   Fields:
                   begin - Beginning date for this search. Format: YYYY-MM-DD
                   end - Ending date for this search. Format: YYYY-MM-DD
        query = query.decode('utf-8') if _version_info[0] < 3 else query
        return self._search.service.search(
            queryParameters=_OrderedDict([
                ('databaseId', 'WOS'),
                ('userQuery', query),
                ('editions', editions),
                ('symbolicTimeSpan', symbolicTimeSpan),
                ('timeSpan', timeSpan),
                ('queryLanguage', 'en')
            ]),
            retrieveParameters=(retrieveParameters or
                                self.make_retrieveParameters(offset, count))
        )

    @_api
    def retrieve(self, queryId, count=100, offset=1, retrieveParameters=None):
        return self._search.service.retrieve(
            queryId=queryId,
            retrieveParameters=(retrieveParameters or
```

```
                    self.make_retrieveParameters(offset, count))
        )

    @_api
    def retrieveById(self, uid, count=100, offset=1, retrieveParameters=None):
        :uid: Web of Science unique record identifier
        return self._search.service.retrieveById(
            databaseId='WOS',
            uid=uid,
            queryLanguage='en',
            retrieveParameters=(retrieveParameters or
                    self.make_retrieveParameters(offset, count))
        )

    @_api
    @_premium
    def citedReferences(self, uid, count=100, offset=1,
                        retrieveParameters=None):
        :uid: Web of Science unique record identifier
        return self._search.service.citedReferences(
            databaseId='',
            uid=uid,
            queryLanguage='en',
            retrieveParameters=(retrieveParameters or
                    self.make_retrieveParameters(offset, count))
        )

    @_api
    @_premium
    def citedReferencesRetrieve(self, queryId, count=100, offset=1,
                    retrieveParameters=None):
        return self._search.service.citedReferencesRetrieve(
            queryId=queryId,
            retrieveParameters=(retrieveParameters or
                    self.make_retrieveParameters(offset, count))
        )
    def citingArticles(self, uid, count=100, offset=1, editions=None,
                       timeSpan=None, retrieveParameters=None):
            Fields:
            collection - Name of the collection
            edition - Name of the edition
            Fields:
            begin - Beginning date for this search. Format: YYYY-MM-DD
            end - Ending date for this search. Format: YYYY-MM-DD
        return self._search.service.citingArticles(
            databaseId='',
            uid=uid,
            editions=editions,
```

```
        timeSpan=timeSpan,
        queryLanguage='en',
        retrieveParameters=(retrieveParameters or
                            self.make_retrieveParameters(offset, count))
    )

@_api
@_premium
def relatedRecords(self, uid, count=100, offset=1, editions=None,
                   timeSpan=None, retrieveParameters=None):
    :uid: A unique item identifier. It cannot be None or empty string.
                   Fields:
                   collection - Name of the collection
                   edition - Name of the edition
                   Fields:
                   begin - Beginning date for this search. Format: YYYY-MM-DD
                   end - Ending date for this search. Format: YYYY-MM-DD
    :retrieveParameters: Retrieve parameters. If omitted the result of
                         make_retrieveParameters(offset, count, 'RS', 'D')
                         is used.
    return self._search.service.relatedRecords(
        databaseId='WOS',
        uid=uid,
        editions=editions,
        timeSpan=timeSpan,
        queryLanguage='en',
        retrieveParameters=(retrieveParameters or
                            self.make_retrieveParameters(offset, count))
    )
```

Триконенко Сергій Володимирович
Прізвище, ім'я по батькові

Факультет інформаційних та прикладних технологій
Факультет

122 Комп'ютерні науки
Шифр і назва спеціальності

Комп'ютерні технології обробки даних
Освітня програма

ДЕКЛАРАЦІЯ АКАДЕМІЧНОЇ ДОБРОЧЕСНОСТІ

Усвідомлюючи свою відповідальність за надання неправдивої інформації, стверджую, що подана кваліфікаційна (магістерська) робота на тему «РОЗРОБКА І ДОСЛІДЖЕННЯ МЕТОДІВ СИНТАКСИЧНОГО АНАЛІЗУ БІБЛІОГРАФІЧНИХ ДАНИХ» є написаною мною особисто.

Одночасно заявляю, що ця робота:

- не передавалась іншим особам і подається до захисту вперше;
- не порушує авторських та суміжних прав, закріплених статтями 21-25 Закону України «Про авторське право та суміжні права»;
- не отримувалась іншими особами, а також дані та інформація не отримувалась у недозволений спосіб.

Я усвідомлюю, що у разі порушення цього порядку моя кваліфікаційна робота буде відхилена без права її захисту, або під час захисту за неї буде поставлена оцінка «незадовільно».

_____ _____

(дата) (підпис здобувача освіти)