Міністерство освіти і науки України
Криворізький національний університет
Кафедра моделювання та програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти магістра

за спеціальністю 121 – Інженерія програмного забезпечення

На тему: Розробка програмного модуля семантичного інформаційного пошуку

Засвідчую, що в цій кваліфікаційній роботі немає запозичень із праць інших авторів без відповідних посилань.

Студент гр. ІПЗ–23-2м_____/ І. В. Данейкін /

Керівник кваліфікаційної роботи _____/ Н. Х. Саітгарєєв /

Економіко-організаційна частина _____/ О. В. Шамрай /

Нормоконтроль _____/ Н. Х. Саітгарєєв /

Завідувач кафедрою _____/ А. М. Стрюк /

Кривий Ріг
2024

Криворізький національний університет

Факультет: Інформаційних технологій

Кафедра: Моделювання та програмного забезпечення

Ступінь вищої освіти: магістр

Спеціальність: 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. кафедри

_____ А. М. Стрюк

«____» _____ 2024 р.

ЗАВДАННЯ

на кваліфікаційну роботу

студента групи ІПЗ–23-2м Данейкіна Іллі Валерійовича

1. Тема: Розробка програмного модуля семантичного інформаційного пошуку затверджена наказом по КНУ № 278с від «15» квітня 2024 р.

2. Термін подання студентом закінченої роботи: «28» листопада 2024р.

3. Вихідні дані по роботі: розробка системи семантичного інформаційного пошуку у файлах.

4. Зміст пояснювальної записки (перелік питань, що їх треба розробити): виконати аналіз існуючих програм-аналогів на ринку, визначити функціонал розроблюваного додатку, спроектувати розроблюваний додаток, розробити програмне забезпечення, здійснити тестування розробленого додатку.

5. Перелік ілюстративного матеріалу: функціональна схема, блок–схема розробленого алгоритму, скріншоти роботи програми.

Календарний план:

№	Найменування етапів кваліфікаційної роботи	Термін виконання
1	Аналіз літературних джерел та огляд інтернет-ресурсів з заданої тематики	09.01.24 – 14.02.24
2	Пошук існуючих методів вирішення проблеми	15.02.24 – 10.03.24
3	Формулювання актуальності роботи і постановка завдань	11.03.24 – 29.03.24
4	Оформлення матеріалів першого розділу роботи	30.03.24 – 15.04.24
5	Визначення об’єкту, предмету та мети дослідження	16.04.24 – 02.05.24
6	Оформлення матеріалів другого розділу роботи	03.05.24 – 12.05.24
7	Розробка математичного та інформаційного забезпечення системи	13.05.24 – 06.06.24
8	Оформлення матеріалів третього розділу роботи	07.06.24 – 25.06.24
9	Розробка функціональної схеми та алгоритму програми	26.06.24 – 11.07.24
10	Розробка програмного забезпечення системи	12.07.24 – 10.09.24
11	Тестування програмного забезпечення	11.09.24 – 04.10.24
12	Оформлення матеріалів четвертого розділу роботи	05.10.24 – 16.10.24
13	Аналіз економічної ефективності інновації	17.10.24 – 02.11.24
14	Оформлення матеріалів п’ятого розділу роботи	03.11.24 – 14.11.24
15	Остаточне оформлення пояснювальної записки	15.11.24 – 27.11.24

Дата видачі завдання: «05» січня 2024р.

Студент: _____ / І. В. Данейкін /

Керівник роботи: _____ / Н. Х. Саітгарєєв /

РЕФЕРАТ

ЛЕМАТИЗАЦІЯ, СТЕМІНГ, ТОКЕНІЗАЦІЯ, МОРФОЛОГІЧНИЙ АНАЛІЗ, АЛГОРИТМИ СЕМАНТИЧНОГО ПОШУКУ, ІНДЕКСУВАННЯ ДАНИХ, КОРЕНЕВІ ФОРМИ СЛІВ, РЕЛЕВАНТНІСТЬ РЕЗУЛЬТАТІВ, ПРИХОВАНІ МАРКІВСЬКІ МОДЕЛІ.

Пояснювальна записка: 82 с., 18 рис., 5 табл., 2 дод., 32 джерел.

Мета кваліфікаційної роботи: розробка програмного модуля семантичного інформаційного пошуку.

Об’єкт дослідження: процес семантичного інформаційного пошуку на базі системи, що включає компоненти для індексування, аналізу та пошуку інформації на основі семантичних зв'язків і контексту запитів користувачів.

У теоретичній частині роботи виконано аналіз існуючих на сьогодні аналогічних програмних рішень на ринку програмного забезпечення. Зазначені сильні та слабкі сторони існуючих програмних продуктів. Обґрунтовані актуальність роботи, мета та сформульовані завдання для розробки програмного модуля семантичного інформаційного пошуку. Розглянуті можливі методи вирішення проблеми, розроблено математичне та інформаційне забезпечення створюваної системи.

У практичній частині кваліфікаційної роботи реалізовано функціональну схему розроблюваного додатку та алгоритм його роботи. Розроблено інтерфейс програмного модуля семантичного інформаційного пошуку та програмну логіку її роботи. Проведено тестування розробленого програмного забезпечення.

Розроблений програмний модуль семантичного інформаційного пошуку може використовуватися широким колом користувачів для пошуку інформації в текстових даних.

ABSTRACT

LEMMATIZATION, STEMMING, TOKENIZATION, MORPHOLOGICAL ANALYSIS, SEMANTIC SEARCH ALGORITHMS, DATA INDEXING, WORD ROOT FORMS, RELEVANCE OF RESULTS, HIDDEN MARKOV MODELS.

Explanatory note: 82 p., 18 fig., 5 tabl., 2 app., 32 references.

The aim of the qualifying work: development of the program module for semantic information retrieval.

Object of research: the system-based semantic information retrieval process that includes components for indexing, analyzing, and searching for information based on semantic relationships and the context of user queries.

In the theoretical part of the work, the analysis of existing similar software solutions in the software market has performed. The strengths and weaknesses of existing software products have indicated. The relevance of the work, the purpose and objectives for the development of the program module for semantic information retrieval have formulated. Possible methods of solving the problem have considered, mathematical and information support of the created system have developed.

In the practical part of the qualification work, the functional scheme of the developed application and the algorithm of its work have implemented. The interface of the program module for semantic information retrieval and the program logic of the application have developed. The developed software has tested.

The developed software module of semantic information retrieval can be used by a wide range of users to search for information in text data.

ЗМІСТ

ВСТУП

У XXI сторіччі кількість доступної інформації постійно збільшується з кожним днем, демонструючи експоненційне зростання. Це відбувається завдяки розвитку технологій зв'язку, появі нових цифрових платформ і збільшенню швидкості обміну даними. Люди постійно стикаються з необхідністю знаходити потрібну інформацію серед величезних масивів даних в Інтернеті. Однак традиційні методи пошуку, які ґрунтуються на ключових словах та індексуванні, часто не справляються з цим завданням. Вони можуть бути не настільки точними та гнучкими, особливо коли інформація стає дедалі різноманітнішою та швидко змінюється.

У таких умовах стає важливо використовувати більш розумні системи пошуку, які здатні розуміти сенс інформації та враховувати контекст запитів користувачів. Семантичний інформаційний пошук являє собою напрямок, який допомагає поліпшити якість пошуку, роблячи його більш точним і релевантним. Такі системи аналізують зміст інформації глибше, розуміють наміри користувача та встановлюють зв'язки між різними поняттями. Це не тільки підвищує точність пошуку, а й робить обробку даних ефективнішою, що особливо важливо, коли ресурси обмежені і потрібно швидко реагувати на запити.

Розробка програмного модуля для семантичного інформаційного пошуку - це достатньо актуальне та важливе завдання, спрямоване на створення інструментів, здатних ефективно працювати з великими обсягами даних і різноманітними користувацькими запитами. Такий модуль має об'єднувати сучасні технології обробки природної мови, машинного навчання та моделювання онтологій. Це дасть змогу системі краще розуміти й інтерпретувати інформацію. У межах цієї роботи розглядаються основні аспекти створення такого програмного забезпечення, включно з архітектурою системи, алгоритмами семантичного аналізу та методами оцінювання

ефективності пошуку. Мета дослідження - створити функціональний і масштабований модуль, який зможе адаптуватися до змін в інформаційному середовищі та задовольняти потреби сучасних користувачів у швидкому доступі до інформації.

Таким чином, ця робота спрямована на подолання обмежень традиційних систем пошуку шляхом впровадження семантичних технологій. Це відкриває нові можливості для опрацювання та використання інформації в різних сферах знань і практичної діяльності, роблячи доступ до потрібної інформації простішим та ефективнішим.

1 АНАЛІЗ МЕТОДОЛОГІЇ СЕМАНТИЧНОГО ПОШУКУ ТА ЙОГО ЗАСТОСУВАННЯ В КОНТЕКСТІ РОБОТИ З ДАНИМИ

1.1 Короткі теоретичні відомості стосовно методології семантичного пошуку

Одним з актуальних на сьогоднішній день напрямків роботи з інформацією є семантичний пошук - це «пошук із сенсом». Це значення може стосуватися різних частин процесу пошуку: розуміння запиту (замість простого пошуку збігів його компонентів у даних), розуміння даних (замість простого пошуку таких збігів) або представлення знань у вигляді способу, придатного для значущого пошуку.

Семантика - це вивчення значення. Таким чином, у двох словах можна сказати, що семантичний пошук - це пошук із сенсом.

Більш зручно осягнути це, поглянувши на протилежний приклад. Лише десяток-півтора років тому пошукові системи, включно з великими пошуковими системами в Інтернеті, все ще були здебільшого лексичними. Під лексичним ми тут маємо на увазі, що пошукова система шукає буквальні збіги слів запиту, введених користувачем, або їхніх варіантів, не намагаючись зрозуміти, що насправді означає весь запит.

Розглянемо запит «Криворізький національний університет», надісланий пошуковій системі. Очевидно, що веб-сайт Криворізького національно університету [1] добре відповідає цьому запиту. Щоб визначити цю сторінку як збіг, пошуковій системі не потрібно розуміти, що насправді означають слова запиту «університет», «національний» і «криворізький» або що вони означають разом. Фактично, веб-сайт університету містить ці три слова в назві.

Усі ці критерії легко перевірити, і самі по собі вони роблять цю сторінку кандидатом на перше місце за цим пошуковим запитом. Жодного глибшого

розуміння того, що насправді означав запит або про що насправді йдеться на веб-сайті, не потрібно під час пошуку.

Сучасні пошукові системи дедалі більше й більше йдуть у напрямі прийняття ширшого спектра запитів, фактично намагаючись їх «зрозуміти» та надаючи найвідповіднішу відповідь у найвідповіднішій формі, а не просто перелік збігаючихся за текстом документів.

Наприклад, розглянемо два запити «вчені-комп'ютерники» і «жінки-комп'ютерники, що працюють над семантичним пошуком». Перший запит короткий і простий, другий довший і складніший. Обидва є наочними прикладами того, що називається семантичним пошуком. Наступне обговорення не залежить від точної форми цих запитів. Їх можна сформулювати як запити за ключовими словами, як зазначено вище. Також їх можна сформулювати у вигляді повних запитів природною мовою. Або їх можна сформулювати абстрактною мовою запитів. Головне полягає у тому, що саме запитують зазначені запити.

Для людини мета обох цих запитів цілком зрозуміла: користувач, скоріше за все, шукає вчених певного типу. Ймовірно, було б непогано скласти їхній список із деякою базовою інформацією про кожного (наприклад, зображенням і посиланням на їхній профіль у соцмережі чи на сайті університету). Для запиту щодо вчених, дослідження яких тісно пов’язані з інформаційними технологіями, Вікіпедія надає сторінку з відповідним списком і відповідними словами запиту.

Для другого запиту (жінки-комп'ютерники, що працюють над семантичним пошуком) не існує жодної веб-сторінки або іншого документа з відповідним списком, не кажучи вже про документ, який відповідає словам запиту. З огляду на специфіку запиту, також малоймовірно, що хтось коли-небудь буде вручну складати такий список у якому б то не було форматі і підтримувати його. Треба звернути увагу, що обидва списки постійно змінюються з часом, оскільки в будь-який момент до них можуть приєднатися нові дослідники.

Фактично, навіть окремі веб-сторінки, що відповідають запиту, навряд чи міститимуть більшу частину слів запиту. Вчений-комп'ютерник зазвичай не розміщує слова «вчений-комп'ютерник» на своїй домашній сторінці. Жінка-комп'ютерник навряд чи розмістить слово «жінка» на своїй домашній сторінці. На домашній сторінці, ймовірно, є розділ, присвячений дослідницьким інтересам конкретного вченого, але цей розділ не обов'язково містить слово «робота» (можливо, там є схоже слово, а можливо, такого слова взагалі немає, а просто список тем). Семантичний пошук теми, ймовірно, буде вказано на відповідній веб-сторінці, хоча, можливо, в іншому формулюванні, наприклад, інтелектуальний пошук або пошук знань.

Таким чином, обидва запити є хорошими прикладами того, як пошук повинен виходити за рамки простого лексичного зіставлення слів запиту, щоб надати користувачеві задовільний результат. Крім того, обидва запити (зокрема, другий) вимагають об'єднання інформації з кількох різних джерел для задовільної відповіді на запит. Ці джерела інформації можуть бути різних видів: (неструктурований) текст, а також (структуровані) бази знань.

Точного визначення того, що таке семантичний пошук, не існує. Насправді семантичний пошук для різних людей означає багато різних речей. Дослідники з різних спільнот працюють над безліччю проблем, пов'язаних із семантичним пошуком, часто не знаючи про відповідну роботу в інших спільнотах.

1.2 Види даних та види семантичного пошуку

Семантичний пошук може провадитися за текстом (природною мовою) або із використанням баз знань (які складаються відповідно зі структурованих записів). Ці два типи даних можуть бути об'єднані. Наприклад, текст природною мовою може бути збагачений семантичною розміткою, яка ідентифікує згадки сутностей із бази знань. Або можна об'єднати кілька баз знань із різними схемами, як у семантичній мережі [2].

Таблиця 1.1 - Класифікація досліджень семантичного пошуку за базовими даними та за парадигмою пошуку

	Пошук за ключовими словами	**Структурований пошук**	**Природна мова (NLP)**
Текст	Виконання пошуку за ключовими словами в тексті	Структуроване вилучення даних з тексту	Відповіді на питання в тексті
Бази знань	Виконання пошуку за ключовими словами в базах знань	Структурований пошук в базах знань	Відповіді на питання в базах знань
Комбіновані дані	Виконання пошуку за ключовими словами в комбінованих даних	Напівструктурований пошук в комбінованих даних	Відповіді на питання в комбінованих даних

В той же час можливий семантичний пошук за зображеннями, аудіо, відео та іншими об'єктами, що мають за своєю суттю нетекстове представлення [3,4]. Для таких видів даних семантичний пошук може бути доволі доречним. Наприклад, припустимо, що користувач шукає фотографію певної людини. Майже напевно користувача не цікавить точне розташування пікселів, які використовуються для представлення зображення. Її може навіть цікавити не конкретний ракурс, вибір або умови освітлення знімка, а тільки продемонстрований об'єкт. Існує деякий збіг із пошуком за текстовими

даними, включно зі спробами зіставити нетекстові функції з текстовими та використанням тексту, що супроводжує нетекстовий об'єкт (наприклад, підпис до зображення). Але здебільшого пошук за нетекстовими даними - це інший напрямок, що вимагає зовсім інших методів та інструментів [5].

Особливим випадком зображень і аудіоданих є скани текстових документів і мови. Базові дані також є текстовими і можуть бути витягнуті з використанням методів оптичного розпізнавання символів (OCR) і автоматичного розпізнавання мови (ASR). Варто зазначити, що «семантичні методи», описані в цьому огляді, можуть бути корисними в процесі розпізнавання тексту. Наприклад, як в OCR, так і в ASR семантичне розуміння можливих текстових інтерпретацій може допомогти вирішити, яка інтерпретація є найбільш підходящою.

1.3 Потенційні напрямки використання семантичного пошуку

Розглянемо деякі можливі варіанти використання апарату семантичного пошуку та потенційні напрямки для створення відповідного програмного забезпечення (рисунок 1.1).

Семантичний пошук у текстових документах може застосовуватися у різних сферах, де необхідний ефективний пошук інформації у файлах відповідного формату. Це можуть бути корпоративні інформаційні системи, державні установи та архіви, юридичні чи медичні компанії, журналістські чи медіа-агенства. Застосування семантичного пошуку дозволить швидко знаходити потрібні документи або інформацію в них - наприклад, можна знайти всі документи, що стосуються певного проекту, навіть якщо конкретні ключові слова не згадуються безпосередньо. Також впровадження зазначеного напрямку дозволить аналізувати документи для вилучення ключових фраз, сутностей (імена осіб, організації, дати) і автоматично сортувати їх за релевантністю. Реалізація описаного алгоритму може базуватися на побудові індекса документів, використанні методів обробки природної мови для

вилучення семантичної інформації та реалізації алгоритмів семантичного пошуку, які враховують контекст і синонімію.

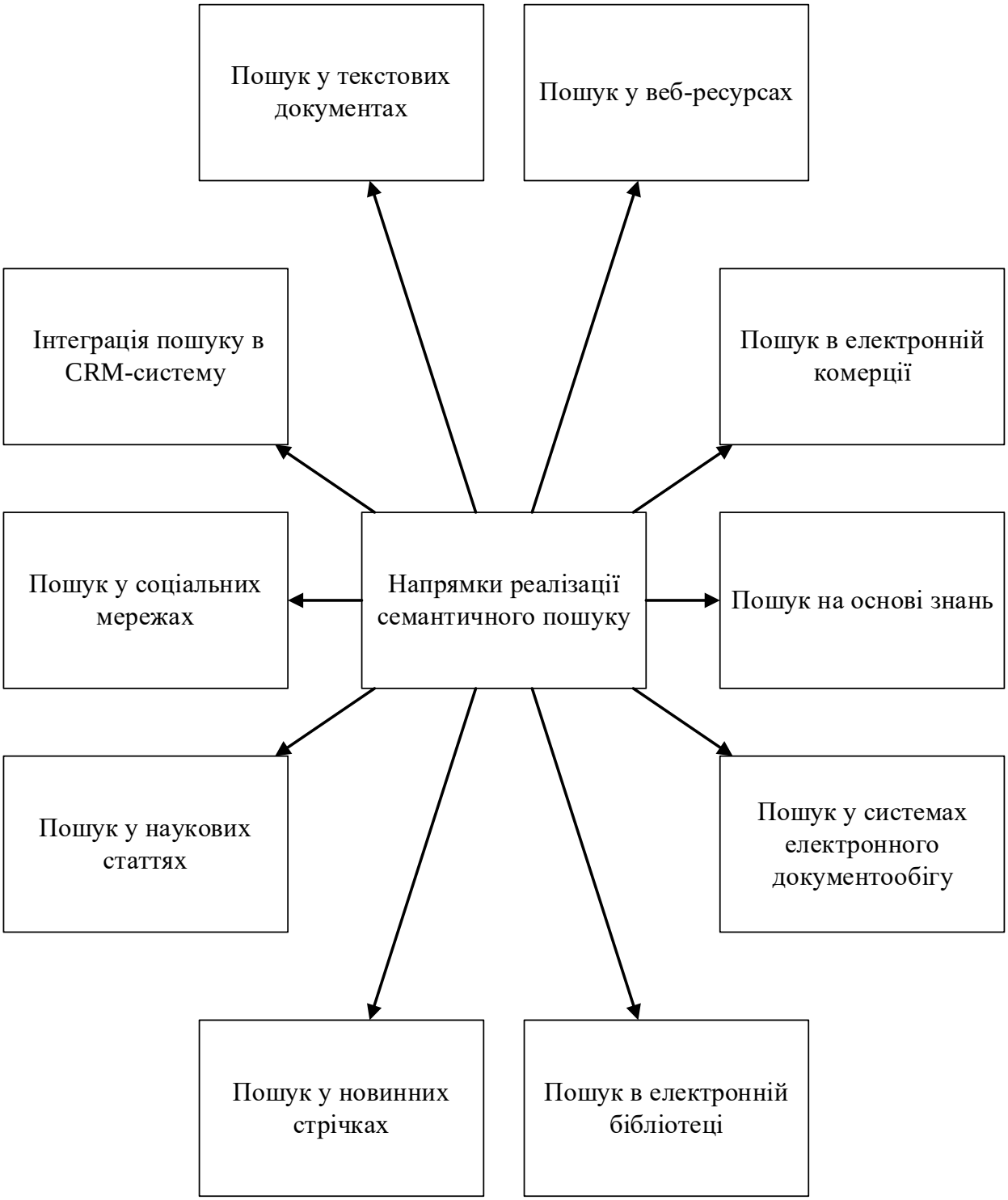

Рисунок 1.1 – Потенційні напрямки застосування семантичного пошуку та реалізації програмного забезпечення

Створення програмного модуля для семантичного пошуку інформації на веб-сторінках може бути використана для аналізу і досліджень ринку, моніторингу брендів та репутаційного менеджменту власної компанії, вирішення інших питань, які є підкласами даної задачі та описані далі в матеріалі поточного розділу. В цілому описана система може забезпечити автоматичний збір даних з веб-ресурсів про продукти, ціни, новини ринку; аналіз конкурентів на основі семантичного аналізу контенту, що дозволяє визначити стратегії конкурентів, основні характеристики продуктів і тенденції на ринку; а також формулювання рекомендацій для бізнесу - щодо виходу на ринок, зміни стратегії або впровадження нових продуктів. Функціонувати зазначений модель може на базі використання веб-скрапінгу для збирання даних та застосування моделей машинного навчання для аналізу контенту веб-сторінок [6].

Використання семантичного пошуку в електронній комерції [7] може стати важливим інструментом для покращення якості пошуку та рекомендацій в інтернет-магазині. Він може забезпечити кращий користувацький досвід під час пошуку товарів, а також суттєво підвищити конверсію продажів завдяки видачі більш релевантних результатів і персоналізованих торгівельних рекомендацій. Під час введення пошукових запитів при застосуванні семантичного підходу сайт може пропонувати варіанти користувачеві автодоповнення, враховуючи популярні запити та синоніми. Також застосування описаного методу пошуку може допомогти аналізувати пошукові запити користувачів та надавати аналітику для оптимізації пошуку в інтернет-магазині, забезпечувати пошук товарів на основі зображень та текстового опису, реалізувати каталог товарів з семантичними атрибутами для ефективнішої класифікації та індексації товарів в магазині, пропонувати покупцеві товари, які є семантично подібними до тих, які він вже переглядав або купував нещодавно.

Інтеграція модуля семантичного пошуку в CRM-систему утворює потенціал для покращення обслуговування клієнтів, підвищення ефективності

роботи менеджерів, а також для автоматизації процесів обробки клієнтських запитів. Варто зазначити, що CRM-системи часто використовують для зберігання великих обсягів текстових даних, таких як електронні листи, чати з клієнтами, замітки після дзвінків тощо. Семантичний пошук дозволяє аналізувати вхідні електронні листи та чати з клієнтами, виділяючи ключові запити, скарги, питання або потреби; виділяти важливі сутності, такі як імена, дати, продукти або проблеми, з текстових даних для полегшення їх обробки та використання в подальшій роботі; аналізувати текст з урахуванням контексту, щоб краще розуміти зміст запиту клієнта, навіть якщо він сформульований неочевидним чином. Також зазначений модуль може забезпечити пошук релевантної інформації для менеджерів з обслуговування клієнтів, автоматичний аналіз і поділ на категорії запитів клієнтів, аналіз історії взаємодій з клієнтами.

Застосування семантичного пошуку у соціальних мережах може стати гарним інструментом для опрацювання великого обсягу інформації, що генерується користувачами. Воно допомагає ефективно відстежувати настрої, виявляти ключові теми та тренди, що обговорюються в соціальних мережах, а також аналізувати громадську думку. Дані при цьому можуть автоматично збиратися з різних соціальних мереж через офіційні API [8].

Програма для семантичного пошуку на основі знань може допомогти у пошуку та інтерпретації інформації з великих і складних баз знань. Вона може бути використана там, де необхідний доступ до структурованої інформації. Це можуть бути такі напрямки, як академічні дослідження і наукові бібліотеки, бізнес-аналітика і управлінські інформаційні системи, навчальні платформи, консалтинг, інформаційні системи в охороні здоров'я.

Реалізація семантичного пошуку у наукових статтях може створити ефективний інструмент для дослідників, що допоможе швидко знаходити релевантну інформацію в наукових джерелах та економити час на аналіз великої кількості літератури. Такого роду програмне забезпечення може значно спростити процес пошуку наукових матеріалів, підвищити точність

результатів і забезпечити глибший аналіз наукових зв'язків між статтями. Серед можливих сценаріїв використання подібного продукту можна виділити індексацію наукових статей з різних джерел (збирання наукових статей, інтеграцію з власними дослідженнями); аналіз наукових текстів за допомогою методів обробки природної мови, що включає аналіз ключових слів і фраз, алгоритми розпізнавання сутностей та семантичну сегментацію; пошук статей за темою, автором та ключовими словами з урахуванням семантичних зв'язків; візуалізацію результатів пошуку і графів взаємозв'язків між науковими статтями; персоналізацію пошуку і рекомендацій.

Ще одним дещо схожим напрямком можна вважати семантичний пошук в документах електронної бібліотеки [9]. Це може бути дотичним до наукових пошуків, в тім може стосуватися і бібліотек, наприклад, з художньою літературою. Тут варто реалізувати автоматичну індексацію бібліотечних фондів, автоматичне оновлення індексу при додаванні нових матеріалів до них, аналіз запитів користувача з урахуванням контексту, пошук з урахуванням синонімів та за тематичними зв'язками. Окремою корисною опцією може бути спрощення роботи з утворенням бібліографічного списку та його приведення до чинних стандартів, а також генерація цитувань та пропонування рекомендацій на основі попередніх запитів.

При роботі з великими обсягами документів у різних форматах доцільно використовувати семантичного пошуку, особливо якщо мова йде про системи електронного документообігу. Це спрощує використання запитів, сформульованих природною мовою, що містять складні поняття та контекст, та підвищує результативність їх застосування. В системі може бути реалізована підтримка різних форматів документів, індексація архівів, оновлення індексу при додаванні або зміні документів, що забезпечує актуальність результатів пошуку. Також тут можуть використовуватися технології обробки природної мови для аналізу тексту документів і вилучення з них потрібної семантичної інформації (витяг ключових фраз і термінів, розпізнавання сутностей тощо). Можлива реалізація семантичного пошуку з

підтримкою розширених запитів, що забезпечує пошук за складними запитами та з урахуванням синонімів та семантичних зв'язків, ранжування результатів за релевантністю та фільтрацію. Використання такого програмного забезпечення також дозволить реалізувати моніторинг змін у документах та автоматизувати їх категоризацію.

Може знайти своє застосування означена технологія пошуку і під час роботи з новинними ресурсами. Вона з великою вірогідністю продемонструє хороші результати при аналізі великого обсягу новинних статей, що постійно оновлюються. Апарат семантичного пошуку може використовуватися журналістами, аналітиками, маркетологами, а також широкою аудиторією звичайних користувачів, які бажають швидко знаходити потрібні їм новини. При цьому вони можуть відстежувати популярні теми та тренди, а також фільтрувати новини за потрібними критеріями. Система може забезпечувати автоматичну індексацію новинних статей з різних онлайн джерел - новинних порталів, блогів, соціальних мереж, оброблюючи різні формати даних; реалізовувати семантичний пошук з можливістю фільтрації за категоріями і датою. Доцільно реалізувати можливість виявлення настроїв та їх аналіз - визначати тональність новинних статей (позитивна, негативна або нейтральна), їх зміну по регіонах та графіки цих змін протягом певного періоду часу.

1.4 Існуючі програмні засоби для реалізації семантичного інформаційного пошуку

1.4.1 Elasticsearch

Наразі існує суттєва кількість програмних засобів, що використовують принципи семантичного інформаційного пошуку. Одним з них є ElasticSearch.

ElasticSearch [10] - це розподілений пошуковий і аналітичний рушій з відкритим вихідним кодом, що написаний на мові Java, та який підтримує велику кількість типів даних, включно з текстовими, числовими, геопросторовими, структурованими і неструктурованими.

ElasticSearch, по суті, являє собою документоорієнтовану базу даних, оптимізована під всілякі операції пошуку. Розуміючи це і маючи базове уявлення про концепції документоорієнтованого підходу до зберігання даних, використання цього інструменту стає більш простим та наочним для користувача.

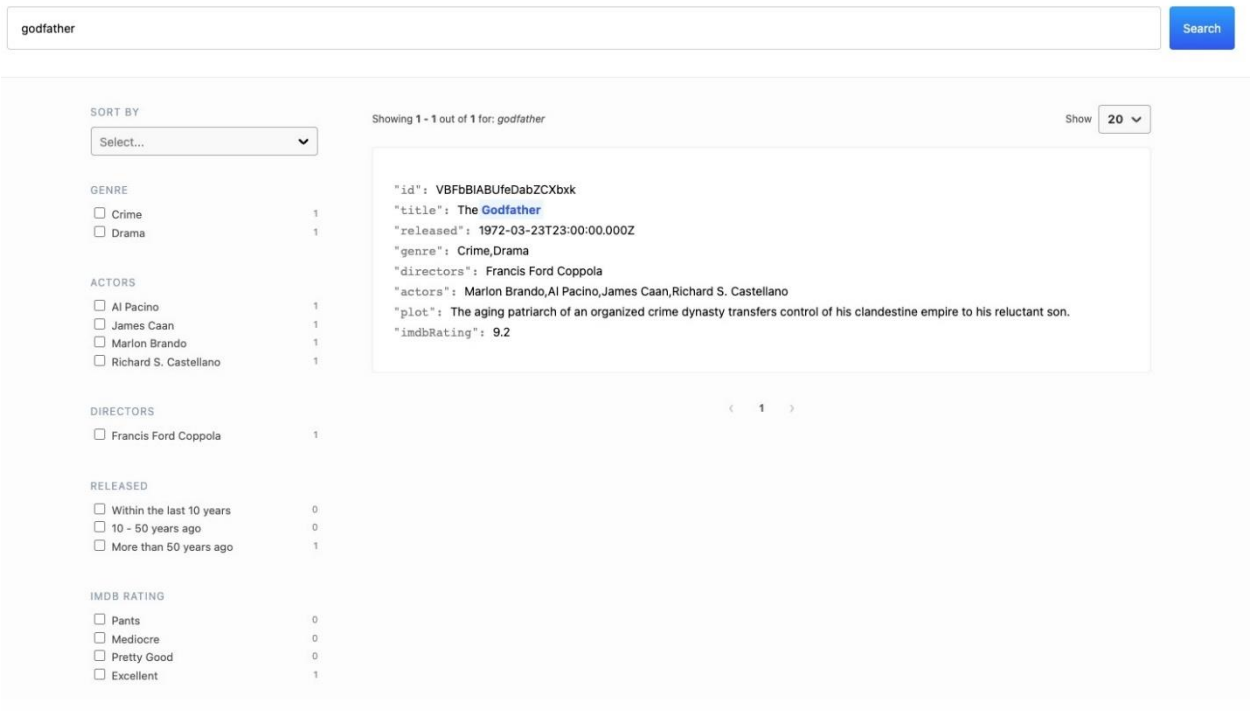

Рисунок 1.2 – Пошуковий UI ElasticSearch [11]

За роки успішного існування на ринку ElasticSearch став центральним елементом екосистеми ELK Stack. ELK - це акронім трьох продуктів компанії Elastic: ElasticSearch (пошуковий і аналітичний движок), Logstash (конвеєр обробки даних) і Kibana (інтерфейс для візуалізації даних). Сьогодні можна виділити два найпопулярніші сценарії використання ElasticSearch:

а) движок для повнотекстового пошуку;

б) сховище логів і метрик в ELK Stack.

Будь-який пошуковий запит в ElasticSearch перед безпосереднім виконанням потрапляє в аналізатор. Він являє собою конвеєр, який складається з декількох частин: character filter, tokenizer і token filter.

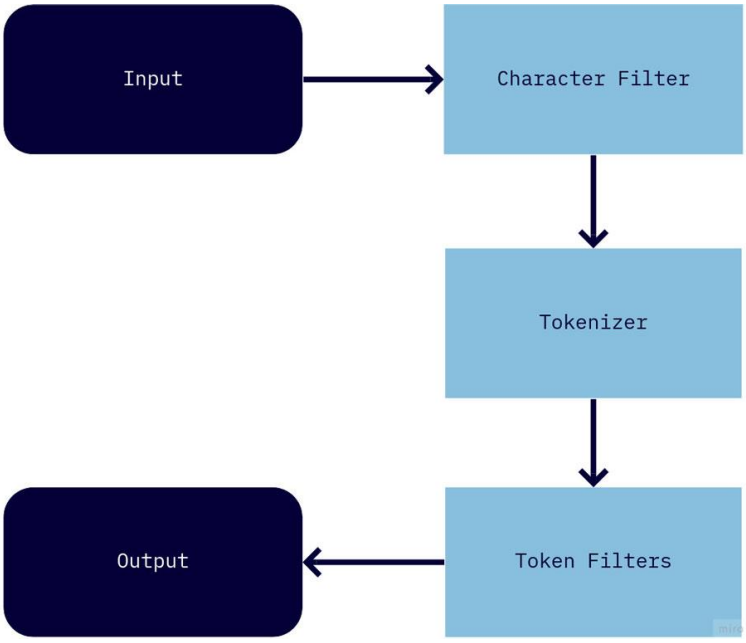

Рисунок 1.3– Аналізатор ElasticSearch [12]

ElasticSearch надає набір вбудованих аналізаторів для базових потреб, однак, найімовірніше, користувачеві доведеться написати власний аналізатор для вирішення своїх задач [12].

Для підвищення зручності розробки, перегляду та налагодження пошукових запитів і їхніх результатів використовується Kibana [13].

Kibana представляє собою засіб реалізації візуального інтерфейсу, який дає змогу проводити дослідження та відображати дані журналу, що були зібрані у відповідних кластерах Elasticsearch [14].

Kibana дозволяє надавати форму даним і переміщатися по Elastic Stack. З Kibana користувач може:

а) шукати, спостерігати та захищати свої дані та використовувати функціонал від пошуку документів до аналізу журналів і пошуку вразливостей безпеки;

б) аналізувати свої дані, шукати приховані інсайти, візуалізувати знайдене у вигляді діаграм, індикаторів, карт, графіків тощо та об'єднувати їх на інформаційній панелі;

в) керувати, відстежувати та захищати Elastic Stack. При цьому забезпечується контроль того, які користувачі мають доступ до тих чи інших функцій;

г) використовувати домашню сторінку аналітики для більш наочної візуалізації даних.

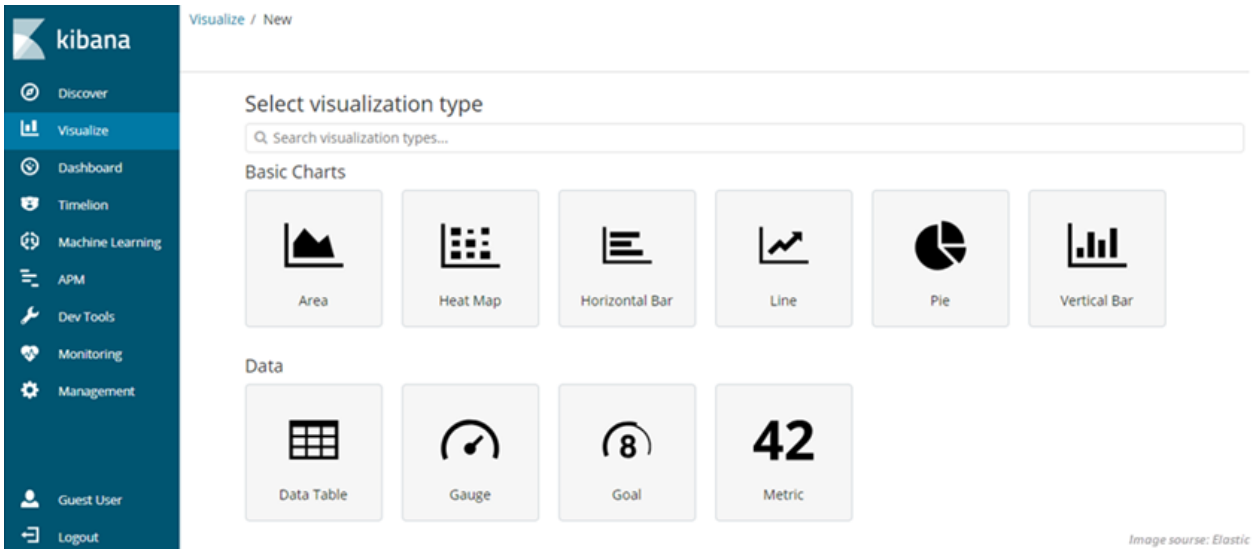

Рисунок 1.4 – Інструмент візуалізації Kibana

Kibana призначена для адміністраторів, аналітиків та бізнес-користувачів. Як для адміністратора, його роль полягає в управлінні Elastic Stack - від створення і розгортання до отримання даних Elasticsearch в Kibana і подальшого управління даними. Як для аналітика, Kibana допоможе шукати інсайти в даних, візуалізувати їх на інформаційних панелях та ділитися висновками. Бізнес-користувач, своєю чергою, зможе переглядати існуючі дашборди і заглиблюватися в інфографічні деталі.

Kibana працює з усіма типами даних. Це може бути структурований або неструктурований текст, числові дані, часові ряди, геопросторові дані, журнали, метрики, події безпеки тощо. Незалежно від типу даних, Kibana допоможе виявляти закономірності та взаємозв'язки і візуалізувати результати [15].

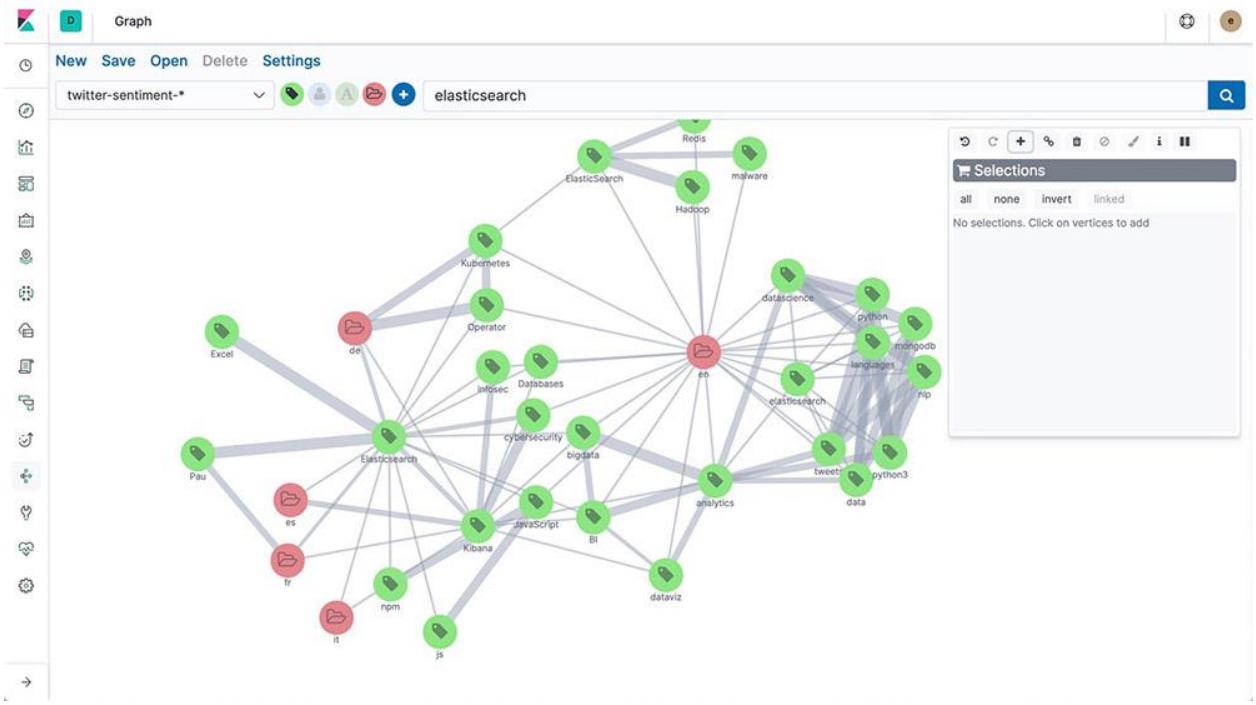

Рисунок 1.5 – Дослідження зв’язків у ElasticSearch з використанням інструменту Kibana

Серед проблем при використанні Kibana спеціалісти відмічають часті проблеми при використанні різних версій програмного забезпечення (наприклад, при додаванні плагінів).

1.4.2 Azure Cognitive Search

Azure Cognitive Search [16] від Microsoft Azure [17] - це хмарна пошукова служба (Search as a Service), яка надає програмістам інфраструктуру, API та інструменти, необхідні для створення розширених можливостей пошуку на основі приватних і гетерогенних колекцій даних, а потім використання їх у веб-додатках, мобільних додатках і бізнес-рішеннях.

Як зазначають розробники [18], незважаючи на те, що пошук сам по собі є повноцінною послугою, компанії мають можливість створювати індивідуальні рішення, інтегруючи свої моделі управління даними. Крім того, як і у випадку з усіма продуктами та службами Azure, користувачам доступна

потужна інфраструктура безпеки та конфіденційності від Microsoft, яка допомагає захистити інформацію як компанії, так і самих клієнтів.

Перш за все, використання Azure Cognitive Search передбачає організацію даних у структурованому вигляді та їхню індексацію, щоб користувачі могли легко їх шукати. Сервіс надає два різні рушії індексації: технологію обробки природної мови (Natural Language Processing або NLP [19]), що належить Microsoft, або аналізатори бібліотек з відкритим вихідним кодом Apache Lucene [20].

При індексації, наприклад, текстових документів, Azure Cognitive Search проаналізує вміст кожного документа і створює індекс, який пов'язує ключові слова з документами. Цей індекс дасть змогу Azure Cognitive Search швидко знаходити документи, що відповідають певним пошуковим запитам.

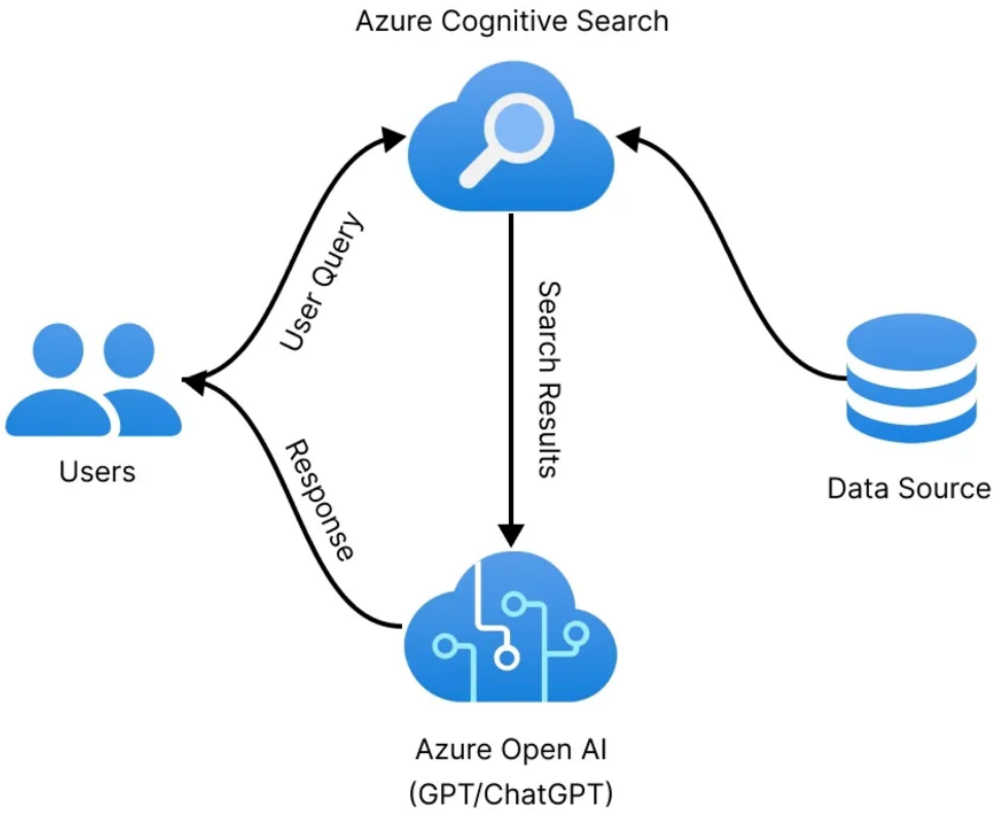

Рисунок 1.6 – Функціональна схема реалізації семантичного пошуку в Azure Cognitive Search [21]

Після індексації даних користувачі зможуть виконувати пошук, використовуючи певні ключові слова або фрази. Коли користувач вводить пошуковий запит, Azure Cognitive Search переглядає індекс даних, щоб знайти збіги між ключовими словами і раніше проіндексованими даними.

Результати будуть ранжовані Cognitive Search на основі їхньої релевантності. Існує кілька факторів, що визначають релевантність результату, наприклад, частота ключових слів у документі або їхнє розташування в ньому.

Відмінною рисою Azure Cognitive Search є впровадження можливостей штучного інтелекту для значного поліпшення функціональності пошуку.

Сервіс може використовувати оптичне розпізнавання символів (OCR) для вилучення тексту з відсканованих зображень або документів без необхідності додаткового втручання. Використання алгоритмів машинного навчання також дає змогу Cognitive Search розуміти зміст тексту і дає користувачам можливість знаходити інформацію, навіть якщо документ не містить саме те ключове слово, яке вони шукали.

Таким чином, Azure Cognitive Search може знаходити потрібну користувачеві інформацію, беручи до уваги контекстну близькість до теми пошуку, синоніми ключових слів, які використовуються в індексованих результатах, та інші дискримінаційні критерії, щоб звузити коло релевантних результатів, не жертвуючи потенційно значущими кореляціями.

Система демонструє впевнені кроки до здатності інтерпретувати аудіозаписи, зображення та тексти, щоб для пошуку певного контенту можна було застосовувати семантичний пошук, використовуючи наявність синонімів для пошуку певного документу, або ж розпочинати пошук незалежно від мови, якою було зроблено запит, чи мови, якою було написано документи.

Слід також зазначити, що реалізована інтеграція функцій Azure Cognitive Search з Sharepoint [22], яка дає змогу компаніям розширювати можливості пошуку в Sharepoint, використовуючи розширені функції Azure Cognitive Search, який тепер може підключатися до сайтів і бібліотек

застосунку для індексації контенту, як-от документи, списки та інші типи даних, присутніх у ньому. Інтегрувавши Cognitive Search Azure з SharePoint, організації тепер можуть створити уніфіковану систему пошуку в усіх своїх сховищах контенту, включно із сайтами SharePoint, зовнішніми базами даних та іншими джерелами даних.

Таким чином, Azure Cognitive Search пропонує рішення для індексації різноманітних джерел інформації та додавання додаткового контексту до результатів пошуку.

1.5 Висновки про поточний стан питання семантичного пошуку інформації

Аналіз інтернет-джерел продемонстрував, що у сфері семантичного інформаційного пошуку існує безліч інструментів, що забезпечують ефективний доступ до великих масивів даних. Серед найбільш популярних і широко використовуваних систем можна виділити Elasticsearch та Azure Cognitive Search. Обидва рішення пропонують потужні можливості для індексування та пошуку інформації, проте мають свої особливості та відмінності.

Elasticsearch являє собою розподілену пошукову та аналітичну систему з відкритим вихідним кодом, засновану на бібліотеці Lucene. Вона має високу масштабованість, гнучкість і підтримує складні запити, що робить її популярним вибором для організацій, які потребують потужних інструментів для обробки великих обсягів даних. Elasticsearch надає багатий набір API, інтеграцію з різними платформами та підтримує розширені функції, такі як повнотекстовий пошук, агрегації та аналітика в реальному часі. Однак, налаштування та управління Elasticsearch може бути складним завданням, що вимагає значних технічних знань і досвіду.

З іншого боку, Azure Cognitive Search від Microsoft являє собою хмарний сервіс, що надає можливості пошуку як сервісу. Він інтегрований з іншими хмарними сервісами Azure, що полегшує його впровадження та управління.

Azure Cognitive Search пропонує інтуїтивно зрозумілий інтерфейс для створення індексів, налаштування пошукових параметрів та інтеграції з додатками. Крім того, сервіс містить вбудовані функції машинного навчання і штучного інтелекту, що дають змогу поліпшити якість пошуку через семантичний аналіз і обробку природної мови. Незважаючи на зручність використання, Azure Cognitive Search може бути менш гнучким порівняно з Elasticsearch у плані налаштування та масштабування, особливо для специфічних вимог.

Крім Elasticsearch і Azure Cognitive Search, на ринку присутні й інші інструменти, такі як Apache Solr [23], Sphinx [24], Algolia [25] і Amazon OpenSearch Service [26]. Apache Solr, аналогічно до Elasticsearch, базується на Lucene і надає потужні можливості для повнотекстового пошуку та аналітики, але також потребує значних ресурсів для налаштування та підтримки. Sphinx орієнтований на високопродуктивний пошук і часто використовується у веб-додатках, однак його функціональність може бути обмежена порівняно з більш сучасними рішеннями. Algolia пропонує хмарний сервіс пошуку з акцентом на швидкість і простоту інтеграції, що робить його привабливим для розробників, які прагнуть швидко впровадити пошукові функції. Amazon OpenSearch Service, раніше відомий як Amazon Elasticsearch Service, надає керовану версію Elasticsearch, полегшуючи його розгортання в хмарному середовищі AWS.

Усі перераховані системи мають високу функціональність і можливості для налаштування, проте їхня складність може стати бар'єром для звичайних користувачів, які не володіють глибокими технічними знаннями. Налаштування, управління та оптимізація таких систем вимагають значних зусиль і ресурсів, що робить їх менш доступними для невеликих організацій або індивідуальних користувачів.

Таким чином, незважаючи на наявність потужних інструментів для семантичного інформаційного пошуку, їхня складність і вимогливість до ресурсів створюють необхідність у розробці простіших і зручніших рішень.

Для більшості користувачів важливо мати доступ до інструментів, які легко розгортаються, не потребують глибоких технічних знань і забезпечують ефективний пошук без необхідності складного налаштування. Розробка таких інструментів може значно розширити можливості семантичного пошуку, зробивши його доступним і корисним для широкої аудиторії, включно з малими підприємствами, освітніми установами та індивідуальними користувачами.

2 ФОРМУЛЮВАННЯ НАПРЯМКІВ ДОСЛІДЖЕНЬ ПРИ РОЗРОБЦІ СЕМАНТИЧНОЇ СИСТЕМИ ІНФОРМАЦІЙНОГО ПОШУКУ

Як було показано вище, розвиток інформаційних технологій і стрімке зростання обсягів даних, які створюють і поширюють у цифровому просторі, висувають нові вимоги до методів пошуку та обробки інформації. В умовах інформаційного перенасичення традиційні механізми пошуку, засновані на простому співставленні ключових слів, часто виявляються неефективними для задоволення потреб користувачів у точній і релевантній інформації. Це підкреслює необхідність розроблення більш інтелектуальних і адаптивних систем, здатних враховувати семантичні зв'язки та контекст запитів.

Суспільство сьогодні характеризується експоненціальним збільшенням обсягу доступної інформації, що ускладнює процеси її пошуку та обробки. В умовах глобалізації та цифровізації економіки підприємства, освітні установи та звичайні користувачі стикаються з необхідністю швидкого і точного доступу до релевантної інформації. Семантичний інформаційний пошук, що має здатність глибоко аналізувати зміст даних і розуміти наміри користувачів, стає критично важливим інструментом для підвищення ефективності інформаційних систем. Актуальність цієї роботи зумовлена потребою у створенні досконаліших інструментів пошуку, здатних справлятися зі зростаючими обсягами даних і різноманітністю інформаційних запитів.

Ідея цієї роботи полягає в розробці програмного модуля семантичного інформаційного пошуку, який використовує сучасні технології оброблення природної мови для підвищення точності та релевантності результатів пошуку. Пропонований модуль має забезпечувати користувачам інтуїтивно зрозумілий інтерфейс і високу продуктивність під час обробки запитів.

Об'єктом дослідження є система семантичного інформаційного пошуку, що включає компоненти для індексування, аналізу та пошуку інформації на основі семантичних зв'язків і контексту запитів користувачів.

Предметом дослідження, своєю чергою, виступають методи та алгоритми семантичного аналізу даних, архітектурні рішення для інтеграції різних технологій обробки інформації, а також критерії оцінки ефективності роботи системи семантичного інформаційного пошуку.

Метою даної роботи є розробка функціонального програмного модуля семантичного інформаційного пошуку, здатного ефективно обробляти великі обсяги даних і забезпечувати точні та релевантні результати пошуку. Для досягнення цієї мети планується:

а) дослідити і вибрати найбільш підходящі методи і алгоритми семантичного аналізу;

б) розробити архітектуру системи, що забезпечує інтеграцію різних компонентів обробки інформації;

в) реалізувати програмний модуль і протестувати його на різних наборах даних;

д) оцінити ефективність розробленої системи за критеріями точності, швидкості та зручності використання;

е) запропонувати рекомендації щодо подальшого вдосконалення та масштабування системи.

Таким чином, формулювання напрямів досліджень спрямоване на створення сучасного і водночас простого інструменту, здатного значно поліпшити процеси пошуку та обробки текстової інформації, задовольняючи потреби сучасних користувачів в умовах постійно збільшуючогося інформаційного простору.

3 ВИЗНАЧЕННЯ ПІДХОДІВ ДЛЯ ВИРІШЕННЯ ЗАВДАННЯ СЕМАНТИЧНОГО ІНФОРМАЦІЙНОГО ПОШУКУ ТА ЇХ МАТЕМАТИЧНА ФОРМАЛІЗАЦІЯ

Семантичний інформаційний пошук виступає як спроба вирішити проблеми, що пов'язані з необхідністю глибокого розуміння сенсу запитів і вмісту документів. Для досягнення цієї мети використовують різні підходи. Аналіз літературних та інтернет джерел по заданій тематиці допоміг з’ясувати, що одними з основних методів для вирішення поставленого завдання можна вважати стемінг, лематизації та обробку природної мови (Natural Language Processing, NLP). Однак, перш за все, необхідно виконати попередню підготовку тексту та його токенізацію.

Первинні операції по підготовці тексту включають в себе перетворення всіх символів тексту в нижній регістр, що сприяє уніфікації текстових даних, зменшуючи кількість унікальних токенів і знижуючи розмірність даних. В українській мові відмінність між регістрами зазвичай не несе семантичного навантаження і може призводити до надмірності даних. Видалення пунктуації та спеціальних символів, своєю чергою, використовується через те, що ці символи часто не несуть значущої інформації при семантичному пошуку і можуть розглядатися як шумові дані, що заважають ефективному аналізу тексту.

Розглянемо докладніше, на чому базуються зазначені методи семантичного пошуку, як вони математично описуються, та як можуть бути втілені в життя.

3.1 Токенізація

Токенізація є одним із ключових етапів попередньої підготовки тексту і являє собою процес поділу тексту на окремі елементи - токени, які можуть бути словами, фразами або іншими значущими одиницями. Токени слугують

основою для побудови індексів і виконання пошукових запитів. Токенізація виконується після переходу до однакового регістру і видалення розділових символів, що забезпечує більш точний та ефективний поділ тексту на токени. В українській мові токенізація має враховувати певні особливості мови, включно зі складними структурами слів і наявністю сполучних елементів.

Процес токенізації може бути реалізовано з використанням різних методів. При цьому важливо використовувати метод, який забезпечує необхідне співвідношення точності і продуктивності, враховуючи вимоги системи пошуку та характеристики оброблюваних текстів.

Ефективна попередня підготовка тексту і токенізація є критичними для досягнення високої якості семантичного пошуку. Ці процеси сприяють поліпшенню якості індексації та оптимізації наступних етапів обробки. Таким чином, правильна токенізація полегшує завдання стемінгу, лематизації та семантичного аналізу, підвищуючи загальну точність системи.

3.2 Стемінг

Стемінг являє собою процес приведення словоформ до їхньої базової або кореневої форми шляхом механічного видалення суфіксів і префіксів. Це сприяє уніфікації та скороченню розмірності даних. Стемінг дає змогу знизити варіативність слів, що полегшує завдання пошуку та індексації. Однак, незважаючи на свою ефективність у зменшенні розмірності даних, стемінг може іноді призводити до втрати семантичної інформації, що обмежує його застосування в контексті семантичного пошуку.

Стемінг є ключовим етапом передобробки текстових даних у системах інформаційного пошуку. У контексті семантичного пошуку стемінг відіграє важливу роль, оскільки дозволяє покращити якість пошуку шляхом урахування різних морфологічних варіацій слів, зберігаючи при цьому їхнє смислове навантаження.

Українська мова має багату морфологію, включаючи різні закінчення для іменників, дієслів, прикметників та інших частин мови. Це створює

складні умови для ефективного стемінгу, що потребують урахування специфічних морфологічних особливостей мови. Застосування стемінгу до української мови включає розробку алгоритмів, здатних коректно опрацьовувати різні афікси та закінчення, характерні для української морфології.

Для української мови стемінг може бути реалізований з використанням правил на основі морфологічних структур або із застосуванням статистичних методів. В математичній моделі стемінгу для української мови враховуються ймовірності появи різних суфіксів та префіксів, а також їх вплив на формування кореня слова.

При математичному моделюванні процесу стемінгу можна виділити різні підходи. Одним з них може бути використання ланцюгів Маркова [27].

Ланцюги Маркова - це математична модель, що описує систему, яка переходить з одного стану до іншого. Головна властивість такої системи - відсутність пам'яті: ймовірність переходу до наступного стану залежить тільки від поточного стану і не залежить від того, як система дійшла до нього.

Варто зазначити, що ланцюги Маркова - це потужний інструмент для обробки послідовних даних, таких як текст. У контексті стемінгу, моделі Маркова можуть ефективно імітувати процес перетворення форм слів у їх основні або кореневі форми, враховуючи ймовірнісні залежності між символами або морфемами слова.

У завданнях обробки природної мови, насамперед при виконанні стемінгу, марківські моделі можуть використовуватися для моделювання послідовності символів або морфем у словах, що дозволяє передбачати ймовірні перетворення словоформ у їхні стеми.

В контексті стемінгу марківські моделі можуть бути використані для моделювання процесу видалення суфіксів та префіксів, а також для ідентифікації кореневої основи слова. Особливо це стосується прихованих марківських моделей. Розглянемо основні етапи застосування прихованих марківських моделей для стемінгу українських слів.

Прихована марківська модель складається з таких компонентів:

а) стани (S), що можуть представляти різні морфеми або позиції в слові (наприклад, корінь, суфікс, префікс);

б) спостережувані символи (V) - символи або морфеми словоформи, з яких складається слово;

в) перехідні ймовірності (A), що визначають ймовірності переходу від одного стану до іншого;

д) імовірності емісії (B) - ймовірності появи конкретного спостережуваного символу в даному стані;

е) початкові ймовірності (π) - ймовірності того, що слово починатиметься з певного стану.

Математично параметри прихованої марківської моделі можна визначити наступним чином

$$\lambda = (\mathrm{A}, \mathrm{B}, \pi) \tag{3.1}$$

де $\mathrm{A} = \{a_{\mathrm{ij}}\}$; $\mathrm{B} = \{b_{\mathrm{j}}(\mathrm{k})\}$; $\pi = \{\pi_{\mathrm{i}}\}$

де, в свою чергу,

$a_{\mathrm{ij}} = \mathrm{P}(S_j \mid S_i)$ - ймовірність переходу зі стану S_i у стан S_j;

$b_{\mathrm{j}}(\mathrm{k}) = \mathrm{P}(V_k \mid S_j)$ - імовірність емісії символу V_k у стан S_j;

$\pi_{\mathrm{i}} = \mathrm{P}(S_i)$ – початкова ймовірність стану S_i.

Розглянемо приклад використання ланцюгів Маркова для здійснення операції стемінгу. Використаємо для цього слово «*говорять*» та виконаємо необхідні операції.

Першою чергою, відбувається розбиття зазначеного слова на окремі символи.

В другу чергу, визначається стан для кожного символу. Стани можуть включати в себе наступні сутності: «корінь», «суфікс», «закінчення». При

цьому визначається найбільш вірогідна послідовність станів, наприклад, згідно алгоритму Вітебрі [28].

Алгоритм Вітербі відіграє важливу роль у визначенні найімовірнішої послідовності станів, що відповідає заданій словоформі. Це дає змогу ефективно виділяти кореневу основу слова, мінімізуючи ймовірність помилок під час перетворення різних форм слова до його стеми. Сьогодні він вважається одним із найнадійніших методів, не дивлячись на деякі обмеження, що пов’язані з обробкою морфологічної складності мови.

Приклад визначення станів наведено у таблиці 3.1. Для першого символу "г" обчислюються ймовірності початкових станів, наприклад, тільки "корінь" має високу ймовірність емісії "г".

Таблиця 3.1 – Приклад визначення станів згідно алгоритму Вітебрі

№	**Символ**	**Частина мови**
1.	«г»	корінь
2.	«о»	корінь
3.	«в»	корінь
4.	«о»	корінь
5.	«р»	корінь
6.	«я»	суфікс
7.	«т»	суфікс
8.	«ь»	закінчення

Таким чином, при здійсненні стемінгу виконується визначення стему «говор» на основі кореневих станів.

Цікаво те, що аналіз існуючих бібліотек продемонстрував фактичну відсутність модулів для реалізації стемінгу текстів на українській мові. Тому було вирішено використати методику, розглянуту в науковій статті [29], яка полягала в розробці регулярних виразів для різних частин мови. Вона була

запропонована на основі алгоритму Портера [30], що відсікає суфікси та закінчення, не використовуючи бази коренів слів. В роботі було адаптовано алгоритм під українську мову шляхом визначення регулярних виразів для ідентифікації частин мови по закінченнях та суфіксах.

Таблиця 3.2 – Регулярні вирази для реалізації стемінгу української мови

№	Частина мови	Регулярний вираз
1.	Дієприслівник	/((ив|ивши|ившись))$/
2.	Інфінітив	/(ти|учи|ячи|вши|ши|ати|яти|ючи|)$/
3.	Рефлексивне дієслово	/(с[яьи])$/
4.	Прикметник	/(ими|ій|ий|а|е|ова|ове|ів|є|їй|єє|єє| я|ім|ем|им|ім|их|іх|ою|ими|іми|у|ю|ого|ому|ої)$/
5.	Дієприкметник	/(ий|ого|ому|им|ім|а|ій|у|ою|ій|і|их| ими|их)$/
6.	Дієслово	/(сь|ся|ив|ать|ять|у|ю|ав|али|учи|ячи| вши|ши|е|ме|ати|яти|є)$/
7.	Іменник	/(а|ев|ов|е|ями|ами|еи|и|ей|ой|ий|й| иям|ям|ием|ем|ам|ом|о|у|ах|иях|ях|ы|ь|ию|ью|ю| ия|ья|я|і|ові|ї|ею|єю|ою|є|еві|ем|єм|ів|їв|\'ю)$/
8.	Частина після першого голосного, або кінець слова, що не містить голосних	/^(.*?[аеиоуюяіїє])(.*)$/
9.	Словотвірні частини	/[^аеиоуюяіїє][аеиоуюяіїє]+ [^аеиоуюяіїє]+[аеиоуюяіїє].*сть?$/

Таким чином, стемінг дозволяє привести слово до кореневої форми, що дозволяє, в свою чергу, здійснювати семантичний інформаційний пошук, ґрунтуючись на однокореневості слів.

3.3 Лематизація

Лематизація є ще одним методом обробки природної мови і займає важливе місце у системах семантичного інформаційного пошуку. Основне завдання лематизації полягає у приведенні словоформ до їхньої базової або словникової форми - леми. На відміну від вищеописаного стемінгу, який часто використовує прості евристичні методи для видалення афіксів, лематизація враховує морфологічні особливості мови, забезпечуючи більш точне й осмислене перетворення слів. У контексті семантичного пошуку лематизація сприяє поліпшенню якості індексації та підвищенню релевантності результатів пошуку за рахунок уніфікації різних форм одного й того самого слова.

Лематизація допомагає скоротити варіативність словоформ, що, своєю чергою, зменшує розмірність даних і полегшує завдання зіставлення запитів користувачів з індексованими документами. Більше того, лематизація дає змогу системі розуміти семантичні зв'язки між словами, що є основою для реалізації просунутих пошукових механізмів.

Використання лематизації в системах семантичного пошуку сприяє поліпшенню точності пошуку - шляхом приведення всіх форм слова до єдиної леми, система може ефективно знаходити релевантні документи незалежно від форми використання слова в запиті. Також підвищується релевантність результатів через врахування семантичних зв'язків між словами, що сприяє точнішій відповідності наміру користувача і результатів, що повертаються. Ще одним з позитивних ефектів є зниження розмірності даних, адже зменшення кількості унікальних токенів за рахунок об'єднання різних форм одного слова спрощує процес індексування і прискорює пошук.

Математично процес лематизації можна представити у вигляді морфологічних моделей. Останні ґрунтуються на правилах, що описують структуру слів та їхнього морфемного складу. У рамках лематизації такі моделі формалізують процес розбору словоформ на морфеми і подальшого відновлення леми.

Якщо прийняти W за множину всіх словоформ мови, L - за множину лем, а M за множину морфем, то функція лематизації

$$\psi: W \to L \tag{3.2}$$

визначатиметься як

$$\psi(w) = l \tag{3.3}$$

де $w \in W$ — словоформа,

$l \in L$ — відповідна лема.

Своєю чергою, морфологічну модель можна подати як множину правил R, кожне з яких описує перетворення певної морфемної структури словоформи на лему:

$$R = \{r_1, r_2, \dots, r_n\} \tag{3.4}$$

де кожне правило r_i має вигляд:

$$\boldsymbol{r_i}: \text{словоформа} \to \text{лема}$$

Дослідження продемонстрували, що існують бібліотеки для реалізації операцій лематизації україномовних текстів. Однією з них може бути NHunspell [31] — .NET-обгортка для популярного інструмента Hunspell [32], який використовується для перевірки орфографії та розбиття слів на склади. Хоча Hunspell в основному призначений для перевірки правопису, його можна

використовувати для отримання базових форм слів, що може бути корисним для лематизації.

Таким чином, лематизація є важливим компонентом систем семантичного інформаційного пошуку, забезпечуючи уніфікацію та стандартизацію текстових даних.

4 ПРОЕКТУВАННЯ ПРОГРАМНОГО МОДУЛЮ ДЛЯ СЕМАНТИЧНОГО ІНФОРМАЦІЙНОГО ПОШУКУ

4.1 Формулювання вимог до функціональних можливостей програмного модуля, що розроблюється

Як зазначалося вище, сьогодні постає гостра потреба в ефективних системах пошуку, здатних знаходити релевантні дані не тільки за ключовими словами, а й за смисловим змістом. Розроблення системи семантичного інформаційного пошуку має задовольняти низку вимог, що забезпечують її функціональність та ефективність.

Система має підтримувати опрацювання текстових документів українською мовою, враховуючи лінгвістичні особливості та мовні нюанси. Це передбачає коректне розпізнавання та інтерпретацію слів, фраз і граматичних конструкцій, характерних для української мови.

Необхідно реалізувати функцію попереднього опрацювання текстів, що передбачає приведення всіх символів до нижнього регістру, видалення пунктуації та спеціальних символів, а також токенізацію - розбиття тексту на окремі слова або токени. Це забезпечить однаковість даних і підготує їх для подальшого аналізу.

Важливим компонентом є можливість стемінгу та лематизації. Стемінг дає змогу зводити слова до їхньої кореневої форми, полегшуючи зіставлення різних форм одного й того самого слова. Він являє собою процес спрощення словоформ до їхніх базових коренів – стемів – які можуть не завжди відповідати реальним словам. Наприклад, слова "говорив", "говорить" і "говоріння" можуть бути зведені до кореня "говор". Лематизація перетворює слова в їхню вихідну словникову форму або лему, враховуючи морфологічні особливості мови. Таким чином, ті самі слова приведуться до форми "говорити". Ці операції необхідні для усунення варіативності словоформ, що дає змогу системі більш ефективно порівнювати та індексувати тексти, а також

покращує якість пошуку, зменшуючи кількість нерелевантних результатів. При цьому користувач повинен мати можливість вмикати або вимикати ці функції через відповідні налаштування інтерфейсу, адаптуючи роботу системи під свої потреби.

Побудова інвертованого індексу є необхідною для прискорення процесу пошуку. Індекс має динамічно оновлюватися в разі зміни налаштувань системи, наприклад активації чи деактивація стемінгу та лематизації. Це гарантує, що пошук завжди буде виконуватися за актуальними даними, які відповідають поточним налаштуванням користувача.

Система має надавати зручний та інтуїтивно зрозумілий інтерфейс для завантаження та обробки документів. Користувач повинен мати можливість обирати директорії з файлами. Після завантаження система має відображати статистичну інформацію, наприклад, кількість опрацьованих файлів, сумарний обсяг даних у кілобайтах і загальне число символів.

Під час введення пошукового запиту користувач має отримувати швидкий відгук системи. Час, витрачений на пошук, має відображатися, демонструючи ефективність роботи програми. Результати пошуку мають надаватися в упорядкованому вигляді, відображаючи ступінь релевантності кожного знайденого документа щодо запиту. Інформація про релевантність допомагає користувачеві швидко визначити найбільш підходящі матеріали.

Система має забезпечувати можливість перегляду вмісту знайдених документів прямо з інтерфейсу. При цьому ключові слова або фрази, що відповідають пошуковому запиту, мають бути візуально виділені, полегшуючи навігацію по тексту і прискорюючи пошук потрібної інформації всередині документа.

Система має бути оптимізована для роботи з великими обсягами даних, забезпечуючи високу продуктивність і стабільність. Вона повинна ефективно використовувати ресурси і масштабуватися залежно від обсягу оброблюваної інформації.

Нарешті, важливо забезпечити гнучкість і настроюваність системи. Користувач повинен мати можливість легко змінювати налаштування й адаптувати роботу програми під свої конкретні завдання та уподобання. Це підвищить задоволеність від використання системи і розширить сферу її застосування.

У сукупності ці вимоги формують функціональні можливості системи семантичного інформаційного пошуку, здатної ефективно обробляти й аналізувати тексти українською мовою, надаючи користувачам швидкий і точний доступ до необхідної інформації.

4.2 Формулювання вимог до засобів кодування та апаратного забезпечення для розгортання програмного модуля

Вдале формулювання вимог стосовно засобів кодування та вимог до апаратної складової є важливим етапом для розробки надійного програмного модулю.

Таблиця 4.1 демонструє перелік запропонованих до використання засобів створення програмного модуля, а також вимог до апаратної складової, на якій планується розгортання програмного комплексу семантичного інформаційного пошуку.

Таблиця 4.1 – Перелік вимог до засобів кодування при створенні програмного модулю

Опис	Вимоги
Мова програмування	C#
Платформа розробки	.NET Framework 4.5 або вище
Середовище розробки	Visual Studio 2015 або вище
Операційна система	Windows 7 або вище
Можливість застосування сторонніх бібліотек	Так

Таблиця 4.2 – Перелік вимог до апаратного забезпечення при розгортанні системи семантичного інформаційного пошуку

Опис	Мінімальні вимоги
Процесор	1,5 GHz
Оперативна пам'ять	3 ГБ
Відеокарта	Вбудована або дискретна
Місце на жорсткому диску (HDD/SSD)	200 МБ
Монітор	Роздільна здатність 1024x768
Клавіатура та миша	Так
Принтер	Опціонально (для друку даних)

Виконання встановлених вимог до версій програмних засобів та характеристик апаратного забезпечення є важливим аспектом для ефективної розробки та подальшого функціонування програмного модулю семантичного інформаційного пошуку.

4.3 Структурна побудова програмного модуля семантичного інформаційного пошуку

У розроблюваному програмному модулі семантичного інформаційного пошуку мають бути реалізовано ключові компоненти, що дають змогу ефективно опрацьовувати й аналізувати текстові дані українською мовою. Основними логічними блоками, що забезпечують функціональність системи, є токенізація, стемінг і лематизація. Їхній взаємозв'язок та інтеграція в загальний процес опрацювання даних забезпечують релевантність результатів пошуку.

Архітектура модуля побудована таким чином, щоб забезпечити послідовне опрацювання текстових даних. Процес починається з отримання вихідного тексту, який передається в блок токенізації. Отримані токени потім проходять через блоки стемінга і лематизації, залежно від обраних користувачем налаштувань і необхідного рівня обробки.

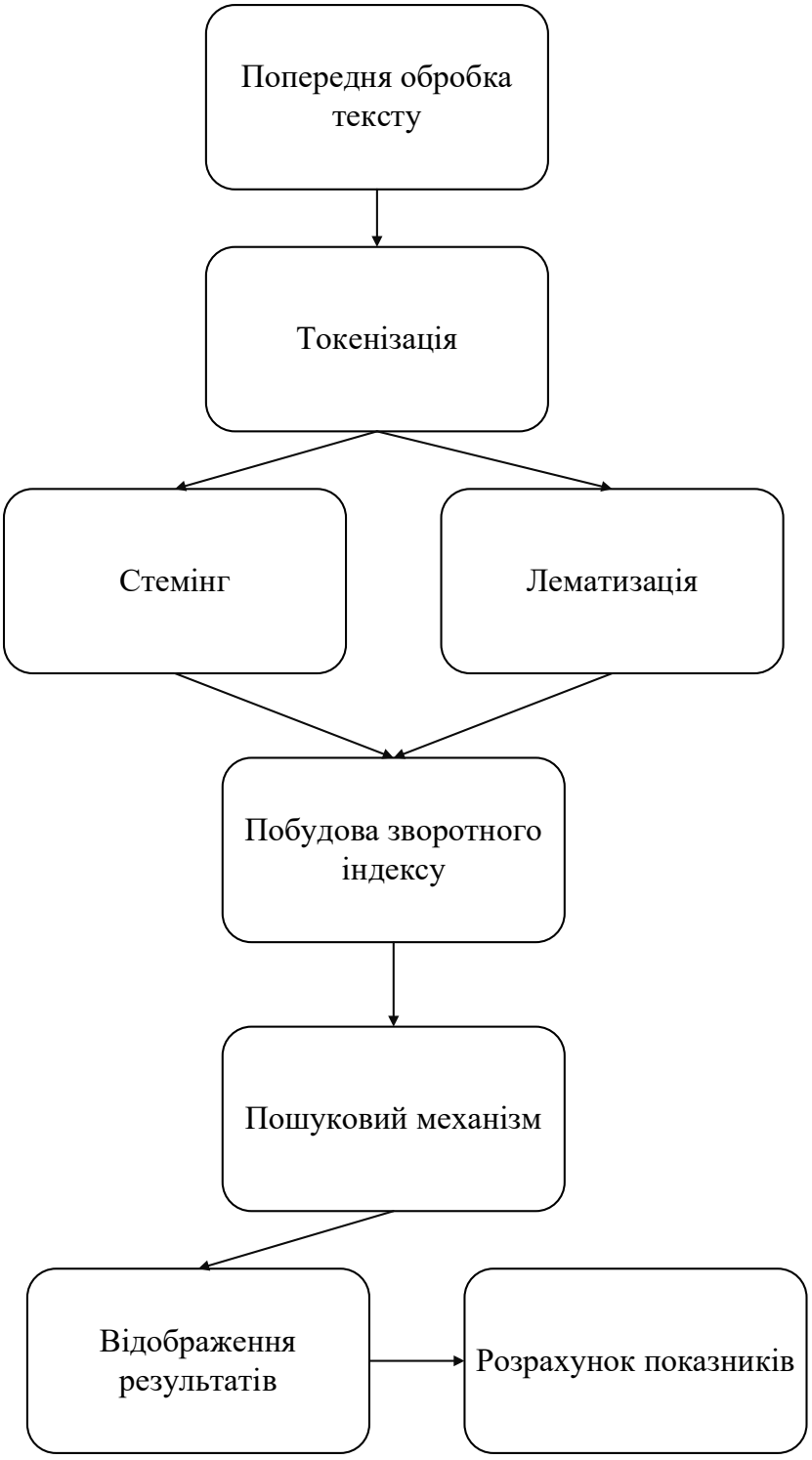

Рисунок 4.1 – Архітектура програмних модулів системи семантичного інформаційного пошуку

Токенізація слугує першим етапом у процесі обробки текстової інформації. Цей логічний блок відповідає за розбиття вхідного тексту на окремі елементи - токени, якими зазвичай є слова або розділові знаки.

Правильна токенізація забезпечує основу для подальших етапів обробки, оскільки всі подальші операції застосовуються саме до отриманих токенів.

У програмі токенізацію реалізовано з використанням регулярних виразів і спеціальних алгоритмів, що дають змогу коректно розділяти текст на слова, видаляти непотрібні символи та приводити дані до уніфікованого формату. Цей блок тісно пов'язаний із попередньою обробкою тексту, де виконується нормалізація тексту, включно з приведенням до нижнього регістру і видаленням пунктуації.

Реалізація стемінгу базується на спеціалізованих алгоритмах, адаптованих під морфологію української мови. Цей блок взаємодіє з результатами токенізації, приймаючи на вхід отримані токени та перетворюючи їх на стеми. Отримані стеми потім використовуються під час побудови інвертованого індексу та в процесі зіставлення з пошуковими запитами.

Лематизація, в свою чергу, реалізована з використанням лематизаторів, здатних обробляти українську мову. Цей блок тісно інтегрований із токенізацією, оскільки для коректної роботи йому необхідні правильно визначені токени. Лематизація використовується в програмі для поліпшення якості пошуку, особливо під час опрацювання складних запитів і текстів з багатою морфологією.

Важливою особливістю системи є можливість гнучкого управління цими блоками. Користувач може вмикати або вимикати стемінг і лематизацію, адаптуючи роботу програми під конкретні завдання. Наприклад, для швидкого пошуку за ключовими словами може бути достатньо стемінгу, тоді як для більш точного семантичного аналізу кращою є лематизація.

Після обробки токенів блоками стемінга і лематизації результати використовуються для побудови інвертованого індексу. Цей індекс є основою для швидкого та ефективного пошуку, даючи змогу системі швидко знаходити документи, що містять необхідні терміни або їхні форми. У процесі пошуку

запит користувача також проходить через ті самі блоки обробки, що забезпечує коректне зіставлення з індексом.

Логічні блоки токенізації, стемінгу та лематизації не існують ізольовано, а тісно пов'язані з інтерфейсом користувача та налаштуваннями системи. Користувацький інтерфейс надає засоби для вибору необхідних опцій обробки, які безпосередньо впливають на роботу відповідних блоків. Це забезпечує інтерактивність системи та дає змогу користувачеві контролювати процес пошуку.

У разі зміни налаштувань система автоматично адаптує послідовність і набір застосовуваних логічних блоків. Наприклад, якщо користувач вимикає лематизацію, відповідний блок виключається з ланцюжка обробки, і результати перераховуються з урахуванням нових параметрів.

Результати роботи логічних блоків зберігаються і використовуються для оптимізації наступних пошукових операцій. Інвертований індекс, побудований на основі оброблених даних, дає змогу системі швидко знаходити релевантні документи без необхідності повторного повного опрацювання тексту. Це значно підвищує продуктивність системи, особливо під час роботи з великими обсягами даних.

4.4 Алгоритм роботи програмного модуля семантичного інформаційного пошуку

Нижче наведено докладний опис блок-схеми алгоритму модуля семантичного інформаційного пошуку, що відображає детерміновану послідовність дій програми.

Процес починається з ініціалізації системи. Користувач надає шлях до директорії, що містить текстові документи у форматі .txt. Система автоматично сканує зазначену папку, ідентифікуючи та витягуючи вміст кожного файлу.

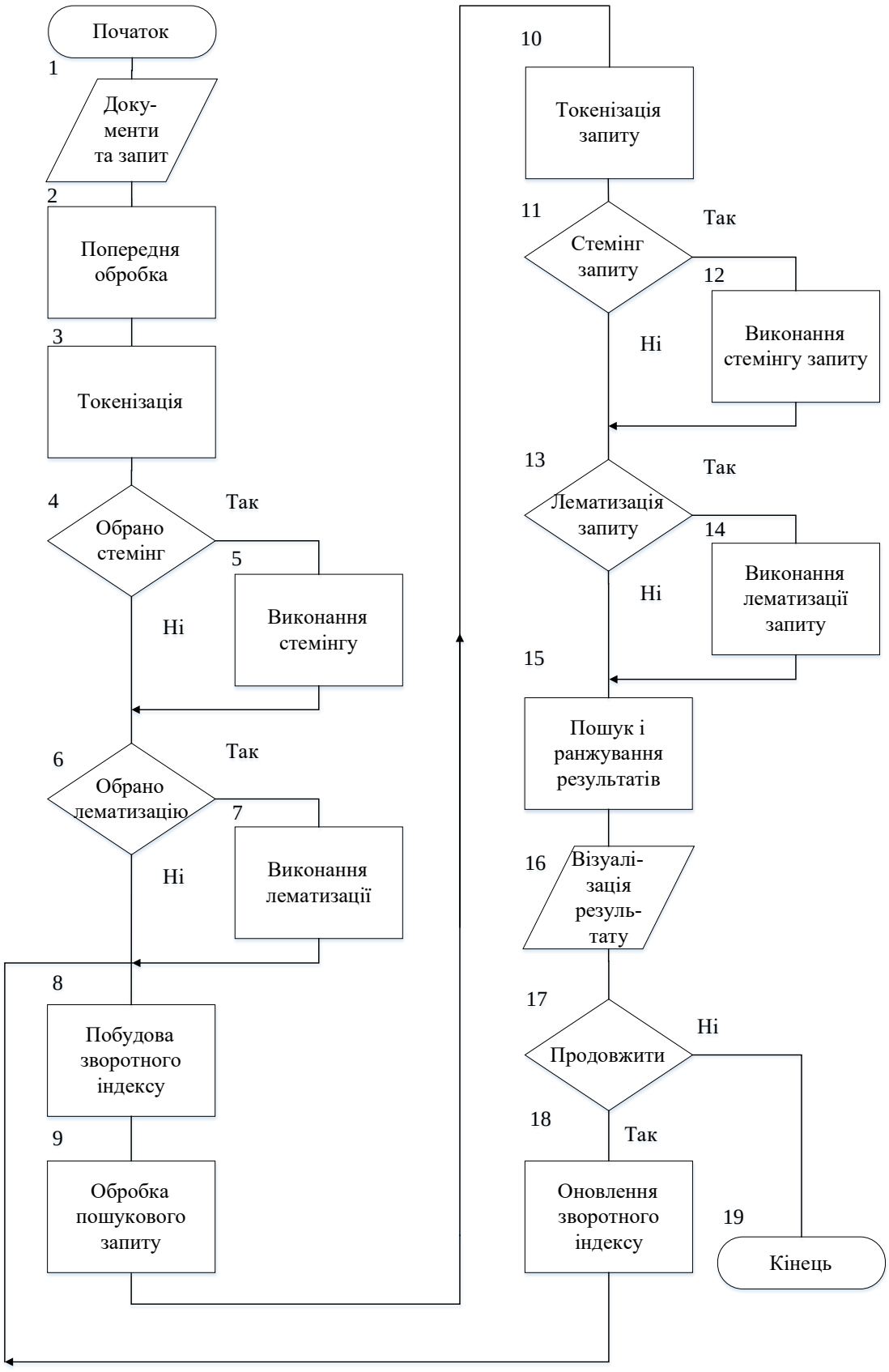

Рисунок 4.2 – Алгоритм роботи програмного модуля семантичного інформаційного пошуку

Після успішного завантаження документів система переходить до етапу попередньої обробки тексту. На цьому етапі текст кожного документа приводиться до нижнього регістру для уніфікації даних і видалення зайвих символів, таких як пунктуація та спеціальні знаки, за допомогою регулярних виразів. Це забезпечує чистоту й однорідність тексту.

Наступний крок - токенізація, де очищений текст розбивається на окремі елементи, які являють собою слова або значущі фрази. Токенізація виконується з урахуванням особливостей української мови, включно з правильним поділом словоформ і врахуванням специфічних граматичних конструкцій. Отримані токени слугують основою для подальших операцій обробки.

Після токенізації система здійснює стемінг. Це дає змогу зменшити варіативність словоформ і спростити процес індексації та пошуку. Також виконується лематизація. Ці дві операції взаємодоповнюють одна одну, забезпечуючи комплексну обробку тексту для семантичного пошуку.

Після виконання стемінгу та лематизації, оброблені токени використовуються для побудови інвертованого індексу. Цей індекс являє собою структуру даних, яка пов'язує кожне слово (або його базову форму) з переліком документів, у яких воно зустрічається. Інвертований індекс значно прискорює процес пошуку.

Коли користувач вводить пошуковий запит, система виконує аналогічні етапи попереднього опрацювання: приведення тексту запиту до нижнього регістру, видалення зайвих символів, токенізацію, стемінг і лематизацію. Це забезпечує зіставлення запиту з уже проіндексованими документами на основі однакових базових форм слів, що підвищує точність пошуку. Після обробки запиту система звертається до інвертованого індексу для пошуку збігів. Кожен знайдений документ оцінюється за ступенем релевантності, яка визначається кількістю збігів токенів і їхнім значенням у контексті запиту. Результати пошуку ранжуються в порядку зменшення співпадінь, надаючи користувачеві найбільш релевантні документи першими.

Результати пошуку відображаються користувачеві з використанням ергономічного інтерфейсу, де кожен документ представлений із зазначенням його релевантності. Додатково, в текстах знайдених документів автоматично підсвічуються слова і фрази, що відповідають пошуковому запиту. Це полегшує навігацію і дає змогу користувачеві швидко орієнтуватися у вмісті документів, виділяючи ключові моменти.

Система також передбачає можливість динамічного оновлення інвертованого індексу в разі додавання нових документів або зміни наявних. Це забезпечує актуальність даних і дає змогу системі ефективно працювати з постійно оновлюваним контентом.

5 ОГЛЯД СТВОРЕНОГО ПРОГРАМНОГО МОДУЛЮ ДЛЯ СЕМАНТИЧНОГО ІНФОРМАЦІЙНОГО ПОШУКУ ТА ДОСЛІДЖЕННЯ ЙОГО РОБОТИ

В ході роботи було реалізовано систему семантичного інформаційного пошуку. Для зручності роботи з ним було розміщено усі елементи в межах однієї форми. Розглянемо інтерфейс створеного програмного модулю.

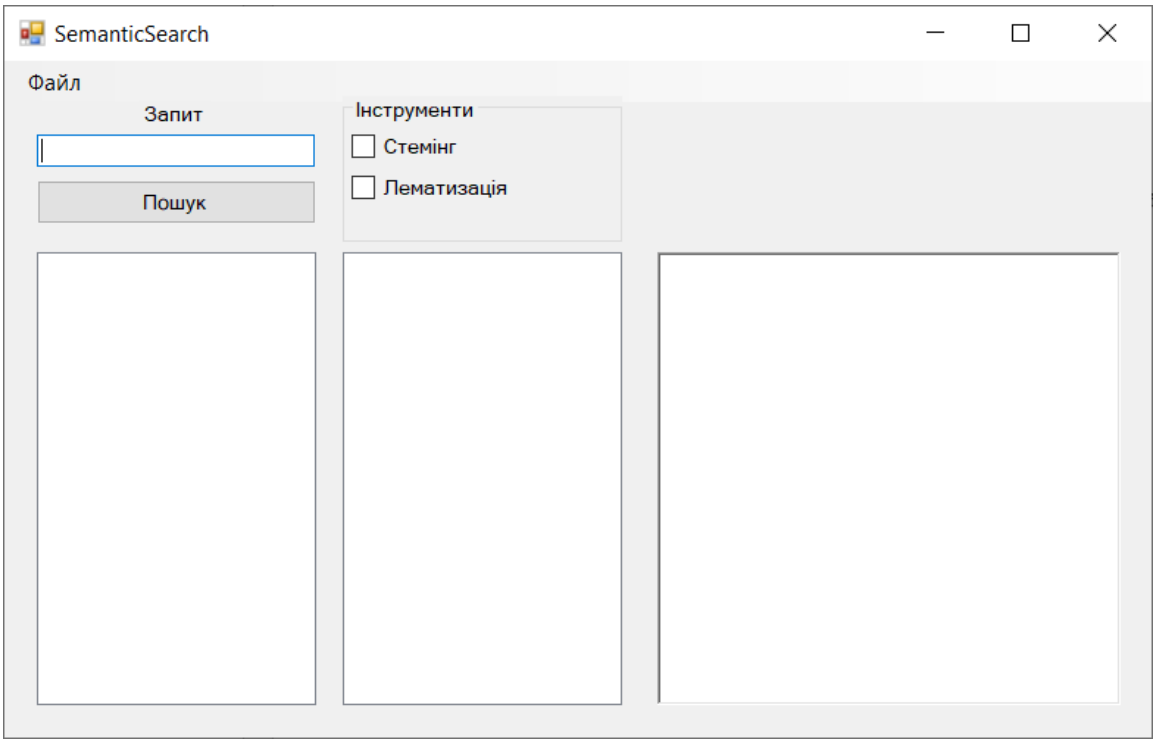

Рисунок 5.1 – Вікно програмного модулю після запуску

Після запуску необхідно завантажити текстові файли для аналізу в систему, що можна зробити шляхом використання меню «Файл» - «Відкрити директорію».

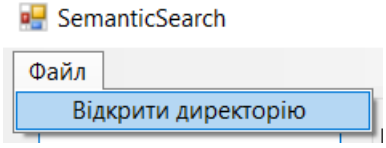

Рисунок 5.2 – Завантаження файлів

У випадку наявності в обраній директорії файлів та їх успішного завантаження, користувачеві буде надано повідомлення про успішну обробку даних.

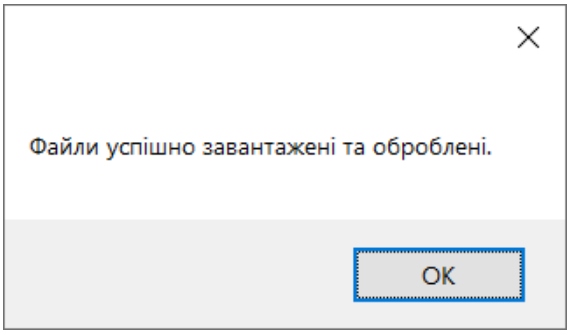

Рисунок 5.3 – Повідомлення про успішну обробку файлів

Після цього у лівому лістбоксі з’явиться перелік усіх завантажених файлів. Далі користувач може переходити безпосередньо до виконання операцій пошуку з використанням текстового поля «Запит».

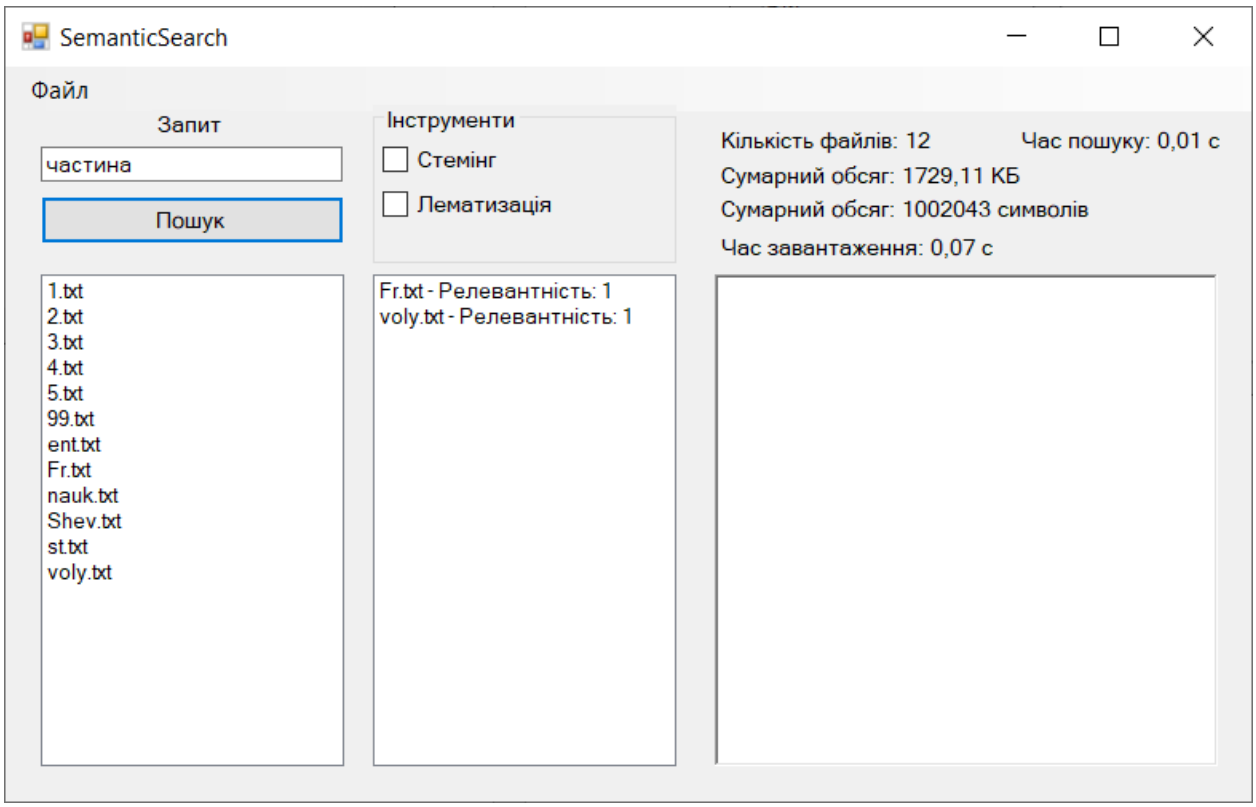

Рисунок 5.4 – Зміна стану після виконання пошуку

Слід зазначити, що в програмному модулі реалізовано три режими пошуку, що обираються у наборі чекбоксів «Інструменти». Це звичайний пошук (по повному співпадінню), пошук з використанням стемінгу (приведенні до кореневої форми шляхом видалення суфіксів і префіксів) та пошук з використанням лематизації (приведенні словоформ до словникової форми - леми). Для звичайного пошуку необхідно залишити усі чекбокси порожніми, а для підключення стемінгу або лематизації поставити відповідну галочку.

Після введення пошукового запиту та натискання кнопки «Пошук» в центральному лістбоксі з'являється перелік релевантних файлів, тобто тих документів, в яких були знайдені співпадіння. На рисунку 5.4 було наведено результати звичайного пошуку на основі повних співпадінь. При виборі потрібного документу в центральному лістбоксі з правої сторони буде наведено текст документа та підсвічено жовтим кольором слова, пошук яких виконувався (рисунок 5.5).

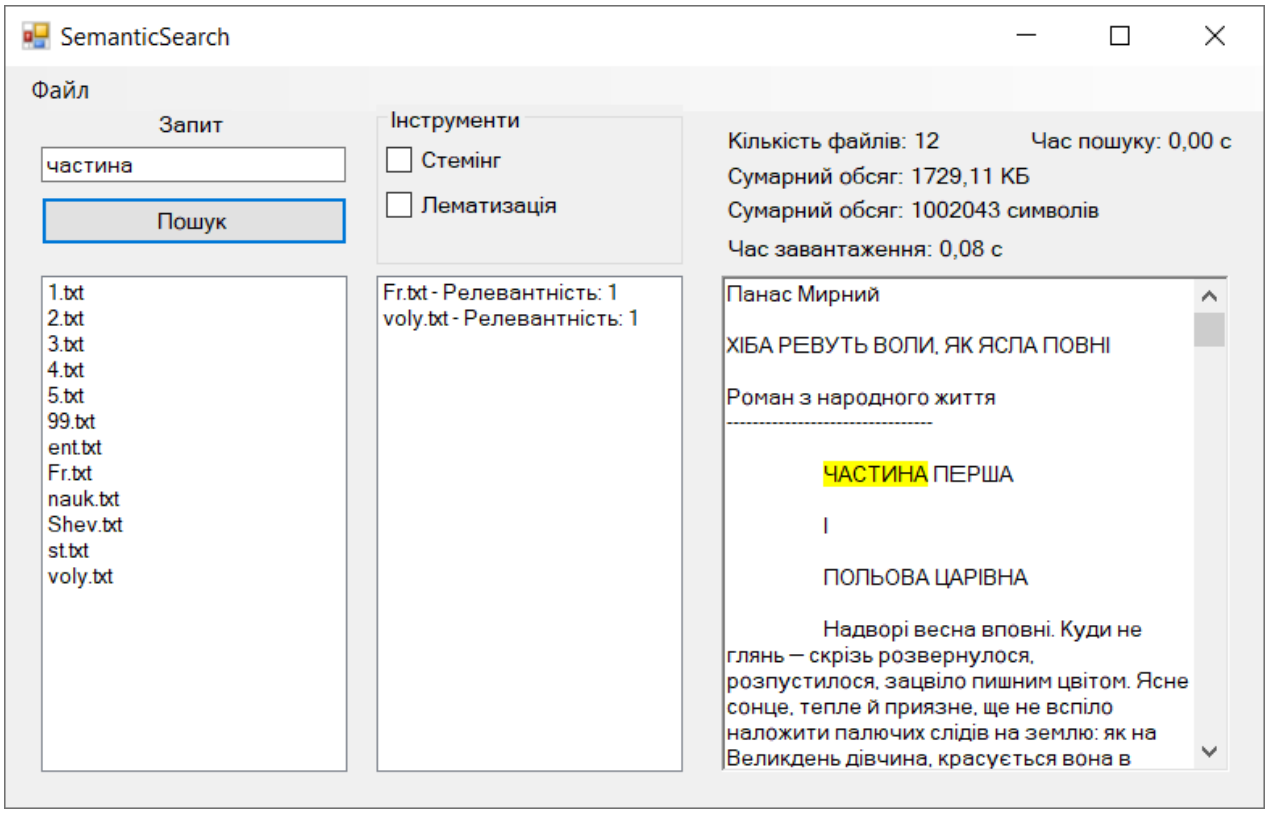

Рисунок 5.5 – Відображення співпадінь в документах

При активації пошуку на основі стемінгу будуть знаходитися однокореневі слова з відмінними суфіксами та закінченнями (рисунки 5.6, 5.7).

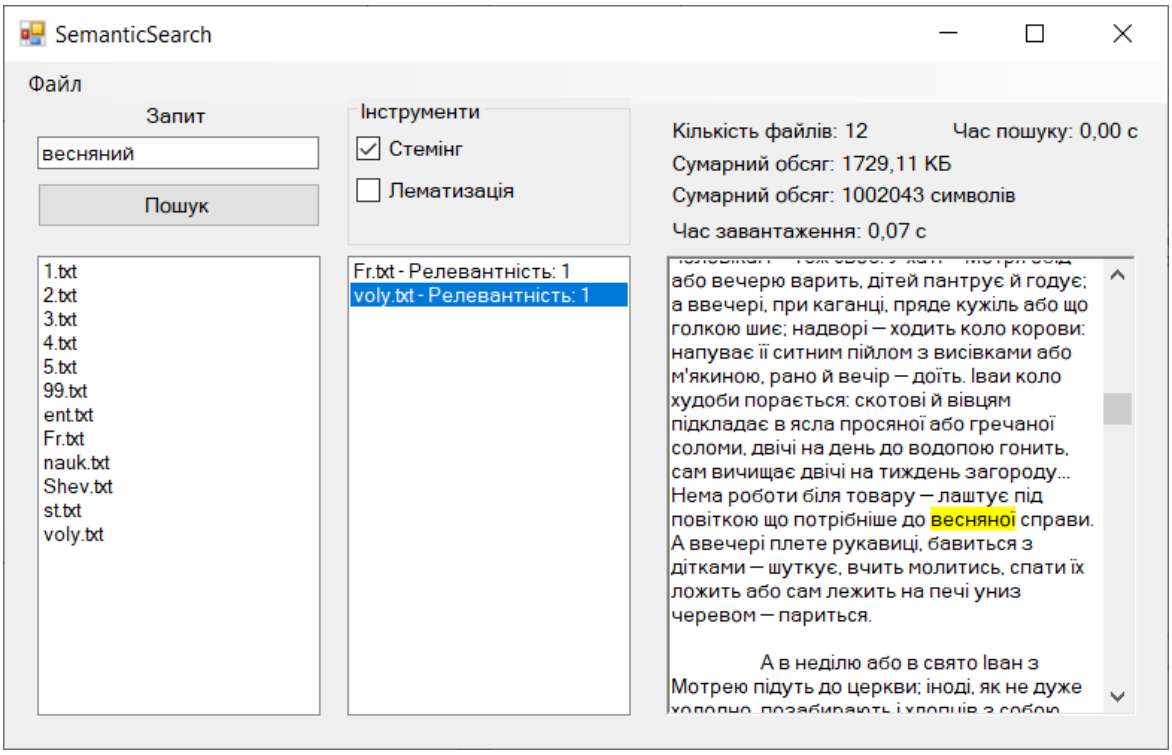

Рисунок 5.6 – Використання стемінгу

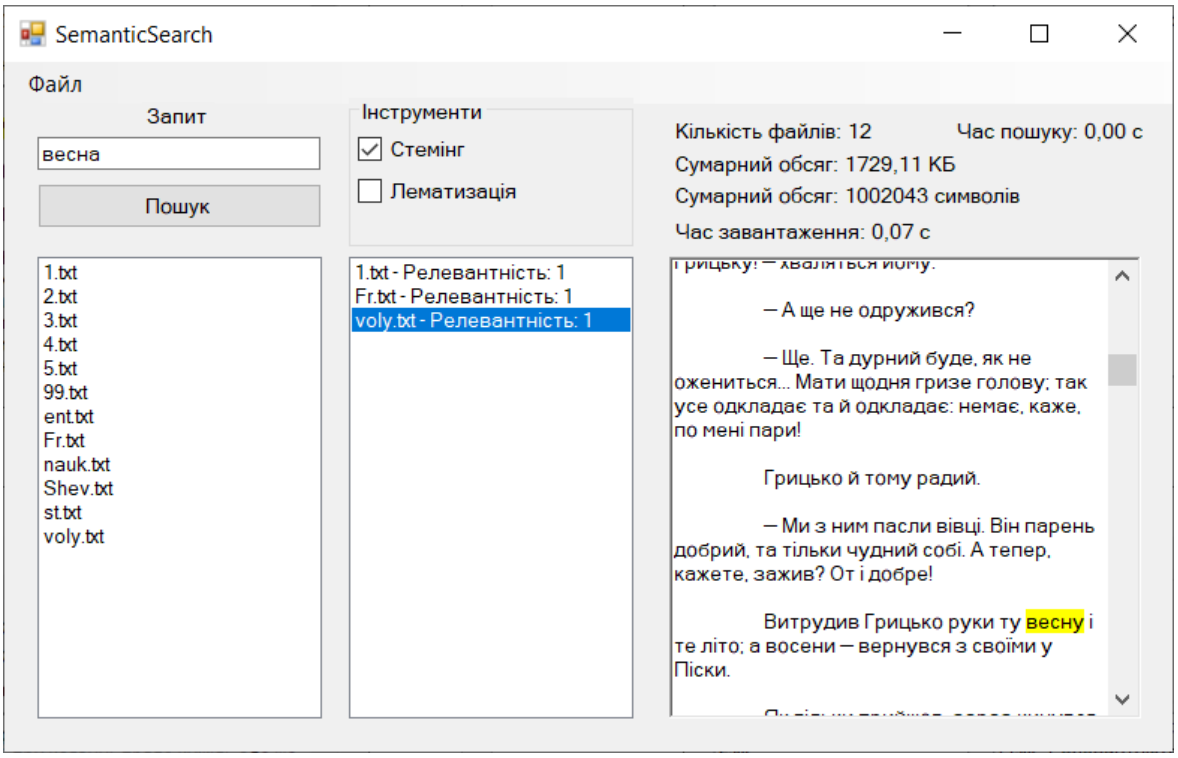

Рисунок 5.7 – Використання стемінгу

У випадку використання лематизації результат буде схожим (рисунок 5.8). В тім, лематизація демонструє кращі результати порівняно зі стемінгом з рахунок урахування морфологічної структури слів, що дає змогу приводити їх до коректних базових форм. Це дозволяє забезпечити більшу семантичну точність, оскільки леми відображають справжнє значення слів. На відміну від цього, стемінг використовує прості алгоритмічні правила для видалення суфіксів і префіксів, що може призводити до утворення неіснуючих або семантично некоректних форм.

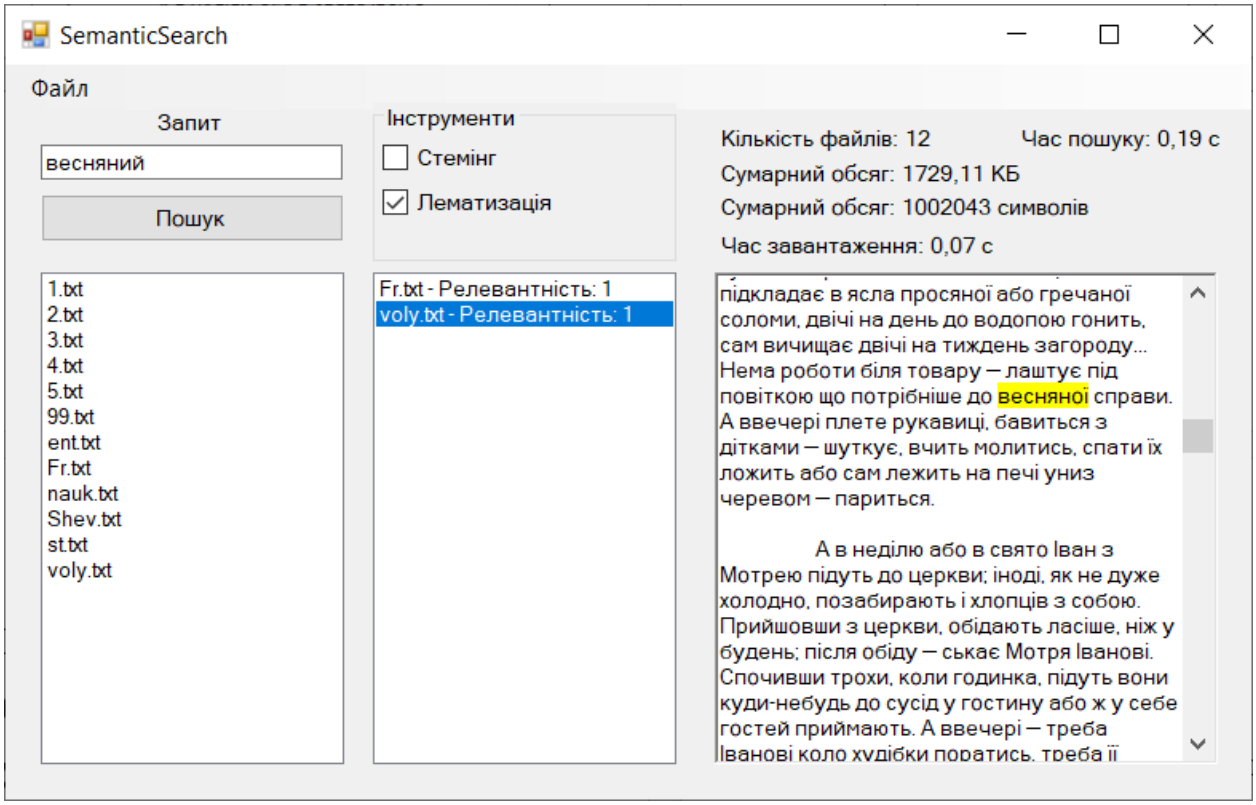

Рисунок 5.8 – Використання лематизації

Також в програмному модулі реалізовано можливість пошуку за фразами або декількома словами. При цьому в центральному лістбоксі відображається кількість співпадінь у документі з тілом пошуку (релевантність). Наприклад, на рисунку 5.9 обидва слова з пошукової строки знайдені в документі voly.txt, тому релевантність дорівнює двом.

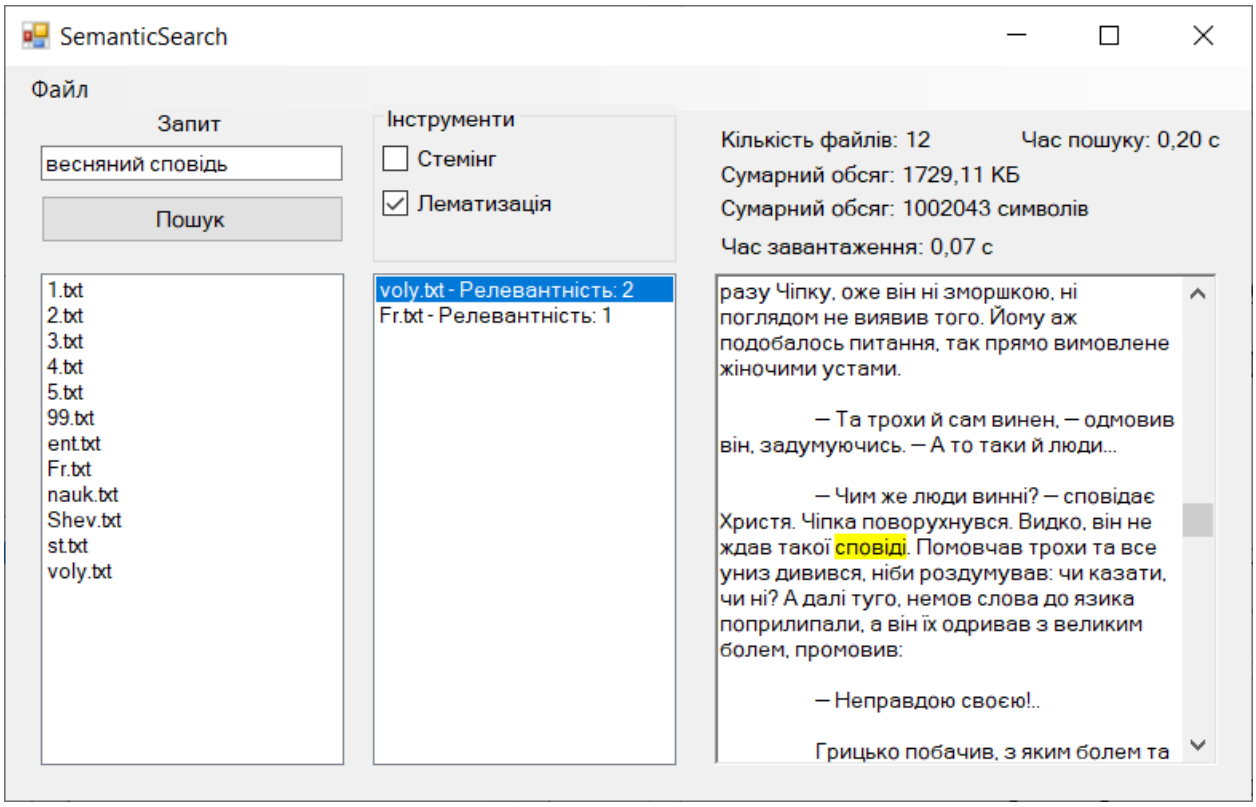

Рисунок 5.9 – Комплексний пошук з використанням лематизації

Над полем відображення тексту розміщено інформаційну панель, яка сповіщає користувача про кількість завантажених файлів, їх сумарний розмір в кілобайтах та символах, час завантаження файлів та час виконання пошукової операції.

В ході роботи було проведено дослідження залежності часу пошуку у файлах та часу їх завантаження залежно від загальної кількості символів у документах. Результати досліджень продемонстрували, що пошук з використанням лематизації потребує значно більше часу, порівняно з пошуком з використанням стемінгу. Це цілком очікуваний результат, з урахуванням того, що лематизація вимагає використання більш складних алгоритмів морфологічного аналізу і ґрунтується на словникових базах даних для точного визначення лем. Також вона включає синтаксичний аналіз і врахування контексту, що збільшує обчислювальні витрати і час обробки.

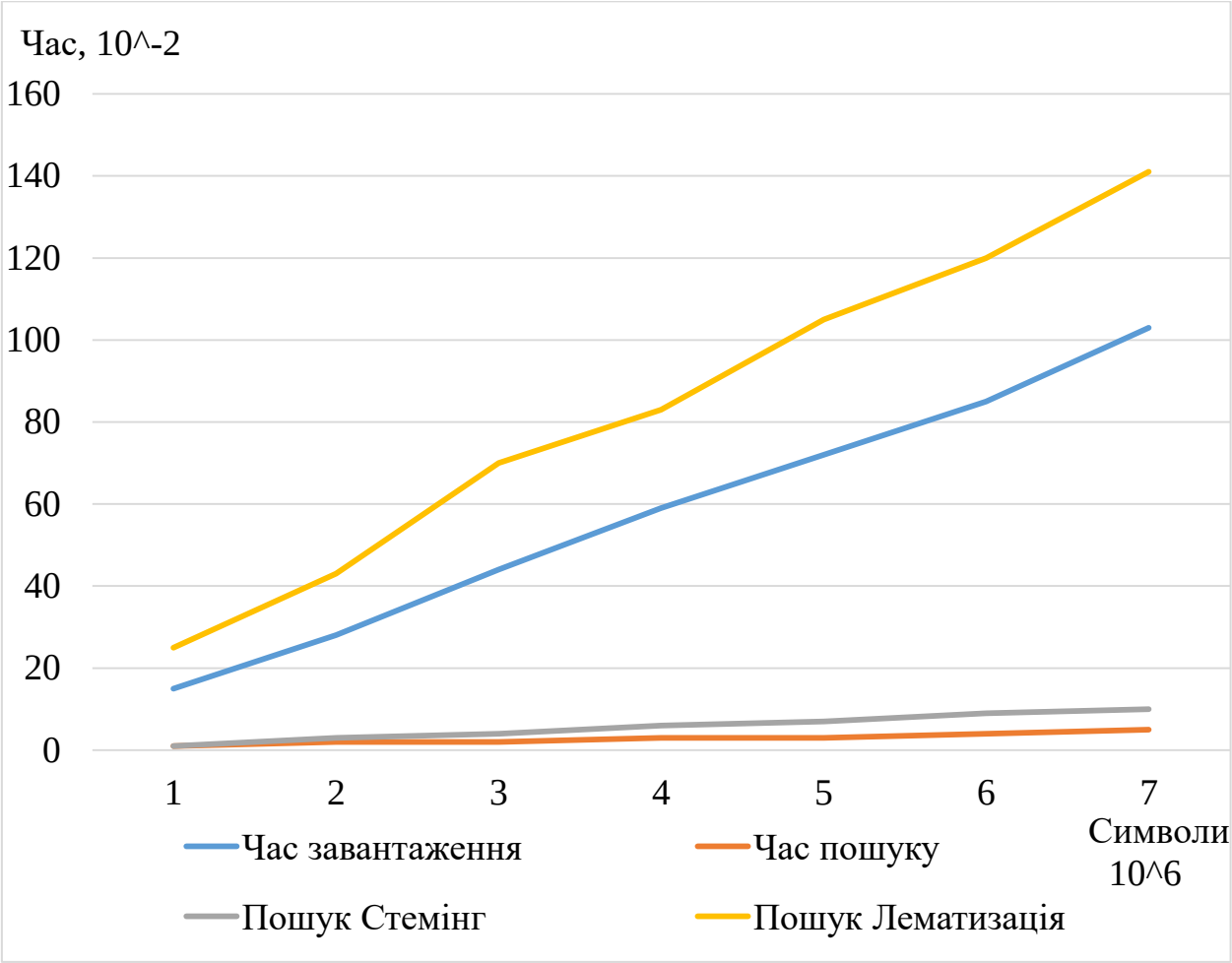

Рисунок 5.10 – Дослідження часу завантаження документів та виконання операцій пошуку залежно від кількості символів

В той же час стемінг, який використовує евристичні правила для обрізки словоформ, демонструє час пошуку, порівняний зі звичним пошуком за повним співпадінням. Його алгоритми менш складні та потребують менше обчислювальних ресурсів, що дає змогу швидко обробляти великі обсяги даних. З урахуванням того, що під час тестування пошуку за обома методами (стемінг та лематизація) були отримані порівняно однакові результати, можна зробити висновки щодо доцільності використання стемінгу при обробках великих масивів текстових даних.

Загалом, отримані залежності носять лінійний характер. Результати повторних експериментів можуть залежати від зміни конфігурації апаратної частини та поточного завантаження центрального процесора.

ВИСНОВКИ

В процесі розроблення програмного модуля семантичного інформаційного пошуку фокус був поставлений на реалізації ключових методів опрацювання природної мови, а саме стемінгу та лематизації. Ці методи показали свою високу ефективність у поліпшенні якості пошуку та підвищенні релевантності знайдених документів. Застосування стемінгу дало змогу скоротити слова до їхньої кореневої основи, що зменшило варіативність словоформ і спростило процес зіставлення пошукових запитів із вмістом документів. Це важливо для української мови, багатої на морфологічні форми та закінчення.

Лематизація, своєю чергою, забезпечила приведення слів до їхньої словникової форми, враховуючи контекст і граматичні особливості. Це дало змогу системі точніше інтерпретувати смислові зв'язки між словами та покращити якість семантичного аналізу текстів. Завдяки цьому користувачі могли знаходити документи, навіть якщо в них використовувалися різні форми одного й того самого слова, що підвищило повноту і точність пошуку.

Практичне впровадження цих методів продемонструвало їхню ефективність у розв'язанні задач семантичного пошуку. Система змогла успішно обробляти великі обсяги текстових даних, забезпечуючи високу швидкість і продуктивність.

В межах цієї дипломної роботи не було включено більш просунуті методи обробки природної мови (NLP) для семантичного пошуку. Основною причиною цього стала фактична відсутність готових реалізацій таких методів для використовуваного інструментарію розроблення на мові C# і обмеженість доступних бібліотек із підтримкою української мови. Крім того, інтеграція методів NLP вимагала б використання додаткових технологій і ресурсів, наприклад запуск окремих серверів на Python або звернення до зовнішніх API, що виходило за рамки поточних завдань і технічних можливостей проєкту.

Незважаючи на ці обмеження, виконана робота заклала міцну основу для подальшого розвитку системи. Реалізовані методи стемінга і лематизації довели свою цінність і ефективність, продемонструвавши значне поліпшення якості пошуку. Система готова до розширення функціональності, і в майбутньому можна розглянути можливість інтеграції таких методів, як розпізнавання частин мови, розбір синтаксичних залежностей, використання синонімів і семантичних векторних уявлень.

Проведена робота показала, що навіть без використання просунутих методів обробки природної мови можна домогтися істотного поліпшення якості семантичного інформаційного пошуку завдяки ефективному застосуванню стемінга і лематизації. Ці методи виявилися корисними для роботи з українською мовою. Система продемонструвала продуктивність і готовність до подальшого розвитку і можливості інтеграції нових технологій семантичного пошуку.

ПЕРЕЛІК ПОСИЛАНЬ

1. Криворізький національний університет [Електронний ресурс] – Режим доступу до ресурсу: https://www.knu.edu.ua/

2. Semantic Search over the Web / ed. by R. De Virgilio, F. Guerra, Y. Velegrakis. Berlin, Heidelberg : Springer Berlin Heidelberg, 2012. URL: https://doi.org/10.1007/978-3-642-25008-8

3. Karmakar S., Dey S., Sahoo S. Towards Semantic Image Search. 2016 IEEE Tenth International Conference on Semantic Computing (ICSC), Laguna Hills, CA, 4–6 February 2016. URL: https://doi.org/10.1109/icsc.2016.113

4. Semantic Video Search / A. W. M. Smeulders et al. 14th International Conference of Image Analysis and Processing - Workshops (ICIAPW 2007), Modena, Italy, 10–13 September 2007. URL: https://doi.org/10.1109/iciapw.2007.39

5. Bast H., Buchhold B., Haussmann E. Semantic Search on Text and Knowledge Bases. Now Publishers, 2016.

6. Understanding semantic search: the future of web browsing [Електронний ресурс] – Режим доступу до ресурсу: https://magnet.co/articles/understanding-semantic-search-the-future-of-web-browsing

7. Semantic Search Guide: What Is It And Why Does It Matter? [Електронний ресурс] – Режим доступу до ресурсу: https://www.bloomreach.com/en/blog/semantic-search-explained-in-5-minutes

8. Social network search based on semantic analysis and learning / F. Kou et al. CAAI Transactions on Intelligence Technology. 2016. Vol. 1, no. 4. P. 293–302. URL: https://doi.org/10.1016/j.trit.2016.12.001

9.Warren P., Alsmeyer D. Applying semantic technology to a digital library: a case study. Library Management. 2005. Vol. 26, no. 4/5. P. 196–205. URL: https://doi.org/10.1108/01435120510596053

10. The Elastic Search AI Platform [Електронний ресурс] – Режим доступу до ресурсу: https://www.elastic.co/

11. Search UI with Elasticsearch [Електронний ресурс] – Режим доступу до ресурсу: https://www.elastic.co/docs/current/search-ui/tutorials/elasticsearch
12. Будуємо розширений пошук з ElasticSearch [Електронний ресурс] – Режим доступу до ресурсу: https://dou.ua/lenta/columns/building-advanced-search-with-elasticsearch/
13. Discover, iterate, and resolve with ES|QL on Kibana [Електронний ресурс] – Режим доступу до ресурсу: https://www.elastic.co/kibana
14. Що таке Kibana і як цей інструмент використовується в тестуванні [Електронний ресурс] – Режим доступу до ресурсу: https://training.qatestlab.com/blog/technical-articles/what-is-kibana-and-how-to-use-this-tool/
15. Kibana — your window into Elastic [Електронний ресурс] – Режим доступу до ресурсу: https://www.elastic.co/guide/en/kibana/current/introduction.html
16. Azure AI Search [Електронний ресурс] – Режим доступу до ресурсу: https://azure.microsoft.com/en-us/products/ai-services/ai-search/
17. The best AI platform to realize the best ideas [Електронний ресурс] – Режим доступу до ресурсу: https://azure.microsoft.com/
18. Azure Cognitive Search [Електронний ресурс] – Режим доступу до ресурсу: https://www.dev4side.com/en/blog/azure-cognitive-search
19. What Is Natural Language Processing (NLP)? [Електронний ресурс] – Режим доступу до ресурсу: https://www.oracle.com/eg/artificial-intelligence/what-is-natural-language-processing/#:~:text=Natural%20language%20processing%20(NLP)%20is,natural%20language%20text%20or%20voice.
20. Welcome to Apache Lucene [Електронний ресурс] – Режим доступу до ресурсу: https://lucene.apache.org/
21. What is Cognitive Search? [Електронний ресурс] – Режим доступу до ресурсу: https://botpenguin.com/glossary/cognitive-search

22. SharePoint. Розумна інтрамережа на вашому мобільному пристрої [Електронний ресурс] – Режим доступу до ресурсу: https://www.microsoft.com/uk-ua/microsoft-365/sharepoint/collaboration

23. Apache Solr [Електронний ресурс] – Режим доступу до ресурсу: https://solr.apache.org/

24. Sphinx. Open Source Search Engine [Електронний ресурс] – Режим доступу до ресурсу: https://sphinxsearch.com/

25. Algolia Електронний ресурс] – Режим доступу до ресурсу: https://www.algolia.com/

26. Open Source Search Engine - Amazon OpenSearch Service [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/ru/opensearch-service/

27. Essentials of finite mathematics: Matrices, linear programming, probability, Markov chains / ed. by B. Brown. New York : Ardsley House Publishers, 1990. 454 p.

28. Viterbi A. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*. 1967. Vol. 13, no. 2. P. 260–269.

29. Глибовець А. М., Точицький В. В. Алгоритм токенізації та стемінгу для текстів українською мовою. *Наукові записки НаУКМА*. *Комп'ютерні науки*. 2017. Т. 198. С. 4-8.

30. Porter M. F. An algorithm for suffix stripping. *Program*. 1980. Vol. 14, no. 3. P. 130–137. URL: https://doi.org/10.1108/eb046814

31. NHunspell [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/Thomas-Maierhofer-Consulting/NHunspell

32. Hunspell [Електронний ресурс] – Режим доступу до ресурсу: https://hunspell.github.io/

Додаток А – Економічна ефективність розробки

Собівартість програмного модуля семантичного інформаційного пошуку розраховується на основі параметрів, зазначених в таблиці А.1.

Таблиця А.1 — Початкові показники для визначення собівартості

Опис	Показник
Складність розробки ПЗ (кількість календарних днів)	12
Ставка розробника ПЗ на місяць, грн	17000
Число робочих годин у місяці, год	160
Додаткова зарплата, %	10
Відсоток відрахувань до соціальних фондів	15
Відсоток загальновиробничіх витрат компанії	100
Відсоток ПДВ в Україні	20

Перед початком виконання розрахунків треба зауважити, що календарний місяць, як правило, містить в собі двадцять днів, що є робочими. З огляну на це, подальші розрахунки проводитемо з урахуванням терміну в 1 календарний місяць.

а) Визначення затрат на носії інформації для інсталяції розроблюваного модулю

$$З_k = \sum Ц_k * n_k \quad (А.1)$$

де $З_К$ — затрати на закупівлю носіїв інформації, грн;

$Ц_К$ — ціна одного носія інформації, грн;

n_k — загальна кількість потрібних носіїв, шт.

Таким чином, затрати на носії інформації для інсталяції розроблюваної програми складуть:

$$З_k = 11*10 = 110 грн \quad (А.2)$$

Затрати на закупівлю носіїв інформації складатимуть 110 грн.

б) Затрати електроенергії для розробки програмного забезпечення складуть:

$$B_E = P_E \sum W_i * t_i \quad (А.3)$$

де P_E — актуальна вартість 1 кВт на годину, грн;

Wi — середньозважена потужність, що споживалася усіми типами обладнання при розробці системи;

t_i - загальний час роботи обладнання.

Варстість одного кіловату електроенергії складає 4,32 грн.

Проведемо числові розрахунки затрат на електроенергію:

$$B_E = 4{,}32 * 0{,}3 * 160 = 207{,}36 грн \quad (А.4)$$

Таким чином, зазначені затрати складатимуть 207,36 грн.

в) Розрахунок основної заробітної плати для програмістів, що розробляють програмний модуль, проводиться наступним чином:

$$З_{осн} = l_{год} * T_{год} \quad (А.5)$$

де $l_{год}$ — тарифна ставка програміста (годинна), грн;

$T_{год}$ — кількість робочих годин програміста протягом місяця (160 годин згідно розрахунків).

При цьому тарифна ставка програміста – розробника даного продукту, складає 106,25 грн.

Таким чином, основна зарплата програміста складе:

$$З_{осн} = 106{,}25 * 160 = 17000грн \quad (А.6)$$

Отже, основна зарплата програміста складає 17000 грн.

г) Додаткова зарплата програміста обчислюється завдяки наступному виразу:

$$З_{дод} = \frac{З_{осн} * Д\%}{100} \quad (А.7)$$

де *Здод* — додаткова зарплата програміста, грн;

Д% — відсоток додаткової зарплати програміста (дорівнює 10%).

Проведемо числові розрахунки додаткової заробітної плати програміста за виразом А.8:

$$З_{дод} = \frac{17000*10}{100} = 1700грн \quad (А.8)$$

Розрахунки показують, що додаткова зарплата програміста дорівнюватиме 1700 грн.

д) Відповідні об’єми відрахувань до соціальних фондів можна визначити наступним виразом:

$$З_{соц} = \frac{(З_{осн} + З_{дод}) * С\%}{100} \quad (А.9)$$

де *Зсоц* — розміри відрахувань, грн;

С% —доля зазначених відрахувань (15%).

Проведемо числові розрахунки зазначеного параметру:

$$З_{соц} = \frac{(17000+1700)*12}{100} = 2244грн \quad (А.10)$$

Таким чином, відрахування до соціальних фондів має відповідно складати 2244 грн;

е) Загальновиробничі витрати при розробці програмного забезпечення можна визначити таким чином:

$$З_{заг} = \frac{З_{осн} * Н_1\%}{100} \quad (А.11)$$

де $З_{заг}$ — відповідно, загальновиробничі витрати, грн;

$Н_1\%$ — запланована доля витрат (100%).

Проведемо числові розрахунки загальновиробничих витрат:

$$З_{заг} = \frac{17000*100}{100} = 17000грн \quad (А.12)$$

Загальновиробничі витрати на створення програмного забезпечення становлять 17000,00 грн.

Шляхом послідовного сумування отриманих вище результатів отримуємо виробничу собівартість:

$$S_{пп} = 110 + 207,36 + 17000 + 1700 + 2244 + 17000 = 38261,36 грн \quad (А.13)$$

де $S_{пп}$ — виробнича собівартість системи, грн.

Розрахована виробнича собівартість складає 38261,36 грн.

В таблиці А.2 наведено планову калькуляцію виробничої собівартості програмного забезпечення.

Таблиця А.2 — Результати планової калькуляції

Пункти	Сума, грн.
Комплектуючі (носії інформації для інсталяції)	110
Затрати на електроенергію:	172,8
Розмір основної зарплати	17000
Розмір додаткової зарплати	1700
Обсяги відрахуваннь в соціальні фонди	2244
Витрати підприємства (загальновиробничі)	17000
Виробнича собівартість (1 місяць)	38261,36
Собівартість розробки системи оцінки ефективності роботи співробітників підприємства	22956,82

Створення програмного модуля семантичного інформаційного пошуку зайняло 12 робочих днів. Відповідно, собівартість створеного ПЗ дорівнює 22956,82 грн. На основі зазначених результатів можна виконати розрахунок економічного ефекту від впровадження програмного забезпечення.

Обчислення економічного ефекту від впровадження в роботу зазначеного модулю є дещо приблизним через складність вибору факторів для оцінки, але можна розглянути економічну ефективність як комбінацію таких факторів:

– оціночній вартості часу роботи користувача програмного модулю;

– збереження часу при використанні програмного забезпечення для семантичного інформаційного пошуку порівняно з іншими методами;

– оціночного числа людей, що будуть використовувати зазначений програмний модуль.

Таким чином, економічний ефект від впровадження програмного забезпечення прийматиме вигляд виразу

$$E = Value * Economy * Count \quad \text{(А.14)}$$

де *Value* – оціночна вартість часу роботи користувача програмного модулю;

Economy – збереження часу при використанні програмного модулю;

Count – оціночна кількість людей, що будуть використовувати зазначений програмний модуль.

Розглянемо запропоновані числові значення для розрахунку економічної ефективності (таблиця А.3).

Розрахуємо економічну ефективність згідно запропонованих табличних значень:

$$E = 21000 * 0{,}1 * 10 = 21000 \text{ грн/місяць} \quad \text{(А.15)}$$

Таблиця А.3 — Пропоновані числові значення для виконання розрахунків

Параметр	**Значення**
Value	7000 грн
Economy	0,2 (20%)
Count	10

Таким чином, економічна ефективність від впровадження програмного забезпечення складе 21000 грн/місяць.

Термін окупності програмного модуля семантичного інформаційного пошуку розраховуємо згідно виразу:

$$T = \frac{C}{E_{еф}} \quad \text{(А.16)}$$

де T – строк окупності програмного забезпечення, міс,

C – собівартість системи, грн,

$Е_{еф}$ – економічний ефект протягом місяця, грн.

Проведемо числові розрахунки терміну окупності:

$$T = 28751{,}1/21000 = 1{,}37 \text{ місяця} \quad \text{(А.17)}$$

Отже, термін окупності програмного модуля семантичного інформаційного пошуку складає приблизно 1,37 місяця, що дозволяє вважати доцільною розробку даного програмного продукту.

Додаток Б - Програмний код

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System.Threading.Tasks;
using System.Windows.Forms;
using NHunspell;


namespace SemanticSearch
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {

        }
        //step3
        private                                                 void
відкритиДиректоріюToolStripMenuItem_Click(object          sender,
EventArgs e)
        {
            using (FolderBrowserDialog folderBrowserDialog = new
FolderBrowserDialog())
            {
                if      (folderBrowserDialog.ShowDialog()      ==
DialogResult.OK)
                {
                    string             selectedPath             =
folderBrowserDialog.SelectedPath;

                    // Вызываем метод для загрузки файлов из
выбранной директории
                    LoadFilesFromDirectory(selectedPath);
                }
            }
        }
```

```
        // Список для хранения документов
        List<Document> documents = new List<Document>();

        // Словник для збереження релевантності документів
        Dictionary<int, int> docRelevance = new Dictionary<int,
int>();

        private void LoadFilesFromDirectory(string directoryPath)
        {
            // Початок вимірювання часу
            Stopwatch stopwatch = new Stopwatch();
            stopwatch.Start();

            // Ініціалізація змінних для статистики
            int totalFiles = 0;
            long totalSizeBytes = 0;
            long totalChars = 0;

            // Очищуємо попередні дані
            documents.Clear();
            listBoxResult.Items.Clear();
            listBoxFiles.Items.Clear();

            // Отримуємо всі файли з розширенням .txt у вибраній
папці
            string[] files = Directory.GetFiles(directoryPath,
"*.txt");

            totalFiles = files.Length;

            foreach (string file in files)
            {
                try
                {
                    // Читаємо вміст файлу
                    string content = File.ReadAllText(file);

                    // Оновлюємо статистику
                    FileInfo fileInfo = new FileInfo(file);
                    totalSizeBytes += fileInfo.Length;
                    totalChars += content.Length;

                    // Створюємо об'єкт Document та додаємо його
до списку
                    Document doc = new Document
                    {
                        FilePath = file,
                        Content = content
                    };

                    documents.Add(doc);
```

```
                    // Додаємо ім'я файлу до listBoxFiles
                    string fileName = Path.GetFileName(file);
                    listBoxFiles.Items.Add(fileName);
                }
                catch (Exception ex)
                {
                    MessageBox.Show($"Помилка при читанні файлу
{file}: {ex.Message}");
                }
            }

            if (documents.Count > 0)
            {
                // Побудова зворотного індексу після завантаження
файлів
                BuildInvertedIndex();
            }

            // Зупинка вимірювання часу
            stopwatch.Stop();
            TimeSpan loadTime = stopwatch.Elapsed;

            // Відображення статистики
            labelFileCount.Text     =     $"Кількість     файлів:
{totalFiles}";
            labelTotalSizeKB.Text     =     $"Сумарний     обсяг:
{totalSizeBytes / 1024.0:F2} КБ";
            labelTotalChars.Text = $"Сумарний обсяг: {totalChars}
символів";
            labelLoadTime.Text     =     $"Час     завантаження:
{loadTime.TotalSeconds:F2} с";

            if (documents.Count > 0)
            {
                MessageBox.Show("Файли  успішно  завантажені  та
оброблені.");
            }
            else
            {
                MessageBox.Show("У  вибраній  директорії  немає
текстових файлів.");
            }
        }


        public class Document
        {
            public string FilePath { get; set; }
            public string Content { get; set; }
            public List<string> Tokens { get; set; }
        }
```

```
        // Обратный индекс: слово -> список ID документов
        Dictionary<string, List<int>> invertedIndex = new
Dictionary<string, List<int>>();

        private void BuildInvertedIndex()
        {
            invertedIndex.Clear();

            for (int i = 0; i < documents.Count; i++)
            {
                Document doc = documents[i];

                // Попередня обробка тексту документа
                List<string> originalTokens;
                doc.Tokens = PreprocessText(doc.Content, out
originalTokens);

                // Побудова зворотного індексу
                foreach (string token in doc.Tokens)
                {
                    if (invertedIndex.ContainsKey(token))
                    {
                        if (!invertedIndex[token].Contains(i))
                        {
                            invertedIndex[token].Add(i);
                        }
                    }
                    else
                    {
                        invertedIndex[token] = new List<int> { i
};
                    }
                }
            }
        }

        private UkrainianLemmatizer lemmatizer = new
UkrainianLemmatizer();

        private List<string> queryTokens = new List<string>();
        private List<string> originalQueryTokens = new
List<string>();


        private List<string> PreprocessText(string text, out
List<string> originalTokens)
        {
            // Приведення тексту до нижнього регістру
            text = text.ToLower();

            // Видалення пунктуації та спеціальних символів
```

```
            text = Regex.Replace(text, "[^\\p{L}\\p{Nd}\\s]",
"");

            // Розбиття тексту на слова (токенізація)
            string[] tokens = text.Split(new char[] { ' ', '\t',
'\n', '\r' }, StringSplitOptions.RemoveEmptyEntries);

            originalTokens = tokens.ToList(); // Зберігаємо
оригінальні токени

            List<string> processedTokens = tokens.ToList();

            // Лематизація
            if (checkBoxLemmatization.Checked)
            {
                UkrainianLemmatizer lemmatizer = new
UkrainianLemmatizer();
                processedTokens = processedTokens.Select(token =>
lemmatizer.Lemmatize(token)).ToList();
            }

            // Стемінг
            if (checkBoxStemming.Checked)
            {
                UkrainianStemmer stemmer = new
UkrainianStemmer();
                processedTokens = processedTokens.Select(token =>
stemmer.Stem(token)).ToList();
            }

            return processedTokens;
        }

        //step4
        private void buttonSearch_Click(object sender, EventArgs
e)
        {
            // Початок вимірювання часу
            Stopwatch stopwatch = new Stopwatch();
            stopwatch.Start();

            // Отримуємо пошуковий запит з текстового поля
            string query = textBoxSearch.Text;

            if (string.IsNullOrWhiteSpace(query))
            {
                MessageBox.Show("Будь ласка, введіть пошуковий
запит.");
                return;
            }

            // Виконуємо пошук
```

```
            List<int> resultDocIds = Search(query);

            // Зупинка вимірювання часу
            stopwatch.Stop();
            TimeSpan searchTime = stopwatch.Elapsed;

            // Відображення часу пошуку
            labelSearchTime.Text = $"Час пошуку:
{searchTime.TotalSeconds:F2} с";

            // Відображаємо результати
            DisplayResults(resultDocIds);
        }

        private List<int> Search(string query)
        {
            // Попередня обробка запиту
            List<string> originalQueryTokens;
            queryTokens = PreprocessText(query, out
originalQueryTokens);

            // Очищаємо попередні дані про релевантність
            docRelevance.Clear();

            // Пошук документів
            foreach (string token in queryTokens)
            {
                if (invertedIndex.ContainsKey(token))
                {
                    foreach (int docId in invertedIndex[token])
                    {
                        if (docRelevance.ContainsKey(docId))
                        {
                            docRelevance[docId]++;
                        }
                        else
                        {
                            docRelevance[docId] = 1;
                        }
                    }
                }
            }

            // Сортуємо результати за релевантністю
            List<int> resultDocIds =
docRelevance.OrderByDescending(kv => kv.Value).Select(kv =>
kv.Key).ToList();

            return resultDocIds;
        }
```

```
        private void DisplayResults(List<int> docIds) //from step6
        {
            // Очищаємо попередні результати
            listBoxResult.Items.Clear();

            if (docIds.Count == 0)
            {
                listBoxResult.Items.Add("Нічого не знайдено.");
            }
            else
            {
                foreach (int docId in docIds)
                {
                    Document doc = documents[docId];

                    //  Відображаємо  назву  файлу  та  кількість
співпадінь
                    string               fileName               =
Path.GetFileName(doc.FilePath);
                    string   displayText   =   $"{fileName}   -
Релевантність: {docRelevance[docId]}";

                    listBoxResult.Items.Add(displayText);
                }
            }
        }




        private  void  listBoxResult_DoubleClick(object  sender,
EventArgs e) //from step5
        {
            if (listBoxResult.SelectedIndex != -1)
            {
                string             selectedFileName             =
listBoxResult.SelectedItem.ToString();

                // Перевіряємо, чи є результати
                if (selectedFileName == "Нічого не знайдено.")
                {
                    return;
                }

                // Знаходимо документ за ім'ям файлу
                Document               selectedDoc               =
documents.FirstOrDefault(doc => Path.GetFileName(doc.FilePath) ==
selectedFileName);

                if (selectedDoc != null)
                {
```

```
                    try
                    {
                        // Відкриваємо файл у стандартному
текстовому редакторі
                        Process.Start(selectedDoc.FilePath);
                    }
                    catch (Exception ex)
                    {
                        MessageBox.Show($"Не вдалося відкрити
файл: {ex.Message}");
                    }
                }
            }
        }

        //step6
        private void listBoxResult_SelectedIndexChanged(object
sender, EventArgs e)
        {
            if (listBoxResult.SelectedIndex != -1)
            {
                string selectedItem =
listBoxResult.SelectedItem.ToString();

                // Перевіряємо, чи є результати
                if (selectedItem == "Нічого не знайдено.")
                {
                    richTextBoxPreview.Clear();
                    return;
                }

                // Отримуємо ім'я файлу без релевантності
                string fileName = selectedItem.Split('-
')[0].Trim();

                // Знаходимо документ за ім'ям файлу
                Document selectedDoc =
documents.FirstOrDefault(doc => Path.GetFileName(doc.FilePath) ==
fileName);

                if (selectedDoc != null)
                {
                    // Отображаем фрагмент текста с выделением
                    DisplayDocumentPreview(selectedDoc);
                }
            }
        }

        private void DisplayDocumentPreview(Document doc)
        {
            // Очищаємо попередній текст
            richTextBoxPreview.Clear();
```

```
            // Отримуємо текст документа
            string previewText = doc.Content.Replace("\r\n",
"\n");

            // Відображаємо текст у RichTextBox
            richTextBoxPreview.Text = previewText;

            // Підсвічуємо знайдені слова з урахуванням стемінгу
та лематизації
            HighlightQueryWordsInText(richTextBoxPreview,
previewText);
        }


        public class WordPosition
        {
            public string Word { get; set; }
            public int StartIndex { get; set; }
            public int Length { get; set; }
        }

        private void HighlightQueryWordsInText(RichTextBox
richTextBox, string text)
        {
            // Зберігаємо поточні позиції курсору
            int selectionStart = richTextBox.SelectionStart;
            int selectionLength = richTextBox.SelectionLength;

            // Замінюємо \r\n на \n
            string lowerText = text.Replace("\r\n",
"\n").ToLower();

            // Створюємо екземпляри лематизатора та стемера
            UkrainianLemmatizer lemmatizer = new
UkrainianLemmatizer();
            UkrainianStemmer stemmer = new UkrainianStemmer();

            // Токенізуємо текст з відображенням позицій кожного
слова
            List<WordPosition> wordsWithPositions = new
List<WordPosition>();
            foreach (Match match in Regex.Matches(lowerText,
@"\b\w+\b"))
            {
                wordsWithPositions.Add(new WordPosition
                {
                    Word = match.Value,
                    StartIndex = match.Index,
                    Length = match.Length
                });
            }
```

```
            // Обробляємо слова з тексту відповідно до налаштувань
            List<WordPosition> processedWordsWithPositions = new
List<WordPosition>();
            foreach (var wp in wordsWithPositions)
            {
                string processedWord = wp.Word;

                // Лематизація
                if (checkBoxLemmatization.Checked)
                {
                    processedWord                                    =
lemmatizer.Lemmatize(processedWord);
                }

                // Стемінг
                if (checkBoxStemming.Checked)
                {
                    processedWord = stemmer.Stem(processedWord);
                }

                processedWordsWithPositions.Add(new WordPosition
                {
                    Word = processedWord,
                    StartIndex = wp.StartIndex,
                    Length = wp.Length
                });
            }

            // Підсвічуємо слова, які відповідають обробленим
токенам запиту
            foreach (var wp in processedWordsWithPositions)
            {
                if (queryTokens.Contains(wp.Word))
                {
                    richTextBox.Select(wp.StartIndex, wp.Length);
                    richTextBox.SelectionBackColor                =
Color.Yellow;
                }
            }

            // Відновлюємо попередні позиції курсору
            richTextBox.SelectionStart = selectionStart;
            richTextBox.SelectionLength = selectionLength;
        }

        private void HighlightQueryWords(RichTextBox richTextBox,
List<string> words)
        {
            // Зберігаємо поточні позиції курсору
            int selectionStart = richTextBox.SelectionStart;
            int selectionLength = richTextBox.SelectionLength;
```

```
            // Переводимо текст у нижній регістр для пошуку
            string text = richTextBox.Text.ToLower();

            foreach (string word in words)
            {
                int startIndex = 0;
                while    ((startIndex    =    text.IndexOf(word,
startIndex)) != -1)
                {
                    // Виділяємо слово
                    richTextBox.Select(startIndex, word.Length);
                    richTextBox.SelectionBackColor              =
Color.Yellow;

                    startIndex += word.Length;
                }
            }

            // Відновлюємо попередні позиції курсору
            richTextBox.SelectionStart = selectionStart;
            richTextBox.SelectionLength = selectionLength;
        }

        private    void    checkBoxStemming_CheckedChanged(object
sender, EventArgs e)
        {
            if (documents.Count > 0)
            {
                // Перебудовуємо зворотний індекс
                BuildInvertedIndex();
            }
        }

        private void checkBoxLemmatization_CheckedChanged(object
sender, EventArgs e)
        {
            if (documents.Count > 0)
            {
                // Перебудовуємо зворотний індекс
                BuildInvertedIndex();
            }
        }

        //стімер
        public class UkrainianStemmer
        {
            private static readonly string VOWEL = "аеиоуюяіїє";
            private static readonly Regex PERFECTIVEGROUND = new
Regex("((ив|ивши|ившись|ыв|ывши|ывшись))$",
RegexOptions.Compiled);
```

```
            private static readonly Regex REFLEXIVE = new
Regex("(с[яьи])$", RegexOptions.Compiled);
            private static readonly Regex ADJECTIVE = new
Regex("(ими|ій|ий|а|е|ова|ове|ів|є|їй|єє|еє|я|ій|ім|им|их|іх|ою|
ою|йми|іми|у|ю|ого|ому|оїї)$", RegexOptions.Compiled);
            private static readonly Regex PARTICIPLE = new
Regex("(ий|ого|ому|им|ім|а|ій|у|ою|і|их|йми|іми)$",
RegexOptions.Compiled);
            private static readonly Regex VERB = new
Regex("(сь|ся|ив|ивши|ився|ившися|ать|ять|уть|ють|у|ю|ав|али|ала
|ало|ати|яти|авши|авшись|яла|яли|яло|є)$",
RegexOptions.Compiled);
            private static readonly Regex NOUN = new
Regex("(а|ев|ов|е|є|иями|ями|ами|еи|и|ей|ий|й|иям|ям|ием|ем|ам|о
м|о|у|ах|ях|и|ь|ию|ью|ю|ія|ья|я|і|ї|їв|ів)$",
RegexOptions.Compiled);
            private static readonly Regex RVRE = new
Regex("^(.*?[аеиоуюяіїє])(.*)$", RegexOptions.Compiled);
            private static readonly Regex DERIVATIONAL = new
Regex(".*[^аеиоуюяіїє][аеиоуюяіїє]+[^аеиоуюяіїє]+[аеиоуюяіїє].*о
сть?$", RegexOptions.Compiled);
            private static readonly Regex SUPERLATIVE = new
Regex("(най|найменш|найбільш)$", RegexOptions.Compiled);
            private static readonly Regex I = new Regex("і$",
RegexOptions.Compiled);
            private static readonly Regex P = new Regex("ь$",
RegexOptions.Compiled);
            private static readonly Regex NN = new Regex("нн$",
RegexOptions.Compiled);

            public string Stem(string word)
            {
                word = word.ToLower();

                // Замінюємо апострофи та інші спеціальні символи
                word = word.Replace("’", "").Replace("'", "");

                Match rvMatch = RVRE.Match(word);
                if (rvMatch.Success)
                {
                    string prefix = rvMatch.Groups[1].Value;
                    string rv = rvMatch.Groups[2].Value;

                    // Step 1
                    if  (!TryReplace(ref  rv,  PERFECTIVEGROUND,
""))
                    {
                        TryReplace(ref rv, REFLEXIVE, "");

                        if (TryReplace(ref rv, ADJECTIVE, ""))
                        {
                            TryReplace(ref rv, PARTICIPLE, "");
```

```
                        }
                        else
                        {
                            if (!TryReplace(ref rv, VERB, ""))
                            {
                                TryReplace(ref rv, NOUN, "");
                            }
                        }
                    }

                    // Step 2
                    TryReplace(ref rv, I, "");

                    // Step 3
                    if (DERIVATIONAL.IsMatch(rv))
                    {
                        TryReplace(ref rv, new Regex("ость?$",
RegexOptions.Compiled), "");
                    }

                    // Step 4
                    if (!TryReplace(ref rv, P, ""))
                    {
                        TryReplace(ref rv, SUPERLATIVE, "");
                        TryReplace(ref rv, NN, "н");
                    }

                    word = prefix + rv;
                }

                return word;
            }

            private bool TryReplace(ref string original, Regex
regex, string replacement)
            {
                string result = regex.Replace(original,
replacement);
                bool replaced = result != original;
                original = result;
                return replaced;
            }
        }

        //lemma
        public class UkrainianLemmatizer
        {
            private readonly Hunspell hunspell;

            public UkrainianLemmatizer()
            {
                string affFile = @"Dictionaries\uk_UA.aff";
```

```
            string dicFile = @"Dictionaries\uk_UA.dic";

            hunspell = new Hunspell(affFile, dicFile);
        }

        public string Lemmatize(string word)
        {
            // Отримуємо базові форми слова
            List<string> stems = hunspell.Stem(word);

            // Якщо основи знайдено, повертаємо першу
            if (stems != null && stems.Count > 0)
            {
                return stems[0];
            }

            // Інакше повертаємо оригінальне слово
            return word;
        }
    }
}
```