

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ
ТЕХНОЛОГІЇ WEB MINING ДЛЯ ПОТРЕБ SEO
RESEARCH AND SOFTWARE IMPLEMENTATION OF A
WEB MINING TECHNOLOGY FOR SEO NEEDS

Виконав(ла): студент(ка) 2 курсу, групи КНТ-222м
Спеціальності 122 Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Системи штучного інтелекту

ЛІНЕЦЬ В.А.

(ПРИЗВИЩЕ та ініціали)

Керівник КОЛПАКОВА Т.О.

(ПРИЗВИЩЕ та ініціали)

Рецензент ГОЛУБ Т.В.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Факультет КНТ
 Кафедра програмних засобів
 Ступінь вищої освіти магістр
 Спеціальність 122 Комп'ютерні науки
(код і найменування)
 Освітня програма (спеціалізація) Системи штучного інтелекту
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
 “ ” 2023 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ЛІНЦЯ Вадима Андрійовича
(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація технології web mining для потреб SEO. Research and Software Implementation of a Web Mining Technology for SEO Needs

керівник проєкту (роботи) к.т.н., доцент, КОЛПАКОВА Тетяна Олексіївна,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “ 14 ” листопада 2023 року № 437

2. Строк подання студентом проєкту (роботи) 01 грудня 2023 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження. 2. Матеріали і методи. 3. Опис програми. 4. Розробка програми 5. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	КОЛПАКОВА Т.О., доцент		
Нормоконтроль	ЛИПОВЕЦЬ М.В., асистент		

7. Дата видачі завдання «08» вересня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1-2 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1-2 тиждень	Розділ 1, 2
3	Вибір мови програмування та інших технологій розробки.	3 тиждень	Розділ 3
4	Розробка архітектури програми.	4-5 тижні	Розділ 3
5	Розробка програми.	6-8 тижні	Розділ 4, 5
6	Тестування та експериментальне дослідження програмного забезпечення.	9 тиждень	Розділ 5
7	Оформлення пояснювальної записки та документів до неї.	10 тиждень	Додатки
8	Нормоконтроль та рецензування.	11 тиждень	
9	Захист роботи.	12 тиждень	

Студент(ка)

Підпис
(підпис)

Вадим ЛІНЕЦЬ
(Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

Тетяна КОЛПАКОВА
(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
116 с., 15 табл., 45 рис., 3 дод., 35 джерел.

WEB MINING, SEO, ВЕБЗАСТОСУНОК, GOLANG, GO, VSCODE, JAVASCRIPT, REACT.

Об'єкт дослідження – процес отримання даних, що потрібні для SEO, з сайтів, що вказує користувач, за допомогою технології web mining.

Предмет дослідження – вебзастосунки та десктопні програми для отримання технічних даних для SEO.

Мета роботи – створення програмного забезпечення для надання можливості отримувати дані для SEO за допомогою технології web mining.

Матеріали, методи та технічні засоби: структурне та об'єктно-орієнтоване програмування, мови програмування Golang та JavaScript, інтегровані середовища розробки програм Microsoft Visual Studio Code, персональний IBM-сумісний комп'ютер з процесором AMD Ryzen 5 5500U, оперативною пам'яттю 32 Гб, 15.6'' монітором, клавіатурою та маніпулятором типу «миша», що функціонує під управлінням операційної системи Ubuntu 23.04 x86_64.

Результати. Створено програмний вебзастосунок, який надає можливість відправити запит на обробку вебсторінки, та отримати всю інформацію, яка буде корисна для цілей SEO, по цій сторінці.

Висновки. Розроблено вебзастосунок який надає можливість отримувати дані для SEO за допомогою технології web mining.

Галузь використання – подібну програму можна використовувати як окрему програму, що запускається для надання можливості передивлятися SEO дані вебсторінок, що цікавлять користувача.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 116 pages, 15 tables, 45 figures, 3 appendixes, 35 sources.

WEB MINING, SEO, WEB APPLICATIONS, GOLANG, GO, VSCODE, JAVASCRIPT, REACT.

The object of the study is the process of obtaining data required for SEO from the sites indicated by the user, with the help of web mining technology.

The subject of research is web applications and desktop applications for obtaining technical data for SEO.

The goal of the work is to create software to provide the ability to obtain data for SEO using web mining technology.

Materials, methods and technical means: structural and object-oriented programming, Golang and JavaScript programming languages, integrated Microsoft Visual Studio Code program development environments, personal IBM-compatible computer with an AMD Ryzen 5 5500U processor, RAM 32 GB, 15.6" monitor, keyboard and mouse type manipulator, operating under Ubuntu 23.04 x86_64 operating system.

The results. A software web application has been created that provides an opportunity to send a request for the processing of a web page and receive all the information that will be useful for SEO purposes on this page.

Conclusions. A web application has been developed that provides an opportunity to receive data for SEO using web mining technology.

Field of Use – A program like this can be used as a stand-alone program that runs to provide the ability to review SEO data of web pages of interest to the user.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	8
Вступ.....	9
1 Аналіз предметної області.....	10
1.1 Дослідження особливостей технології web mining.....	10
1.2 Дослідження особливостей SEO.....	18
1.3 Аналіз програмного забезпечення.....	26
1.4 Висновки за розділом 1 та постановка задачі.....	36
2 Матеріали і методи.....	38
2.1 Підходи реалізації.....	38
2.2 Дослідження особливостей технології PageRank.....	41
2.3 Дослідження особливостей технології TextRank.....	43
2.4 Дослідження особливостей технології whois.....	45
2.5 Висновки за розділом 2.....	46
3 Розробка архітектури програми.....	47
3.1 Аналіз вимог.....	47
3.2 Архітектура.....	48
3.3 Вибір інструментарію розробки.....	50
3.4 Висновки за розділом 3.....	57
4 Основні рішення щодо реалізації компонентів системи.....	58
4.1 Розробка алгоритму функціонування програми.....	58
4.2 Структура підсистеми.....	61
4.3 Опис особливостей програмної реалізації.....	62
4.4 Висновки за розділом 4.....	65

5 Експлуатація, тестування та експериментальне дослідження програми.....	66
5.1 Опис застосування.....	66
5.2 Інструкція з експлуатації програми.....	66
5.3 Висновки за розділом 5.....	76
Висновки.....	77
Перелік джерел посилань.....	78
Додаток А Технічне завдання.....	82
Додаток Б Текст програми.....	86
Додаток В Слайди презентації.....	109

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

- DOM – Document Object Model;
- HTML – HyperText Markup Language;
- ICANN – Internet Corporation for Assigned Names and Numbers;
- IDE – Integrated Development Environment;
- IP – Internet Protocol;
- JSON – JavaScript Object Notation;
- ML – Machine Learning;
- NN – Neural Network;
- SEO – Search Engine Optimization;
- SQL – Structured Query Language;
- TCP – Transmission Control Protocol;
- URL – Uniform Resource Locator;
- VPN – Virtual Private Network;
- XML – Extensible Markup Language;
- БД – база даних;
- СКБД – система керування базами даних.

ВСТУП

Сучасний інтернет є найбільшою інформаційною базою у світі, де щоденно публікується безліч вебсторінок зі статтями, інтернет-магазинами, відео та іншими ресурсами. Завдяки цьому великому обсягу даних, важливим завданням стає ефективне управління та оптимізація вебпростору, зокрема для досягнення кращого рейтингу в пошукових системах (SEO). Одним із перспективних напрямків у цьому контексті є використання технології web mining.

Web mining – це сфера досліджень, спрямованих на аналіз вебданих та видобуток корисної інформації з них. Ця технологія має великий потенціал для вдосконалення SEO-стратегій та підвищення вірогідності, що саме ваш вебресурс користувач отримає першим в різних пошукових системах. Використання технології web mining для потреб SEO є актуальною проблемою, оскільки дані, що отримані завдяки цьому, можуть бути спрямовані на покращення показників пошукової оптимізації, а отже й підвищення рейтингу сайту.

Цей дипломний проєкт спрямований на дослідження різних аспектів web mining для SEO, а також на розробку програмних інструментів та методів для збору, обробки та аналізу вебданих. Дослідження в даній області має важливий практичний вимір, оскільки допоможе вдосконалити методики SEO та сприяти покращенню рейтингу сайтів у пошукових системах.

Метою даного проєкту є розробка та впровадження технології web mining для підвищення SEO-ефективності. Цей проєкт є актуальним і важливим кроком у розвитку сучасних методів вебаналітики та пошукової оптимізації й спрямований на підтримку веброзробників та маркетологів у досягненні вищих показників ефективності в онлайн-середовищі.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Дослідження особливостей технології web mining

Web mining – це термін застосування методів інтелектуального аналізу даних для автоматичного виявлення та вилучення корисної інформації з документів і служб Всесвітньої павутини. Хоча web mining пускає коріння глибоко в інтелектуальний аналіз даних, він не еквівалентний інтелектуальному аналізу даних. Дослідження web mining – це фактично об'єднана область кількох дослідницьких спільнот, таких як бази даних, пошук інформації, штучний інтелект, а також психологія та статистика [1].

Основна задача web mining полягає у видобуванні інформації з сайтів та/або з вебжурналів сайтів, її аналізу та збереженні у базу даних. Загалом весь процес web mining можна поділити на 4 етапи, що продемонстровано на рисунку 1.1 [1]:

- виявлення ресурсів: завдання отримання потрібної інформації з Інтернету;
- вилучення інформації: автоматичний вибір і попередня обробка конкретної інформації з отриманих вебресурсів;
- узагальнення: автоматично виявляє загальні шаблони як на окремих вебсайтах, так і на кількох сайтах;
- аналіз: аналіз отриманого шаблону.



Рисунок 1.1 – Етапи web mining [1]

Виходячи з основних типів місць, з котрих ми отримуємо інформацію, web mining можна умовно поділити на три типи: Web Structure Mining, Web Content Mining та Web Usage Mining, як показано на рисунку 1.2 [1].



Рисунок 1.2 – Типи web mining [1]

1.1.1 Web Usage Mining

Web Usage Mining — це застосування методів інтелектуального аналізу даних для виявлення цікавих моделей використання з вебданих, щоб зрозуміти та краще задовольнити потреби вебдодатків [1]. Він намагається знайти цінну інформацію з вторинних даних, які виникають внаслідок взаємодії користувачів у мережі Інтернет з вебсайтами. Ці дані містять в собі інформацію щодо ідентичності, походження вебкористувачів, а також їх поведінку під час перегляду вебсайтів. Він зосереджується на аналізі даних, що накопичуються під час сеансів та активності в мережі. До цих даних відносяться інформація з журналів проксі-серверів, журналів браузера, журналів доступу до вебсерверів, профілів користувачів, сесій користувачів, запитів користувачів, даних реєстрації, закладок, дій користувачів (наприклад, клацань мишею та прокручування), файли cookie та інші дані, які виникають внаслідок їхньої взаємодії з Інтернетом [2].

Інформація, яка міститься в журналах, залежить від формату журналу і встановлених параметрів на вебсервері. Аналіз вебжурналів може бути корисним для керування корпоративним електронним бізнесом на короткостроковій основі, але справжня цінність цих даних проявляється тоді, коли вони інтегруються з іншою інформацією про контакти з клієнтами. Загальні застосування цієї інформації включають поліпшення зручності використання вебсайту, оптимізацію маршруту до покупки, динамічний контент-маркетинг, створення профілів користувачів на основі аналізу їхньої поведінки та визначення схожості продуктів [1].

Загалом існує три відмінні фази аналізу використання вебсайтів: збір і попередня обробка даних, виявлення знань і аналіз шаблонів, як показано на рисунку 1.3 [1].



Рисунок 1.3 – Фази web usage mining [1]

Етап збору та попередньої обробки даних займається створенням, очищенням даних і перетворенням їх на набір транзакцій користувача, що представляють дії кожного користувача під час його відвідування вебсайту. Цей крок вплине на якість і результат виявлення та аналізу шаблону, тому робити це потрібно дуже акуратно [1].

Метою очищення даних є фільтрація та видалення шуму: невдалих запитів або деяких небажаних даних. Ці методи важливі для будь-якого типу

аналізу вебжурналів, а не лише для аналізу даних, через те, що безпосередньо впливають на якість фінальної підбірки даних.

Завдання ідентифікації користувача складається у виявленні різних сеансів одного користувача з вихідного журналу доступу. Ідентифікація користувача призначена для визначення того, які користувачі переглядають сайт, та які вебсторінки є для них цікавішими або корисними. Труднощі ідентифікації кожного окремого користувача пов'язані з динамічними IP-адресами, використанням VPN тощо. Все це призводить до запису однакових IP-адрес в журнал.

Окрім проблеми ідентифікації користувачів є ще й проблема відтворення шляху користувача по сайту. Причина виникнення такої проблеми полягає в основному в наявності кешу. Якщо користувач клацне назад, щоб відвідати сторінку, копія якої зберігається в кеші, браузер отримає сторінку безпосередньо з кешу. Такий перегляд сторінки ніколи не відобразатиметься в журналі доступу, що спричинить проблему неповного шляху, який потрібно виправити. Щоб виконати це завдання, потрібно звернутися до журналу переходу та топології сайту. Якщо URL-адреса запитованої сторінки не збігається з останньою запитаною прямою сторінкою, це означає, що запитований шлях неповний. Ми можемо виправити неповний шлях за допомогою евристики, наданої реферером і топологією сайту [1].

Виявлення знань або шаблонів є ключовим компонентом web mining, який використовує алгоритми та методи інтелектуального аналізу даних, машинного навчання, статистики та розпізнавання шаблонів тощо. Завдяки цій інформації ми можемо виявити інтереси користувача або спільноти користувачів, а потім побудувати модель інтересів. Зараз зазвичай використовуються методи машинного навчання, в основному включають кластеризацію, класифікацію, виявлення відношень і відкриття моделі порядку [1].

Кластеризація - це метод групування користувачів або елементів даних (сторінок) разом із подібними характеристиками. Це може полегшити

розробку та реалізацію майбутніх маркетингових стратегій. Кластеризація користувачів допомагає виявити групу користувачів, які мають подібний шаблон навігації. Це дуже корисно для визначення демографічних даних користувачів для виконання сегментації ринку в програмах електронної комерції або надання персоналізованого вебвмісту окремим користувачам. Кластеризація сторінок є корисною для пошукових систем Інтернету та постачальників вебпослуг, оскільки її можна використовувати для виявлення груп сторінок із пов'язаним вмістом [3].

Класифікація використовується для відображення елемента даних у попередньо визначені класи. У вебдоміні необхідно витягнути та вибрати атрибути, які найкраще характеризують властивості певного класу чи категорії, щоб створити профіль людей, що належать до цього класу чи категорії. Процес аналізу вебвикористання розуміє наявні дані та поведінку нових екземплярів. Він ідентифікує певний клас/категорію користувача. Методи класифікації використовують машинне навчання (ML), нейронну мережу (NN) і статистику. Для класифікації можна використовувати класифікатори дерева рішень, наївні баєсівські класифікатори, класифікатори k-найближчих сусідів, опорні векторні машини та інші контрольовані індуктивні методи навчання [3].

1.1.2 Web Content Mining

Web Content Mining — це процес вилучення корисної інформації із вмісту вебдокументів [1]. Вміст може складатися з тексту, зображень, аудіо, відео або структурованих записів, таких як списки та таблиці. Можуть бути також метадані та гіперпосилання. Застосування видобутку тексту до вебвмісту було найбільш широко дослідженим. Інтелектуальний аналіз вебконтенту використовує два підходи: перший для видобутку вмісту вебдокументів, та другий для видобутку або покращення результатів пошуку пошукових систем.

Деякі дані вебвмісту є прихованими даними, а деякі генеруються динамічно в результаті запитів і зберігаються в базі даних (БД). Ці дані, як правило, не індексуються. Текстові частини даних вебвмісту складаються з неструктурованих даних, таких як вільні тексти, напівструктурованих даних, таких як HTML-документи, і більш структурованих даних, таких як дані в таблицях або створених базами даних HTML-сторінок. Безсумнівно, велика частина даних вебвмісту є неструктурованими вільними текстовими даними.

На рисунку 1.4 продемонстровано основні підходи до аналізу вебконтенту.

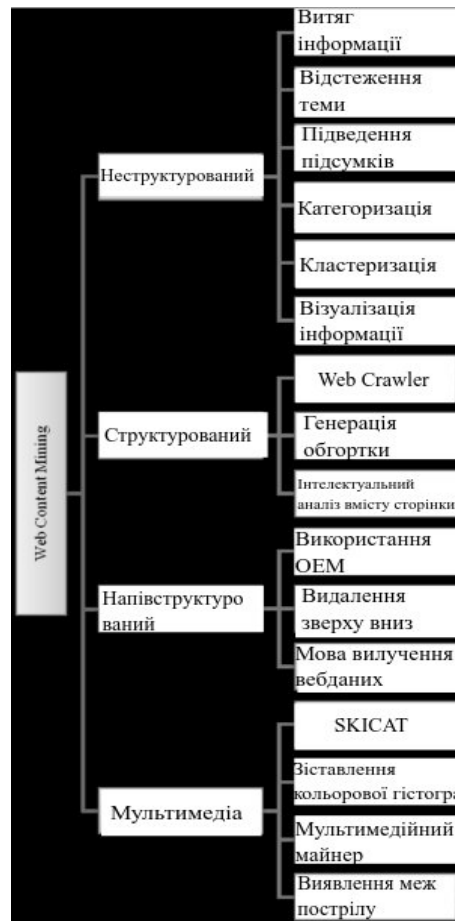


Рисунок 1.4 – Підходи до аналізу вебконтенту [1]

Розглянемо деякі з основних проблем, що найчастіше пов'язані із пошуком вебконтенту. Перша з них – це вилучення даних/інформації. Існує багато методів вилучення структурованих даних з мережі, які також називають генерацією обгортки. По-перше, це написати спеціальну програму для

отримання даних з будь-якого конкретного вебсайту. При цьому підході потрібно багато зусиль, а також це трудомісткий процес. Тому це неможливо. Другий — індукція обгортки або навчання оболонки. У цьому програміст спочатку позначає деякі навчальні сторінки, які використовуються для створення системи навчання, а також з них генеруються деякі правила. Ці правила використовуються для отримання даних вмісту з інших вебсайтів [2].

Друга проблема – це інтеграція вебінформації. Інтернет має величезну кількість даних. Різні вебсайти представляють ту саму інформацію у власному форматі. Навіть на одному вебсайті представлення інформації у двох різних місцях може відрізнятися. Зазвичай методи інтелектуального аналізу вимагають даних у певному стандартному форматі. Ця інформація має бути інтегрована в одному місці, щоб її можна було використовувати для цілей майнінгу. Окрім цього отримання інформації з багатьох баз даних глибокої мережі є складним, оскільки ці величезні дані не можуть бути проіндексовані традиційними пошуковими системами [2].

Третя проблема – це побудова ієрархії понять. Інформації в Інтернеті настільки багато, що для комфортного використання її слід упорядкувати. Але через великий розмір важко організувати всю мережу. Хоча ми можемо організувати результат пошуку за запитом. Вебсторінки, створені на основі ранжирування у відповідь на запит користувача, недостатні для багатьох програм, тому іншим способом організації інформації є ієрархії понять, а іноді й категоризація [2].

Четверта проблема – це автоматична сегментація вебсторінки та видалення шуму. Зазвичай вебсторінка складається з кількох частин: основні області вмісту, меню, оголошення тощо. Не всі з цих даних сприяють вилученню знань і продуктивності вебсайту, а отже результат можна покращити після видалення цього шуму [2].

П'ята проблема – це джерела думок з web mining. У вік ринку зростає конкуренція. У таких ситуаціях компанії вимагають зворотного зв'язку у вигляді думок споживачів у формі опитувань або використовують ручні

методи для отримання відгуків про свої продукти та послуги. І ця інформація є загальнодоступною на самих вебсайтах або у формі блогів на сайтах деяких блогерів. Ця інформація може бути використана для покращення продуктивності вебсайту та його реорганізації [2].

Шоста проблема – це *deep web*. Це невидима або прихована мережа, яка не індексується та не кодується в HTML. Тому пошуковим системам стає важко витягувати інформацію з цього джерела всесвітньої мережі [2].

1.1.3 Web Structure Mining

Web Structure Mining, одна з трьох категорій вебаналізу даних, є інструментом, який використовується для визначення зв'язку між вебсторінками, пов'язаними інформацією або прямим посиланням. Використовується для вивчення топології гіперпосилань з описом посилань або без нього [3]. Основна мета аналізу структур полягає в тому, щоб витягнути раніше невідомі зв'язки між вебсторінками. Інтелектуальний аналіз даних у цій структурі дозволяє компанії зв'язувати інформацію з власного вебсайту, щоб забезпечити навігацію та кластеризувати інформацію на картах сайту. Це дозволяє користувачам отримувати доступ до потрібної інформації за допомогою асоціації ключових слів і аналізу вмісту.

Web Structure Mining витягує шаблони з гіперпосилань в Інтернеті. Гіперпосилання — це структурний компонент, який з'єднує вебсторінку з іншим місцем, розташуванням на тій самій вебсторінці або на іншій вебсторінці. Гіперпосилання, яке з'єднується з іншою частиною однієї сторінки, називається внутрішньодокументним гіперпосиланням, а гіперпосилання, яке з'єднує дві різні сторінки, називається міждокументним гіперпосиланням. Ієрархія гіперпосилань також визначена для того, щоб пов'язана інформація на сайтах була зв'язана з конкурентними посиланнями та з'єднанням через пошукові системи та сторонні посилання. Це дає змогу кластеризувати підключені вебсторінки для встановлення зв'язку між цими

сторінками. Гіперпосилання вказує на весь документ або на певний елемент у документі. Гіпертекст — текст із гіперпосиланнями.

Web Structure Mining також вивчає структуру документа. Він включає аналіз деревоподібної структури структур сторінки для опису використання тегів HTML або XML. Окрім цього він підтримує дерево DOM.

Web Structure Mining виявляє структурну інформацію з Інтернету. Структура типового вебграфа складається з вебсторінок як вузлів і гіперпосилань як ребер, що з'єднують пов'язані сторінки. Будь-яку колекцію, V , сторінок з гіперпосиланнями можна розглядати як орієнтований граф $G = (V, E)$: вузол, що відповідає сторінкам, і спрямоване ребро $(p, g) \in E$ вказує на наявність зв'язку від p до g . Ступінь вузла p — це кількість вузлів, з якими він має зв'язки, а ступінь p — це кількість вузлів, які мають зв'язки з ним. Якщо $W \in V$ є підмножиною сторінок, $G[W]$ можна використовувати для позначення графа, індукованого на W , вузлами якого є сторінки в W , а його ребра відповідають усім зв'язкам між сторінками в W [1].

1.2 Дослідження особливостей SEO

SEO розшифровується як search engine optimization. До задач SEO-спеціаліста входить оптимізація вебсайтів, вебсторінок та вмісту з метою підвищення рейтингу в пошукових системах, таких як Google. SEO — це набір методів, призначених для покращення зовнішнього вигляду, позиціювання та корисності різноманітних типів вмісту в результатах звичайного пошуку. Цей вміст може включати вебсторінки, відеоматеріали, зображення, списки місцевих підприємств та інші активи. Оскільки звичайний пошук є найкращим методом, за допомогою якого люди знаходять онлайн-контент і отримують доступ до нього, використання найкращих методів оптимізації пошукових систем має важливе значення для того, щоб цифровий вміст, який ви публікуєте, міг знаходити та вибирати суспільство, збільшуючи органічний трафік вашого вебсайту [4].

Навчання тому, як створювати вебсайти та сторінки, щоб покращувати, а не шкодити, розміщення цих вебсайтів і вебсторінок у пошуковій системі стало ключовим компонентом еволюції SEO. Цю головну мету SEO іноді називають основним SEO (на відміну від ширших, непрофільних кампаній вебтрафіку, які можуть включати платну рекламу) [5].

Розміщення в пошуковій системі означає, що вебсторінка з'являється в упорядкованому списку результатів пошукового запиту — очевидно, краще, щоб сторінки відображалися вище та ближче до початку списку, який повертає пошукова система у відповідь на запит користувача.

Не всі запити однакові, тому частина ефективного SEO полягає в тому, щоб зрозуміти, які запити важливі для конкретного вебсайту. Відносно легко стати першим результатом пошуку за запитом, який нікого не цікавить.

Очевидно, що залучення трафіку на вебсайт може змінити комерційний успіх і провал. Тож експерти з оптимізації пошукових систем почали дивитися на розміщення в пошуковій системі лише як на один зі своїх інструментів — і розглядати ширший контекст вебтехнологій і бізнес-механізмів, які допомагають створювати та залучати трафік.

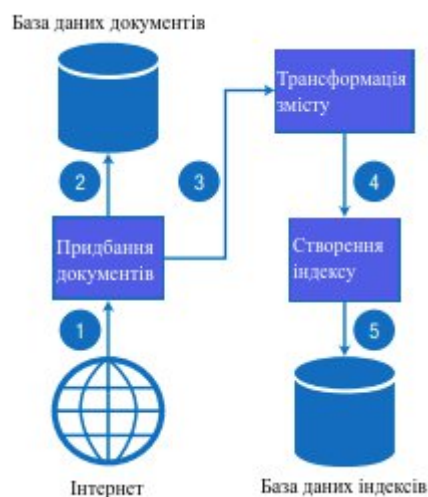


Рисунок 1.5 – Концептуальна схема індексування вебсторінки [6]

Пошукові системи є основою для SEO, але багатьом організаціям бракує знань про те, як вони працюють. На вебсайтах розміщено низку

документів HTML, кожен з яких має унікальний уніфікований показник ресурсу (URL). Пошукова система забезпечує вебпошук, створюючи індекс, процес, прозорий для користувача, і відповідаючи на запити, процес, який вимагає активної участі користувача. На рисунку 1.5 показана концептуальна схема індексування вебсторінки (або документа). Цифри, обведені кружечками, відповідають наступним крокам [6]:

- webcrawler переглядає Інтернет, щоб отримати сторінки HTML;
- webcrawler зберігає ці сторінки в оригінальній формі в базі документів своєї пошукової системи;
- сторінки проходять через трансформації, такі як тег HTML, видалення стоп-слова та формування основи. Перетворення, яке виконує пошукова система, витягує значний текстовий вміст і інформацію про посилання для індексації;
- пошукова система створює індекси шляхом створення прямих і сурогатних представлень сторінок, таких як окремі слова або фрази та їх позиційна інформація на сторінці. Він також записує інформацію про вхідні та вихідні посилання та генерує фрагмент;
- пошукова система зберігає індекси у своїй базі даних індексів.

На рисунку 1.6 показана концептуальна схема процесу запитів і пошуку документів.

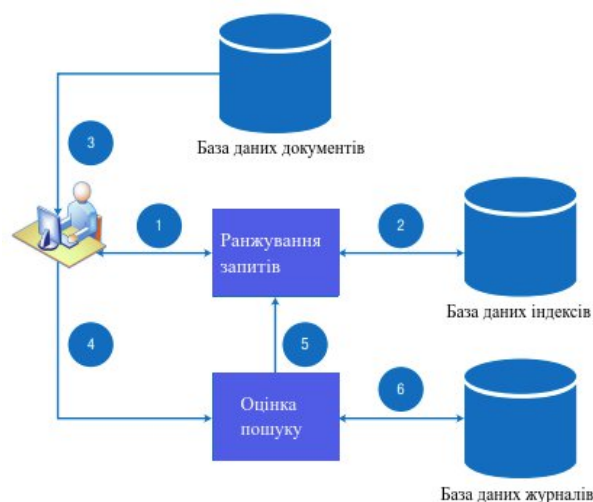


Рисунок 1.6 – Концептуальна схема процесу запитів і пошуку документів [6]

Цифри, обведені кружечками, відповідають наступним крокам [6]:

- користувач використовує браузер пошукової системи, щоб ввести пошуковий запит — як правило, одне ключове слово або коротку фразу;
- алгоритм ранжування запитів пошукової системи створює ранжований список URL-адрес для документів, які він вважає релевантними, на основі бази даних індексу та контекстної інформації в запиті користувача. Потім пошукова система показує користувачеві фрагменти, що відповідають ранжованим URL-адресам у результатах пошуку;
- користувач переглядає фрагменти та клацає певні, щоб отримати відповідні повні документи в оригінальній формі з бази даних документів;
- компонент retrieval evaluation пошукової системи допомагає користувачеві додатково уточнити пошук на основі відгуків про релевантність документа: користувач явно вказує на релевантність (прямий зворотний зв'язок) або натискає відповідні посилання (непрямий зворотний зв'язок);
- використовуючи відгук про релевантність, пошукова система може переформулювати запит користувача та повторно його виконати. Цей процес повторюється, доки користувач не буде задоволений результатами пошуку або не завершить сеанс запиту;
- пошукова система зберігає метаінформацію, таку як запити користувачів, відгуки про релевантність і натиснуті фрагменти в базі даних журналу, яку вона використовує для покращення ефективності пошуку.

Однак не всі компанії дотримуються етичних (white-hat) практик SEO. Окрім них існують ще досить неетичні практики, що називають black-hat практиками. Це практики, що створюють спам, який підриває правдивість рейтингу. За останні роки вони стали більш проблемними, та їх часто використовують компанії, що хочуть підняти свій рейтинг в пошукових системах якомога вище незважаючи на ціну та методи досягнення цієї мети. На сьогодні проблема тільки поглиблюється тим, що з'являються консультанти, які спеціалізуються на використанні таких методів.

Розглянемо основні схеми створення спам-сторінок. Перша з них – це автоматична генерація вебсторінок. Завдяки спеціальним скриптам неетичні SEO спеціалісти можуть масово створювати однакові сторінки на яких будуть розміщені спеціальні ключові слова, завдяки яким пошукові системи будуть вважати що цей сайт підходить під запит користувача. Для того, щоб автоматичні перевірки не виявляли однакових шаблонів, та автоматично не блокували такі сайти, зазвичай використовують синоніми або реальний контент з сайтів, на які не розповсюджується авторське право;

Друга схема – це перенаправлення. Це загалом звичайна практика в інтернеті, що дозволяє додати проміжний шар між самою вебсторінкою та користувачем. Завдяки цьому можна, наприклад, створити перевірку на ботів або перевести користувача зі старого домену сайту на новий. Однак деякі SEO спеціалісти зловживають даною можливістю і створюють багато сторінок, що виглядають для пошукової системи як абсолютно різні з різними тематиками та наповненням, а насправді посилаються на одну й ту саму вебсторінку, яка може й зовсім не містити потрібного користувачу товару, або інформації;

Третя схема – це маскування. Через те, що користувач та робот бачать вебсторінку по-різному стає можливим дати для бота лише ту інформацію, яка вигідна саме тим, хто створював сайт. Через це користувач, що зайде на цю сторінку, може побачити зовсім не те, що хотів побачити, а в найгіршому випадку його комп'ютер навіть може бути заражений вірусом;

Четверта схема – це спам посилань. Через те, що ранг сторінки формується в тому числі на основі того, скільки раз на неї посилаються інші сторінки, та скільки раз вона посилається на інші сторінки сама, можлива схема зі створенням сайтів, що просто містять в собі багато посилань на сторінки, що треба проштовхнути уверх по рейтингу;

П'ята схема – це прихований текст і посилання. Такі дані як правило приховані від користувача за іншими блоками, невидимим шрифтом, або блоками з нулевим розміром. Через це користувач навіть не знає про їх існування, однак про їх існування знають алгоритми, що створюють рейтинг

вебсторінок. Через це вони можуть неправильно помітити цю сторінку, та видати її на запит, інформацію по котрому користувач не побачить на сторінці;

Остання шоста схема – це прохідні сторінки. Такі сторінки містять дуже слабкий з точки зору якості контент, але дуже сильно оптимізовані для покращення свого рангу.

Водночас існують способи етично підняти рейтинг свого сайту. Хоча ці способи й не гарантують швидкого підвищення, однак, при правильно виконаних рекомендаціях, вони дозволять дуже багато виграти на довгій дистанції.

Перший з таких способів – це встановлення тегів заголовка. Теги заголовків є, мабуть, найважливішими тегами SEO на вебсайті, і їх ефективність можна ще більше покращити, поєднавши їх із релевантними ключовими словами. Пошукові системи використовують теги заголовків для визначення теми вебсторінки. Найкраще створити унікальну назву для кожного сайту та зробити назву якомога більш описовою [7]-[8].

Другий спосіб – це встановлення тегів метаопису. Метаописи служать коротким і точним описом вебсторінки, щоб спонукати користувачів клацати фрагменти в результатах пошуку. Метаопис завжди має чесно описувати вміст і містити не більше 140 символів. У поєднанні з ключовими словами з високим рейтингом, пов'язаними із вмістом вебсайту, опис може ще більше підвищити рейтинг [7]-[8].

Третій спосіб – це встановлення тегів зображення. Потрібно вказувати теги ALT і TITLE кожному зображенню, щоб боти могли зрозуміти що знаходиться на цьому зображенні [8].

Четвертий спосіб – це якісний контент. Якісний контент, ефективність і загальна задоволеність користувачів є важливими факторами, які впливають на рейтинг вебсайту в пошукових системах. Погана реалізація одного з цих елементів може безпосередньо призвести до збільшення показника відмов. Показник відмов вимірює відсоток користувачів, які відвідують лише першу сторінку вебсайту, а потім повертаються до пошукової видачі. Якщо вебсайт

має високий показник відмов, пошукова система припускає, що результат не був хорошою відповіддю на цей конкретний пошуковий запит. Зручність використання має великий вплив на SEO, особливо у формі початкового часу завантаження вебсайту. Користувачі не люблять чекати тривалий час, поки сторінка завантажиться. Розв'язанням цієї проблеми є відкладене завантаження, яке дозволяє динамічно завантажувати вміст лише за запитом [7]-[8].

П'ятий спосіб – це спеціальна сторінка 404. Щоразу, коли хтось запитує сторінку, яка недоступна через помилку друку або технічну проблему, сервер відповідає 404 «сторінка не знайдена». Якщо відвідувачі натрапляють на сторінку 404, можливо, вони не захочуть робити повторну спробу знайти початково запитуваний сайт. Відповідно, спеціальна сторінка помилки може надати користувачеві відгуки, а також подальші дії, щоб наблизитися до місця призначення [7].

Шостий спосіб – це мобільна сумісність. Попри те, що загальна кількість відвідувань вебсайтів із мобільних пристроїв зросла з 57% у 2016 році до 63% у 2017 році [7], все ще є досить мало публікацій про SEO з фокусом на адаптивності вебсайтів. Таким чином, кількість користувачів з мобільними пристроями не тільки зростає, але й продовжує зростати [7].

Сьомий спосіб – це щільність ключових слів. Щільність ключових слів описує частоту ключових слів у порівнянні із загальною кількістю слів на сторінці. Загальна кількість слів включає назву, описи та загальний вміст вебсторінки. Це важливо, оскільки пошукові системи використовують ці ключові слова для класифікації сайтів і визначення загальної відповідності конкретним термінам [7].

Восьмий спосіб – це оптимізація поза сторінкою. Пошукова оптимізація за межами сторінки обертається навколо різних методів поза межами вебсайту для підвищення його рейтингу в пошукових системах. Зворотні посилання (інші сайти, що посилаються на ваш вебсайт), також

відомі як вхідні посилання - це найважливіший фактор оптимізації поза сторінками [7].

З усього вище сказаного можна зрозуміти яка інформація потрібна для SEO-спеціаліста:

– контент сторінки. Він потрібен, щоб зрозуміти чи є на сторінці опис того, що вона повинна показувати. Це необхідно для того, щоб покращити вірогідність того, що користувач перейде на сторінку. Окрім цього використання ключових слів підвищить рейтинг сторінки, що ще більше збільшить ймовірність переходу користувача на неї. До того ж завдяки контенту можна зрозуміти якість того, що на ній написано. Чим детальніше та простіше інформація, тим вище ймовірність того, що користувач захоче прочитати її саме на цій сторінці цього сайту. Окрім цього так можна відслідкувати унікальність інформації, що допоможе при покращенні рейтингу сторінки;

– ключові слова зі сторінки. Їх найголовніша роль – це показати ботам, що будуть сканувати цю сторінку, яка інформація є основною. Саме завдяки ключовим словам пошукові системи можуть зрозуміти чи підходить сторінка під запит користувача та на скільки інформація, що представлена на ній, відповідає тому, що хоче побачити користувач, а потім, на основі цієї інформації, сформувати рейтинг сторінок;

– архітектура сайту. Архітектура дуже сильно впливає на рейтинг сайту в пошукових системах. Гарна та грамотна архітектура допомагає пошуковим системам набагато легше індексувати сайт, що своєю чергою веде до підвищення рейтингу сайту. Окрім цього широкі та зручні можливості для переміщення по сайту гарно вплинуть на ймовірність того, що користувач буде повертатись на сайт у майбутньому. Також для пошукової системи важливі внутрішні посилання сторінки, тож треба перевіряти, що вони також присутні на сторінці. Більш того, правильна архітектура зменшить кількість переходів користувача від одної сторінки до іншої, а отже зменшить кількість потрібних завантажувальних інформації, що збільшить популярність вебсайту;

– посилання сторінки. Вони допоможуть зрозуміти на які сторінки посилається та, яку ми передивляємось, що своєю чергою дозволить зрозуміти чи не забагато на сторінці посилань, побачити зріз архітектури сайту, а також на які саме сторінки йдуть посилання.

1.3 Аналіз програмного забезпечення

Вибір програмного забезпечення для ефективного виконання задач SEO є складним завданням, що потребує детального аналізу та порівняння різних продуктів. При виборі програмного забезпечення, необхідно враховувати ряд критеріїв, таких як функціональні можливості, ціна, зручність використання та підтримка користувачів.

Першим критерієм вибору програмного забезпечення є відповідність функціональним потребам SEO-спеціаліста. Необхідно визначити, які саме функції має мати програмне забезпечення для їх реалізації та знайти те, що відповідає якщо не всім, то більшості з потреб.

Другим критерієм є вартість програмного забезпечення та вартість його реалізації. Потрібно зважати на ціну ліцензії, підтримку та навчання користувачів. Важливо визначити, наскільки ефективне та зручне для підприємства є використання програмного забезпечення в порівнянні з його вартістю.

Третім критерієм є зручність використання та навчання користувачів. Програмне забезпечення повинно бути зручним та легким у використанні, а також мати зрозумілий та докладний інтерфейс користувача. Для забезпечення ефективного використання програмного продукту, необхідно забезпечити якісну підготовку та навчання користувачів.

Крім вищезгаданих критеріїв, слід звернути увагу на підтримку та розвиток програмного забезпечення. Необхідно переконатися, що продукт буде активно підтримуватися та розвиватися в майбутньому, оскільки технології постійно змінюються, а потреби спеціалістів можуть зростати. Крім

того, важливо перевірити наявність документації та технічної підтримки продукту, які забезпечать можливість швидко розв'язувати проблеми та отримувати допомогу у випадку потреби.

Ще одним важливим критерієм є сумісність з іншими програмними засобами, що використовуються при роботі. Програмне забезпечення повинно виконувати унікальний функціонал, якого немає в інших програмах, або має замінити собою декілька з них.

Усе вищесказане демонструє нам, що програми для SEO-спеціалістів повинні обиратися дуже ретельно з урахуванням багатьох різноманітних критеріїв. Нижче наведені деякі з популярних SEO інструментів на прикладі яких розроблявся цей проєкт. Порівняння деяких з них наведено у таблиці 1.1.

Таблиця 1.1 – Порівняння SEO інструментів

Категорія	Browseo	ContentKing	Google Lighthouse	Apify
Комплексний аналіз однієї сторінки	+	+	+	+
Аналіз усього сайту	-	+	-	+
Періодичне оновлення інформації	+/-	+	+/-	+
Аналіз PageRank сторінки	-	-	-	-
Аналіз TextRank сторінки	-	-	-	-

1.3.1 Browseo

BROWSEO — це програма, яка дозволяє переглядати будь-яку вебсторінку, не відволікаючись від стилів. Він також виділяє частини сторінки, які мають відношення до SEO. Цей сервіс корисний для дослідження

структури сайтів, виявлення проблем, які можуть впливати на рейтинг в пошукових системах, та для оптимізації вмісту вебсторінок. Зовнішній вигляд сервісу представлено на рисунку 1.7.

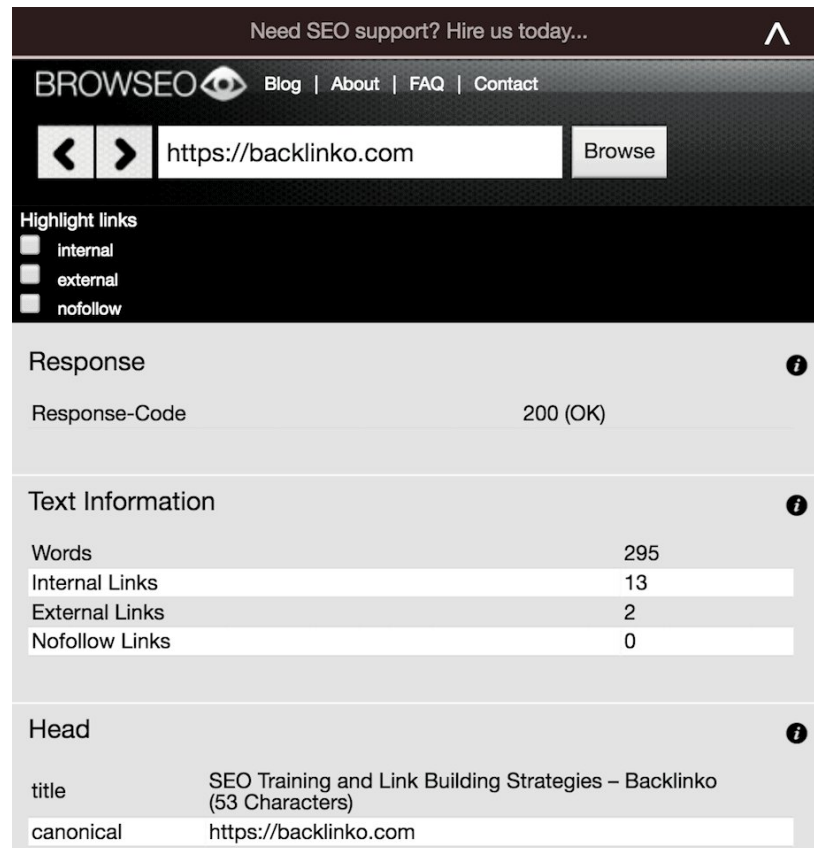


Рисунок 1.7 – Сервіс BROWSEO [9]

Основні функції Browseo містять в собі:

- відображення структури сторінки. Browseo розбиває вміст вебсторінки на основні компоненти, такі як заголовки, метатеги, текстовий контент, посилання і т. д. Це допомагає користувачам швидко зрозуміти, як саме виглядає їхня сторінка для пошукових систем;

- перевірка ключових параметрів SEO. Browseo аналізує такі важливі параметри, як метатеги, ключові слова, заголовки й заголовки другого рівня. Користувач може переконатися, що їхні сторінки оптимізовані для певних ключових запитів;

- ціна. Сервіс повністю безплатний;

- демонстрація шляху перенаправлень. Сервіс може показувати повний шлях перенаправлень зі стартового посилання до фінального результату;

- перегляд того, як буде виглядати посилання у Google.

Однак у цього сервісу є наступні недоліки:

- спрощені можливості порівняння. Browseo, в основному, надає інформацію про окремі сторінки, а не дозволяє легко порівнювати кілька сторінок одночасно. Це може бути не зручно для користувачів, які хочуть аналізувати конкурентів або власні сторінки порівняно;

- обмеження на обсяг сторінки. Browseo може мати обмеження на обсяг сторінки, яку можна проаналізувати. Це може стати проблемою для великих або складних вебсайтів;

- відсутність широких можливостей. Browseo має невеликий функціонал, а отже він може бути недостатнім для повного SEO-аналізу;

- аналіз лише однієї сторінки. Browseo може аналізувати лише одну сторінку сайту, що недостатньо для якісного SEO-аналізу;

- вимога до технічного розуміння. Для користування Browseo ефективно, користувач повинен мати базове розуміння SEO-принципів і технічних аспектів вебсайту. Для новачків це може бути складним.

1.4.2 Clusteric

Clusteric – це високоефективний інструмент для вдосконалення вебсайтів. З його допомогою професійні SEO-спеціалісти можуть проводити аудит сайтів, ретельний аналіз конкурентів, визначати стратегії ключових слів та стежити за їхньою видимістю в пошукових системах. Широкий спектр інструментів Clusteric робить його ідеальним інструментом для оптимізації вебпростору та здобуття кращого рейтингу в пошукових системах. Зовнішній вигляд сервісу представлено на рисунку 1.8.

SEARCH AUDITOR









SHOW/HIDE COL.

	IR	AG	SUBPAGE	GOOGLE_TITLE	T_VISI	EYWORD_CATEGORY	WORD_SUBC	SERP_R
backlinks			http://backli...	Backlinks: 17...	610	Business & ...	Business M...	9180000
best seo tools			http://backli...	SEO Tools: T...	220	Business & ...	Business M...	3940000
adwords keyword p			http://backli...	Google Key...	78	Business & ...	Business M...	374000
backlink			http://backli...	Backlinks: 17...	61	Business & ...	Business M...	37600000
backlinking			http://backli...	Backlinks: 17...	44	Business & ...	Business M...	427000
1 million			http://backli...	We Analyze...	33	Beauty & P...	Perfumes &...	1490000000
cost per action			http://backli...	CPA Marketi...	24	Business & ...	Business M...	78600000
adwords keyword t			http://backli...	Google Key...	20	Business & ...	Business M...	763000
cpa advertising			http://backli...	CPA Marketi...	13	Business & ...	Business M...	17400000
back link			http://backli...	17 Untappe...	12	Business & ...	Business M...	165000000
adwords planner			http://backli...	Google Key...	10	Business & ...	Business M...	533000
adword keyword t			http://backli...	Google Key...	10	Business & ...	Business M...	123000
backlinks seo			http://backli...	SEO Strateg...	10	Business & ...	Business M...	589000
adword planner			http://backli...	Google Key...	8.7	Business & ...	Business M...	50300
cpa networks			http://backli...	CPA Marketi...	6.8	Business & ...	Business M...	976000
effective search en			http://backli...	21 Actionabl...	4.8	Business & ...	Business M...	12300000
adwords tool			http://backli...	Google Key...	4.3	Business & ...	Business M...	2590000
cpa network			http://backli...	CPA Marketi...	3.9	Business & ...	Business M...	4290000
adword keyword p			http://backli...	Google Key...	3.8	Business & ...	Business M...	33300
cost per acquisition			http://backli...	CPA Marketi...	3.4	Finance	Business Fin...	25400000
cpa affiliate			http://backli...	CPA Marketi...	1.8	Business & ...	Business M...	469000
copywriting			http://backli...	16 Actionabl...	1.8	Business & ...	Business M...	20400000

Рисунок 1.8 – Сервіс Clusteric [10]

Плюси Clusteric:

- багатий функціонал. Clusteric дає можливість використовувати дуже багато різноманітних інструментів, що дасть змогу провести всесторонній аналіз з покращення SEO;
- інтуїтивний інтерфейс. Легкий у використанні інтерфейс дозволяє користувачам швидко зорієнтуватися та використовувати всі доступні інструменти.

Мінуси Clusteric:

- складність використання: Щоб отримати максимальну користь від Clusteric, може знадобитися розуміння принципів його роботи;
- вартість: Ціна за використання Clusteric може бути досить великою, особливо для користувачів з обмеженим бюджетом.

1.4.3 ContentKing

ContentKing схожий на багато інших інструментів SEO аудиту на ринку. З однією важливою відмінністю: він завчасно сканує ваш сайт і сповіщає вас про будь-які проблеми, які виникають. Зовнішній вигляд сервісу представлено на рисунку 1.9.

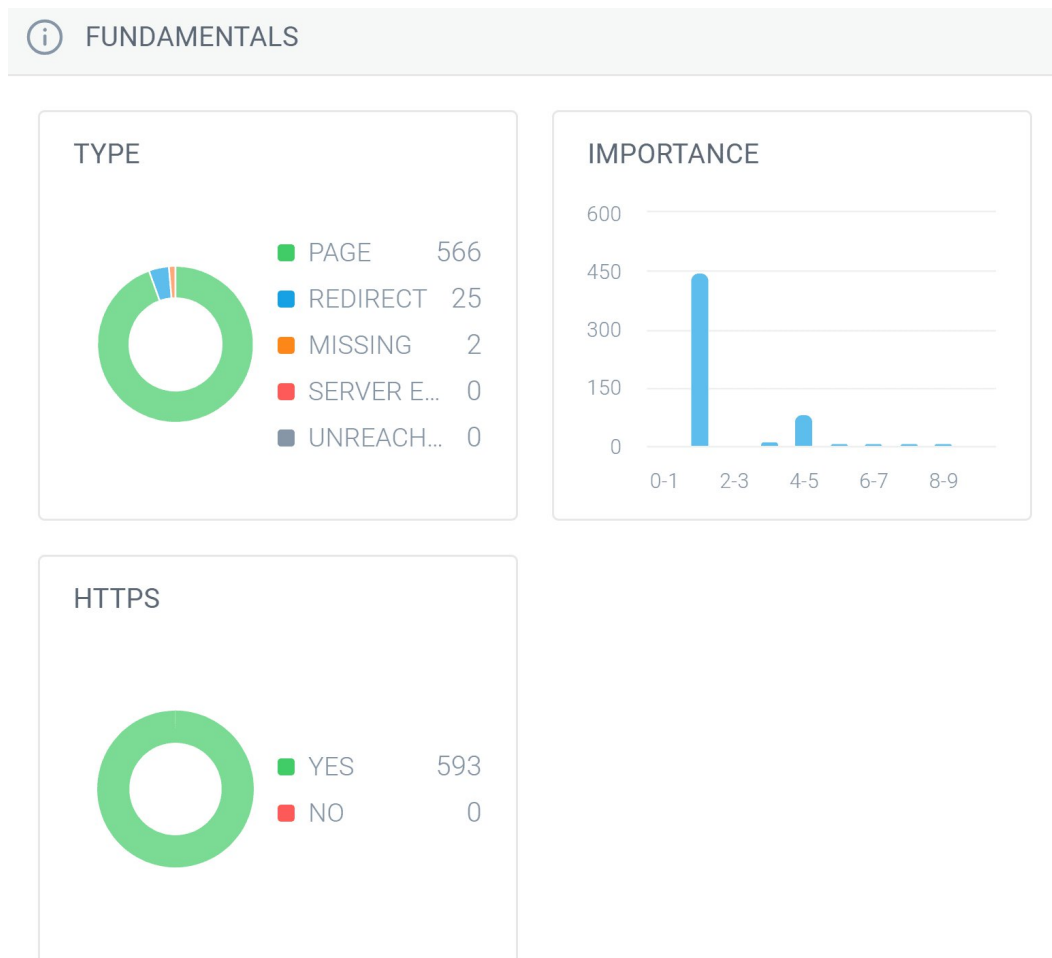


Рисунок 1.9 – Сервіс Clusteric [11]

Плюси ContentKing:

- реальний час. ContentKing пропонує моніторинг в реальному часі, дозволяючи вчасно реагувати на проблеми й зміни на сайті;
- виявлення проблем. Сервіс автоматично виявляє технічні та контентні проблеми на сайті, такі як обривки, неправильні посилання та інші;

– рекомендації. ContentKing надає корисні рекомендації щодо виправлення проблем і покращення SEO оптимізації;

– візуалізація змін. Графічні інструменти допомагають SEO-спеціалістам краще розуміти вплив змін на сайті на його видимість.

Мінуси ContentKing:

– вартість. ContentKing може бути дорогим для невеликих вебпроектів або фрилансерів;

– спеціалізація. ContentKing фокусується на SEO-моніторингу, тому для більш комплексної аналітики сайту може знадобитися використання інших інструментів.

1.4.4 Google Analytics

Google Analytics схожий на багато інших інструментів SEO аудиту на ринку. З однією важливою відмінністю: він завчасно сканує ваш сайт і сповіщає вас про будь-які проблеми, які виникають. Зовнішній вигляд сервісу представлено на рисунку 1.10.

Impressions ?	Clicks ?	Average Position ?	CTR ?
2,071,947 % of Total: 115.11% (1,800,000)	76,597 % of Total: 102.13% (75,000)	22 % of Total: 86.87% (25)	3.70% Avg for View: 4.17% (-11.28%)
550,000 (26.55%)	4,500 (5.87%)	8.1 (34.08%)	0.82%
250,000 (12.07%)	8,000 (10.44%)	29(132.94%)	3.20%
170,000 (8.20%)	6,500 (8.49%)	32(147.43%)	3.82%
140,000 (6.76%)	10,000 (13.06%)	11 (46.08%)	7.14%
90,000 (4.34%)	5,500 (7.18%)	20 (90.42%)	6.11%
75,000 (3.62%)	5,500 (7.18%)	18 (80.35%)	7.33%
75,000 (3.62%)	1,300 (1.70%)	14 (64.28%)	1.73%

Рисунок 1.10 – Сервіс Google Analytics [12]

Плюси Google Analytics:

- відстеження результатів. Google Analytics надає докладну статистику про те, як користувачі взаємодіють із сайтом, що допомагає SEO-спеціалістам вимірювати вплив оптимізації;
- визначення ефективності ключових слів. Можливість відстеження, які ключові слова призводять до більшої кількості відвідувачів та конверсій, допомагає оптимізувати контент;
- кастомізація звітів. SEO-спеціалісти можуть створювати власні звіти та панелі для відображення конкретних метрик, що їх цікавлять;
- безплатний доступ. Google Analytics пропонує базовий безплатний план, що робить його доступним для більшості вебсайтів.

Мінуси Google Analytics:

- складність. Використання Google Analytics може бути важким для новачків, і спеціалістам може знадобитися час для опанування всіх функцій;
- загальні дані. Деякі SEO-спеціалісти можуть вважати, що Google Analytics надає загальні дані, які не завжди відображають всю картину;
- залежність від Google. Якщо ви використовуєте Google Analytics, ви залежите від Google, і це може викликати певні обмеження та конфіденційність даних.

1.4.5 Google Lighthouse

Lighthouse — це автоматизований інструмент із відкритим кодом для покращення якості вебсторінок. Його можна запустити на будь-якій вебсторінці, загальнодоступній або з вимогою автентифікації. Він має аудит продуктивності, доступності, прогресивних вебдодатків, SEO тощо [13].

Lighthouse можна запустити також в Chrome DevTools, з командного рядка або як модуль Node. Користувач повинен надати Lighthouse URL-адресу для перевірки, після чого він запустить серію перевірок сторінки, а потім

створить звіт про те, наскільки сторінка зроблена добре. Після цього можна використовувати невдалі перевірки як індикатори того, як покращити сторінку. Кожен аудит має довідковий документ, у якому пояснюється, чому аудит важливий, а також як це виправити. Зовнішній вигляд сервісу представлено на рисунку 1.11.

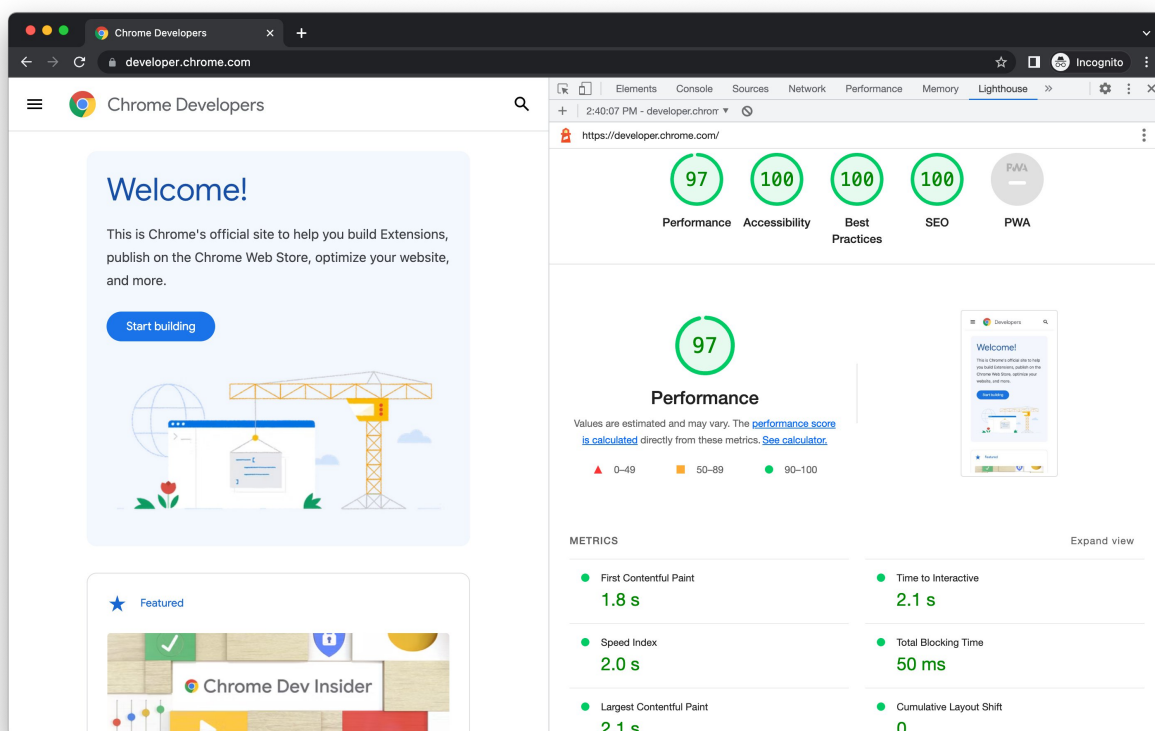


Рисунок 1.11 – Сервіс Google Lighthouse [13]

Плюси Google Lighthouse:

- об'єктивність. Інструмент надає об'єктивні метрики, які допомагають визначити реальну продуктивність вебсайту;
- рекомендації з оптимізації. Lighthouse надає конкретні поради щодо вдосконалення швидкості завантаження та інших аспектів сайту;
- відображення на вебсайті. Ви можете використовувати Lighthouse безпосередньо у веббраузері, що полегшує перевірку сторінок;
- оцінка на мобільних пристроях. Lighthouse оцінює продуктивність сайту на мобільних пристроях, що особливо важливо для SEO.

Мінуси Google Lighthouse:

- оцінка лише однієї сторінки. Lighthouse аналізує окремі сторінки, і вам доведеться перевіряти кожну сторінку окремо;
- не враховує контент. Інструмент не оцінює якість контенту, що також важливо для SEO.

1.4.5 Apify

Apify - це платформа для автоматизації скрапінгу та збору даних з Інтернету. Зовнішній вигляд сервісу представлено на рисунку 1.12.

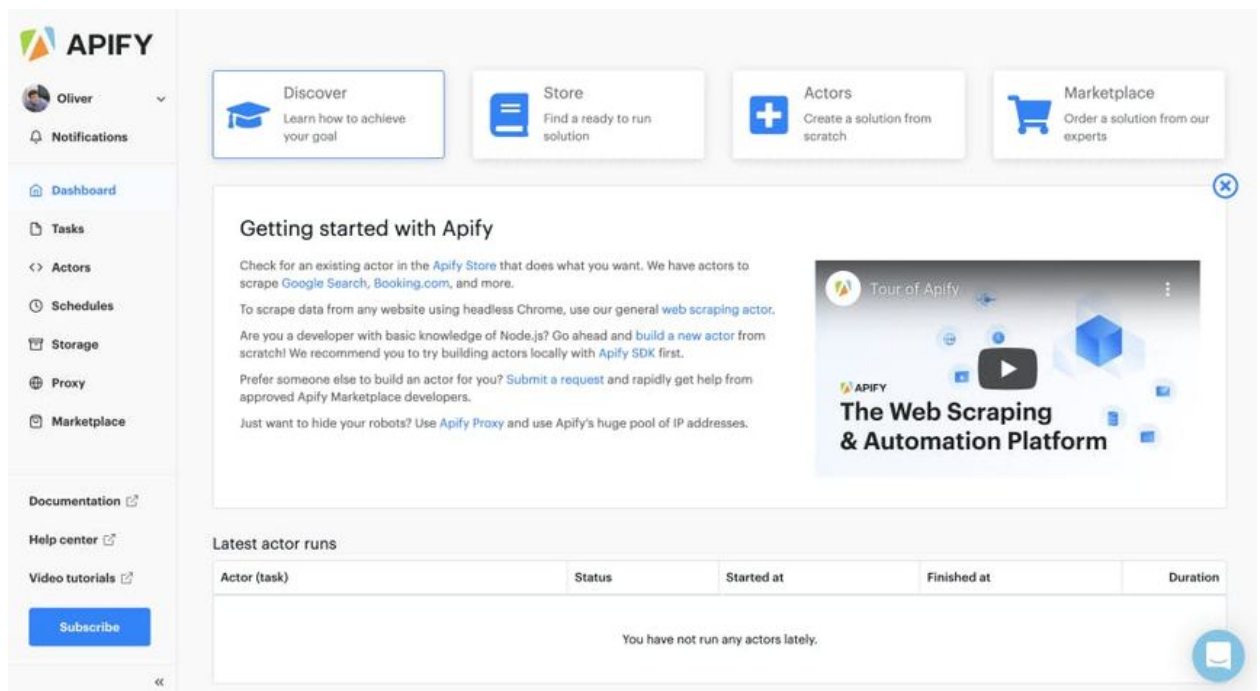


Рисунок 1.12 – Сервіс Apify [14]

Плюси Apify:

- збір і аналіз даних. Apify дозволяє вам збирати велику кількість даних з різних вебсайтів, що може бути корисним для аналізу конкурентів, ключових слів та інших SEO-аспектів;
- автоматизація. Платформа надає інструменти для автоматизованого збору даних, що дозволяє зекономити час та зусилля.

Мінуси Arify:

– вартість. Використання Arify може бути вартісним, особливо для великих обсягів даних або великої кількості скраперів.

1.4 Висновки за розділом 1 та постановка задачі

Як показали проведені дослідження, в сучасному світі SEO-процеси та задачі стають все складнішими. У зв'язку з цим, виникає потреба в ефективних та зручних системах отримання та аналізу даних з вебсайтів, які допоможуть спеціалістам оптимізувати свою роботу та збільшити прибуток.

На жаль, навіть при наявності великої кількості різноманітних реалізацій систем для SEO, знайти ту, яка підходить до абсолютно будь-якої задачі дуже складно. Для більшості спеціалістів це стає проблемою, оскільки вони не можуть дозволити собі витратити багато коштів на кожен окрему програму.

Для розв'язання цих проблем, було прийняте рішення розробити програму, яка буде містити значну кількість необхідних для розв'язання цих задач елементів. Ця програма буде мати додаткові функції, які допоможуть спеціалістам забезпечити ефективний контроль та аналіз даних своєї діяльності. Використання цієї програми дозволить спеціалістам заощадити час та кошти, а також забезпечить їм більш точну та детальну інформацію про сайти, які вони досліджують.

Завдання роботи:

- проаналізувати аналоги створюваного вебзастосунку;
- дослідити технології, що плануються для використання;
- вивчити взаємодію з мовою програмування Golang;
- дослідити принципи роботи технології web mining та SEO;
- створити реалізацію вебзастосунка для потреб SEO;
- протестувати створений вебзастосунок.

Мета роботи – створення програмного вебзастосунка на основі технології web mining для підвищення SEO-ефективності.

Для реалізації цієї задачі програма повинна буде отримувати інформацію з сайту, розбирати її на складові та зберігати у власній базі даних для спрощення роботи з нею надалі.

2 МАТЕРІАЛИ І МЕТОДИ

2.1 Підходи реалізації

Існують два основних розділи оптимізації сторінки для пошукового движку. Кожен з цих підходів неймовірно важливий для подальшого покращення рейтингу сторінки при пошуку. Ці два розділи: On-Page та Off-Page.

On-Page оптимізації пов'язані з наповненням сторінки. До наповнення сторінки відносяться ключові слова, фрази та запитання. Правильно підготовлений вміст дозволяє пошуковим системам зрозуміти, про що йдеться на сторінці сайту, та відобразити її для відповідних пошукових запитів. Інструменти для On-Page оптимізацій фокусуються на наступних можливостях [15]-[16]:

- ключові слова. Дослідження ключових слів передбачає пошук ключових слів з високим обсягом пошуку та релевантністю. Це дозволяє дослідити чи буде вебсторінка відображатися в результатах пошуку вище за рейтингом та яка ймовірність того, що користувач введе саме це ключове слово. Своєю чергою така інформація дозволить зрозуміти чи правильно було обране ключове слово, або потрібно його замінити на інше для покращення рейтингу сторінки;

- вміст. Для того, щоб рейтинг сторінки був вищим, її вміст повинен бути унікальним, інформативним та містити у собі популярну у користувачів інформацію;

- оптимізація ключових слів. Оптимізація ключових слів полягає в ефективному використанні цільових ключових слів у різних частинах сторінки, таких як метазаголовки, метаописи, заголовки та важливих частинах контенту. Ці інструменти SEO на сторінці також можуть використовуватися для контролю щільності ключових слів та пошуку дубльованого контенту на інших сторінках сайту.

Ось деякі з інструментів створених для On-Page оптимізації:

- Google Keyword Planner;
- Google Search Console;
- SEMrush Keyword Tool;
- Moz Keyword Research Tool;
- Soovle;
- Ahrefs Keywords Explorer;
- Google Trends.

Своєю чергою Off-Page оптимізації спрямовані на дії, що напряду зі сторінкою не пов'язані. Одним з ключових показників, який використовують Google та інші пошукові системи, є зв'язки між сайтами. Такими зв'язками є посилання на інші сайти, та backlinks [15]-[16].

Backlinks - це посилання, які створюються, коли зовнішній сайт посилається на сайт користувача. Хоча такі посилання важливі для ранжування, однак не всі такі посилання рівні. Високоякісні backlinks повинні надходити з авторитетних вебсайтів, щоб показати, що інформація на вашому сайті є цінною і їй можна довіряти. Багато факторів сприяють «цінності» backlinks, наприклад [15]:

- популярність, надійність та авторитет вебсайту, що посилається;
- наскільки актуальною є тема вебсайту, що посилається, порівняно зі сторінку, з якою він пов'язаний;
- текст якоря, який використовується на сайті для посилання;
- кількість посилань.

Окрім зворотного зв'язку, Off-Page оптимізації можуть мати форму маркетингу в соціальних мережах, ведення блогів, пов'язаних або не пов'язаних згадок про бренд та закладки від користувачів. Інструменти SEO для Off-Page оптимізації включають [15]:

- різні форми соціальних медіа (Facebook, Instagram, LinkedIn, Twitter тощо);
- Moz Link Explorer;
- SEMrush Link Building Tool;

- Ahrefs Link Building;
- Google Search Console;
- Backlinko.

Окрім On-Page та Off-Page оптимізацій є ще Technical SEO інструменти. Цей тип SEO безпосередньо впливає на ефективність On-Page та Off-Page SEO. Technical SEO орієнтований на різні елементи сайту, включаючи [15]:

- швидкість сайту та час завантаження;
- можливість відкрити сайт на будь-яких пристроях;
- можливість легкої індексації сайту;
- структура сайту;
- безпека сайту;
- перенаправлення та актуальні URL-адреси.

Technical SEO, часто називають фундаментом вебсайту, оскільки він сприяє роботі пошукових систем та визначає загальну якість сайту. Для досягнення оптимального користувацького досвіду необхідно впроваджувати ефективні технічні SEO стратегії, що мають вагомий роль у покращенні позицій в пошукових системах. Інструменти Technical SEO включають, але не обмежуються наступними:

- Screaming Frog;
- SEMrush SEO Tool;
- Siteimprove SEO Tool;
- Google Search Console & Webmaster Tools;
- WebPageTest;
- Moz Bar;
- Google Page Speed Insights.

2.2 Дослідження особливостей технології PageRank

PageRank — це система балів, яка ґрунтується на структурі вебпосилань для визначення важливості вебсторінок. Обчислення PageRank відіграють важливу роль у сучасних системах ранжирування пошуку та привертають широкую увагу дослідників відтоді, як вони були розроблені Google. Після багатьох років розробки алгоритм PageRank тепер часто застосовується до будь-якого графіка чи мережі в будь-якому домені, наприклад у біології, аналізі нейронної та соціальної мережі, бібліометричних індексах тощо [17]. Математичний рейтинг вебсторінок (PageRank) для простої мережі, виражений у відсотках представлено на рисунку 2.1.

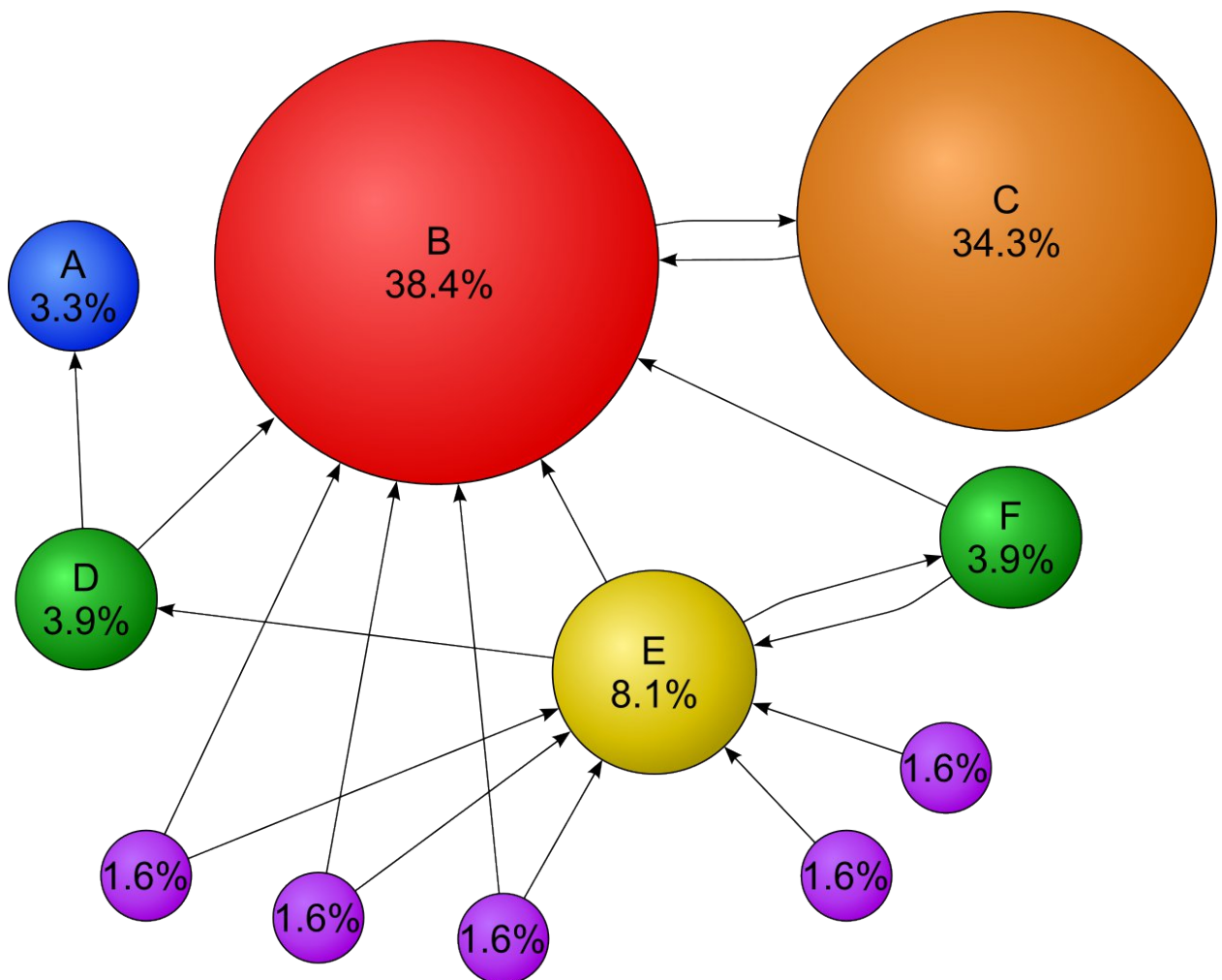


Рисунок 2.1 – Математичний рейтинг вебсторінок [18]

Алгоритм PageRank відповідає ідеї користувача, який переглядає вебсайти, переходячи за посиланнями випадковим чином (випадковий відвідувач). Для обчислення PageRank існує декілька формул. У цій роботі ми розглянемо оригінальну формулу PageRank і зважену версію, яка враховує позицію посилання в статті [19].

Оригінальний PageRank – у наборі статей W ми використовуємо окремі спрямовані посилання $link(w_l, w_2)$ з $w_l, w_2 \in W$, зокрема набір сторінок, які посилаються на сторінку $l(w) = \{w_l | link(w_l, w)\}$ і кількість вихідних посилань $c(w) = |\{w_l | link(w, w_l)\}|$. PageRank сторінки $w_0 \in W$ обчислюється формулою (2.1) [19]:

$$pr(w_0) = (1 - d) + d * \sum_{w_n \in l(w_0)} \frac{pr(w_n)}{c(w_n)}. \quad (2.1)$$

Зважений рейтинг посилань (WLRank) – щоб врахувати відносну позицію посилання в статті, ми адаптуємо формулу (2.1) і вводимо ваги посилань. Ідея полягає в тому, що випадковий відвідувач, швидше за все, не переходить за кожним посиланням на сторінці з однаковою ймовірністю, а може віддати перевагу тим, які знаходяться у верхній частині сторінки. WLRank сторінки $w_0 \in W$ обчислюється формулою (2.2) [19]:

$$wlr(w_0) = (1 - d) + d * \sum_{w_n \in l(w_0)} \frac{pr(w_n) * lw(link(w_n, w_0))}{\sum_{w_m} lw(link(w_n, w_m))}. \quad (2.2)$$

Вагова функція зв'язку lw визначається формулою (2.3):

$$lw(link(w_1, w_2)) = 1 - \frac{first_occurence(link(w_1, w_2), w_1)}{|tokens(w_1)|}. \quad (2.3)$$

Щоб сформувати правильну ймовірнісну модель, вага окремого посилання нормалізується відповідно до ваг усіх вихідних посилань сторінки

у формулі (2.2). Якщо ми встановлюємо вагу кожного вхідного посилання однаковим значенням (наприклад, 1) отримуємо вихідну формулу PageRank (див. формулу (2.1)). Використовувані допоміжні функції формули (2.3) можна описати так [19]:

– $first_occurrence(link(w_1, w_2), w_1)$ – номер маркера першого входження посилання (w_1, w_2) на відповідній сторінці w_1 . Нумерація tokenів починається з 1 (тобто перше слово/посилання);

– $tokens(w_1)$ – загальна кількість tokenів сторінки w_1 . Tokenізація виконується наступним чином: ми розбиваємо текст статті відповідно до пробілів, але не розбиваємо посилання (наприклад, [[бурий ведмідь|ведмеді]] розглядається як один token).

Обидві формули (2.1) і (2.2) застосовуються ітераційно, поки бали не збіжаться. Змінна d називається «фактором демпфування»: у моделі випадкового відвідувача вона враховує можливість доступу до сторінки через адресний рядок браузера замість доступу через посилання з іншої сторінки [19].

2.3 Дослідження особливостей технології TextRank

TextRank - це алгоритм для автоматичного виділення ключових слів та сортування текстових даних на основі їх важливості, що використовує ідею PageRank, але застосовує її до тексту. Він визначає важливість слів та фраз у тексті, допомагаючи при його аналізі.

Для завдання автоматизованого узагальнення TextRank моделює будь-який документ у вигляді графіка, використовуючи речення як вузли. Функція для обчислення подібності речень потрібна для складання ребер між ними. Ця функція використовується для зважування країв графіка, чим вище схожість між реченнями, тим важливішим буде край між ними у графіку. У домені Random Walker, який часто використовується в PageRank, ми можемо

сказати, що ми швидше переходимо від одного речення до іншого, якщо вони дуже схожі [20]. Візуальна репрезентація представлена на рисунку 2.2.

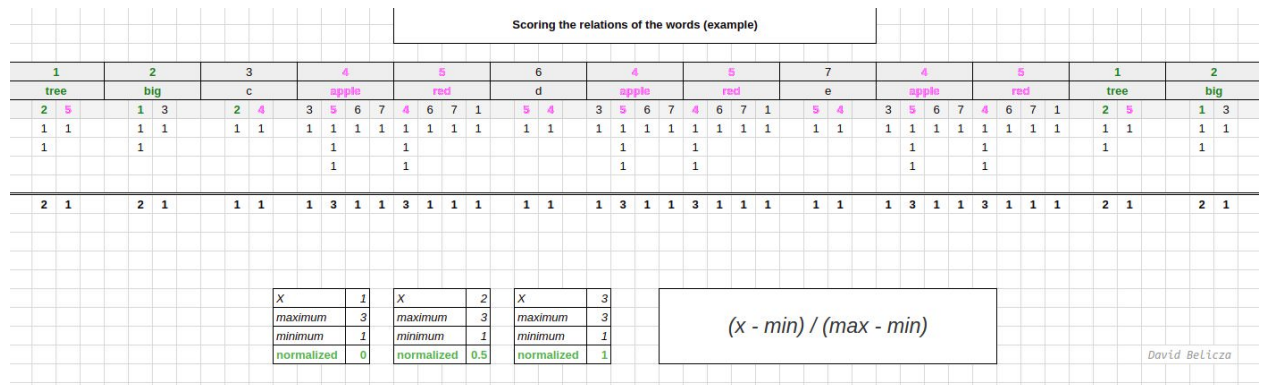


Рисунок 2.2 – Просте візуальне представлення алгоритму TextRank [21]

TextRank використовує співвідношення словникового запасу для сортування наступних ключових слів на основі вікна спільного виникнення, яке може витягувати ключові слова з тексту безпосередньо та краще відображати семантичну інформацію між текстовими одиницями. По-друге, TextRank може знайти найбільш інформативні елементи з тексту, розділивши текст на кілька компонентів, таких як слова та речення, та створивши текстовий графік між компонентами. По-третє, TextRank широко застосовується до реальних програм, включаючи визначення достовірності вебконтенту, витяг із резюме з кількох документів та класифікацію тексту, і раніше не потрібно жодного процесу навчання чи навчання [22].

Функція, що представлена в оригінальному алгоритмі, може бути формалізована як формула (2.4) [20]:

$$Sim(S_i, S_j) = \frac{|\{wk | wk \in S_i \& wk \in S_j\}|}{\log(|S_i|) + \log(|S_j|)} \quad (2.4)$$

Цей процес призводить до створення графіку, що репрезентує документ, і застосовує PageRank для визначення значущості кожного вузла.

Потім обираються та виводяться найбільш важливі речення у тому ж порядку, як вони розташовані у документі.

2.4 Дослідження особливостей технології whois

WHOIS — це протокол запитів/відповідей на основі TCP, який широко використовується для надання інформаційних послуг користувачам Інтернету. Хоча він спочатку використовувався для надання послуги «білих сторінок» та інформації про зареєстровані доменні імена, поточне розгортання охоплює набагато ширший спектр інформаційних послуг [23].

WHOIS є простим у використанні інструментом для перевірки доступності доменного імені. Якщо доменне ім'я доступне, ви можете зареєструвати його через реєстратора. Якщо доменне ім'я вже зайнято, ви можете побачити, хто його зареєстрував. Підтвердження зобов'язань вимагає від ICANN «запровадити заходи для підтримки своєчасного, необмеженого та публічного доступу до точної та повної інформації WHOIS». Служба WHOIS є безплатною та загальнодоступною. В цілому можна сказати що це каталог, що містить контактну та технічну інформацію зареєстрованих власників доменних імен (надалі «реєстранти»). Будь-хто, кому потрібно знати, хто стоїть за доменним іменем вебсайту, може подати запит на цю інформацію через WHOIS. Дані збираються та надаються реєстраторами та реєстрами згідно з умовами їхніх угод з ICANN [24].

WHOIS не єдиним централізованим реєстром даних. Навпаки, реєстраційні дані зберігаються в різних місцях і адмініструються кількома реєстрами та реєстраторами, які встановлюють власні правила для WHOIS відповідно до мінімальних вимог, визначених у контрактах з ICANN. На рисунку 2.3 демонструється приклад мережі співвідношень WHOIS.

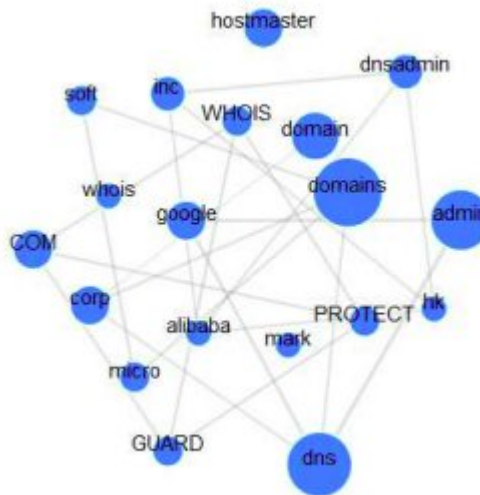


Рисунок 2.3 – Мережа співвідношень WHOIS [25]

Система WHOIS містить в собі клієнтів WHOIS, сервери WHOIS, сховища даних WHOIS і самі дані WHOIS (реєстраційні записи доменних імен). По суті, "WHOIS" може охоплювати будь-що з наступного:

- інформація, яка збирається під час реєстрації доменного імені або ресурсу IP-нумерації та згодом стає доступною через Службу WHOIS і, можливо, оновлюється протягом життя ресурсу;
- служби WHOIS, які надають публічний доступ до інформації про реєстрацію доменних імен, як правило, через програми, які реалізують протокол WHOIS або вебінтерфейс.

2.5 Висновки за розділом 2

Було розглянуто можливі підходи до реалізації вебзастосунку та досліджено які дані та метрики повинен демонструвати розроблений вебзастосунок. Також було проаналізовано особливості технологій PageRank, TextRank та WHOIS, що будуть реалізовані у вебзастосунку для збільшення можливостей програми.

3 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

3.1 Аналіз вимог

Проведений аналіз аналогів програми показав, що вона повинна відповідати наступним функціональним вимогам:

- надавати можливість вказати сторінки, що не треба оброблювати;
- демонструвати користувачу структуру введеного сайту;
- рахувати внутрішній pagerank сторінок;
- надавати повну інформацію о сайті;
- рахувати кількість посилань на різні файли;
- рахувати кількість унікальних посилань;
- рахувати кількість слів, символів, параграфів;
- рахувати кількість помилок при спробі отримати сторінку сайту;
- рахувати кількість різних http-кодів;
- рахувати час, що потрібен на отримання сторінки;
- отримувати та зберігати метадані сторінки;
- отримувати та зберігати текст сторінки;
- періодично оновлювати інформацію.

Окрім цього аналіз показав, що програма повинна забезпечувати наступні нефункціональні вимоги:

- зрозумілий і адаптивний інтерфейс;
- зручність введення інформації для початку роботи програми;
- зручність перегляду структури сайту;
- зручність перегляду інформації про сайт в цілому, та про окремі його сторінки;
- можливість легко доробити програму під нові вимоги.

За даними вимогами була розроблена діаграма, що демонструє можливості користувачів по роботі з програмою. Ця діаграма представлена на рисунку 3.1.

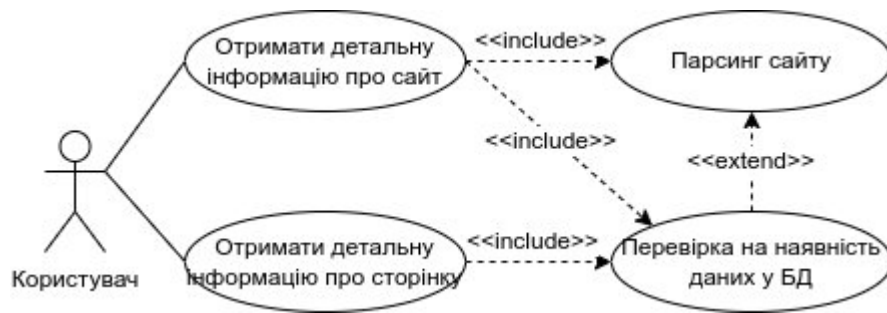


Рисунок 3.1 – Діаграма прецедентів користувача

3.2 Архітектура

Програма буде складатись з трьох основних частин: фронтенду, бекенду та з бази даних. Кожна з них відповідає за виконання певного функціонала, що допомагає досягнути цілей програми. Своєю чергою бекенд буде складатись з менших компонентів, таких як crawler та cron. Така архітектура дозволяє розробляти та підтримувати кожний компонент окремо, що полегшує подальшу розробку та підтримку програми.

Кожна з частин буде взаємодіяти з іншими сервісами та обмінюватися даними з ними, однак сервіси будуть проєктуватися з метою додання можливості подальшого масштабування. Це допоможе уникнути проблем залежності між сервісами та забезпечити більш гнучкий та масштабований підхід до розробки програмного забезпечення, що своєю чергою дозволить нам зробити дуже стійку систему.

У цілому, така архітектура дозволить розробникам програмного забезпечення зосередитися на конкретних функціях та сервісах і робити зміни та покращення окремих сервісів, не впливаючи на роботу інших.

Узагальнена архітектура системи представлена на рисунку 3.2. Цей рисунок містить блок-схему, що показує взаємодію між сервісами та типи переданих програмою даних.

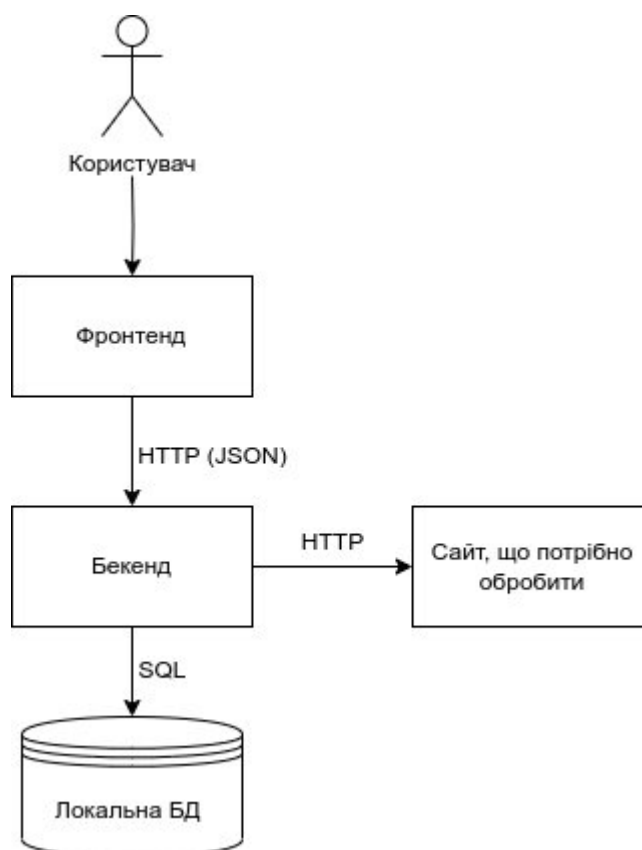


Рисунок 3.2 – Узагальнена архітектура системи

Фронтенд являє собою вебсторінку в браузері, що поставляється разом з програмою, та потрібна для графічного представлення даних, що поставляє бекенд.

Бекенд – це основне тіло програми, що відповідає за отримання запитів користувачів, їх обробку, отримання інформації з сайтів, її аналіз, збереження результатів в БД та формування відповіді на запит користувача, що буде містити усі необхідні дані. Окрім цього на бекенд покладається задача періодичного оновлення інформації для того, щоб користувач міг отримувати найактуальніші дані.

Локальна БД являє собою базу даних в якій буде зберігатися актуальна інформація про сайти, що дозволить зменшити навантаження на сервер, уникнути частих запитів даних з конкретного сайту та більш оперативно реагувати на запити користувачів.

Сайт, що потрібно обробити – це сайт, який хоче проаналізувати користувач.

3.3 Вибір інструментарію розробки

3.3.1 Особливості мови програмування Golang

Початковий дизайн Go було розпочато у вересні 2007 року Робертом Гріземером, Робом Пайком, Кеном Томпсоном та іншими. Go був офіційно анонсований у листопаді 2009 року, з впровадженнями, випущеними для платформ Linux і Mac OS X. Go є виразним, лаконічним, чистим і ефективним. Він швидко компілюється в машинний код, але має зручність збирання сміття та потужність відображення під час виконання. Це швидка статично типізована скомпільована мова, яка виглядає як динамічно типізована інтерпретована мова [26]. Вона відзначається декількома ключовими рисами, які роблять її привабливою для багатьох розробників:

- ефективність та швидкість. Go відома своєю високою швидкодією та низьким споживанням ресурсів. Це робить її ідеальною для розробки високонавантажених серверів та мікросервісів;
- конкурентність. Однією з сильних сторін Go є вбудована підтримка паралельного програмування через горутини та канали. Це дозволяє легко створювати ефективні багатозадачні програми;
- простота та читабельність. Синтаксис Go був спеціально розроблений для того, щоб бути чітким і простим у використанні. Це сприяє розвитку додатків, які легко зрозуміти та підтримувати;
- вбудована підтримка мультиплатформності. Golang надає можливість компілювати код для різних платформ, що дозволяє створювати кросплатформні додатки;
- вбудована підтримка тестування. Golang має вбудовану систему тестування, що дозволяє науковцям легко та ефективно валідувати свій код та впевнитися у його коректності. Це особливо важливо в наукових дослідженнях, де точність та надійність є критичними аспектами;

– сильна спільнота розробників. Зі спільнотою розробників Go постійно зростає, і вона активно підтримує мову, розробляє сторонні бібліотеки та публікує корисні статті та матеріали. Існують численні ресурси для вивчення та розвитку Go, такі як офіційна документація Go.

Go є дуже привабливою мовою для розробки високоефективних та надійних додатків, особливо в галузі серверної розробки. Завдяки простому синтаксису та вбудованим можливостям паралельного програмування, Go став популярним вибором для розробників, які шукають ефективні та швидкодіючі рішення. Префікс виклику функції ключовим словом `go` запускає асинхронний виклик функції. Ці асинхронні виклики функцій у Go називаються `goroutines`. `Goroutines` вважаються «легкими», і контекст середовища виконання Go перемикає їх на потоки операційної системи. Програмісти Go широко використовують `goroutines` як для симетричних, так і для асиметричних завдань. Дві або більше підпрограм можуть обмінюватися даними через передачу повідомлень або спільну пам'ять, що є найбільш часто використовуваним засобом передачі даних у Go [27].

Однією із найголовніших задач кожної мови програмування є створення системи керування пам'яттю. Загалом дуже важливо усунути накладні витрати програміста з достатньо низькими накладними витратами та без значної затримки. Це дозволить розробникам сконцентруватись на написанні коду, а не на ручному управлінні пам'яттю, що забирає достатньо багато часу та призводить до багатьох помилок. Потрібно пам'ятати, що Go дає програмісту значний контроль над розташуванням і розподілом пам'яті, набагато більше, ніж це типово для мов зі збиранням сміття [26].

Імпорт бібліотек досить унікальний у `Golang`. Замість того, щоб просто вказати назву бібліотеки, потрібно вказати посилання Github. На відміну від `Node.js` або `Python`, тут немає файлу зі списком усіх необхідних бібліотек. Для встановлення залежностей потрібен інтерфейс командного рядка `Golang`. Інтерфейс командного рядка проходить граф залежностей програм і визначає, які бібліотеки потрібно імпортувати [28].

Спільнота Go продовжує активно розвивати цю мову, створюючи нові бібліотеки та інструменти. Крім того, Go стала популярною мовою для розробки мікросервісів і контейнеризації завдяки інструментам, таким як Docker та Kubernetes.

У таблиці 3.1 наведено порівняння мови програмування Golang з іншими мовами програмування [29].

Таблиця 3.1 – Порівняння мов програмування

Категорія	Golang	Python	Java	C++
Синтаксис	Простий	Простий	Середній	Складний
Компіляція	Статична	Динамічна	Статична та динамічна	Статична
Пам'ять	Автоматичне збирання сміття	Автоматичне збирання сміття	Автоматичне збирання сміття	Ручне управління
ООП	Так	Так	Так	Так
Кросплатформа	Так	Так	Так	Так
Швидкодія	Високий рівень швидкості	Зазвичай повільніший	Середня швидкодія	Дуже швидкий
Веброзробка	Популярна	Популярна	Популярність зменшується	Не популярна через складність

Узагальнюючи попередні аргументи, вважаю, що Golang є оптимальним вибором мови програмування. Це рішення дозволить значно спростити процес розробки й підтримки проєкт завдяки чітко визначеному синтаксису та широкому спектру бібліотек, які пропонує Golang. Окрім того, висока ефективність виконання коду цією мовою робить її ідеальним вибором для проєкту з вимогами до продуктивності та масштабованості. У результаті, використання Golang забезпечує надійність і гнучкість реалізації, що робить її однією з найкращих мов програмування для розробки цього проєкту.

3.3.2 Вибір СКБД

PostgreSQL, MySQL, Oracle та SQL Server є популярними СКБД, які використовуються для збереження та управління даними в різних типах додатків та вебсайтів.

PostgreSQL - це потужна та розширювана СКБД з відкритим вихідним кодом, яка підтримує різні типи даних, включаючи географічні дані та JSON. PostgreSQL має високу надійність та продуктивність, а також багатий функціонал, що дозволяє його використовувати для великих проєкт. PostgreSQL підтримує широкий діапазон типів даних, включаючи власні типи, які можуть бути розширені користувачами. Ця СКБД також підтримує багатопотокову обробку і має гнучкі можливості налаштування. Вона відома своєю стабільністю та надійністю, а також підтримує масштабування та реплікацію [30].

MySQL - є однією з найпопулярніших СКБД у світі та також має відкритий вихідний код. Окрім цього MySQL має багато різних варіантів розповсюдження, включаючи комерційний та безплатний, простий та зрозумілий інтерфейс для адміністрування та налаштування баз даних, який дозволяє легко встановлювати та налаштовувати бази даних. Вона добре підходить для вебсайтів та додатків невеликих та середніх розмірів. MySQL підтримує масштабування, але може бути менш надійною та стабільною порівняно з PostgreSQL [31].

Oracle - це комерційна СКБД, яка використовується в бізнесі та великих корпораціях. Вона має високу продуктивність, широкий функціонал і підтримує багатопотокову обробку. Oracle має вбудовану підтримку мережевої реплікації, що дозволяє віддаленим користувачам взаємодіяти з СКБД безпосередньо. Oracle також має високі вимоги до обладнання, що може зробити його недоступним для менших компаній або проєктів з обмеженим бюджетом [32].

SQL Server - це СКБД, яка розробляється компанією Microsoft та використовується в середовищі Windows. Вона має високу продуктивність та забезпечує підтримку різноманітних функцій, що дозволяє її використовувати для великих підприємств. SQL Server має хорошу підтримку програмного забезпечення Microsoft, що дозволяє ефективно використовувати їх разом. Вона також має вбудований механізм планування запитів, що дозволяє зменшити час виконання запитів. SQL Server використовує пропрієтарний формат зберігання даних, що може зробити його менш переносним порівняно з PostgreSQL та MySQL. Окрім цього, SQL Server може бути менш розширеною та менш підходити для використання у вебсайтах та додатках невеликих та середніх розмірів [33].

Порівняння цих СКБД наведено у таблиці 3.2 [34].

Таблиця 3.2 – Порівняння СКБД

Категорія	PostgreSQL	MySQL	Oracle	SQL Server
Відкрите джерело	Так	Так	Ні	Ні
Ліцензія	PostgreSQL License	GPL	Proprietary	Proprietary
Розширюваність	Вертикальна	Вертикальна, комплексна	Вертикальна, горизонтальна	Вертикальна, комплексна
Підтримка SQL	Повна	Повна	Повна	Повна
Підтримка JSON	Так	Так	Так	Так
Підтримка XML	Так	Так	Так	Так
Підтримка NoSQL	Так	Часткова	Ні	Ні
Підтримка Full-text пошуку	Так	Так	Так	Так
Реплікація	Так	Так	Так	Так
Індексування	Високий	Високий	Високий	Високий
Підтримка зберігання BLOB	Так	Так	Так	Так
Швидкість роботи	Висока	Середня	Висока	Висока

Як базу даних було вирішено обрати об'єктно-реляційну систему керування базами даних PostgreSQL. Вона була обрана через те, що ця БД є дуже швидкою, містить багато різноманітного функціонала та є доволі популярною, а отже й багато розробників мають досвід у роботі з нею.

3.3.3 Visual Studio Code як основне середовище розробки проєкту

Visual Studio Code — це легкий, але потужний редактор вихідного коду, який працює на робочому столі та доступний для Windows, macOS і Linux. Він поставляється з вбудованою підтримкою JavaScript, TypeScript і Node.js і має багату екосистему розширень для інших мов і середовищ виконання (таких як C++, C#, Java, Python, PHP, Go, .NET) [35].

Крім загальної ідеї легкості та швидкого запуску, Visual Studio Code має доповнення коду IntelliSense для змінних, методів та імпортованих модулів, графічне налагодження, лінтування, редагування кількома курсорами, підказки параметрів та інші потужні функції редагування, швидку навігацію та можливість рефакторингу коду, а також підтримку Git. Більшу частину цього було адаптовано з технології Visual Studio.

Власне код Visual Studio створено з використанням оболонки Electron, Node.js, TypeScript і протоколу Language Server Protocol. Завдяки великій підтримці з боку розробників VSCode оновлюється щомісяця. Рівень підтримки розширень різниться для різних мов програмування починаючи від простого підсвічування синтаксису та підбору дужок до налагодження та рефакторингу. Однак, навіть якщо мовний сервер для мови програмування недоступний, кожен користувач може додати базову підтримку за допомогою розфарбовувачів TextMate.

Код Visual Studio Code є відкритим за ліцензією MIT. Однак сам продукт Visual Studio Code постачається за стандартною ліцензією на продукти Microsoft, оскільки він має невеликий відсоток індивідуальних

налаштувань Microsoft. Однак, попри комерційну ліцензію, VSCode залишається повністю безплатним продуктом.

Порівняння VSCode з іншими популярними IDE наведено у таблиці 3.3.

Таблиця 3.3 – Порівняння популярних IDE

Категорія	VSCode	Sublime Text	IntelliJ IDEA	Atom
Відкрите джерело	Так	Ні	Ні	Так
Швидкість роботи	Швидка	Дуже швидка	Середня	Швидка
Підтримка різних мов програмування	Широка	Широка	Середня	Широка
Відладка	Так	Ні	Так	Так
Підтримка Git	Так	Так	Так	Так
Підтримка Docker	Так	Ні	Так	Так
Підтримка IntelliSense	Так	Ні	Так	Так
Вбудований термінал	Так	Ні	Так	Так
Ціна	Безплатний	Платний	Платний	Безплатний

Розглядаючи усе зазначене вище, можна стверджувати, що VSCode є дуже гідним вибором для створення програм мовою програмування Go. Його високий рівень кастомізації та розширюваності завдяки багатофункціональній системі плагінів робить його ідеальним інструментом для програмістів. Кожен користувач може налаштувати й доповнити його функціонал під свої власні потреби та вимоги, створюючи при цьому ефективне та зручне середовище розробки, що налаштовано саме під нього. Таким чином, VSCode відкриває широкі можливості для розробників Go, допомагаючи їм швидко і продуктивно створювати якісний програмний код.

3.4 Висновки за розділом 3

Було розглянуто особливості мови програмування Golang, оглянуто функціонал та можливості середовища розробки Visual Studio Code, проаналізовано плюси та мінуси різних СКБД. Також було описано архітектуру програми, розроблено схему таблиць БД та розроблено загальну схему роботи проєкту.

4 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

4.1 Розробка алгоритму функціонування програми

Після аналізу вимог до проєктованої системи та розробки архітектури системи було створено загальний алгоритм функціонування бекенду програми, який продемонстровано на рисунку 4.1.

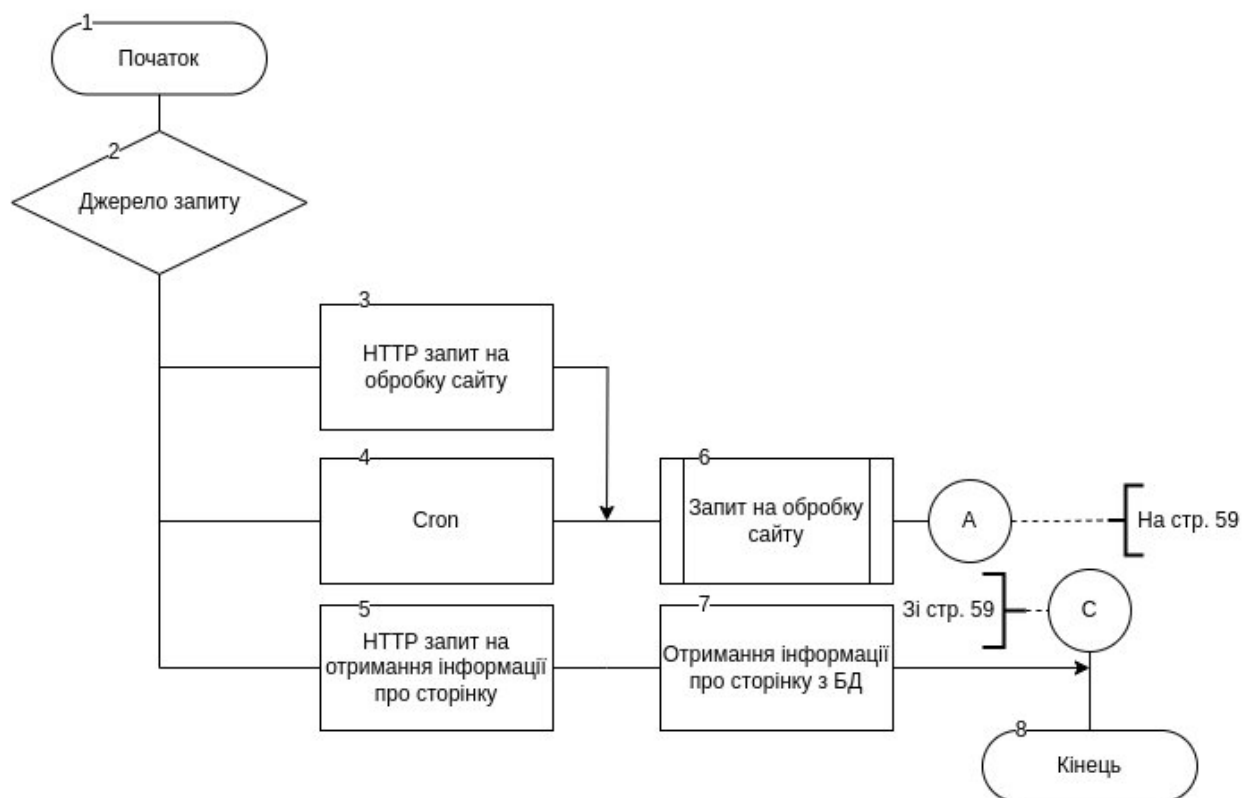


Рисунок 4.1 – Загальний алгоритм функціонування бекенду програми

У блоці №2 рисунку 4.1 передбачено, що джерелом запиту може бути як користувач, що відправить запит через фронтенд, так і сторп, який спрацює через проміжок часу.

У блоках №3 та №5 рисунку 4.1 передбачено, що користувач буде робити запит через фронтенд або через будь-який інший інструмент для посилання http запитів та сервер повинен відреагувати на нього належним чином.

У блоці №4 рисунку 4.1 передбачено, що через встановлений у налаштуваннях проміжок часу запуститься cron, та почне оновлювати дані у БД завдяки блоку №6.

Блок №6 рисунку 4.1 являє собою функцію, що повинна виконуватися при http запиті користувача на сервер. Детальний алгоритм цієї функції показано на рисунку 4.2.

У блоці №7 рисунку 4.1 передбачено отримання збереженої інформації про сторінку з БД, та передачу цієї інформації користувачу.

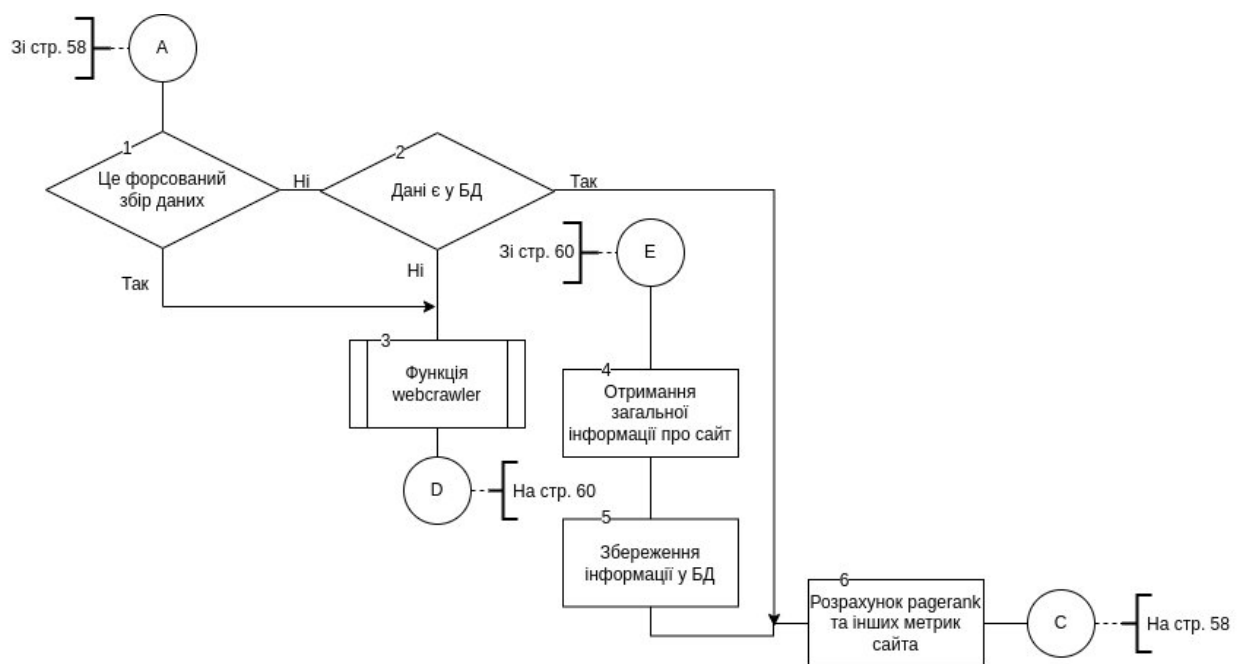


Рисунок 4.2 – Запит на обробку сайту

У блоці №1 рисунку 4.2 передбачено перевірку чи форсований збір даних. Якщо користувач вирішив оновити інформацію не чекаючи коли це зробить cron, то програма переходить до виконання блоку №3. В іншому випадку виконується блок №2.

У блоці №2 рисунку 4.2 передбачено перевірку чи є дані по цьому сайту у БД. Якщо дані наявні та це не форсований збір, то програма переходить до виконання блоку №6. В іншому випадку виконується блок №3.

Блок №3 рисунку 4.2 являє собою функцію, що сторінку з сайту, та отримує з неї корисну інформацію. Детальний алгоритм цієї функції

представлений на рисунку 4.3. Після виконання цієї функції починається виконання блоку №4.

Блок №4 рисунку 4.2 отримує загальну інформацію про сайт. До загальної інформації відноситься інформація типу: власник сайту, його email, строки створення, оновлення тощо. Далі виконання програми переходить до блоку №5.

Блок №5 рисунку 4.2 відповідає за збереження всієї отриманої раніше інформації у БД. Далі виконання програми переходить до блоку №6.

Блок №6 рисунку 4.2 відповідає за розрахунок динамічної інформації. До такої інформації відноситься кількість слів, символів, параграфів на всіх сторінках, pagerank, кількість помилок при спробі отримати сторінку тощо. Далі виконання програми закінчується і дані повертаються користувачу.

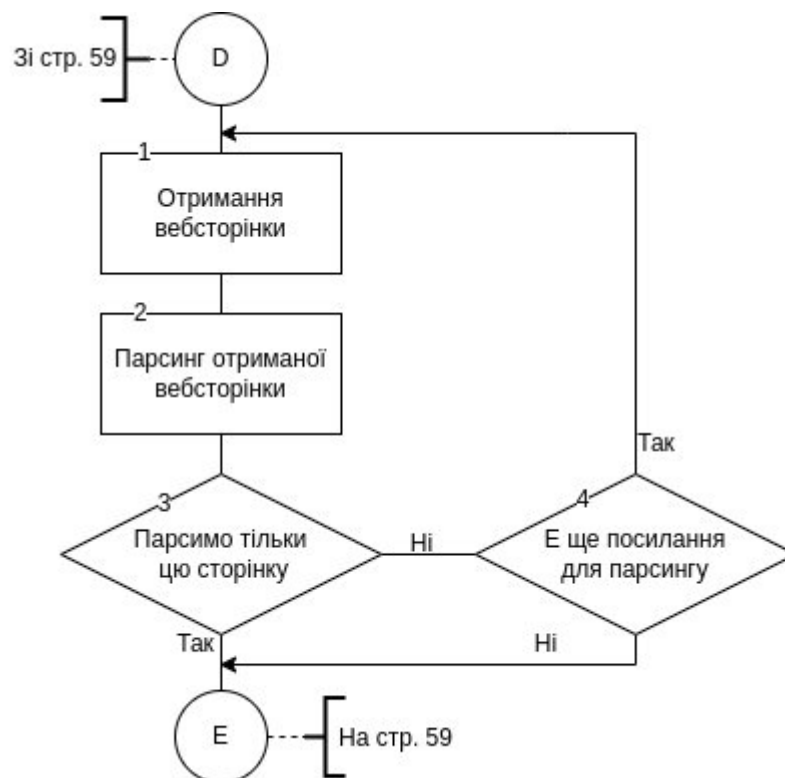


Рисунок 4.3 – Функція webcrawler

Блок №1 рисунку 4.3 отримує вебсторінку. Сторінка являє собою html документ з інформацією. Далі виконання програми переходить до блоку №2.

Блок №2 рисунку 4.3 парсить отриману вебсторінку і витягує з неї всю корисну для SEO інформацію. Далі інформація сортується за категоріями та записується у відповідні поля. Далі виконання програми переходить до блоку №3.

У блоці №3 рисунку 4.3 передбачено перевірку чи парсимо ми лише одну сторінку. Якщо ми хочемо распарсити увесь сайт, то програма переходить до виконання блоку №4. В іншому випадку функція завершує роботу і повертає зібрану інформацію.

У блоці №4 рисунку 4.3 передбачено перевірку чи є в нас ще не перевірені сторінки для парсингу. Якщо такі сторінки ще є, то програма переходить до виконання блоку №1. В іншому випадку функція завершує роботу і повертає зібрану інформацію.

4.2 Структура підсистеми

Всього в програмі існує 2 головних сервіси: Cron та Crawler. Їх опис наведений у таблиці 4.1. Окрім них в програмі також існує вебсервер, що приймає http запити та СКБД PostgreSQL, що зберігає дані. Їх взаємодія, а також внутрішній функціонал продемонстровано на рисунку 4.4.

Таблиця 4.1 – Складові програми

Назва сервісу	Опис
Cron	Цей сервіс відповідає за періодичне оновлення даних сайтів.
Crawler	Цей сервіс відповідає за парсинг, категоризацію та структурування даних.

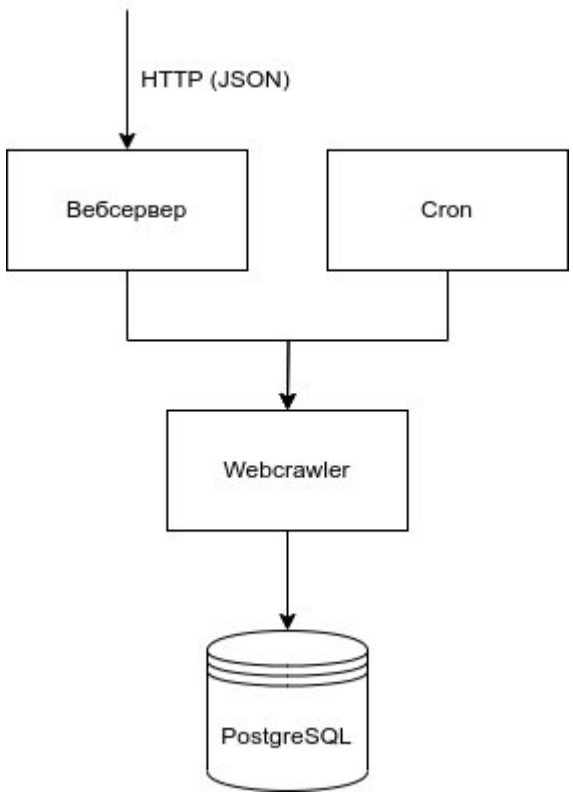


Рисунок 4.4 – Структура підсистеми

4.3 Опис особливостей програмної реалізації

4.3.1 Функціонал сервісу Crawler

Сервіс crawler складається з чотирьох функцій, що виконують базовий функціонал для парсингу сайту. Назви функцій та їх опис представлені у таблиці 4.2.

Таблиця 4.2 – Складові блоку Crawler

Назва функції	Опис
1	2
Start	Відповідає за отримання інформації про сайт, запуск функції PageWalker, розрахунок динамічних параметрів та збереження даних. На виході віддає всю інформацію про сайт.

Продовження таблиці 4.1

1	2
PageWalker	Відповідає за отримання інформації зі сторінки, запуск функції ParsePage та рекурсивний обхід посилань на інші сторінки. На виході віддає структуру сайту зі всією інформацією по кожній сторінці.
ParsePage	Відповідає за парсинг вебсторінки, запуск функції crawlerFunc та сортування посилань сторінки й видалення дублікатів. На виході віддає підготовлену інформацію сторінки.
crawlerFunc	Відповідає за проходження по всім тегам, видобуток тексту та посилань. На виході віддає підготовлену інформацію сторінки.

4.3.2 Структура БД

Після ретельного аналізу вимог та створення алгоритму функціонування програми було прийняте рішення щодо побудови структури бази даних. Визначено, що необхідно створити дві таблиці: "link_data", для зберігання інформації про посилання, та "sites", для утримання даних про вебсайти. Це рішення максимально відповідає необхідним для реалізації програми вимогам і забезпечить ефективне управління даними.

Таблиця "link_data" буде містити основну інформацію про кожну сторінку, посилання на попередню сторінку, відсортовані по категоріях посилання та час відклику сторінки.

Таблиця "sites" буде містити всю інформацію з сайту, базове посилання, дати створення, оновлення тощо.

Структура та зв'язки всіх таблиць показані на рисунку 4.5.

sites		link_data	
domain_id	text	link 	text
base_url 	text	status_code	int
punycode	text	error	text
dns_sec	boolean	text	text
name_servers	text[]	images	text[]
status	text[]	audio	text[]
whois_server	text	video	text[]
created_date	text	hyperlinks	text[]
updated_date	text	internal_links	text[]
expiration_date	text	metadata	json
exclude	text[]	parent_link	text
headers	json	fonts	text[]
registrar	json	files	text[]
registrant	json	ping	numeric
administrative	json		
technical	json		
billing	json		

Рисунок 4.5 – Загальна структура всіх таблиць у БД

4.3.3 Інтерфейс програми

Інтерфейс програми являє собою html сторінку, що підтримує необхідний для роботи програми функціонал. Базовий її варіант можна запустити з папки frontend у проєкті. Зовнішній вигляд інтерфейсу представлений на рисунку 4.6.

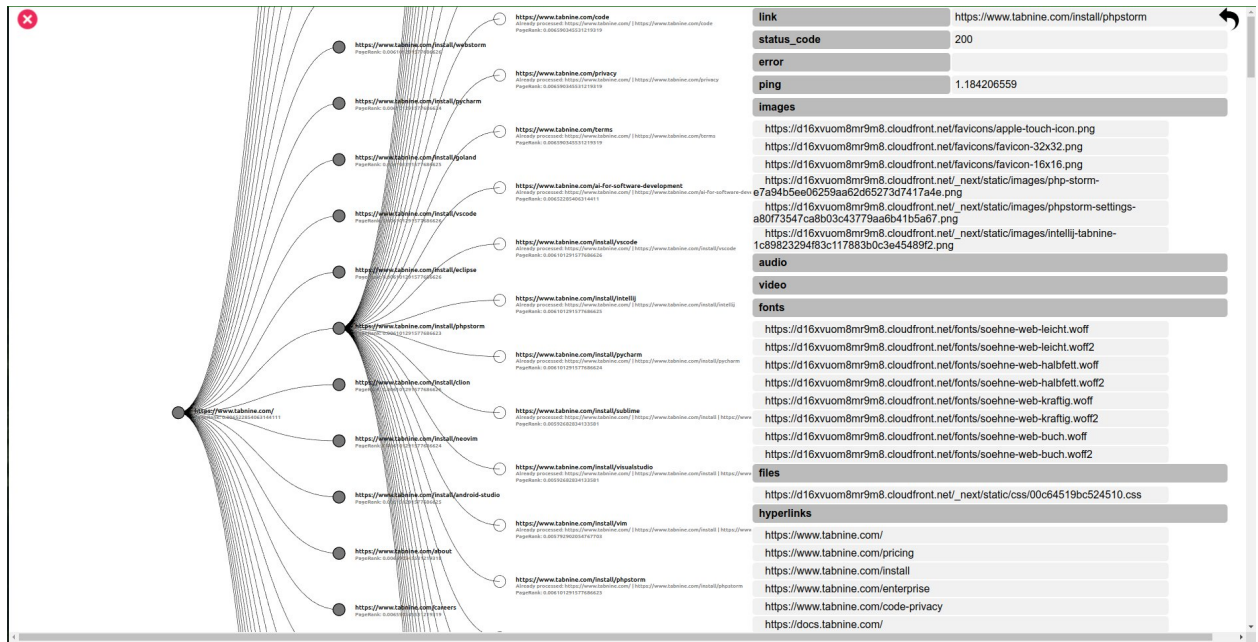


Рисунок 4.6 – Зовнішній вигляд базового інтерфейсу програми

4.4 Висновки за розділом 4

В розділі було розглянуто основні рішення щодо реалізації програми, наведено детальний список розроблених сервісів та їх базовий функціонал. Окрім цього, було запропоновано інтерфейс програми.

5 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

5.1 Опис застосування

Цей вебзастосунок призначений для реалізації інструменту для SEO-спеціалістів мовою програмування Golang. Він може використовуватися для аналізу сайту, його окремих сторінок, отримання внутрішнього pagerank і в результаті для підвищення рейтингу сайту у пошукових системах.

Крім того, завдяки тому, що цей проєкт розміщений у репозиторії GitHub, будь-який розробник може використовувати його як основу для власних проєктів.

Для використання проєкту потрібен комп'ютер, що здатен запускати та підтримувати контейнери Docker. Окрім цього може знадобитися встановлена мова програмування Golang версії не меншої за 1.21. Для запуску проєкту використовувався персональний IBM-сумісний комп'ютер з процесором AMD Ryzen 5 5500U, оперативною пам'яттю 32 Гб, 15.6'' монітором, клавіатурою та маніпулятором типу «миша», що функціонує під управлінням операційної системи Ubuntu 23.04 x86_64.

5.2 Інструкція з експлуатації програми

5.2.1 Керівництво системного програміста

Перед спробою запуску програми потрібно перевірити коректність налаштувань та створити .env файл поряд зі скомпільованим бінарним. Приклади файлів знаходяться у файлі під назвою .env.example. Для створення .env файлу достатньо перейменувати файл .env.example у .env та змінити налаштування у самому файлі відкривши його через будь-який текстовий редактор.

Для коректного запуску програми потрібно скомпілювати та запустити програму піднявши до цього контейнер з PostgreSQL та переконавшись в

правильності налаштувань. Це також треба мати на увазі при спробі запустити проєкт через Docker.

Протестувати програму можна завдяки вбудованим засобам тестування Golang, через програму Postman або її аналоги, або через спрощений інтерфейс, що надається разом з програмою.

5.2.2 Керівництво програміста

Для розробки проєкту використовувалося IDE Microsoft Visual Studio Code. Для подальшої розробки або перероблення цієї програми рекомендовано використовувати саме її, оскільки програма вже містить всі необхідні налаштування для роботи з цією IDE.

Загалом проєкт складається з 11 папок: “.vscode”, “cmd”, “config”, “frontend”, “internal”, “migrations”, “pkg”, “server”, “settings”, “structs”, “tools”. Окрім цього в корені є 5 файлів: “.env.example”, “main.go”, “.gitignore”, “go.mod”, “go.sum”. Вся структура представлена на рисунку 5.1.

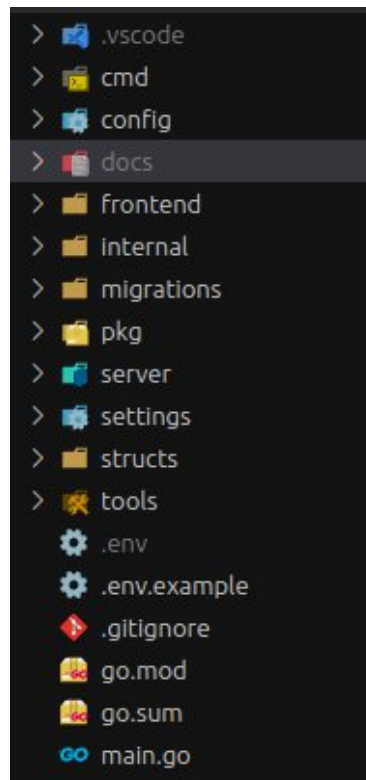


Рисунок 5.1 – Структура кореневої папки програми

В таблиці 5.1 представлений детальний опис елементів кореневої папки проєкту.

Таблиця 5.1 – Опис елементів кореневої папки проєкту

Назва елементу	Опис
.vscode	В цій папці знаходяться налаштування для запуску програми у режимі налагоджування
cmd	В цій папці знаходиться точка запуску всіх частин програми
config	В цій папці знаходиться структури для конфігурації проєкту
frontend	В цій папці знаходиться файли, які потрібні для запуску вебсторінки.
internal	В цій папці знаходяться сервіси Cron та Crawler
migrations	В цій папці знаходяться міграції для БД
pkg	В цій папці знаходиться ефункціона сторонніх бібліотек
server	В цій папці знаходиться точка запуску вебсервера
settings	В цій папці знаходяться налаштування проєкту,що не були винесені у .env файл
structs	В цій папці знаходяться структури, що використовуються по всьому проєкту
tools	В цій папці знаходяться інструменти для роботи програми
main.go	Точка входу в програму
.env.example	Файл з прикладом налаштувань

Продовження таблиці 5.1

1	2
.gitignore	Цей файл зберігає в собі назви та шаблони файлів, що не повинні потрапити у GitHub
go.mod	В цьому файлі знаходяться залежності проєкту від інших бібліотек. Окрім цього там зберігається системна інформація щодо назви проєкту та версії Golang
go.sum	В цьому файлі знаходяться хеші залежностей проєкту від інших бібліотек

Розглянемо папку frontend. Її вміст показано на рисунку 5.2.

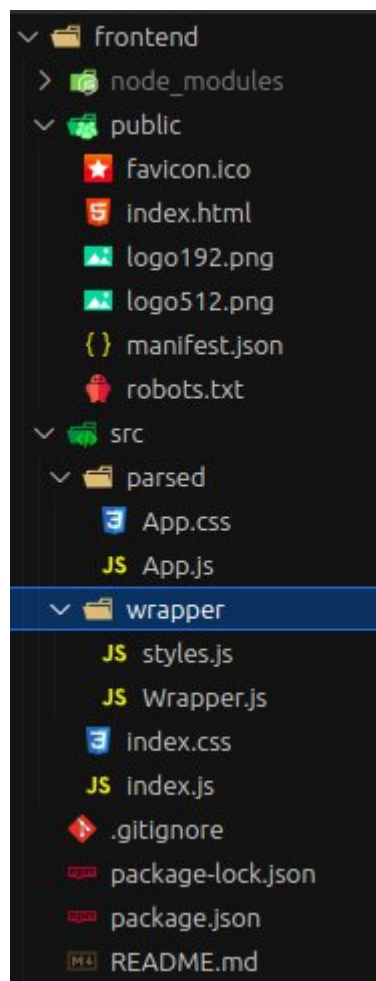


Рисунок 5.2 – Структура папки frontend

В таблиці 5.2 представлений детальний опис елементів папки public.

Таблиця 5.2 – Опис елементів папки public

Назва елементу	Опис
favicon.ico, logo192.png, logo512.png	Іконки програми у різних розмірах
index.html	Шаблон вебсторінки
robots.txt	Інформація для пошукових движків
manifest.json	Інформація про вебсторінку

В таблиці 5.3 представлений детальний опис елементів папки src.

Таблиця 5.3 – Опис елементів папки src

Назва елементу	Опис
parsed/App.css	В цьому файлі вказані стилі які повинні застосовуватися до сторінки з інформацією про опрацьований сайт
parsed/App.js	В цьому файлі вказана структура сторінки з інформацією про опрацьований сайт, а також код, що бере цю інформацію з серверу та реагує на дії користувача
wrapper/styles.js	В цьому файлі вказані стилі які повинні застосовуватися до сторінки з вводом параметрів для подальшої обробки сайту
index.css	В цьому файлі вказані стилі які повинні застосовуватися до всіх сторінок
index.js	В цьому файлі вказана основа структури сторінки

Продовження таблиці 5.3

1	2
package.json	Це файл з переліком використаних бібліотек
wrapper/Wrapper.js	В цьому файлі вказана структура сторінки з вводом параметрів для подальшої обробки сайту, а також код, що відправляє цю інформацію на сервер та реагує на дії користувача

Розглянемо папку internal. Її вміст показано на рисунку 5.3.

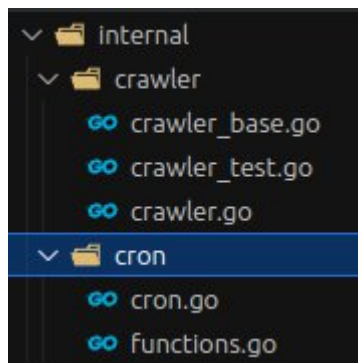


Рисунок 5.3 – Структура папки internal

В таблиці 5.4 представлений детальний опис елементів папки cron.

Таблиця 5.4 – Опис елементів папки cron

Назва елементу	Опис
cron.go	В цій папці знаходиться точка старту cron
functions.go	В цій папці знаходяться функції, які буде викликати cron

В таблиці 5.5 представлений детальний опис елементів папки crawler.

Таблиця 5.5 – Опис елементів папки crawler

Назва елементу	Опис
crawler_base.go	В цьому файлі знаходяться функції парсингу сторінок сайту
crawler_test.go	В цьому файлі знаходяться функції тестування
crawler.go	В цьому файлі знаходиться точка входу для парсингу сайту

Розглянемо папку pkg. Її вміст показано на рисунку 5.4.

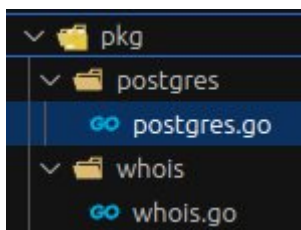


Рисунок 5.4 – Структура папки pkg

В таблиці 5.6 представлений детальний опис елементів папки pkg.

Таблиця 5.6 – Опис елементів папки sgop

Назва елементу	Опис
postgres/postgres.go	В цьому файлі знаходяться функції запису та читання даних з БД
whois/whois.go	В цьому файлі знаходиться код для роботи з WHOIS сервісом

Розглянемо папку server. Її вміст показано на рисунку 5.5.

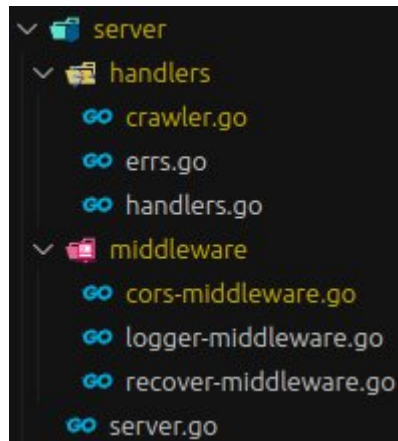


Рисунок 5.5 – Структура папки server

В таблиці 5.7 представлений детальний опис елементів папки server

Таблиця 5.7 – Опис елементів папки server

Назва елементу	Опис
handlers	В цій папці знаходяться функції, що отримують та виконують запити користувача
middleware	В цій папці знаходяться допоміжні функції для кращої роботи сервера
server.go	Цей файл є точкою запуску вебсерверу

В таблиці 5.8 представлений детальний опис елементів папки handlers.

Таблиця 5.8 – Опис елементів папки handlers

Назва елементу	Опис
crawler.go	В цьому файлі знаходяться функції для запитів пов'язаних з обробкою сторінки
errs.go	В цьому файлі знаходяться функції пов'язані з обробкою помилок
handlers.go	В цьому файлі знаходиться точка входу для функцій виконання запитів користувача

В таблиці 5.9 представлений детальний опис елементів папки middleware.

Таблиця 5.9 – Опис елементів папки middleware

Назва елементу	Опис
cors-middleware.go	В цьому файлі знаходиться функція налаштувань cors політик
logger-middleware.go	В цьому файлі знаходиться функція логування запитів користувачів
recover-middleware.go	В цьому файлі знаходиться функція відновлення роботи серверу при критичних помилках

5.2.3 Керівництво оператора

Після запуску програми необхідно перейти за адресою, яка вказана у файлах конфігурації (за замовчуванням це 127.0.0.1:1140). Після цього користувач побачить вікно вводу адреси, наведене на рисунку 5.6.

Parse site

Site url

Only this page

☐

Force collect

☐

Exclude url

Remove exclude

Submit

Reset

Рисунок 5.6 – Вікно вводу адреси

В цьому вікні необхідно ввести адресу сайту, додати адреси, що потрібно виключити з опрацювання, визначити чи будемо ми парсити тільки одну сторінку та чи будемо форсувати перезбір інформації. Після цього користувач побачить головний інтерфейс загрузки, що представлений на рисунку 5.7.



Рисунок 5.7 – Інтерфейс загрузки програми

Після того, як бекенд опрацює інформацію, користувач побачить основний інтерфейс програми. Цей інтерфейс наведений на рисунку 5.8.

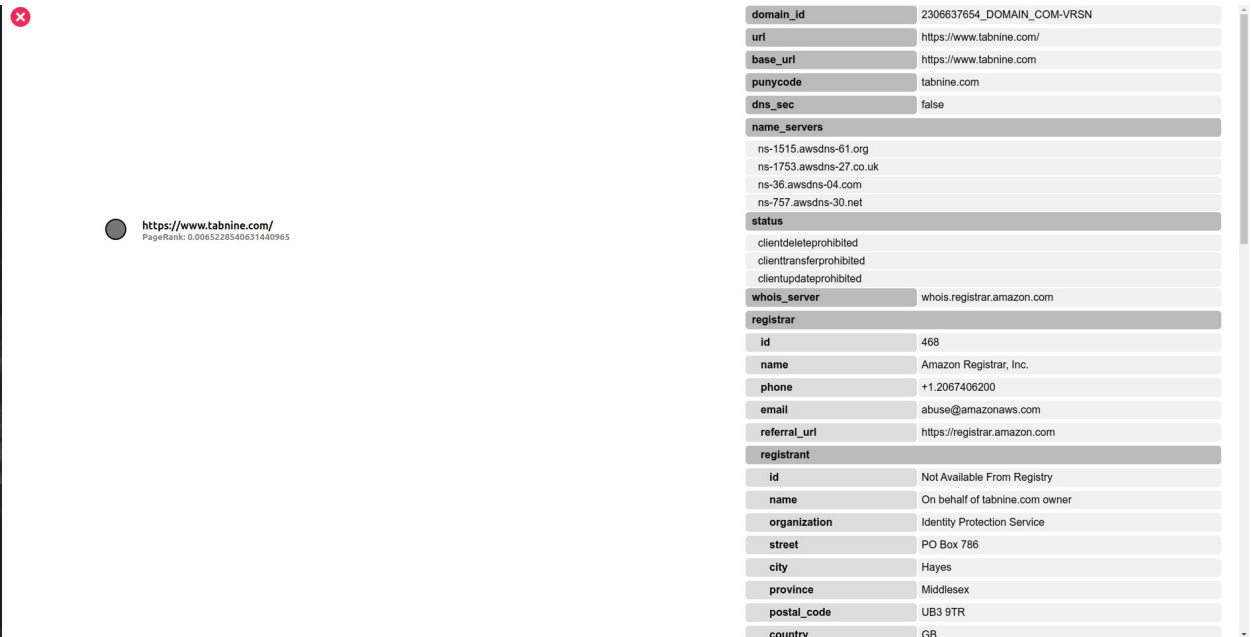


Рисунок 5.8 – Основний інтерфейс програми

Після цього користувач може натиснути на кружечок, що являє собою сторінку сайту, і побачить всі посилання, що йдуть з нього, та інформацію о цій вебсторінці. Приклад наведений на рисунку 5.9.

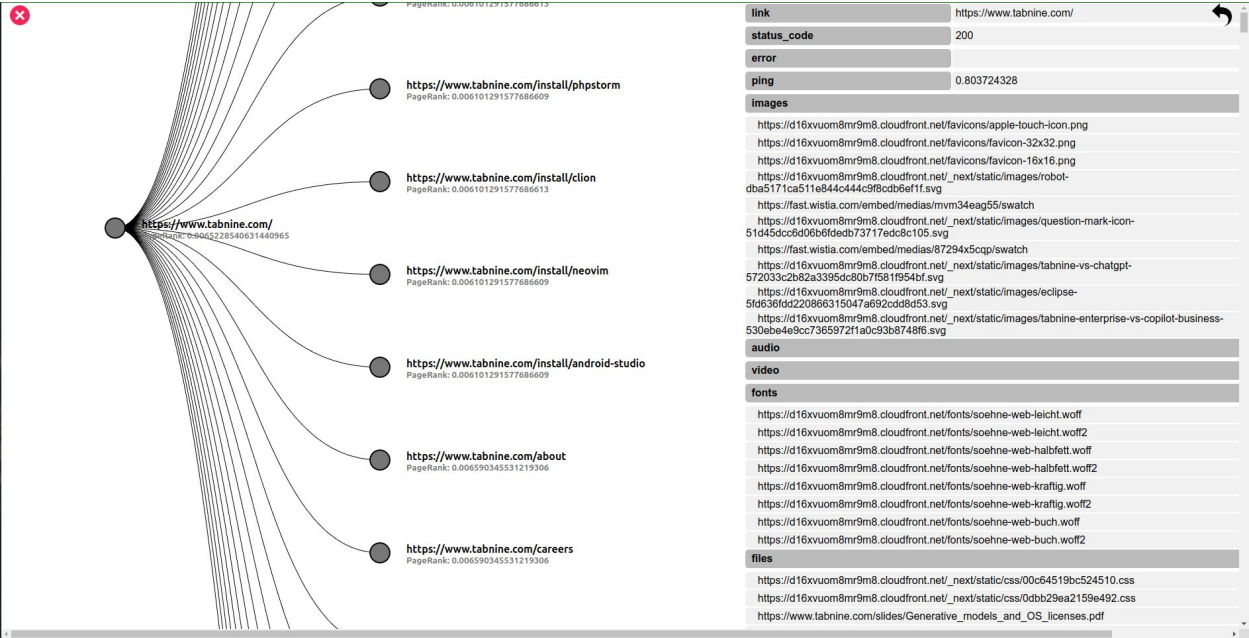


Рисунок 5.9 – Інформація про сторінку сайту

5.3 Висновки за розділом 5

В розділі було розглянуто призначення та способи застосування розробленого вебзастосунку, описані системні вимоги, надані інструкції з експлуатації програми для системного програміста, програміста та користувача.

ВИСНОВКИ

В даній роботі були вирішені наступні завдання:

- проаналізовані аналоги створюваної програми;
- досліджені технології, що планувалися для використання;
- вивчена взаємодія з мовою програмування Golang;
- досліджені методи роботи з web mining;
- проаналізовані алгоритми роботи PageRank, TextRank та WHOIS;
- створена реалізація програми для видобутку даних для SEO;
- протестовано створену програму.

В результаті проведеної роботи було створено програмну реалізацію вебзастосунку, що інтегрує в собі методи PageRank, TextRank та WHOIS, а також технологію web mining, для комплексного аналізу всіх сторінок сайту для потреб SEO-спеціалістів.

Web mining є дуже корисною технологією, що дозволяє отримувати дані з сайтів, категоризувати їх, аналізувати та робити висновки для покращення функціонала на основі обробленої інформації.

Pagerank допомагає виявити сторінки вірогідність потрапити на які є найбільшою. Завдяки цьому можна передбачити які на які сторінки варто посилатися частіше і що користувач буде бачити або купляти з більшою ймовірністю.

Для створення невеликих програм дуже зручно використовувати мову програмування Golang через її простоту, швидкість та легкість.

Розробленою програмною реалізацією може користуватися абсолютно будь-яка людина або компанія для отримання інформації та аналізу сайтів для подальшого покращення рейтингу сайту. Завдяки створеній архітектурі дана програмна реалізація є легко масштабованою.

Проведені випробування показують, що створена програмна реалізація відповідає всім вимогам замовника та являється економічно доцільною для впровадження.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Shyam Nandan Kumar. World towards Advance Web Mining: A Review / Shyam Nandan Kumar // American Journal of Systems and Software / Shyam Nandan Kumar. – Bhopal: Science and Education Publishing, 2015. – С. 44–61.
2. Santosh Kumar. A Study on Different Aspects of Web Mining and Research Issues / Santosh Kumar, Ravi Kumar., 2021. – (IOP Conference Series: Materials Science and Engineering).
3. Prem Sagar Sharma. Web Page Ranking Using Web Mining Techniques: A Comprehensive Survey / Prem Sagar Sharma, Divakar Yadav, R. N. Thakur., 2022. – 19 с.
4. Ron Berman. The Role of Search Engine Optimization in Search Marketing / Ron Berman, Zsolt Katona., 2022. – 18 с. – (Marketing Science).
5. Harold Davis. Search Engine Optimization Building Traffic and Making Money with SEO / Harold Davis., 2006. – 41 с.
6. Venkat N. Gudivada. Understanding Search-Engine Optimization / Venkat N. Gudivada, Dhana Rao, Jordan Paris., 2015. – 10 с.
7. Lukas Felbinger. State-of-the-art white hat SEO methods / Lukas Felbinger. – St. Pölten, 2018. – 5 с.
8. Ihya Syihabul Fadli. Implementation of White Hat SEO-Based Digital Incubator Platform / Ihya Syihabul Fadli, Nuke Puji Lestari, Ananda Alifia Putri // International Transactions on Education Technology / Ihya Syihabul Fadli, Nuke Puji Lestari, Ananda Alifia Putri., 2023. – С. 122–130.
9. Browseo [Electronic resource]. – Access mode: <https://www.browseo.net>.
10. Clusteric [Electronic resource]. – Access mode: <http://clusteric.com/>.
11. ContentKing [Electronic resource]. – Access mode: <https://www.contentkingapp.com>.
12. Google Analytics [Electronic resource]. – Access mode: <https://marketingplatform.google.com/about/analytics/>.

13. Lighthouse [Electronic resource]. – Access mode: <https://developer.chrome.com/docs/lighthouse/overview/>.
14. Apify [Electronic resource]. – Access mode: <https://apify.com>.
15. Different Types of SEO Tools [Electronic resource]. – Access mode: <https://www.firstpagemarketing.com/blog/different-types-of-seo-tools/>.
16. Performance comparison among local and foreign universities websites using seo tools / Kamran Rasheed, Muhammad Noman, Muhammad Imran та ін.]. – Pakistan, 2018. – 14 с. – (Ictact Journal On Soft Computing).
17. Xueyuan Tan. A new extrapolation method for PageRank computations / Xueyuan Tan. // Journal of Computational and Applied Mathematics. – 2016. – №313. – С. 383–392.
18. A simple illustration of the Pagerank algorithm [Electronic resource]. – Access mode: <https://en.wikipedia.org/wiki/File:PageRanks-Example.svg>.
19. Andreas Thalhammer. PageRank on Wikipedia: Towards General Importance Scores for Entities / Andreas Thalhammer, Achim Rettinger // Lecture Notes in Computer Science / Andreas Thalhammer, Achim Rettinger., 2016. – (AIFB, Karlsruhe Institute of Technology). – С. 227–240.
20. Variations of the Similarity Function of TextRank for Automated Summarization / Federico Barrios, Federico L'opez, Luis Argerich, Rosita Wachenchauzer. – 2016.
21. Representation how works the simplest text ranking algorithm [Electronic resource]. – Access mode: <https://camo.githubusercontent.com/677ac753ab40abaf9dcc91a50d90fde8d73190a48d32baab6a279208d31d6a43/68747470733a2f2f692e696d6775722e636f6d2f5255644466427a2e6a7067>.
22. An Empirical Study of TextRank for Keyword Extraction / Mingxi Zhang, Xuemin Li, Shuibo Yue, Liuqian Yang. – 2020.
23. L. Daigle. WHOIS Protocol Specification [Electronic resource]. / L. Daigle // Network Working Group. – 2004. – Access mode: <https://datatracker.ietf.org/doc/html/rfc3912>.

24. Rahul Gautam. Analysis and implementation of whois domain lookup / Rahul Gautam. – Palwal: Advanced Institute of Technology and Management, 2016. – 7 c. – (International Journal of Technical Research & Science).
25. Masahiro Kuyama. Method for Detecting a Malicious Domain by using WHOIS and DNS features / Masahiro Kuyama, Yoshio Kakizaki, Ryoichi Sasaki // The Third International Conference on Digital Security and Forensics / Masahiro Kuyama, Yoshio Kakizaki, Ryoichi Sasaki. – Tokyo: The Society of Digital Information and Wireless Communications, 2016. – (Asia Pacific University of Technology and Innovation). – C. 74–80.
26. Bareen Shaikh K. Quantitative evaluation by applying metrics to existing “Go” with other Programming Language / Bareen Shaikh K, Sangeeta Mahesh Borde. – Alandi: MIT College Of Arts, Commerce & Science, 2013. – 5 c.
27. Milind Chabbi. A Study of Real-World Data Races in Golang / Milind Chabbi, Murali Krishna Ramanathan. – 2022. – C. 1–16.
28. Daniel Hunt. Recreating top using golang / Daniel Hunt., 2018. – 30 c.
29. Aman Goel. Best Programming Language to Learn in 2020 (for Job & Future) / Aman Goel. – 2020. – C. 1–13.
30. PostgreSQL [Electronic resource]. – Access mode: <https://www.postgresql.org>
31. MySQL [Electronic resource]. – Access mode: <https://www.mysql.com>.
32. Oracle [Electronic resource]. – Access mode: <https://www.oracle.com/database/>
33. SQL Server [Electronic resource]. – Access mode: <https://www.microsoft.com/en-us/sql-server>
34. Comparing Database Management Systems: MySQL, PostgreSQL, MSSQL Server, MongoDB, Elasticsearch, and others [Electronic resource]. – 2023. – Access mode: <https://www.altexsoft.com/blog/business/comparing-database-management-systems-mysql-postgresql-mssql-server-mongodb-elasticsearch-and-others/>

35. Documentation for Visual Studio Code [Electronic resource]. – Access mode: <https://code.visualstudio.com/docs>

ДОДАТОК А
Технічне завдання

Вступ

Програмне забезпечення направлене на допомогу SEO-спеціалістам у підвищенні рейтингу сайтів через надання їм необхідної інформації шляхом видобутку цієї інформації з сторінок сайту. Завдяки вбудованим інструментам та технологіям SEO-спеціаліст отримує дуже велику кількість корисної інформації за невеликий проміжок часу.

A.1 Підстави для розробки

Підставою для розробки було завдання на дипломну кваліфікаційну роботу за темою «Дослідження та програмна реалізація технології web mining для потреб SEO» на основі наказу № 437 від 14 листопада 2023 р. за Національним університетом «Запорізька політехніка».

A.2 Призначення розробки

Розробка призначена для допомоги SEO-спеціалістам у підвищенні рейтингу сайту через надання їм необхідної інформації шляхом видобутку цієї інформації зі сторінок сайту.

A.3 Основні вимоги до програми

A.3.1 Вимоги до функціональних характеристик

Вимоги до функціональних характеристик складаються з наступних:

- надавати можливість вказати сторінки, що не треба оброблювати;
- демонструвати користувачу структуру введеного сайту;
- рахувати внутрішній pagerank сторінок;
- надавати повну інформацію о сайті;
- рахувати кількість посилань на різні файли;

- рахувати кількість унікальних посилань;
- рахувати кількість слів, символів, параграфів;
- рахувати кількість помилок при спробі отримати сторінку сайту;
- рахувати кількість різних http-кодів;
- рахувати час, що потрібен на отримання сторінки;
- отримувати та зберігати метадані сторінки;
- отримувати та зберігати текст сторінки;
- періодично оновлювати інформацію.

Окрім цього аналіз показав, що програма повинна забезпечувати наступні нефункціональні вимоги:

- зрозумілий і адаптивний інтерфейс;
- зручність введення інформації для початку роботи програми;
- зручність перегляду структури сайту;
- зручність перегляду інформації про сайт в цілому, та про окремі його сторінки;
- можливість легко доробити програму під нові вимоги.

А.3.2 Вимоги до надійності

Надійність роботи системи має забезпечуватися надійністю організації технічної складової серверної сторони, надійністю використовуваних бібліотек, надійністю системи управління базами даних та обробленням даних, які вводить користувач для уникнення проблем з перетворенням типів.

А.3.3 Умови експлуатації

Для експлуатації програми необхідно забезпечити виділений сервер або хостинг, який буде відповідати необхідній кількості запитів на оброблення, що передбачається відповідним масштабом системи.

A.3.4 Вимоги до складу та параметрів технічних засобів

Апаратні вимоги до сервера:

- жорсткий диск з вільним простором – 0.5 Гб;
- оперативна пам'ять – 4 Гб;
- процесор: 2 ядра, 2,8 ГГц частота.

Клієнт представлений вебсторінкою у браузері користувача. Для виконання вимог до нього потрібний встановлений сучасний браузер з доступом до мережі інтернет.

A.4 Порядок контролю та приймання

Контроль за виконанням дипломної кваліфікаційної роботи покладається на керівника. Остаточне рішення приймається екзаменаційною комісією під час захисту роботи.

ДОДАТОК Б
Текст програми

Б.1 Текст файла crawler_base.go

```

package crawler

func (cb *Base) PageWalker(page string, exclude []string, onlyThisPage, forceCollect bool,
headers map[string]string) (structs.SiteStruct, error) {
    pageParsedURL, err := url.Parse(page)
    if err != nil {
        return structs.SiteStruct{}, fmt.Errorf("CrawlerBase.PageWalker url parse error: %w", err)
    }

    var siteStruct structs.SiteStruct
    if !forceCollect {
        siteStruct, err = cb.postgres.GetFullData(fmt.Sprintf("%s://%s", pageParsedURL.Scheme,
pageParsedURL.Host), page, onlyThisPage)
        if onlyThisPage && err == nil {
            data, err := cb.postgres.GetCrawlerData(page)
            if err == nil {
                siteStruct.Hierarchy = &structs.Hierarchy{CrawlerData: data}
            }
        }
    }

    if err != nil || reflect.DeepEqual(siteStruct, structs.SiteStruct{}) || forceCollect {
        if err != nil && !errors.Is(err, gorm.ErrRecordNotFound) {
            log.Error().Str("function", "PageWalker").Err(err).Msg("CrawlerBase.PageWalker postgres
GetFullData error")
        }

        whoisParsed, err := cb.whois.WhoIS(pageParsedURL)
        if err != nil {
            log.Error().Str("function", "PageWalker").Err(err).Msg("CrawlerBase.PageWalker whois
error")
        } else if whoisParsed.Domain == nil {
            whoisParsed.Domain = new(whoisparser.Domain)
            log.Error().Str("function", "PageWalker").Msg("CrawlerBase.PageWalker whoisParsed
Domain empty")
        }

        crawler := newCrawler(page, exclude)
        hierarchy, err := crawler.PageWalker(page, onlyThisPage, headers)
        if err != nil {
            log.Error().Str("function", "PageWalker").Err(err).Msg("CrawlerBase.PageWalker url
parse error")
        }

        hierarchy.ParentLink = fmt.Sprintf("%s://%s", pageParsedURL.Scheme,
pageParsedURL.Host)
        siteStruct = structs.SiteStruct{
            DomainID: whoisParsed.Domain.ID,
            BaseURL:   fmt.Sprintf("%s://%s", pageParsedURL.Scheme, pageParsedURL.Host),
            Punycode:  whoisParsed.Domain.Punycode,

```

```

DNSec:      whoisParsed.Domain.DNSec,
NameServers: whoisParsed.Domain.NameServers,
Status:     whoisParsed.Domain.Status,
WhoisServer: whoisParsed.Domain.WhoisServer,
CreateDate: whoisParsed.Domain.CreatedDate,
UpdatedDate: whoisParsed.Domain.UpdatedDate,
ExpirationDate: whoisParsed.Domain.ExpirationDate,
Registrar:  whoisParsed.Registrar,
Registrant:  whoisParsed.Registrant,
Administrative: whoisParsed.Administrative,
Technical:   whoisParsed.Technical,
Billing:     whoisParsed.Billing,
Hierarchy:   &hierarchy,
Exclude:     pq.StringArray(exclude),
Headers:     headers,
}

if err := cb.postgres.SaveSiteStruct(siteStruct, onlyThisPage, forceCollect); err != nil {
    log.Error().Str("function", "PageWalker").Err(err).Msg("CrawlerBase.PageWalker postgres
SaveSiteStruct error")
}
} else {
    tools.SetParents(siteStruct.Hierarchy)
}

siteStruct.URL = page
siteStruct.ProcessedHyperlinks = 1
siteStruct.StatusCodesCounter = make(map[int]int64)

graph := pagerank.NewGraph[string]()
siteStruct.LinkHierarchy = tools.HierarchyProcess(&siteStruct, siteStruct.Hierarchy,
make(map[string][]structs.LinkHierarchy), graph)

ranks := make(map[string]float64)
graph.Rank(0.85, 0.000001, func(node string, rank float64) {
    ranks[node] = rank
})

tools.SetPageRank(&siteStruct.LinkHierarchy, ranks)
siteStruct.UniqueHyperlinks = tools.UniqueHyperlinks(siteStruct.Hierarchy)
return siteStruct, nil
}

func NewCrawlerBase(postgres Postgres, whois WhoIS) *Base {
    return &Base{postgres: postgres, whois: whois}
}

```

Б.2 Текст файла crawler.go

```

package crawler

type Crawler struct {

```

```

    visited []string
}

func (c *Crawler) PageWalker(page string, onlyThisPage bool, headers map[string]string)
(hierarchy structs.Hierarchy, err error) {
    resp, err := resty.New().SetHeaders(headers).R().Get(page)
    if err != nil {
        hierarchy.Link = page
        hierarchy.StatusCode = resp.StatusCode()
        hierarchy.Error = err.Error()
        return hierarchy, err
    }

    u, err := url.Parse(page)
    if err != nil {
        hierarchy.Link = page
        hierarchy.StatusCode = resp.StatusCode()
        hierarchy.Error = err.Error()
        return hierarchy, err
    }

    data, err := c.ParsePage(bytes.NewReader(resp.Body()), u)
    if err != nil {
        hierarchy.Link = page
        hierarchy.StatusCode = resp.StatusCode()
        hierarchy.Error = err.Error()
        return hierarchy, err
    }

    data.Link = page
    data.StatusCode = resp.StatusCode()

    hierarchy.CrawlerData = data
    hierarchy.Ping = float64(resp.Time()) / float64(time.Second)
    if !onlyThisPage {
        process := make([]string, 0, len(data.Hyperlinks))
        for _, link := range data.Hyperlinks {
            if strings.Contains(link, fmt.Sprintf("%s://%s", u.Scheme, u.Host))
            && !tools.CheckVisited(c.visited, link) {
                c.visited = append(c.visited, link)
                process = append(process, link)
            }
        }

        for _, link := range process {
            child, err := c.PageWalker(link, onlyThisPage, headers)
            if err != nil {
                log.Error().Str("function", "PageWalker").Err(err).Msg("PageWalker error")
            }

            child.ParentLink = hierarchy.Link
            child.Parent = &hierarchy
        }
    }
}

```

```

        hierarchy.Childrens = append(hierarchy.Childrens, child)
    }
}

return hierarchy, err
}

func (c *Crawler) ParsePage(page io.Reader, baseURL *url.URL) (structs.CrawlerData, error) {
    if page == nil || baseURL == nil {
        return structs.CrawlerData{}, errors.New("Page or URL is null")
    }

    doc, err := html.Parse(page)
    if err != nil {
        return structs.CrawlerData{}, fmt.Errorf("Page parse error: %w", err)
    }

    data := c.crawlerFunc(doc)
    for i, link := range data.Hyperlinks {
        switch {
        case strings.HasPrefix(link, "//"):
            data.Hyperlinks[i] = baseURL.Scheme + "://" + link
        case strings.HasPrefix(link, "/"):
            data.Hyperlinks[i] = fmt.Sprintf("%s://%s", baseURL.Scheme, baseURL.Host) + link
        case strings.HasPrefix(link, "?"):
            baseURL.RawQuery = ""
            data.Hyperlinks[i] = baseURL.String() + link
        }
    }

    data.Images = tools.PrepareLinks(data.Images, baseURL.String())
    data.Audio = tools.PrepareLinks(data.Audio, baseURL.String())
    data.Video = tools.PrepareLinks(data.Video, baseURL.String())
    data.Files = tools.PrepareLinks(data.Files, baseURL.String())
    data.Fonts = tools.PrepareLinks(data.Fonts, baseURL.String())

    data.Audio = tools.Compact(data.Audio)
    data.Video = tools.Compact(data.Video)
    data.Hyperlinks = tools.Compact(data.Hyperlinks)
    data.Images = tools.Compact(data.Images)
    data.InternalLinks = tools.Compact(data.InternalLinks)
    data.Files = tools.Compact(data.Files)
    data.Fonts = tools.Compact(data.Fonts)
    return data, nil
}

func (c *Crawler) crawlerFunc(node *html.Node) structs.CrawlerData {
    var data structs.CrawlerData

    if node.Type == html.TextNode && !slices.Contains(settings.NotAllowedTags,
node.Parent.Data) {
        text := tools.PrepareCrawlerText(node.Data)

```

```

if text != "" {
    data.Text = text
}
} else if node.Type == html.ElementNode {
    switch {
    case node.Data == "meta":
        meta := make(map[string]string)

        for _, attr := range node.Attr {
            meta[attr.Key] = attr.Val
        }

        data.Metadata = append(data.Metadata, meta)
    case node.Data == "link" || node.Data == "a":
        LINK_LOOP:
        for _, attr := range node.Attr {
            if attr.Key == "href" && !strings.Contains(attr.Val, "javascript:void(0)") {
                if strings.HasPrefix(attr.Val, "#") {
                    data.InternalLinks = append(data.InternalLinks, attr.Val)
                } else {
                    for _, imgFormat := range settings.ImageExtensions {
                        if strings.HasSuffix(strings.Split(attr.Val, "?")[0], imgFormat) {
                            data.Images = append(data.Images, attr.Val)
                            break LINK_LOOP
                        }
                    }
                }

                for _, fontFormat := range settings.FontsExtensions {
                    if strings.HasSuffix(strings.Split(attr.Val, "?")[0], fontFormat) {
                        data.Fonts = append(data.Fonts, attr.Val)
                        break LINK_LOOP
                    }
                }

                for _, fileFormat := range settings.FilesExtensions {
                    if strings.HasSuffix(strings.Split(attr.Val, "?")[0], fileFormat) {
                        data.Files = append(data.Files, attr.Val)
                        break LINK_LOOP
                    }
                }

                for _, videoFormat := range settings.VideoExtensions {
                    if strings.HasSuffix(strings.Split(attr.Val, "?")[0], videoFormat) {
                        data.Video = append(data.Video, attr.Val)
                        break LINK_LOOP
                    }
                }

                for _, audioFormat := range settings.AudioExtensions {
                    if strings.HasSuffix(strings.Split(attr.Val, "?")[0], audioFormat) {
                        data.Audio = append(data.Audio, attr.Val)
                        break LINK_LOOP
                    }
                }
            }
        }
    }
}

```

```

    }
}

data.Hyperlinks = append(data.Hyperlinks, attr.Val)
}

break
}
}
case node.Data == "source":
    var links []string
    for _, attr := range node.Attr {
        if attr.Key == "src" {
            links = append(links, attr.Val)
            break
        }
    }

    if node.Parent.Type == html.ElementNode {
        if node.Parent.Data == "video" {
            data.Video = append(data.Video, links...)
        } else if node.Parent.Data == "audio" {
            data.Audio = append(data.Audio, links...)
        }
    }
    if node.Parent.Parent.Type == html.ElementNode {
        if node.Parent.Parent.Data == "video" {
            data.Video = append(data.Video, links...)
        } else if node.Parent.Parent.Data == "audio" {
            data.Audio = append(data.Audio, links...)
        }
    }
}
case node.Data == "img":
    for _, attr := range node.Attr {
        if attr.Key == "src" && !strings.HasPrefix(attr.Val, "data:image/") {
            data.Images = append(data.Images, attr.Val)
            break
        }
    }
}
}

for child := node.FirstChild; child != nil; child = child.NextSibling {
    toAdd := c.crawlerFunc(child)
    data.Merge(slices.Contains(settings.HTMLBlocks, child.Data), toAdd)
}

return data
}

func newCrawler(firstPage string, exclude []string) *Crawler {
    return &Crawler{visited: append(exclude, firstPage)}
}

```

Б.3 Текст файла postgres.go

```

package postgres

func (p *Postgres) SaveSiteStruct(siteStruct structs.SiteStruct, onlyThisPage, force bool) error {
    baseReq := p.db.Model(&siteStruct).Session(&gorm.Session{FullSaveAssociations: true})

    if onlyThisPage && force {
        baseReq.Omit("Hierarchy.ParentLink")
    }

    err := baseReq.Where("base_url = ?", siteStruct.BaseURL).Save(&siteStruct).Error
    if err != nil {
        return fmt.Errorf("Error saving site structure: %w", err)
    }

    return nil
}

func (p *Postgres) GetFullData(link, url string, onlyThisPage bool) (structs.SiteStruct, error) {
    var result structs.SiteStruct

    baseReq := p.db.Model(&result)
    if !onlyThisPage {
        baseReq = baseReq.Preload("Hierarchy", preloadHierarchy)
    }

    err := baseReq.Where("base_url = ?", link).First(&result).Error
    if err != nil {
        return structs.SiteStruct{}, fmt.Errorf("Error getting site structure: %w", err)
    }

    if onlyThisPage {
        crawlerData, err := p.GetCrawlerData(url)
        if err != nil {
            return structs.SiteStruct{}, fmt.Errorf("Error getting crawler data: %w", err)
        }

        result.Hierarchy = &structs.Hierarchy{CrawlerData: crawlerData, ParentLink: link}
    }

    return result, nil
}

func (p *Postgres) GetSites() ([]structs.SiteStruct, error) {
    var result []structs.SiteStruct

    err := p.db.Model(&structs.SiteStruct{}).Find(&result).Error
    if err != nil {
        return nil, fmt.Errorf("Error getting sites: %w", err)
    }
}

```

```

    return result, nil
}

func (p *Postgres) GetCrawlerData(link string) (structs.CrawlerData, error) {
    var result structs.CrawlerData

    err := p.db.Model(&structs.Hierarchy{}).Where("link = ?", link).First(&result).Error
    if err != nil {
        return structs.CrawlerData{}, fmt.Errorf("Error getting crawler data: %w", err)
    }

    return result, nil
}

func NewPostgres(cfg *config.Config) (postgres *Postgres, err error) {
    postgres = new(Postgres)
    postgres.db, err = newDB(cfg.DSN, cfg.MigrationURL)
    if err != nil {
        return
    }

    return
}

func (p *Postgres) Stop(ctx context.Context) error {
    sql, err := p.db.WithContext(ctx).DB()
    if err != nil {
        return fmt.Errorf("Postgres.Stop get database error: %w", err)
    }

    err = sql.Close()
    if err != nil {
        return fmt.Errorf("Postgres.Stop connection close error: %w", err)
    }

    return nil
}

func newDB(dsn, migrationsURL string) (*gorm.DB, error) {
    if err := migrations(dsn, migrationsURL); err != nil {
        return nil, fmt.Errorf("Postgres.newDB migrations error: %w", err)
    }

    client, err := gorm.Open(postgres.Open(dsn), &gorm.Config{
        Logger: glog.Default.LogMode(glog.Silent),
    })
    if err != nil {
        return nil, fmt.Errorf("Postgres.newDB gorm connection open error: %w", err)
    }

    return client, nil
}

```

```

}

func migrations(dsn, migrationsURL string) error {
    db, err := sql.Open("postgres", dsn)
    if err != nil {
        return fmt.Errorf("Postgres.migrations sql connection open error: %w", err)
    }

    driver, err := mpostgres.WithInstance(db, &mpostgres.Config{})
    if err != nil {
        return fmt.Errorf("Postgres.migrations create migration instance error: %w", err)
    }

    m, err := migrate.NewWithDatabaseInstance(
        migrationsURL,
        "postgres",
        driver,
    )
    if err != nil {
        return fmt.Errorf("Postgres.migrations migration init error: %w", err)
    }

    if err := m.Up(); err != nil && !errors.Is(err, migrate.ErrNoChange) {
        return fmt.Errorf("Postgres.migrations migration error: %w", err)
    }

    return nil
}

```

Б.4 Текст файлу common.go

```

package tools

func PrepareCrawlerText(text string) string {
    text = strings.TrimSpace(strings.ReplaceAll(text, " ", ""))
    if text == "\n" {
        return ""
    }

    return text
}

func CheckWisited(wisited []string, link string) bool {
    for _, wisited := range wisited {
        if strings.HasSuffix(wisited, "*") {
            wisited = wisited[:len(wisited)-1]
            if strings.HasPrefix(link, wisited) {
                return true
            }
        } else if wisited == link {
            return true
        }
    }
}

```

```

    }

    return false
}

func HierarchyProcess(
    resp *structs.SiteStruct,
    hierarchy *structs.Hierarchy,
    hyperlinks map[string][]structs.LinkHierarchy,
    graph *pagerank.Graph[string],
) (res structs.LinkHierarchy) {
    if resp == nil || hierarchy == nil {
        return structs.LinkHierarchy{}
    }

    resp.Images += int64(len(hierarchy.Images))
    resp.AudioLinks += int64(len(hierarchy.Audio))
    resp.VideoLinks += int64(len(hierarchy.Video))
    resp.Files += int64(len(hierarchy.Files))
    resp.Fonts += int64(len(hierarchy.Fonts))
    resp.Hyperlinks += int64(len(hierarchy.Hyperlinks))
    resp.ProcessedHyperlinks += int64(len(hierarchy.Childrens))
    resp.InternalLinks += int64(len(hierarchy.InternalLinks))
    resp.Paragraphs += int64(len(strings.Split(hierarchy.Text, "\n")))
    resp.Words += int64(len(strings.Split(strings.ReplaceAll(hierarchy.Text, "\n", " "), " ")))
    resp.Symbols += int64(len(strings.ReplaceAll(hierarchy.Text, "\n", "")))
    resp.StatusCodesCounter[hierarchy.StatusCode]++

    tr := textrank.NewTextRank()
    rule := textrank.NewDefaultRule()
    language := textrank.NewDefaultLanguage()
    algorithmDef := textrank.NewDefaultAlgorithm()
    tr.Populate(hierarchy.Text, language, rule)
    tr.Ranking(algorithmDef)

    phrases := textrank.FindPhrases(tr)
    p := make([]structs.Phrase, 0, len(phrases))
    for _, phrase := range phrases {
        p = append(p, structs.Phrase{
            Left: phrase.Left,
            Right: phrase.Right,
            Weight: phrase.Weight,
            Qty: phrase.Qty,
        })
    }

    hierarchy.Phrases = p

    words := textrank.FindSingleWords(tr)
    w := make([]structs.Word, 0, len(words))
    for _, word := range words {
        w = append(w, structs.Word{

```

```

        Word: word.Word,
        Weight: word.Weight,
        Qty: word.Qty,
    })
}

hierarchy.Words = w

sentences := textrank.FindSentencesByRelationWeight(tr, 20)
for _, sentence := range sentences {
    hierarchy.Sentences = append(hierarchy.Sentences, sentence.Value)
}

if hierarchy.Error != "" {
    resp.Errors++
}

res.Link = hierarchy.Link
res.Attributes = make(map[string]string)

childrenLinks := make([]string, 0, len(hierarchy.Childrens))
for i, child := range hierarchy.Childrens {
    childrenLinks = append(childrenLinks, child.Link)
    processed := HierarchyProcess(resp, &hierarchy.Childrens[i], hyperlinks, graph)
    res.Children = append(res.Children, processed)

    if links, ok := hyperlinks[child.Link]; ok {
        var path string
        for root := &hierarchy.Childrens[i]; root != nil; root = root.Parent {
            path = root.Link + " | " + path
        }

        path = strings.TrimSuffix(path, " | ")
        for _, link := range links {
            link.Attributes["Already processed"] = path
        }
    }
}

for _, link := range hierarchy.Hyperlinks {
    graph.Link(hierarchy.Link, link, 1.0)
    if strings.Contains(link, resp.BaseURL) && !slices.Contains(childrenLinks, link) {
        linkHierarchy := structs.LinkHierarchy{Link: link, Attributes: make(map[string]string)}

        if links, ok := hyperlinks[link]; ok && len(links) > 0 && len(links[0].Attributes) > 0 {
            linkHierarchy.Attributes["Already processed"] = links[0].Attributes["Already processed"]
        } else {
            hyperlinks[link] = append(hyperlinks[link], linkHierarchy)
        }

        res.Children = append(res.Children, linkHierarchy)
    }
}

```

```

    }

    for _, link := range hierarchy.Audio {
        graph.Link(hierarchy.Link, link, 1.0)
    }

    for _, link := range hierarchy.Video {
        graph.Link(hierarchy.Link, link, 1.0)
    }

    for _, link := range hierarchy.Files {
        graph.Link(hierarchy.Link, link, 1.0)
    }

    for _, link := range hierarchy.Fonts {
        graph.Link(hierarchy.Link, link, 1.0)
    }

    for _, link := range hierarchy.Images {
        graph.Link(hierarchy.Link, link, 1.0)
    }

    return res
}

func SetPageRank(hierarchy *structs.LinkHierarchy, ranks map[string]float64) {
    hierarchy.Attributes["PageRank"] = cast.ToString(ranks[hierarchy.Link])

    for i := range hierarchy.Children {
        SetPageRank(&hierarchy.Children[i], ranks)
    }
}

func SetParents(hierarchy *structs.Hierarchy) {
    if hierarchy == nil {
        return
    }

    for i := range hierarchy.Childrens {
        hierarchy.Childrens[i].Parent = hierarchy
        SetParents(&hierarchy.Childrens[i])
    }
}

func UniqueHyperlinks(hierarchy *structs.Hierarchy) int64 {
    if hierarchy == nil {
        return 0
    }

    return int64(len(uniqueHyperlinksProcessor(hierarchy)))
}

```

```

func uniqueHyperlinksProcessor(hierarchy *structs.Hierarchy) []string {
    var result []string

    result = slices.Clone(hierarchy.Hyperlinks)
    for i := range hierarchy.Childrens {
        links := uniqueHyperlinksProcessor(&hierarchy.Childrens[i])
        result = append(result, links...)
    }

    return Compact(result)
}

func PrepareLinks(links []string, baseURL string) []string {
    for i, link := range links {
        if !strings.Contains(link, "http://") && !strings.Contains(link, "https://") {
            url := strings.Split(baseURL, "/")
            if len(url) > 0 {
                links[i] = strings.Join(url[:len(url)-1], "/") + link
            }
        }
    }

    return links
}

func Compact[T comparable](array []T) []T {
    res := make([]T, 0, len(array))

    for _, elem := range array {
        if !slices.Contains(res, elem) {
            res = append(res, elem)
        }
    }

    return slices.Clip(res)
}

```

Б.5 Текст файлы handlers/crawler.go

```
package handlers
```

```

func (h *Handler) GetSiteStruct(ctx echo.Context) error {
    var siteParseReq structs.SiteParseReq
    if err := ctx.Bind(&siteParseReq); err != nil {
        log.Error().Str("function", "GetSiteStruct").Err(err).Msg("Failed to bind to siteParseReq")
        return ctx.JSON(http.StatusBadRequest, newHTTPError(ErrBind))
    }

    headers := make(map[string]string)
    for key, values := range ctx.Request().Header {
        headers[key] = strings.Join(values, ",")
    }
}

```

```

    resp, err := h.crawler.PageWalker(siteParseReq.URL, siteParseReq.Exclude,
siteParseReq.OnlyThisPage, siteParseReq.ForceCollect, headers)
    if err != nil {
        log.Error().Str("function", "GetSiteStruct").Err(err).Msg(ErrGetSiteStruct.Error())
        return ctx.JSON(http.StatusBadRequest, newHTTPError(ErrGetSiteStruct))
    }

    return ctx.JSON(http.StatusOK, resp)
}

func (h *Handler) GetDetails(ctx echo.Context) error {
    resp, err := h.postgres.GetCrawlerData(ctx.QueryParam("link"))
    if err != nil {
        log.Error().Str("function", "GetDetails").Err(err).Msg(ErrGetDetails.Error())
        return ctx.JSON(http.StatusInternalServerError, newHTTPError(ErrGetDetails))
    }

    tr := textrank.NewTextRank()
    rule := textrank.NewDefaultRule()
    language := textrank.NewDefaultLanguage()
    algorithmDef := textrank.NewDefaultAlgorithm()
    tr.Populate(resp.Text, language, rule)
    tr.Ranking(algorithmDef)

    phrases := textrank.FindPhrases(tr)
    p := make([]structs.Phrase, 0, len(phrases))
    for _, phrase := range phrases {
        p = append(p, structs.Phrase{
            Left:  phrase.Left,
            Right: phrase.Right,
            Weight: phrase.Weight,
            Qty:   phrase.Qty,
        })
    }

    resp.Phrases = p

    words := textrank.FindSingleWords(tr)
    w := make([]structs.Word, 0, len(words))
    for _, word := range words {
        w = append(w, structs.Word{
            Word:  word.Word,
            Weight: word.Weight,
            Qty:   word.Qty,
        })
    }

    resp.Words = w

    sentences := textrank.FindSentencesByRelationWeight(tr, 20)
    for _, sentence := range sentences {

```

```

    resp.Sentences = append(resp.Sentences, sentence.Value)
}

return ctx.JSON(http.StatusOK, resp)
}

```

Б.6 Текст файла server.go

```

package server

type HTTPServer struct {
    router *echo.Echo

    cfg      *config.Config
    handlers *handlers.Handler
}

func NewHTTPServer(
    cfg *config.Config,
    handlers *handlers.Handler,
) *HTTPServer {
    return &HTTPServer{
        cfg:      cfg,
        handlers: handlers,
    }
}

func (s *HTTPServer) Start(_ context.Context) error {
    s.router = echo.New()
    s.router.Use(middleware.LoggerMiddleware(),                middleware.CorsMiddleware(),
middleware.RecoverMiddleware())

    s.router.GET("/parse_site", s.handlers.GetSiteStruct)
    s.router.GET("/details", s.handlers.GetDetails)

    go func() {
        if err := s.router.Start(s.cfg.AppAddr); err != nil && !errors.Is(err, http.ErrServerClosed) {
            log.Fatal().Str("function", "Start").Err(err).Msg("Server start error")
        }
    }()

    return nil
}

func (s *HTTPServer) Stop(ctx context.Context) error {
    return s.router.Shutdown(ctx)
}

```

Б.7 Текст файла cron.go

```
package cron

type Crawler interface {
    PageWalker(page string, exclude []string, onlyThisPage, forceCollect bool, headers
map[string]string) (siteStruct structs.SiteStruct, err error)
}

type Postgres interface {
    GetSites() ([]structs.SiteStruct, error)
}

type Cron struct {
    s *gocron.Scheduler
    crawler Crawler
    postgres Postgres
}

func (c *Cron) UpdateSites() {
    sites, err := c.postgres.GetSites()
    if err != nil {
        log.Error().Str("function", "UpdateSites").Err(err).Msg("cron.UpdateSites postgres.GetSites
error")
        return
    }

    p := pool.New().WithMaxGoroutines(settings.MaxCronGoroutines)
    for _, site := range sites {
        p.Go(func(site structs.SiteStruct) func() {
            return func() {
                _, err := c.crawler.PageWalker(site.BaseURL+"/", site.Exclude, false, true, site.Headers)
                if err != nil {
                    log.Error().Str("function", "UpdateSites").Err(err).Msg("cron.UpdateSites
crawler.PageWalker error")
                }
            }
        })(site))
    }
}

func (c *Cron) Start(_ context.Context) error {
    c.s = gocron.NewScheduler(time.UTC)
    _, err := c.s.Every(1).Day().Do(c.UpdateSites)
    if err != nil {
        return fmt.Errorf("Cron start error: %w", err)
    }

    c.s.StartAsync()
    return nil
}
```

```

func (c *Cron) Stop(_ context.Context) error {
    if c.s != nil {
        c.s.Stop()
        c.s.Clear()
    }

    return nil
}

func NewCron(crawler Crawler, postgres Postgres) *Cron {
    return &Cron{crawler: crawler, postgres: postgres}
}

```

Б.8 Текст файла app.go

```

package cmd

func Exec() fx.Option {
    return fx.Options(
        fx.Provide(
            config.Get,
            postgres.NewPostgres,
            fx.Annotate(
                annotationDupl[postgres.Postgres],
                fx.As(new(crawler.Postgres)),
                fx.As(new(handlers.Postgres)),
                fx.As(new(cron.Postgres)),
            ),
            fx.Annotate(whois.NewWhoIS, fx.As(new(crawler.WhoIS))),
            fx.Annotate(crawler.NewCrawlerBase,
                fx.As(new(handlers.Crawler)),
                fx.As(new(cron.Crawler)),
            ),
            cron.NewCron,
            handlers.NewHandler,
            server.NewHTTPServer,
        ),
        fx.Invoke(
            prepareHooks,
        ),
    )
}

func annotationDupl[T any](v *T) *T {
    return v
}

func prepareHooks(server *server.HTTPServer, postgres *postgres.Postgres, _ *cron.Cron, lc
fx.Lifecycle) {
    lc.Append(fx.Hook{OnStart: cron.Start, OnStop: cron.Stop})
    lc.Append(fx.Hook{OnStop: postgres.Stop})
    lc.Append(fx.Hook{OnStart: server.Start, OnStop: server.Stop})
}

```

```
}
```

Б.9 Текст файла App.js

```
import Tree from "react-d3-tree";
import { JsonTable } from "react-json-to-html";
import "./App.css";
import { Component } from "react";
import axios from "axios";
import { useLocation, Link } from "react-router-dom";
import _ from "lodash";

class App extends Component {
  constructor(props) {
    super(props);
    this.state = {
      showHideReturn: false,
      siteStruct: { Loading: "wait..." },
      hierarchy: [{ name: "Loading..." }],
    };

    this.json = {};
    this.setJson = this.setJson.bind(this);
    this.resetJson = this.resetJson.bind(this);
    this.url = this.props.url;
    this.only_this_page = this.props.only_this_page;
    this.force = this.props.force;
    this.exclude = this.props.exclude;
  }

  async getData(url, only_this_page, force, exclude) {
    const searchParams = new URLSearchParams();
    searchParams.append("url", url);
    searchParams.append(
      "only_this_page",
      only_this_page === undefined ? false : only_this_page
    );
    searchParams.append("force_collect", force === undefined ? false : force);
    if (this.exclude !== undefined && exclude.length > 0) {
      exclude.forEach((excl, i) => {
        searchParams.append("exclude", excl);
      });
    }

    const res = await axios(
      "http://localhost:1140/parse_site?" + searchParams.toString()
    );
    return res.data;
  }

  componentDidMount() {
    if (!_isEqual(this.state.siteStruct, { Loading: "wait..." })) {
```

```

this.getData(this.url, this.only_this_page, this.force, this.exclude)
  .then((data) => {
    const { hierarchy, ...rest } = data;
    this.json = rest;
    this.setState({ siteStruct: rest, hierarchy: hierarchy });
    this.forceUpdate();
  })
  .catch((err) => {
    console.log(err);
  });
}
}

setJson(node) {
  axios
    .get("http://localhost:1140/details?link=" + node.data.name)
    .then((response) => {
      const { text, ...data } = response.data;
      if (typeof data.metadata !== "string") {
        data.metadata = JSON.stringify(data.metadata);
      }

      data.text = text;
      this.json = data;
      this.setState({ showHideReturn: true });
      this.forceUpdate();
    })
    .catch((err) => {
      console.log(err);
    });
}

resetJson() {
  this.json = this.state.siteStruct;
  this.setState({ showHideReturn: false });
  this.forceUpdate();
}

render() {
  return (
    <div className="App">
      <div className="Tree">
        <Link to="/">
          <button id="close">
            
          </button>
        </Link>
      </Tree>
    </div>
  );
}

```

```

        data={this.state.hierarchy}
        initialDepth={0}
        collapsible={true}
        transitionDuration={0}
        nodeSize={{ x: 400, y: 140 }}
        onClick={this.setJson}
      />
    </div>
    <div className="Json">
      {this.state.showHideReturn && (
        <button id="return">
          
        </button>
      )}
      <JsonTable json={this.json} />
    </div>
  </div>
);
}
}

const WrappedComponent = (props) => {
  const location = useLocation();
  const { url, only_this_page, force, exclude } = location.state;

  return (
    <App
      url={url}
      only_this_page={only_this_page}
      force={force}
      exclude={exclude}
      {...props}
    />
  );
};

export default WrappedComponent;

```

Б.10 Текст файлу Wrapper.js

```

import { BrowserRouter, Routes, Route, Link } from "react-router-dom";
import WrappedComponent from "../parsed/App";
import Styles from "../styles";
import { Form, Field } from "react-final-form";
import arrayMutators from "final-form-arrays";
import { FieldArray } from "react-final-form-arrays";

```

```

export default function Wrapper() {
  return (
    <BrowserRouter>
      <Routes>
        <Route
          path="/"
          element={
            <Styles>
              <h1>Parse site</h1>
              <Form
                onSubmit={function () {}}
                mutators={{
                  ...arrayMutators,
                }}
                render={{(
                  handleSubmit,
                  form: {
                    mutators: { push, pop },
                  }, // injected from final-form-arrays above
                  pristine,
                  form,
                  submitting,
                  values,
                ) => {
                  return (
                    <form onSubmit={handleSubmit}>
                      <div>
                        <label>Site url</label>
                        <Field name="url" component="input" />
                      </div>
                      <div>
                        <label>Only this page</label>
                        <Field
                          name="only_this_page"
                          component="input"
                          type="checkbox"
                        />
                      </div>
                      <div>
                        <label>Force collect</label>
                        <Field name="force" component="input" type="checkbox" />
                      </div>
                      <div className="buttons">
                        <button
                          type="button"
                          onClick={() => push("exclude", undefined)}
                        >
                          Exclude url
                        </button>
                        <button type="button" onClick={() => pop("exclude")}>
                          Remove exclude
                        </button>
                      </div>
                    </form>
                  )
                }}
              />
            </Styles>
          </Route>
      </Routes>
    </BrowserRouter>
  )
}

```

```

</div>
<FieldArray name="exclude">
 (({ fields }) =>
    fields.map((url, index) => (
      <div key={url}>
        <label>Url №{index + 1}</label>
        <Field
          name={`$${url}`}
          component="input"
          placeholder="URL"
        />
        <span
          onClick={() => fields.remove(index)}
          style={{ cursor: "pointer" }}
        >
          ✕
        </span>
      </div>
    ))
  )
</FieldArray>

<div className="buttons">
  <Link to="/parsed" state={values}>
    <button
      type="submit"
      disabled={submitting || pristine}
    >
      Submit
    </button>
  </Link>
  <button
    type="button"
    onClick={form.reset}
    disabled={submitting || pristine}
  >
    Reset
  </button>
</div>
</form>
);
}}
/>
</Styles>
}
/>
<Route path="/parsed" element={<WrappedComponent />} />
</Routes>
</BrowserRouter>
);
}

```

ДОДАТОК В
Слайди презентації

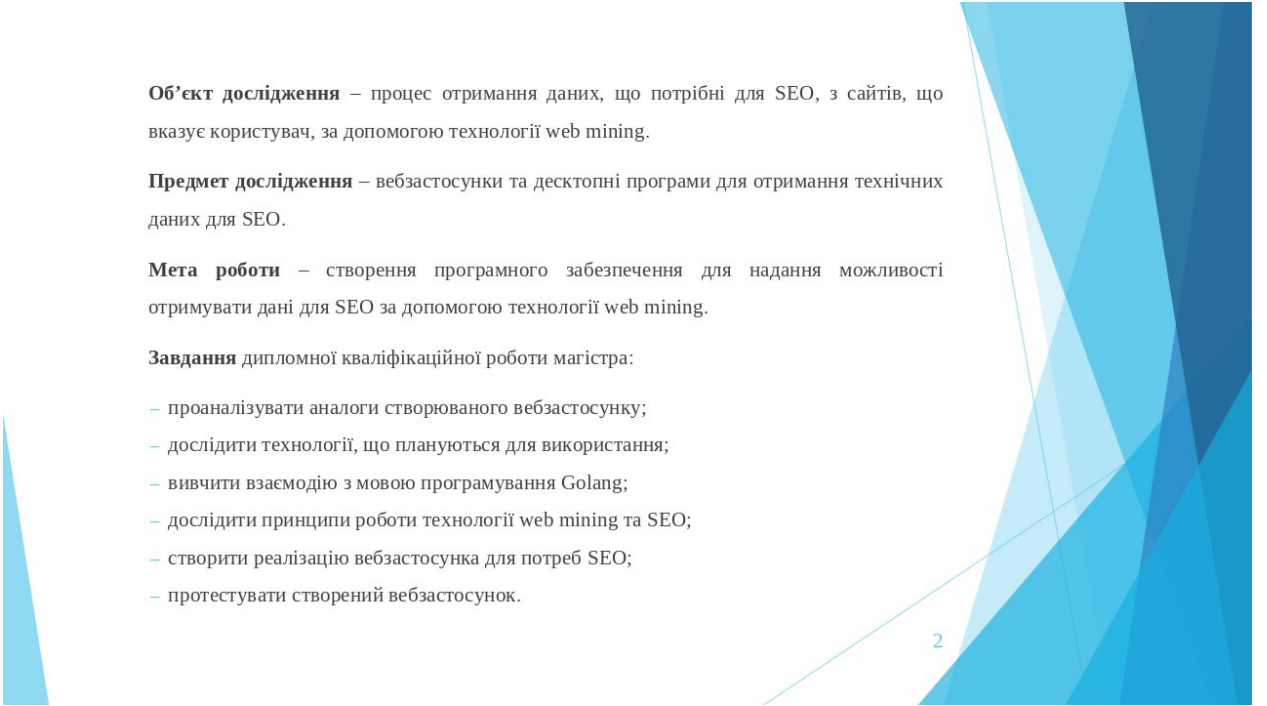


Дослідження та програмна реалізація технології web mining для потреб SEO RESEARCH AND SOFTWARE IMPLEMENTATION OF A WEB MINING TECHNOLOGY FOR SEO NEEDS

Виконав ст. гр. КНТ-222м ЛІНЕЦЬ В. А.
Керівник к.т.н., доцент — КОЛПАКОВА Т. О.

2023

Рисунок В.1 – Слайд № 1



Об'єкт дослідження – процес отримання даних, що потрібні для SEO, з сайтів, що вказує користувач, за допомогою технології web mining.

Предмет дослідження – вебзастосунки та десктопні програми для отримання технічних даних для SEO.

Мета роботи – створення програмного забезпечення для надання можливості отримувати дані для SEO за допомогою технології web mining.

Завдання дипломної кваліфікаційної роботи магістра:

- проаналізувати аналоги створюваного вебзастосунку;
- дослідити технології, що плануються для використання;
- вивчити взаємодію з мовою програмування Golang;
- дослідити принципи роботи технології web mining та SEO;
- створити реалізацію вебзастосунка для потреб SEO;
- протестувати створений вебзастосунок.

2

Рисунок В.2 – Слайд № 2

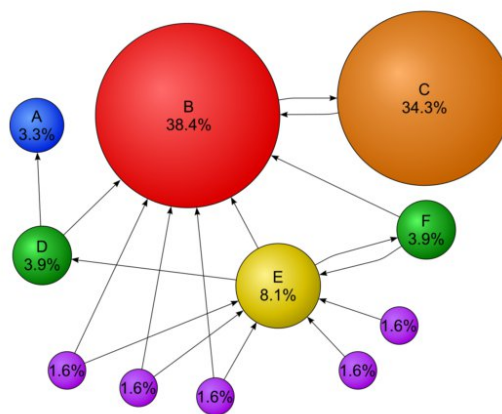
Порівняння аналогів

	Browseo	ContentKing	Google Lighthouse	Apify	Розробка
Комплексний аналіз однієї сторінки	+	+	+	+	+
Аналіз усього сайту	-	+	-	+	+
Періодичне оновлення інформації	+/-	+	+/-	+	+
Аналіз PageRank сторінки	-	-	-	-	+
Аналіз TextRank сторінки	-	-	-	-	+

3

Рисунок В.3 – Слайд № 3

Технологія PageRank



4

Рисунок В.4 – Слайд № 4

До

де

- $$wlr(w_0) = (1-d) + d * \sum_{w_n \in l(w_0)} \frac{pr(w_n) * w(link(w_n, w_0))}{\sum_{w_m} kv(link(w_n, w_m))},$$

де

- $$lw(link(w_1, w_2)) = 1 - \frac{first_occurrence(link(w_1, w_2), w_1)}{|tokens(w_1)|},$$

де

- 5

	5	1		2	
	row	tree		big	
1	6	7	1	2	5
1	1	1	1	1	1
1			1		1
1					1
1					
0	1	1	1	2	1

David Belicza

6

Д

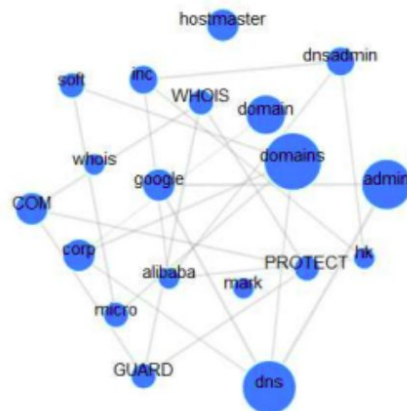
$$Sim(S_i, S_j) = \frac{||wk|wk \in S_i \& wk \in S_j||}{log(|S_i|) + log(|S_j|)},$$

де

- 6

Рисунок В.6 – Слайд № 6

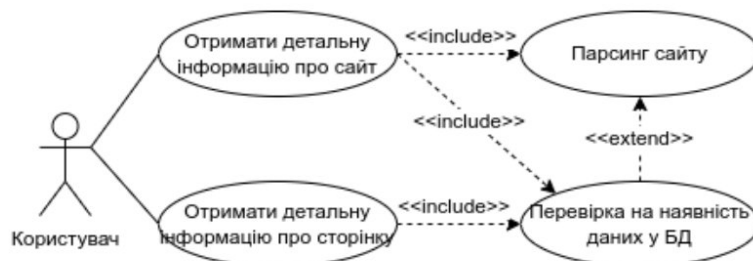
Технологія WHOIS



7

Рисунок В.7 – Слайд № 7

Діаграма прецедентів користувача



8

Рисунок В.8 – Слайд № 8

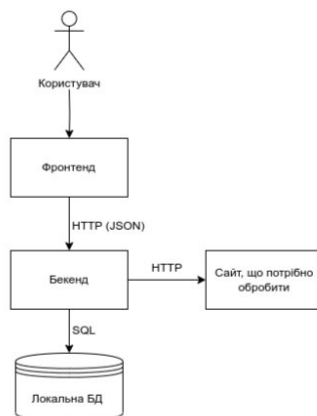
Схема бази даних

sites		link_data	
domain_id	text	link	text
base_url	text	status_code	int
punycode	text	error	text
dns_sec	boolean	text	text
name_servers	text[]	images	text[]
status	text[]	audio	text[]
whois_server	text	video	text[]
created_date	text	hyperlinks	text[]
updated_date	text	internal_links	text[]
expiration_date	text	metadata	json
exclude	text[]	parent_link	text
headers	json	fonts	text[]
registrar	json	files	text[]
registrant	json	ping	numeric
administrative	json		
technical	json		
billing	json		

9

Рисунок В.9 – Слайд № 9

Архітектура програми



10

Рисунок В.10 – Слайд № 10



Інтерфейси користувача

12

Рисунок В.12 – Слайд № 12

Висновки

В даній роботі були вирішені наступні завдання:

- проаналізовані аналоги створюваної програми;
- досліджені технології, що планувалися для використання;
- вивчена взаємодія з мовою програмування Golang;
- досліджені методи роботи з web mining;
- проаналізовані алгоритми роботи PageRank, TextRank та WHOIS;
- створена реалізація програми для видобутку даних для SEO;
- протестовано створену програму.

В результаті проведеної роботи було створено програмну реалізацію вебзастосунку, що інтегрує в собі методи PageRank, TextRank та WHOIS, а також технологію web mining, для комплексного аналізу всіх сторінок сайту для потреб SEO-спеціалістів. Завдання атестаційної роботи магістра повністю виконано.

13

Рисунок В.13 – Слайд № 13