МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ПРИРОДОКОРИСТУВАННЯ
ФАКУЛЬТЕТ МЕХАНІКИ, ЕНЕРГЕТИКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

другого (магістерського) рівня вищої освіти

на тему: «**Розробка інформаційної системи автоматизованого аналізу сайтів на основі ASP.NET**»

Виконав: здобувач 6 курсу групи Іт-61

Спеціальності 126 «Інформаційні системи та технології»
(шифр і назва)

Мельничук О. А.
(Прізвище та ініціали)

Керівник: Пташник В. В
(Прізвище та ініціали)

Рецензент: _____
(Прізвище та ініціали)

ДУБЛЯНИ-2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ПРИРОДОКОРИСТУВАННЯ
ФАКУЛЬТЕТ МЕХАНІКИ, ЕНЕРГЕТИКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Другий (магістерський) рівень вищої освіти
Спеціальність 126 «Інформаційні системи та технології»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ А.М. Тригуба, д.т.н., проф.
«____»_____ 2022 р.

ЗАВДАННЯ
на кваліфікаційну роботу здобувачу
Мельничуку Олегу Андрійовичу

1. Тема роботи: Розробка інформаційної системи автоматизованого аналізу сайтів на основі ASP.NET

Керівник роботи Пташник Вадим Вікторович, к.т.н., доцент
Затверджені наказом по університету від 12.05.2022 № 62/к-с.
2. Строк подання студентом роботи 16.12.2022 р.
3. Вихідні дані до роботи:
1. Статистичні дані вебсайтів виробничих підприємств. 2. Науково-технічна і довідкова література. 3. Методика побудови імітаційних моделей та проєктування ІС. 4. Методика моделювання та опрацювання даних. 5. Рекомендації пошукових систем.
4. Зміст розрахунково-пояснювальної записки:
1. Аналіз стану питання в теорії та практиці та постановка завдання
2. Обгрунтування, вибір та реалізація інструментарію вирішення задачі
3. Результати вирішення задачі
4. Охорона праці
5. Визначення ефективності інформаційної системи
Висновки
Список використаних джерел
Додатки

5. Перелік графічного матеріалу: 1 та 2 – Тема, мета, завдання роботи;
3 – Існуючі автоматизовані системи для аналізу сайтів; 4 – Вибір засобів реалізації;
5 – UML-діаграма класів; 6 – Розробка бази даних інформаційної системи;
7 – Розробка вебінтерфейсу інформаційної системи; 8, 9 та 10 – Розроблена інформаційна системи.

6. Консультанти з розділів:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1, 2, 3, 5	*Пташник В.В., доцент кафедри інформаційних технологій*		
4	*Городецький І.М., доцент кафедри управління проектами та безпеки виробництва*		

7. Дата видачі завдання 12.05.2022 р.

Календарний план

№ з/п	Назва етапів дипломного проекту	Строк виконання етапів роботи	Примітка
1.	*Дослідження стану питання в теорії та практиці та написання першого розділу роботи*	*12.05.22-01.06.22*	
2.	*Вивчення інформаційних технологій для виконання завдань роботи, написання другого розділу*	*01.06.22-30.06.22*	
3.	*Розробка інформаційної системи та виконання третього розділу*	*01.07.22-31.07.22*	
4.	*Написання розділу «Охорона праці та безпека в надзвичайних ситуаціях»*	*01.08.22-31.08.22*	
5.	*Дослідження ефективності розробленої інформаційної системи, оцінення розроблених пропозицій, написання п'ятого розділу роботи*	*01.09.22-30.09.22*	
6.	*Завершення оформлення розрахунково-пояснювальної записки та презентаційних матеріалів*	*01.10.22-31.10.22*	
7.	*Завершення роботи в цілому*	*01.11.21-30.11.22*	

Здобувач _____ О. А. Мельничук
(підпис)

Керівник роботи _____ В. В. Пташник
(підпис)

УДК 004.9 : 631.1

Розробка інформаційної системи автоматизованого аналізу сайтів на основі ASP.NET Мельничук О. А. Кафедра ІТ – Дубляни, Львівський НУП, 2022. Кваліфікаційна робота: 87 с. текст. част., 29 рис., 4 табл., 7 арк. ілюстраційного матеріалу, 31 джерел.

Наведено особливості просування вебсайтів в пошукових системах. Проведено огляд існуючих інформаційних систем. Сформульовано невирішену науково-прикладну задачу створення інформаційної системи для автоматизованого аналізу вебсайтів.

Об’єкти дослідження – пошукова оптимізації вебсайтів, рекомендації пошукових систем, інформаційна система автоматичного аналізу сайтів та методи проєктування інформаційних систем.

Метою кваліфікаційної роботи є розробка інформаційної системи для автоматизованого аналізу сайтів на основі ASP.NET.

Окреслено задачі розробки інформаційної системи для автоматизованого аналізу сайтів. Вибрано засоби реалізації. Проведено проєктування інформаційної системи для заданих умов. Виконано функціональне моделювання діяльності інформаційної системи.

Подано особливості реалізації інформаційної системи автоматизованого аналізу сайтів на основі ASP.NET та практичне її використання.

Розроблено заходи із охорони праці та безпеки у надзвичайних ситуаціях під час використання інформаційної системи.

Визначено показники ефективності запропонованої інформаційної системи автоматизованого аналізу сайтів.

Ключові слова: інформаційна система, проєктування, аналіз, пошукові системи, вебсайт, вебдодаток, ASP.NET.

UDC 004.9: 631.1

Development of information system of automated site analysis on the basis of ASP.NET. Melnychuk O. A. Department of Information Technologies. Dubliany, Lviv National Environmental University, 2022. Qualification Paper: 87 p. of text part, 29 fig., 4 tables, 7 sheets of illustrative materials, 31 sources.

The features of website promotion in search engines are given. The existing information systems have been examined. The outstanding scientific and applied task of creation of information system for automated analysis of websites has been formulated.

The objects of research are search optimization of websites, recommendations of search systems, information system of automatic analysis of sites and methods of designing information systems.

The purpose of the qualification paper is to develop an information system for automated analysis of sites on the basis of ASP.NET.

The tasks of development of information system for automated analysis of sites have been outlined. The means of realization have been chosen. The information system for the given conditions has been designed. The functional modeling of the information system activity has been performed.

The features of the information system implementation of automated website analysis on the basis of ASP.NET and its practical use are presented.

The measures on health and safety in emergency situations during the use of the information system have been developed.

The indicators of effectiveness of the suggested information system of automated website analysis are determined.

Keywords: information system, design, analysis, search systems, website, web application, ASP.NET.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

B2B – Business-To-Business;

B2C – Business-To-Consumer;

CSS – Cascading Style Sheets;

DNX – Dot Net Execution Environment;

HTML – Hypertext Markup Language;

HTTP – Hypertext Transfer Protocol;

IIS – Internet Information Services;

MVC – Model-View-Controller;

SEO – Search Engine Optimization;

SQL – Structured Query Language;

UML – Unified Modeling Language

URL – Uniform Resource Locator;

XML – Extensible Markup Language;

ВСТУП

Інтернет посів вагоме місце у житті сучасного суспільства. Він забезпечує швидкий обмін інформацією на великих відстанях та широко застосовуються для просування та продажу товарів чи послуг, змінюючи традиційні принципи взаємодії бізнесу з клієнтами.

Незважаючи на зростання популярності мобільних додатків та соціальних мереж, вебсайти залишаються одним з основних інструментів бізнесу. Вони дозволяють знайомити потенційних клієнтів з товарами та послугами бізнесу та оформляти замовлення віддалено, без візиту до магазину чи офісу.

Намагаючись збільшити кількість відвідувачів вебсайту компанії використовують різні способи. Один з найбільш популярних – пошукова оптимізація, яка дозволяє покращити позиції сайту в результатах пошуку. Недоліком пошукової оптимізації є те, що вона не дає миттєвих результатів та вимагає великої кількості рутинної роботи з оптимізації сайту. Для виявлення і виправлення потенційних проблем в оптимізації сайту спеціалістам з пошукової оптимізації часто доводиться вручну перевіряти всі сторінки вебсайту, що вимагає батаго часу. Проте частину цієї роботи можна автоматизувати за допомогою автоматизованих систем.

Тому метою даної роботи є розробка актуальної інформаційної системи автоматизованого аналізу сайтів, яка б виконувала рутинну роботу з пошуку можливих проблем в пошуковій оптимізації сайту.

Дана система повинна завантажувати, аналізувати всі сторінки вебсайту та складати звіт для користувача. При цьому система повинна бути доступною з будь-яких пристроїв, легко налаштовуватися під вимоги конкретного підприємства і передбачати подальше розширення набору функцій.

Об’єктом дослідження даної роботи є процес збільшення популярності вебсайтів компаній, які продають товари або послуги через Інтернет.

Предметом дослідження даної роботи є сучасні технології, які можуть зменшити об’єм рутинної праці спеціалістів з пошукової оптимізації вебсайтів.

Результатом даної роботи є розроблена інформаційна система автоматизованого аналізу сайтів, яка може бути використана в процесі пошукової оптимізації і налаштована відповідно до вимог конкретного підприємства.

РОЗДІЛ 1

АНАЛІЗ СТАНУ ПИТАННЯ В ТЕОРІЇ ТА ПРАКТИЦІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1. Вивчення джерел трафіку для вебсайтів

Сьогодні конкуренція в мережі Інтернет є високою, й отримати нових відвідувачів на сайт дуже непросто. Особливо це стосується нових сайтів які поки не відомі широкій публіці. Пошук можливостей для росту кількості відвідувачів (трафіку) варто почати з аналізу джерел, які потенційно можуть приводити на сайт нових людей.

Кожному переходу на вебсайт відповідає своє джерело трафіку – канал, з якого користувачі потраплять на ресурс. Таким джерелом можуть бути платформи різних типів, наприклад, це можуть бути інші вебсайти, пошукові системи, соціальні мережі, мобільні додатки, месенджери, рекламні платформи, Email чи SMS повідомлення. Також користувач може зайти відразу на потрібний йому сайт. Такі переходи на сайт називають прямим трафіком.

Основними джерелами трафіку, які приводять на сайти найбільшу кількість відвідувачів, є:

- Пошукові системи – переходи через посилання з пошукової системи, яке користувачі отримують у відповідь на свій пошуковий запит. Такий тип трафіку також називають органічним.
- Прямий трафік – прямий перехід на сайт. Користувач відразу заходить на потрібний сайт, вручну вводячи адресу сайту або користуючись закладками в браузері. Також до прямого трафіку відносять переходи, джерело яких не можна встановити.
- Соціальні мережі – перехід на сайт з соціальних мереж. Користувач заходить на сайт через посилання, яке він знаходить в соціальній

мережі. До цього джерела відносять переходи з таких ресурсів, як Facebook, Twitter чи Instagramm. Враховуються переходи як з вебверсій соціальних мереж, так і з їх мобільних додатків.

– Email повідомлення – перехід на сайт через посилання з email листа. Дане джерело трафіку характерне для комерційних сайтів, які ведуть розсилку email повідомлень для свої клієнтів.

У 2020 році компанія GrowthBadger провела дослідження [1] джерел трафіку для найбільших вебсайтів світу. У дослідженні були використані данні сервісу SimilarWeb, які включають 3,25 мільярдів відвідувань вебсайтів.

У результаті дослідження компанія GrowthBadger прийшла до таких висновків:

– Пошукові системи є найбільшим джерелом трафіку для вебсайтів і забезпечують понад 60% відвідувань (рис. 1.1). Для більшості галузей доля переходів з пошуку перевищує 50%. При цьому деякі галузі більше залежать від пошукового трафіку, ніж інші.
– Галузь, яка найбільше залежить від пошукових систем,– це медицина. Для вебсайтів цієї категорії доля пошукового трафіку становить 88% від усіх переходів на сайт. Також від пошукового трафіку дуже залежать такі галузі, як туризм, дизайн, фінанси, мода та громадське харчування, для яких доля пошукового трафіку перевищує 60%.
– Пошукові системи надають у 8 разів більше трафіку, ніж соціальні мережі, та 2 рази більше, ніж прямі заходи.
– Найбільшим джерелом трафіку після пошукових систем є прямий трафік, який приносить більше 27% переходів на сайт.

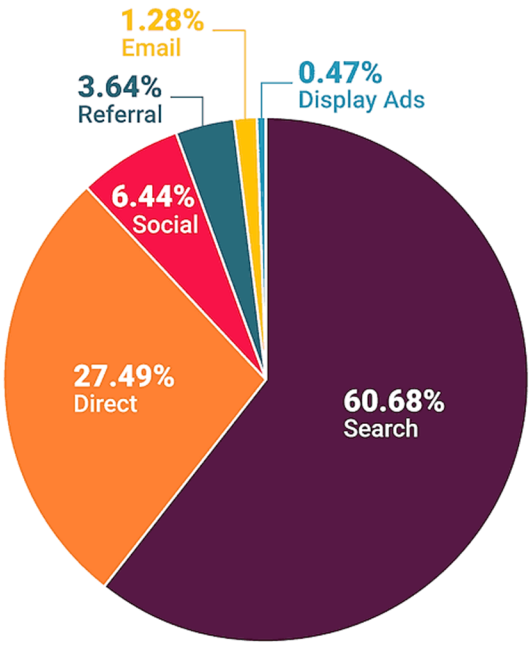


Рисунок 1.1 – Відсоток переходів з пошукових систем у всіх галузях згідно з дослідженням GrowthBadger

Серед пошукових систем беззаперечним лідером є Google. Згідно з даним сервісу Statcounter [2] за період з травня 2021 по травень 2022 року частка Google на світовому ринку пошукових систем становить майже 92% (рис. 1.2). Таку ж частку в 92% Google має і на українському ринку пошуку [3].

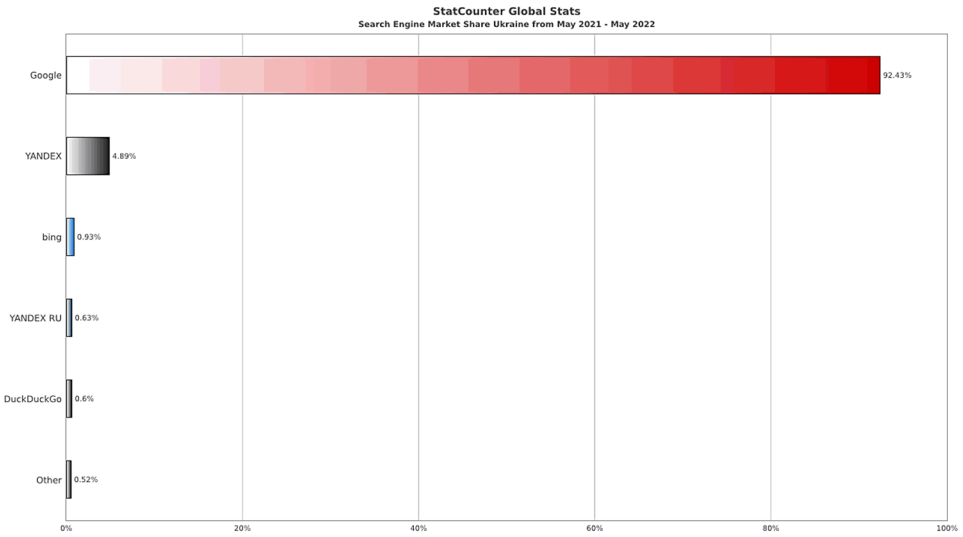


Рисунок 1.2 – Частка Google на українському ринку пошукових систем згідно з даними сервісу Statcounter

При цьому переходи з різних джерел трафіку не є рівнозначними для бізнесу. Так переходи з пошуку з більшою ймовірністю приводять на сайт користувачів, які створюють замовлення.

У 2021 році компанія FirstPageSage [4] опублікувала дослідження різних джерел трафіку згідно з відсотком замовлень, які приносили ці переходи. У дослідженні були використанні дані 150 клієнтів компанії за 2016-2021 роки.

У результаті дослідження компанія FirstPageSage визначила, що пошуковий трафік є найбільш ефективним у сфері B2B. Тут відсоток конверсії відвідування в замовлення становив 2,6%. Ефективність інших джерел трафіку в сфері B2B наведена в таблиці 1.1.

Таблиця 1.1 – Середній коефіцієнт конверсії за джерелом трафіку для B2B.

Джерело трафіку	Коефіцієнт конверсії в замовлення для B2B
Пошукові системи	2,6%
Email	2,4%
Прямі заходи	1,9%
Соціальні мережі	1,7%
Реклама в пошукових системах (PPC)	1,5%
Перенаправлення	1,1%
Реклама в соціальних мережах	0,9%
Медійна реклама	0,3%

У сфері B2C пошуковий трафік теж демонструє високі результати (2,1% конверсій) поступаючись тільки Email розсилкам (2,8%) та соціальним мережам (2,4%). Ефективність інших джерел трафіку в сфері B2C наведена в таблиці 1.2.

Таблиця 1.2 – Середній коефіцієнт конверсії за джерелом трафіку для B2C

Джерело трафіку	Коефіцієнт конверсії в замовлення для B2C
Email	2,8%
Соціальні мережі	2,4%
Пошукові системи	2,1%
Реклама в соціальних мережах	2,1%
Перенаправлення	1,8%
Прямі заходи	1,6%
Реклама в пошукових системах (PPC)	1,2%
Медійна реклама	0,7%

Орієнтуючись на результати вказаних досліджень, можна стверджувати, що пошукові системи є одним з основних джерел трафіку для будь-якого бізнесу. Тому при розробці і просуванні вебсайту необхідно орієнтуватися в першу чергу на пошукові системи, їх вимоги та рекомендації. Це дозволить уникнути типових помилок, які можуть обмежити кількість переходів з пошукових систем.

1.2. Вивчення рекомендацій пошукових систем для розробників вебсайтів

Принцип роботи пошукових систем заснований на індексації вебресурсів в мережі Інтернет та їх ранжування згідно з власними алгоритмами. Роботи пошукових систем постійно обходять відомі їм вебсторінки, аналізують їх вміст та додають у власну базу даних (пошуковий індекс). Отримавши запит від користувача, пошукова система

підбирає сторінки з індексної бази, ранжує їх згідно з релевантністю запиту, а потім віддає їх у вигляді списку. При цьому сторінки, які виводяться першими, отримують більшу частину переходів.

Для максимально повної індексації вебсайту та високого ранжування його сторінок він повинен виконувати рекомендації пошукових систем, які приводяться в їх офіційній документації.

У цьому розділі ми проаналізуємо основні рекомендації пошукової системи Google [5], які стосуються технічних питань сканування, індексування та ранжування сторінок вебсайту.

1.2.1. Структура URL

Для індексації та ранжуваня вебсторінок Google рекомендує створювати структуру URL-адресів яка буде простою та зрозумілою людині. В URL рекомендується використовувати осмислені слова, а не довгі цифрові ідентифікатори. Якщо вебсайт відвідують користувачі різних регіонів, необхідно створити таку структуру URL, яка дозволить легко виділяти фрагменти, що вказують на регіон. Слова які містяться в URL необхідно розділяти за допомогою знака дефіс.

Згідно з документацією Google, складна структура URL, наприклад, з великої кількістю параметрів, може ускладнити сканування сайту. Внаслідок цього робот Google може створювати значне навантаження на сервер. Крім того, є можливість, що йому не вдасться повністю просканувати весь контент сайту.

1.2.2. Метадані вебсторінок

Правильне зазначення метаданих гарантує, що Google зможе розпізнати HTML розмітку ваших сторінок. Google намагається розпізнати HTML-код, навіть якщо він неправильний або не відповідає стандарту

HTML, але помилки в розмітці можуть викликати проблеми з вашим сайтом у Google Пошуку. Наприклад, якщо у розділі <head> використовується неприпустимий елемент, Google проігнорує всі елементи, вказані після нього.

Усередині розділу <head> для метаданих можна використовувати лише такі елементи HTML: title, meta, link, script, style, base, noscript, template. При виявленні будь-якого іншого елемента Google буде вважати його кінцем розділу <head> і не буде зчитувати частину розділу, що залишилася в <head>.

При індексації сторінок роботи Google розпізнають та враховують значення наступних метатегів:

- <meta name="description" content="A description of the page" />
  Короткий опис сторінки. У деяких випадках воно може відображатися як фрагмент тексту в результатах пошуку.
- <meta name="robots" content="..., ..." />
  <meta name="googlebot" content="..., ..." />
  Інструкції зі сканування та індексування сторінки, призначені для всіх пошукових систем (robots) або тільки для Google (googlebot).
- <meta name="google" content="nositelinkssearchbox" />
  Вимкнення вікна пошуку на вашому сайті в результатах Google.
- <meta name="googlebot" content="notranslate" />
  Вимкнення автоматичного перекладу вашого сайту в результатах Google.
- <meta name="google" content="nopagereadaloud" />
  Заборона на озвучування тексту для продуктів Google.
- <meta http-equiv="Content-Type" content="..." charset=..." />
  <meta charset="..." >

Тип контенту та набір символів сторінки. Рекомендується використовувати набір символів Unicode/UTF-8.

– <meta http-equiv="refresh" content="..." url=..." />

Перенаправлення користувачів на новий URL через певний час.

– <meta name="viewport" content="...">

Інформація для браузера із вказівками щодо обробки сторінки на мобільному пристрої.

1.2.3. HTTP-заголовки

Заголовок X-Robots-Tag можна вказувати в HTTP-відповіді, що надсилається з певної URL-адреси. У заголовках X-Robots-Tag підтримуються ті ж самі директиви, як у метатегах robots. Наприклад, для того щоб заборонити індексації сторінки потрібно вказати X-Robots-Tag: noindex.

1.2.4. Коди статусу HTTP

Коли сервер, на якому розміщено сайт, отримує запит клієнта (наприклад, браузера або пошукового робота), у відповідь він надсилає код статусу HTTP.

Для індексації пошуковою системою Google сторінки повинні повертати код 200 (success), 201 (created) або 202 (accepted). При отримані коду 3xx (redirection) робот Google перейде по новому URL та спробує індексувати цю сторінку. При отримані коду 429 або 5xx (server errors) робот Google сповільнить індексацію сайту і з часом видалить сторінку з індексу. Сторінки, що повертають код 204 (no content) або 4xx (client errors), не індексуються.

1.2.5. Файл robots.txt

Якщо заборонити роботу Googlebot доступ до вебсторінок за допомогою налаштувань вебсервера або в файлі robots.txt, то він не зможе сканувати та індексувати розміщені на них матеріали. Це зазвичай призводить до погіршення позицій контенту в результатах пошуку Google.

robots.txt – це звичайний текстовий файл, який знаходиться в кореневій папці сайту та вміщує в себе інструкції (директиви) для пошукових роботів. Директивами для пошукових систем є:

- user-agent – обов'язкова директива, яка вказує назву пошукового робота, до якого відносяться вказані нижче інструкції.
- disallow – директива для обмеження доступу до каталогів вебсайту чи окремих сторінок.
- allow – директива для відкриття доступу до каталогів вебсайту чи окремих сторінок.
- sitemap – директива, яка вказує на шлях до карти сайту в форматі XML.

1.2.6. Посилання на сторінці

Роботи Google можуть переходити лише за посиланнями у правильно вказаних тегах <a> з атрибутом href та дійсними URL-адресами. Інші формати посилань не підтримуються. Це означає, що Google не може обробляти теги <a> без атрибуту href та теги, які використовують події скриптів та функціонують як посилання.

Вихідні посилання на інші сайти, які присутні на вебсторінці, рекомендується маркувати за допомогою атрибуту rel у тегу <a>. Для рекламних посилань Google рекомендує використовувати атрибут rel="sponsored", а для посилань, опублікованих користувачами, – атрибут rel="ugc", для інших зовнішніх посилань – атрибут rel="nofollow".

1.2.7. XML карта сайту

Sitemap.xml – це карта сайту (список всіх сторінок) в форматі XML. У файлі Sitemap міститься інформація про те, як організовано контент на сайті. Вона допомагає Google та іншим пошуковим системам точніше індексувати матеріали сайту. Наприклад, у файлі Sitemap можна вказати, які розділи сайту найбільш важливі, і повідомити додаткову інформацію про них (коли сторінка востаннє оновлювалася, чи існують її версії іншими мовами тощо). Також до файлу Sitemap можна додати додаткові відомості про контент різних типів, у тому числі про відео, зображення та новини.

Google рекомендує створювати файл Sitemap.xml для сайтів з великою кількістю сторінок, а також для сайтів з великим обсягом мультимедійного контенту (відео та зображення) або новин.

1.3. Вивчення існуючих досліджень на тему пошукової оптимізації вебсайтів

Рекомендації пошукової системи Google не можуть включати всіх проблем, з якими стикаються власники і розробники сайтів, тому важливо вивчити досвід компаній, які займаються пошуковою або SEO-оптимізацією сайтів для своїх клієнтів.

У 2021 році українська компанія Serpstat опублікувала власне дослідження [6] найбільш розповсюджених помилок в SEO-оптимізації сайтів. Дослідження базується на перевірці 58 тисяч сайтів та 288 мільйонів сторінок.

Згідно з дослідженням Serpstat найбільш розповсюдженим типом помилок є помилки в роботі мікророзміткою, яка використовується для позначення різних типів контенту на сторінці. Ці проблеми зустрічаються на 30% сайтів (рис. 1.3). Серед цих помилок частіше всього зустрічається

відсутність мікророзмітки Twitter Card (44% випадків), Open Graph (30%) і Schema.org (26%).

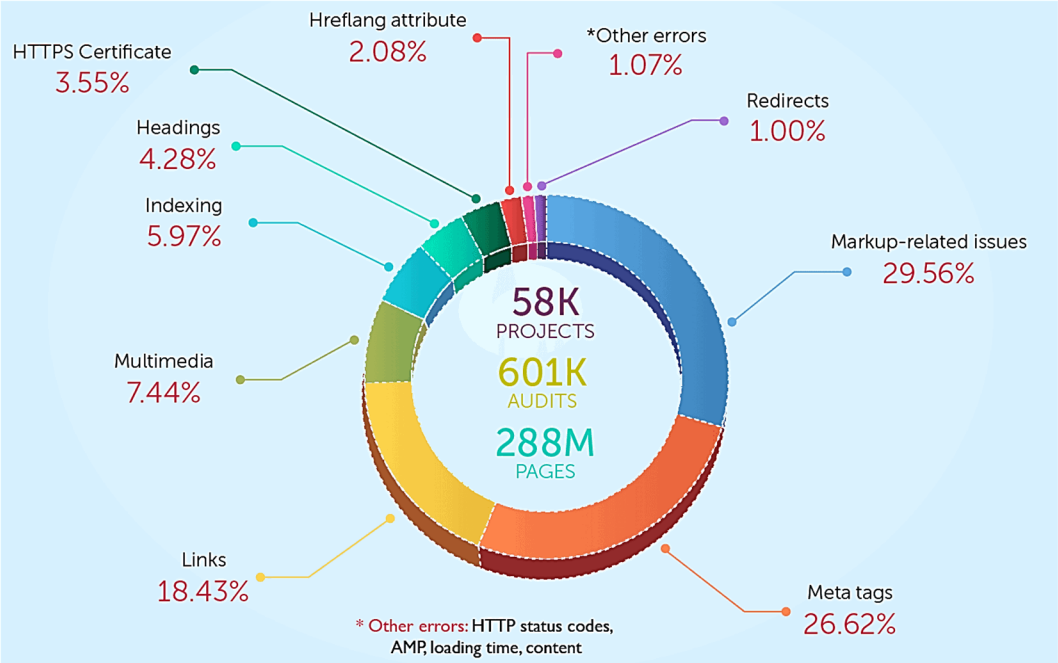


Рисунок 1.3 – Найбільш розповсюджені типи помилок в пошуковій оптимізації згідно з дослідженням Serpstat

На другому місці знаходяться помилки в роботі з метатегами і заголовками сторінки. Найбільш розповсюджені помилки цього типу – надто довгий заголовок Title (29% випадків) (рис. 1.4) і мета-тег Description (21%). Компанія Serpstat рекомендує дотримуватися максимальної довжини в 65 символів для Title і 300 символів для Description.

Також зустрічаються інші проблеми з метатегами та заголовками, наприклад, дублювання Title/Description, відсутність тегів Title/Description, дублювання тега H1, надто короткі теги та інше.

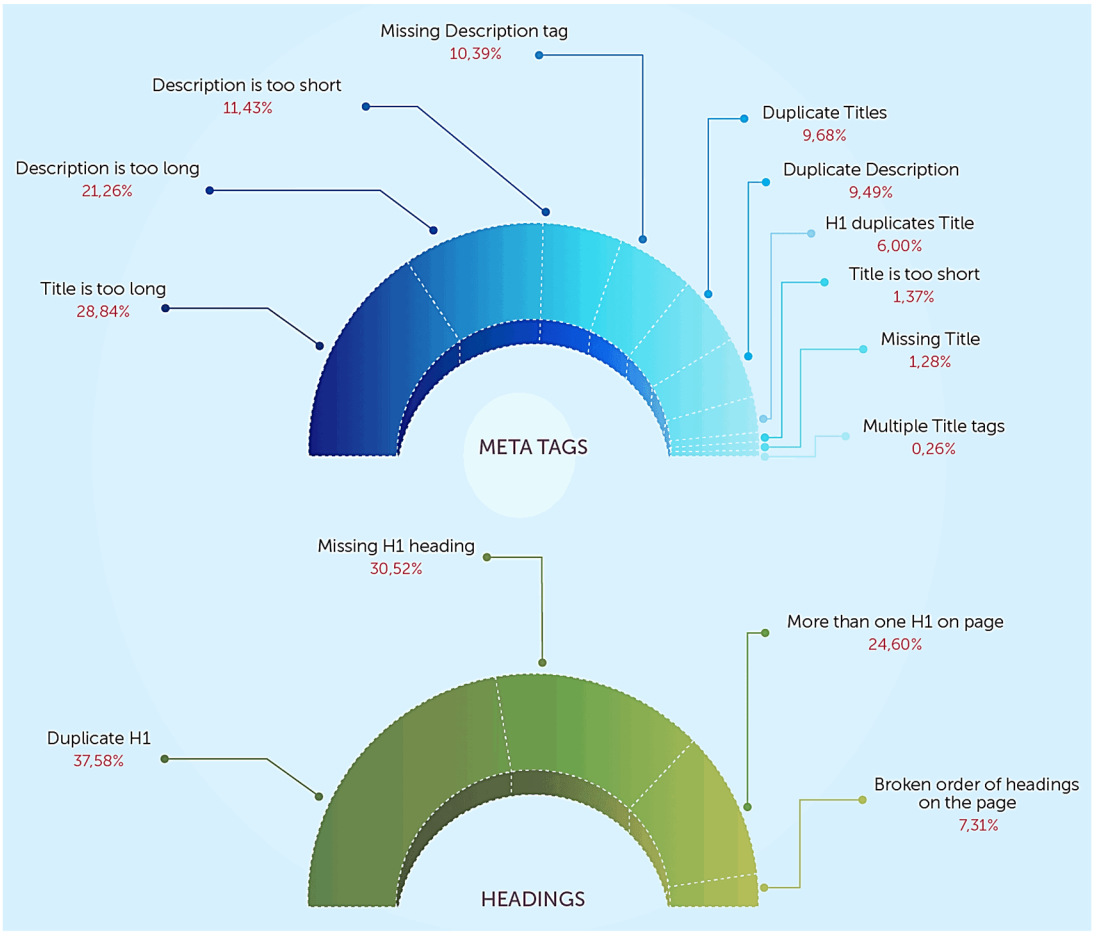


Рисунок 1.4 – Розповсюдженість помилок в роботі з метатегами і заголовками сторінки згідно з дослідженням Serpstat

На третьому місці за розповсюдженням знаходяться помилки в розмітці посилань. Розробники сайтів часто не використовують атрибут rel="nofollow" (63% випадків) для зовнішніх посилань або навпаки використовують цей тег для внутрішніх посилань (34%). Згідно з документацією Google цей атрибут варто застосовувати тільки для зовнішніх посилань.

Схоже дослідження [7] в 2021 році випускала інша українська компанія – SiteChecker. У цьому дослідженні було проаналізовано 52 тисячі сайтів та 6 мільйонів сторінок (рис. 1.5). Результати, які отримала команда SiteChecker, дуже схожі на результати Serpstat.

Всього було виявлено 21 мільйон помилок, які впливають на пошукову оптимізацію сайту. 24% з цих помилок були оцінені як критичні, тобто такі, що можуть серйозно вплинути на результати сайту в пошуку.

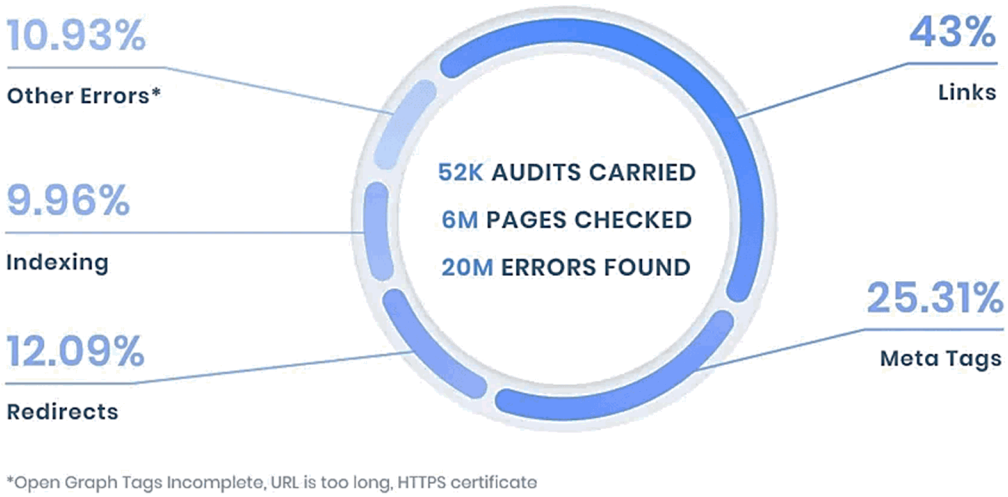


Рисунок 1.5 – Результати дослідження компанії SiteChecker

Серед критичних помилок найбільш розповсюдженими виявилися помилки, пов’язані з посиланнями та метатегами title, h1 та description. У 43% випадків один з цих метатегів був відсутній на сторінці, ще в 21% випадків на сторінці знаходилось два заголовки h1.

Також 9% сторінок відповідали HTTP кодом 4xx та 2% кодом 5xx, що повністю виключає можливість їх індексації та відображення в результатах пошуку.

Виконання рекомендацій пошукових систем та виправлення помилок, на які вказують наведені вище дослідження, відносяться до внутрішньої оптимізації сайту. Ці методи оптимізації сайту є основою сучасних підходів до SEO-оптимізації сайтів. Тільки повністю виконавши всі рекомендації з внутрішньої оптимізації сайту, є сенс переходити до більш складних та вартісних способів таких, як зовнішня оптимізація сайту чи робота з поведінковими факторами.

Особливістю внутрішньої оптимізації є необхідність перевірки великої кількості елементів сайту таких, як заголовки, метатеги, посилання, коди відповіді HTTP і т. д. Це вимагає від спеціаліста великої кількості ручної роботи і витрати часу.

Альтернативою є розробка автоматизованих систем для аналізу сайтів, які будуть автоматично перевіряти всі сторінки сайту та виявляти типові помилки у внутрішній оптимізації ресурсу.

1.4. Вивчення існуючих автоматизованих систем для аналізу сайтів

Для роботи над внутрішньою оптимізацією сайтів існує велика кількість інструментів, але більшість із них не є повністю автоматичними і не дають користувачу конкретних рекомендацій щодо виправлення знайдених проблем. Вони лише полегшують збір даних із сайту, тоді як роботу з аналізу отриманих даних доводиться виконувати спеціалісту.

Прикладом такого інструменту є розширення для браузера SEO META in 1 CLICK (рис 1.6) [8]. Це розширення збирає дані, які стосуються внутрішньої оптимізації, та відображає їх у вигляді списків, які стосуються різних аспектів внутрішньої оптимізації (заголовки, зображення, посилання, соціальні мережі та інше).

Недоліком цього інструменту є те, що він дозволяє аналізувати лише одну сторінку за раз, не дозволяє експортувати дані, а також не дає чітких рекомендацій, щодо виявлених на сторінці проблем.

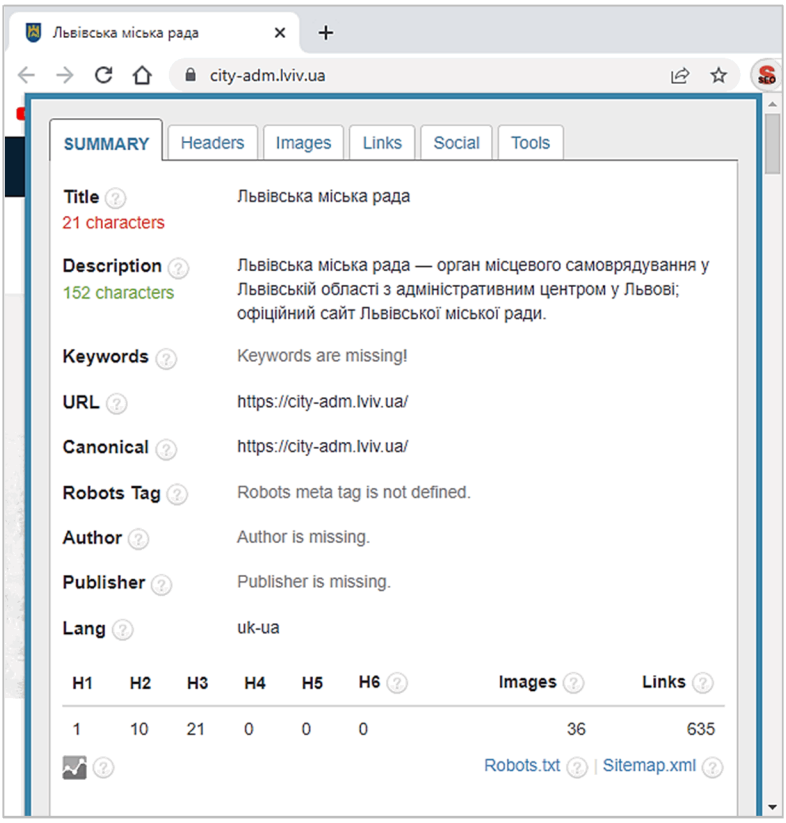
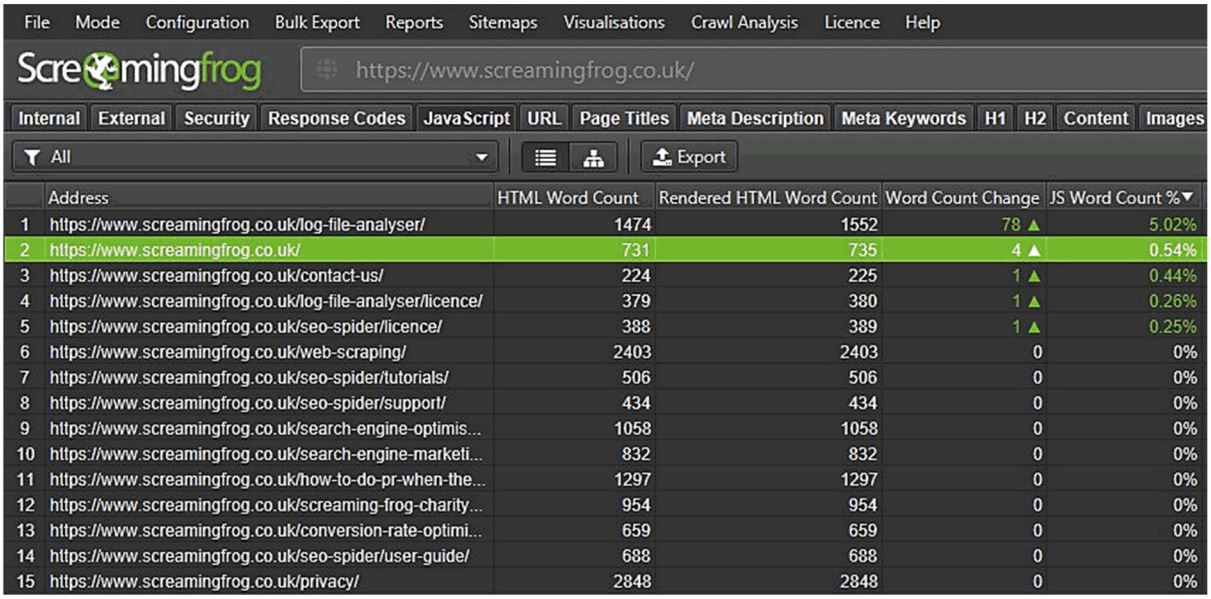


Рисунок 1.6 – Інтерфейс розширення для браузера SEO META in 1 CLICK

Ще один тип інструментів для аналізу сайтів – це програми для парсингу, наприклад Screaming Frog SEO (рис. 1.7) [9] або Netpeak Spider (рис. 1.8) [10], яка розробляється українською компанією Netpeak.

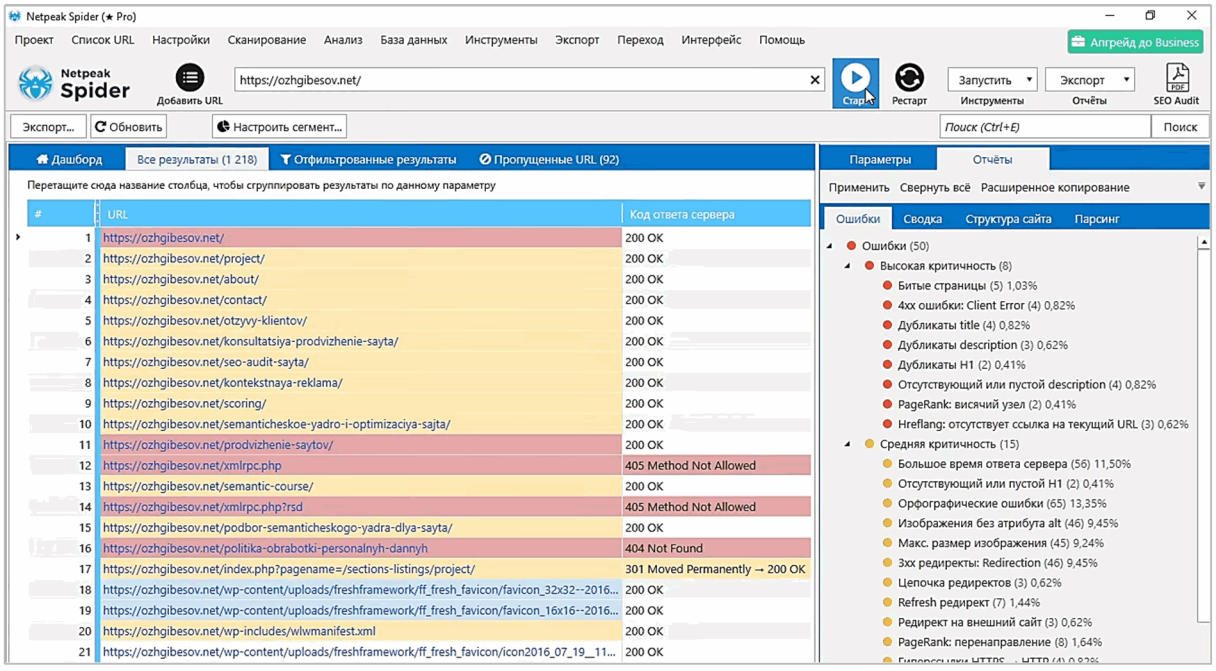
Такі програми завантажують усі сторінки сайту та відображують їх вміст у вигляді великої таблиці, де кожен рядок відповідає одному URL, а стовпець одному з параметрів сторінки. В таких таблицях виводиться інформація про значення метатегів, заголовків, кодів відповіді HTTP, об'єм контенту, швидкість завантаження сторінки та інше. Також програми парсери повідомляють про деякі типові помилки у внутрішній оптимізації сайту.



	Address	HTML Word Count	Rendered HTML Word Count	Word Count Change	JS Word Count %
1	https://www.screamingfrog.co.uk/log-file-analyser/	1474	1552	78 ▲	5.02%
2	https://www.screamingfrog.co.uk/	731	735	4 ▲	0.54%
3	https://www.screamingfrog.co.uk/contact-us/	224	225	1 ▲	0.44%
4	https://www.screamingfrog.co.uk/log-file-analyser/licence/	379	380	1 ▲	0.26%
5	https://www.screamingfrog.co.uk/seo-spider/licence/	388	389	1 ▲	0.25%
6	https://www.screamingfrog.co.uk/web-scraping/	2403	2403	0	0%
7	https://www.screamingfrog.co.uk/seo-spider/tutorials/	506	506	0	0%
8	https://www.screamingfrog.co.uk/seo-spider/support/	434	434	0	0%
9	https://www.screamingfrog.co.uk/search-engine-optimis...	1058	1058	0	0%
10	https://www.screamingfrog.co.uk/search-engine-marketi...	832	832	0	0%
11	https://www.screamingfrog.co.uk/how-to-do-pr-when-the...	1297	1297	0	0%
12	https://www.screamingfrog.co.uk/screaming-frog-charity...	954	954	0	0%
13	https://www.screamingfrog.co.uk/conversion-rate-optimi...	659	659	0	0%
14	https://www.screamingfrog.co.uk/seo-spider/user-guide/	688	688	0	0%
15	https://www.screamingfrog.co.uk/privacy/	2848	2848	0	0%

Рисунок 1.7 – Інтерфейс програми Screaming Frog SEO

Перевагою даного типу програм є те, що вони дозволяють аналізувати відразу весь сайт та виявляти проблеми на сторінках, які розміщенні глибоко в структурі сайту. Проте проблеми, які не виявляються автоматично, необхідно шукати самостійно, аналізуючи зібрані програмою дані. Тому робота з парсерами все ще вимагає великих витрат часу.



#	URL	Код ответа сервера
1	https://ozhgibesov.net/	200 OK
2	https://ozhgibesov.net/project/	200 OK
3	https://ozhgibesov.net/about/	200 OK
4	https://ozhgibesov.net/contact/	200 OK
5	https://ozhgibesov.net/otzyvy-klientov/	200 OK
6	https://ozhgibesov.net/konsultatsiya-prodvizhenie-sayta/	200 OK
7	https://ozhgibesov.net/seo-audit-sayta/	200 OK
8	https://ozhgibesov.net/kontekstnaya-reklama/	200 OK
9	https://ozhgibesov.net/scoring/	200 OK
10	https://ozhgibesov.net/semanticheskoe-yadro-i-optimizaciya-sajta/	200 OK
11	https://ozhgibesov.net/prodvizhenie-saytov/	200 OK
12	https://ozhgibesov.net/xmlrpc.php	405 Method Not Allowed
13	https://ozhgibesov.net/semantic-course/	200 OK
14	https://ozhgibesov.net/xmlrpc.php?rsd	405 Method Not Allowed
15	https://ozhgibesov.net/podbor-semanticheskogo-yadra-dlya-sayta/	200 OK
16	https://ozhgibesov.net/politika-obrabotki-personalnyh-dannyh	404 Not Found
17	https://ozhgibesov.net/index.php?pagename=/sections-listings/project/	301 Moved Permanently → 200 OK
18	https://ozhgibesov.net/wp-content/uploads/freshframework/ff_fresh_favicon/favicon_32x32--2016...	200 OK
19	https://ozhgibesov.net/wp-content/uploads/freshframework/ff_fresh_favicon/favicon_16x16--2016...	200 OK
20	https://ozhgibesov.net/wp-includes/wlwmanifest.xml	200 OK
21	https://ozhgibesov.net/wp-content/uploads/freshframework/ff_fresh_favicon/icon2016_07_19__11...	200 OK

Рисунок 1.8 – Інтерфейс програми Netpeak Spider

До автоматизованих систем аналізу сайтів можна віднести такі українські вебсервіси, як Serpstat.com (розділ Аудит сайту) (рис. 1.10) [11] та Sitechecker.pro (розділ Site Audit) (рис. 1.9) [12].

Як і парсери, ці сервіси обходять усі сторінки сайту та виявляють помилки у внутрішній оптимізації сайту. Проте дані представляються по-іншому, користувачу пропонується список знайдених помилок, а не список усіх сторінок сайту.

При цьому знайдені помилки розділяються на групи згідно з пріоритетом (критичні помилки, попередження, ймовірні помилки та інше). Це дозволяє організувати роботу над сайтом та в першу чергу виправити найбільш серйозні проблеми. Обидва сервіси пропонують досить детальний опис виявлених проблем та надають рекомендації щодо їх виправлення.

У сервісі Serpstat.com присутня функція автоматичної перевірки сайту згідно зі встановленим розкладом. Проте відсутнє інформування користувача, щоб отримати дані нової перевірки, потрібно заходити на сайт. У Sitechecker.pro автоматична перевірка за розкладом відсутня.

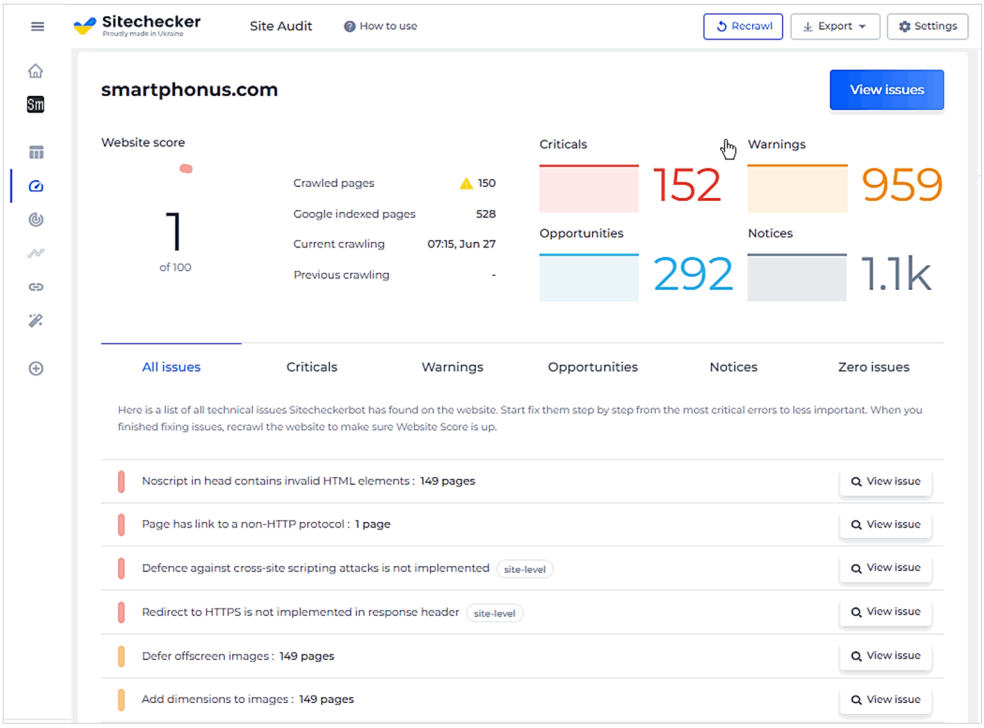


Рисунок 1.9 – Інтерфейс вебсервісу Sitechecker.pro

До інших недоліків обох цих сервісів можна віднести те, що вони не надають посилань на документацію пошукових систем. А вебсервіс Sitechecker.pro вимагає підтвердження власності на сайт та не має україномовного інтерфейсу, оскільки орієнтується в першу чергу на західних SEO-спеціалістів.

Крім того, обидва сервіси є частиною більших вебсервісів, які пропонують цілий комплекс інструментів для просування вебсайту. Наприклад, окрім пошуку помилок, сервіси Serpstat.com та Sitechecker.pro пропонують послуги з підбору ключових слів, аналізу зворотних посилань, моніторингу позицій у пошукових системах та інше. Тому ціна за користування ними є досить високою.

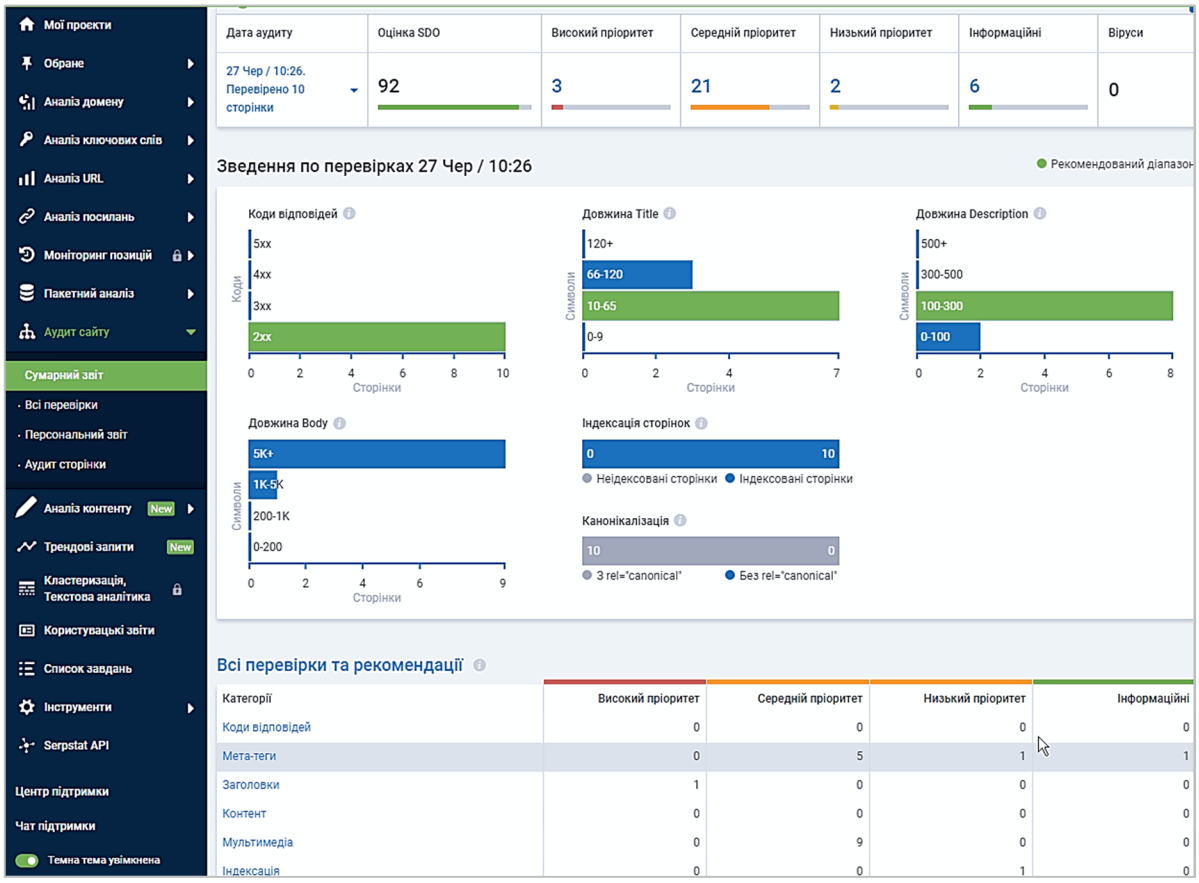


Рисунок 1.10 – Інтерфейс вебсервісу Serpstat.com

1.5. Постановка задачі на розробку

Відповідно до вивчених досліджень основним джерелом трафіку для вебсайтів є пошукові системи, найбільшою з яких є пошукова система Google, яка займає 92% ринку пошуку. Відсоток пошукового трафіку на вебсайтах залежно від галузі становить від 50 до 88%. При цьому пошуковий трафік є найбільш ефективним для бізнесу, коефіцієнт його конверсії в замовлення становить від 2,1% для B2C сфери та 2,6% для B2B.

Тому при розробці і підтримці сайту є сенс в його оптимізації під рекомендації пошукових систем. Це дозволить уникнути проблем під час індексації сайту пошуковими системами, а також підвищить позиції його сторінок в результатах пошуку.

Вивчивши рекомендації пошукових систем, стало зрозуміло, що така оптимізація вимагає великого об'єму робіт та витрат часу. Особливо, якщо сайт складається з великої кількості сторінок.

Для вирішення цієї проблеми є сенс в розробці автоматизованої системи аналізу сайтів, яка б виконувала перевірку всіх сторінок сайту на відповідність рекомендаціям пошукових систем та основним принципам SEO-оптимізації. Така система дозволить зменшити об'єм робіт, необхідних для оптимізації сайту, та зосередити ресурси підприємства саме на виправлені знайдених проблем, а не на їх пошуку.

На даний момент на українському ринку SEO-оптимізації сайтів присутні два інструмента, які можна вважати автоматизованими системами аналізу сайтів. Це вебсервіси Serpstat.com та Sitechecker.pro. Вони виконують автоматичний аналіз сайтів та повідомляють про знайдені проблеми. При цьому користувача повідомляють про критичність проблеми, причини її виникнення та надаються рекомендації, щодо її усунення. Проте дані сервіси не дають посилань на документацію пошукових систем та перенавантажені додатковими функціями. В

Sitechecker.pro відсутня автоматична перевірка сайту згідно з розкладом та підтримка української мови в інтерфейсі.

Враховуючи необхідність великої кількості однотипної ручної праці з оптимізації сайту та відсутність оптимальних інструментів для автоматизації, які б закрили всі потреби SEO-спеціалістів, є доцільність в розробці інформаційної системи автоматизованого аналізу сайтів.

Розроблена система повинна обходити всі сторінки сайту, виявляти помилки та автоматично складати звіт для користувача. Звіт повинен формуватися українською мовою та включати пояснення щодо виявлених помилок, посилання на документацію пошукових систем і надавати поради щодо їх усунення. Повторна перевірка сайту повинна відбуватися автоматично згідно зі встановленим розкладом, а про її результати користувача потрібно повідомити через email.

Висновки до розділу 1

- Пошукові системи є основним джерелом відвідувачів для вебсайтів, при цьому переходи з пошукових систем є одним з найбільш ефективних джерел для бізнесу. Тому для отримання максимальної кількості відвідувачів із пошукових систем є сенс в оптимізації комерційних сайтів під вимоги пошукових систем.
- Процес оптимізації вебсайту під вимоги пошукових систем включає велику кількість однотипної роботи щодо пошуку можливих проблем. Більшу частину цієї роботи можна автоматизувати за допомогою автоматизованої системи аналізу сайтів.
- На українському ринку вже існують інструменти для автоматичного пошуку проблем в оптимізації сайтів, проте вони не закривають усіх потреб SEO-спеціалістів. Часто такі програми перенасичені додатковими функціями, погано документовані та не завжди оснащені україномовним інтерфейсом.
- Враховуючи наявну проблему та недоліки існуючого програмного забезпечення, було сформовано завдання на розробку інформаційної системи аналізу сайтів.

РОЗДІЛ 2

ОБҐРУНТУВАННЯ ТА ВИБІР ІНСТРУМЕНТАРІЮ ДЛЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Аналіз вимог до інформаційної системи

Інформаційна система аналізу сайтів, яка розробляється, повинна експлуатуватись на комп'ютері зі швидким та стабільним доступом до мережі Інтернет, оскільки під час аналізу сайту необхідно завантажити та проаналізувати всі його сторінки. Тому аналіз великих сайтів може потребувати великої кількості запитів до сервера вебсайту та зайняти велику кількість часу. У випадку розробки настольної програми для аналізу сайтів користувач може зіткнутися зі сповільненням роботи його комп'ютера під час роботи такої програми. Також настольні програми доступні тільки для користувачів платформи, для якої вони були розроблені, що робить їх менш універсальними.

Альтернативою є розробка інформаційної системи у вигляді вебдодатку, який буде розміщуватись на хостингу та зможе опрацювати завдання користувача цілодобово, незалежно від робочого комп'ютера користувача та швидкості його підключення до Інтернету. Такий підхід також використовується деякими існуючими системами аналізу сайтів, наприклад, вебсервісами Serpstat.com та Sitechecker.pro.

Для забезпечення стабільної роботи інформаційної системи оптимальним варіантом буде саме розробка вебдодатку.

2.1. Аналіз доступних технологій для розробки інформаційної системи

На даний момент для розробки вебдодатків існує велика кількість мов програмування та фреймворків для них. Фреймворк – це програмний продукт, який прискорює та спрощує розробку програмного забезпечення. При використанні фреймворка необхідно написати лише той код, який є специфічним для продукту і реалізує логіку його роботи. Фреймворк забезпечує готові інструменти для роботи з базою даних, автентифікації користувачів, підтримки сеансів, захисту від атак, кешування, відокремлення логіки вебдодатку від представлення та інше.

Перед початком розробки інформаційної системи були розглянуті актуальні мови програмування, які використовуються для розробки вебдодатків, та найбільш популярні фреймворки для них.

2.1.1. Python та Django

Django – популярний серед великих компаній фреймворк для веброзробки на Python. Django має Model-View-Template (MVC) структуру і дотримується принципів проєктування DRY та Convention over configuration. Django пропонує зручну систему шаблонів з успадкуванням, вбудовані можливості автентифікації, просту маршрутизацію та багато інших можливостей. Особливий пріоритет надається безпеці. Фреймворк реалізує багато важливих принципів захисту самостійно, наприклад, запобігає виконанню коду на рівні шаблонів. Цей фреймворк працює з такими базами даних, як PostgreSQL, MySQL, SQLite та Oracle. Також Django включає вебсервер для стадії розробки.

2.1.2. JavaScript та Express

Express – це популярний JavaScript-фреймворк, який працює на базі Node.js. Він розширює Node.js інструментами, які полегшують розробку сучасних вебдодатків. Його використовують великі компанії, а також інші фреймворки, наприклад, Kraken, Sails та Loopback. Express пропонує надійний механізм маршрутизації для обробки динамічних URL-адрес, дозволяє додавати додаткове програмне забезпечення до будь-якої точки конвеєра обробки запитів, спрощує налагодження коду, надає шаблонізатор для динамічного рендерингу сторінок HTML на стороні сервера.

2.1.3. PHP та Laravel

Laravel – це популярний MVC-фреймворк для PHP. Laravel характеризується високим рівнем безпеки та захистом від проникнення SQL-коду та крос-сайтового скриптингу. Шаблонізатор цього фреймворку не лише дозволяє стандартизувати та повторно використовувати шаблони, але й дає розробнику свободу використовувати вбудовані засоби PHP. Модель, представлення та контролери чітко відокремлено одне від одного, що спрощує спільну роботу над проєктами. Основна проблема Laravel – недостатня продуктивність порівняно з Django чи Express. Для важких проектів це може стати суттєвим мінусом.

2.1.4. Java та Spring

Spring є найпопулярнішим інструментом розробки промислових додатків на Java. Найчастіше Spring використовується у великих корпоративних проєктах, це характерно для Java та пов'язаних із ним інструментів. Але завдяки універсальності застосовувати Spring можна і в інших випадках. Його завданням є дати розробникам більше свободи у проєктуванні та реалізації. Основна філософія Spring – універсальність.

Spring пропонує розробникам фундамент, на основі якого можна реалізувати будь-який додаток. Ще одна особливість – модульність. Spring складається з багатьох компонентів, які забезпечують фреймворку функціональність. Основними компонентами є: модуль АОП (аспектно-орієнтоване програмування), модуль доступу до даних, модуль транзакцій, модуль MVC, модуль авторизації та аутентифікації.

2.1.5. C# та ASP.NET Core

ASP.NET Core – це нова версія фреймворку ASP.NET, яка в основному призначена для роботи на платформі .NET. Даний фреймворк має відкритий вихідний код та розробляється корпорацією Майкрософт. ASP.NET Core підтримує декілька платформ (Windows, Linux і Mac), містить вбудований контейнер IoC для автоматичного впровадження залежностей, дозволяє використовувати та керувати сучасними front-end фреймворками (AngularJS, ReactJS, Umber, Bootstrap), підтримує різні вебсервери (IIS, Apache).

2.2. Обґрунтування вибору технології для розробки інформаційної системи

Після вивчення наявних технологій для розробки вебдодатків було прийнято рішення розробляти інформаційну систему аналізу сайтів із використанням мови програмування C# та фреймворку ASP.NET Core. Front-end частина додатку буде розроблятися з використанням мови розмітки гіпертексту HTML, каскадних таблиць стилей CSS та front-end фреймворку Bootstrap, який вже доступний в ASP.NET Core.

2.2.1. Основні відомості про мову програмування C#

На даний момент мова програмування C# є одною з найпотужніших та найбільш популярних мов програмування у сфері інформаційних технологій. Ця мова дозволяє розробляти найрізноманітніші проєкти: від простих додатків для мобільних пристроїв, до високонавантажених вебпорталів.

Мова C# відноситься до сімейства мов із C-подібним синтаксисом, тому синтаксично схожа на такі відомі мови, як Java та C++. Це об'єктно-орієнтовна мова з відносно строгою типізацією, оскільки в останніх версіях були додані деякі можливості динамічної типізації. Мова C# підтримує перевантаження операторів, події, делегати, замикання, анонімні функції, атрибути, ітератори, узагальнені типи та методи.

Перейнявши багато у своїх попередників – мов C, C++, Java, Smalltalk та опираючись на різні методики й підходи до розробки, ця мова взяла тільки краще та виключила деякі моделі, які зарекомендували себе як проблематичні. Наприклад, в C#, на відміну від C++, не підтримується багаторазове наслідування класів, але присутнє багаторазове наслідування інтерфейсів.

Мова C# продовжує розвиватись. З кожною версією з'являється все більше можливостей, як наприклад, лямбда вирази, асинхронні методи, динамічне зв'язування тощо.

2.2.2. Особливості платформи .NET Core

.NET Core – це модульна версія .NET Framework із підтримкою переносу на інші платформи. .NET Core є підмножиною повної версії .NET Framework та підтримує основні можливості для реалізації функцій додатків, необхідних для повторного використання цього коду на різних платформах.

Раніше структура середовища .NET мала вигляд, зображений на рисунку 2.1, в якій кожна цільова платформа підтримувала свою, окрему вертикаль: свою модель додатку, функціональну базу та середовище виконання.

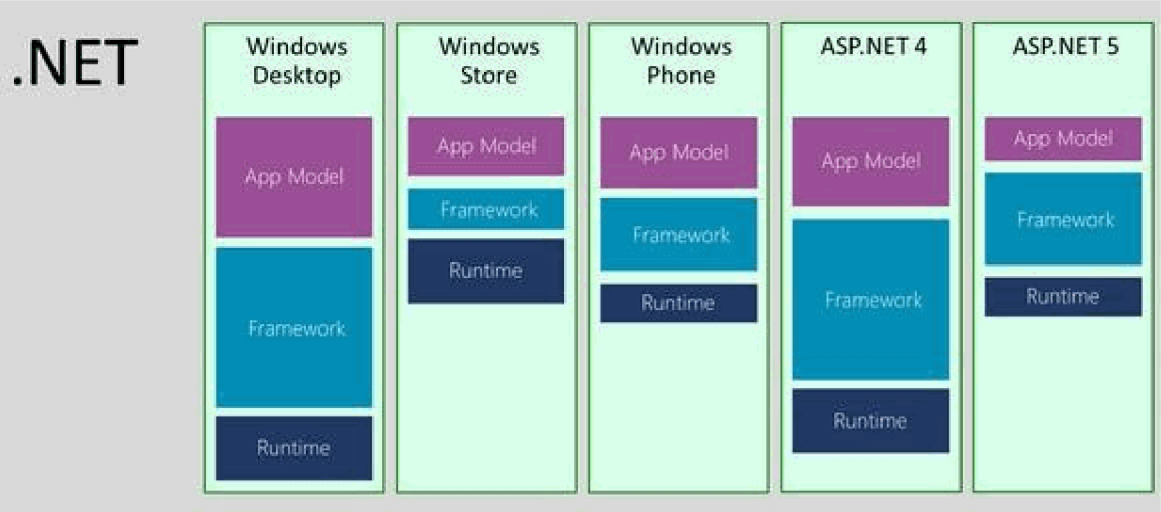


Рисунок 2.1 – Структура середовища .NET Framework

Це добре працює коли розробник націлений на одну вертикаль із даної структури. Кожна вертикаль надає набір API, який оптимізований для її функціональності. Проблема з'являється тоді, коли розробник хоче використовувати декілька вертикалей, оскільки з'являється завдання написання коду, який міг би виконувався на різних вертикалях.

Ця проблема сприяла переосмисленню структури середовища .NET і в результаті появі .NET Core. Нова структура зображена на рисунку 2.2. Вона дозволяє уникнути недоліків попередньої версії, оскільки .NET Core – це модульна реалізація, яка може використовуватися великим набором вертикалей, починаючи з дата-центрів і закінчуючи сенсорними пристроями, доступна з відкритим вихідним кодом і підтримувана на Windows , Linux і Mac OSX .

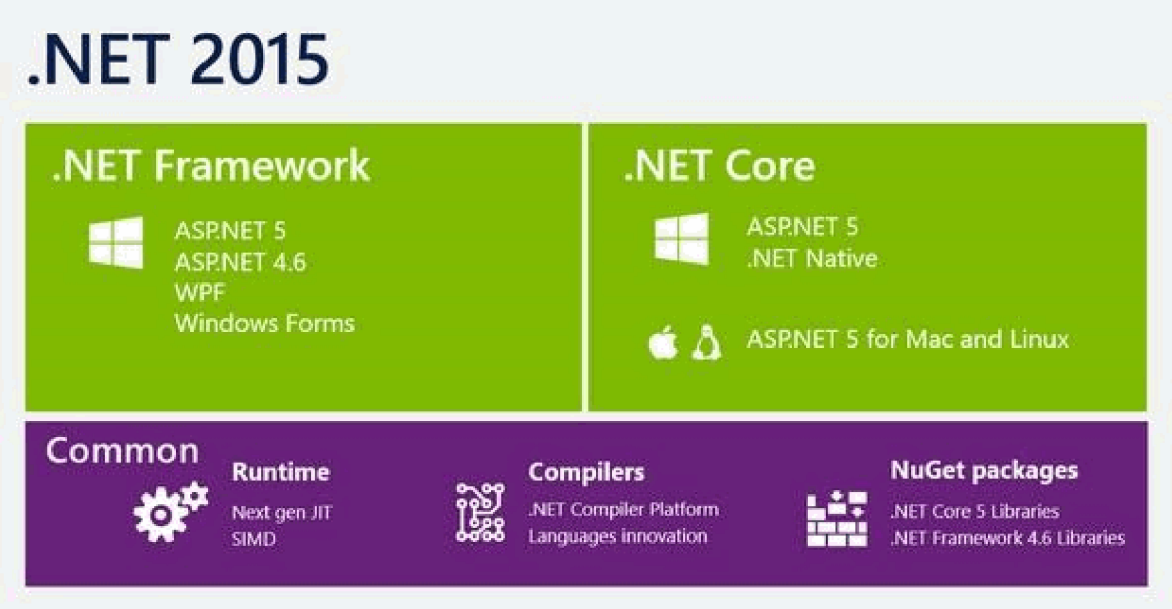


Рисунок 2.2 – Структура середовища .NET Core

Сам .NET Core скаладається з бібліотек, що носять назву "CoreFX", і з середовища виконання "CoreCLR". Обидва ці компоненти доступні через пакетний менеджер NuGet.

Основною перевагою .NET Core є можливість переносу, яка дозволяє позбавити вебдодаток залежності від наявності на цільовій машині конкретної версії платформи .NET. Дозволяється розгортати пакети CoreCLR разом із додатком, незалежно від версії .NET, яка використовується на комп'ютері. При цьому є можливість розгорнути на одній машині відразу кілька додатків, які будуть працювати з різними версіями CoreCLR .

CoreFX включає в себе набір бібліотек, що представляють інструменти з управління, доступу до консолі, колекцій, діагностики, системи введення-виведення, JSON, LINQ, XML тощо. Кожна з цих бібліотек має мінімальне число залежностей від інших бібліотек, що спрощує процес переносу та розгортання пакетів CoreFX.

Для створення додатків на базі .NET Core використовується кросплатформенне середовище виконання DNX. DNX – це середовище

виконання і SDK, які необхідні для розробки та запуску додатків .NET на платформах Windows, Linux та Mac. При цьому середовище DNX дозволяє запускати не тільки вебдодатки, але і нативні мобільні додатки, консольні програми тощо. Таким чином, з'являється можливість розробляти програму на одній платформі, а запускати на іншій в тому випадку, якщо на ній розгорнута сумісна версія DNX.

2.2.3. Особливості платформи ASP.NET Core

Платформа ASP.NET 5 – технологія від Microsoft, для створення вебдодатків. З однієї сторони, ASP.NET 5 є новим етапом розвитку платформи ASP.NET. Проте з іншої сторони, це зовсім нова технологія та реорганізація існуючої платформи. Саме тому платформу було перейменовано в ASP.NET Core. ASP.NET Core є повністю кросплатформенною та має відкритий вихідний код, який доступний на GitHub.

Додаток ASP.NET Core може використовувати два різних середовища: .NET Core і .NET Framework.

Завдяки модульності всі компоненти, які необхідні для вебдодатку, можуть завантажуватися, як окремі модулі, через пакетний менеджер NuGet. Крім того, на відміну від попередніх версій платформи, тепер немає необхідності користуватися бібліотекою System.Web.dll.

ASP.NET Core більш оптимізована для використання хмарних технологій. При розгортанні вебдодатку можна використовувати традиційний вебсервер IIS. Але також можна використовувати кросплатформенний вебсервер Kestrel. Структура платформи ASP.NET Core зображена на рисунку 2.3.

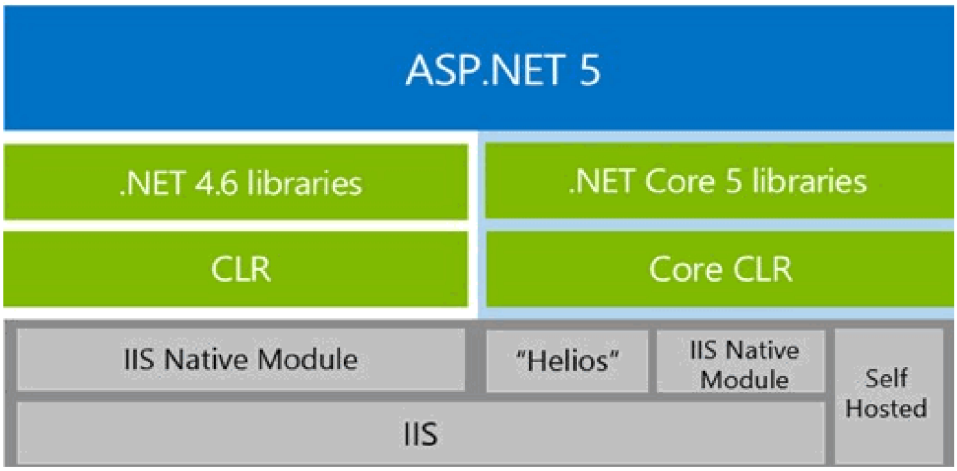


Рисунок 2.3 – Структура платформи ASP.NET

У структурі платформи можна виділити наступні ключові особливості, які відрізняють її від попередника:

- новий конвеєр HTTP-запитів з модульною структурою;
- можливість розгортати додаток як на IIS, так і на базі свого власного процесу;
- використання платформи .NET Core і її функціональності;
- розповсюдження пакетів платформи за допомогою NuGet;
- спрощена конфігурація для роботи у хмарі;
- вбудована підтримка для інверсії управління;
- кросплатформенність, можливість розробки і розгортання додатків ASP.NET на Windows, Mac і Linux.

2.2.4. ASP.NET Core MVC

За замовчуванням у ASP.NET Core застосовується шаблон проєктування MVC (Model-View-Controller), від якого й походить концептуальна назва проєкту.

Проте буде неправильно ототожнювати ASP.NET Core тільки з фреймворком ASP.NET Core MVC. Фреймворк ASP.NET Core MVC працює поверх платформи ASP.NET Core і використовується для того, щоб

спростити процес розробки програмних продуктів. Проте розробник не зобов’язаний використовувати MVC, він може використовувати чистий ASP.NET Core і вибудовувати на ньому всю логіку програми.

Шаблон проєктування MVC не є новою ідеєю в архітектурі програмних продуктів, він був розроблений в кінці 1970-х років в компанії Xerox як спосіб організації графічних компонентів у мові Smalltalk.

Концепція шаблону MVC передбачає розподіл програми на три компоненти:

- Модель (model) – описує дані, які використовуються в програмі, а також логіку, яка пов'язана безпосередньо з даними, наприклад, логіку валідації даних. Зазвичай об'єкти моделей зберігаються в базі даних. У MVC моделі представлені двома основними типами: моделі представлень, які застосовуються представленнями для відображення і передачі даних, і моделі домену, які описують логіку управління даними. Модель може зберігати дані та містити логіку управління цими даними. У той же час модель не повинна включати в себе логіку взаємодії з користувачем і не повинна визначати механізм обробки запиту. Крім того, модель має містити логіку відображення даних в представленні.
- Представлення (view) відповідає за візуальну частину або інтерфейс, доступний користувачеві. Часто це звичайна html-сторінка, через яку користувач взаємодіє з додатком. Також представлення може містити логіку, яка пов'язана з відображенням даних. При цьому представлення не повинно містити логіку обробки запиту користувача або управління даними.
- Контролер (controller) – основний компонент MVC, який забезпечує зв'язок між користувачем та програмою, представленням і моделлю. Він містить логіку опрацювання запиту користувача. Контролер отримує дані від користувача, обробляє їх та залежно від результатів

обробки відправляє користувачеві певний висновок, наприклад, у вигляді представлення, наповненого даними моделей.

Модель є незалежним компонентом, зміни контролера або представлення не повинні впливати на модель. Контролер і представлення відносно незалежні компоненти. Так, з представлення можна звертатися до певного контролера, а з контролера створювати представлення, але при цьому нечасто їх можна змінювати незалежно один від одного.

Така структура компонентів вебдодатку дозволяє реалізувати концепцію розділення відповідальності, в якій кожен компонент відповідає за свою частину функціональності системи. Це спрощує роботу над окремими компонентами, додаток з такою структурою простіше розробляти і підтримувати.

2.2.5. Entity Framework

Framework Core – це обгортка для реляційної бази даних, яка дозволяє представити базу даних в об'єктно-орієнтованій парадигмі. Ця технологія була розроблена компанією Microsoft для застосунків на платформі .NET Core. Завдяки Entity Framework Core застосунки, написані з використанням платформи .NET Core, дозволяють зберігати дані, використовуючи реляційну модель даних. За допомогою технології Entity Framework компанія Microsoft спростила завдання розробникам під час роботи з базою даних, оскільки тепер читання, зберігання, видалення та запит даних відбувається за допомогою Entity Framework.

При цьому база даних у Entity Framework представляє собою реляційну модель. Презентація даних у реляційній моделі є відображенням таблиць з рядками. Робота з даними у реляційній базі даних відбувається за допомогою запитів, які у свою чергу реалізовуються мовою SQL. Технологія Entity Framework реалізовує перетворення абстрактних об'єктів

у форму зберігання даних, яка прийнята в реляційній моделі, а саме: рядки, таблиці та запити до них.

Важливою перевагою Entity Framework є використання запитів LINQ для отримання даних. За допомогою LINQ ми можемо отримувати об’єкти сутностей із бази даних, а також отримувати об'єкти, які пов'язані різними асоціативними зв'язками.

Іншим ключовим поняттям в Entity Framework є блок EDM. Цей блок зіставляє класи сутностей із реальними таблицями в базі даних.

Entity Framework передбачає три можливі варіанти взаємодії з базою даних:

- Database first: Entity Framework самостійно генерує набір класів, які відображають модель даних уже існуючої бази даних;
- Model first: розробник розробляє модель даних, на основі якої Entity Framework надалі генерує реальну базу даних на сервері;
- Code first: розробник розробляє класи моделі даних, які будуть зберігатися в базі даних. Після чого Entity Framework на базі створених класів моделі створює базу даних з відповідними таблицями.

Висновки до розділу 2

- Інформаційна система аналізу сайтів повинна бути розроблена у вигляді вебдодатку. Це дозволить уникнути проблем з недостатньою пропускною здатністю Інтернет підключення користувача та забезпечить доступ до системи з будь-яких пристроїв.
- Для розробки інформаційної системи необхідно використовувати один із популярних фреймворків для розробки вебдодатків. Це дозволить значно скоротити терміни розробки системи.
- Оптимальним вибором для розробки інформаційної системи аналізу сайтів є мова програмування C# та фреймворк ASP.NET Core, який підтримує шаблон проєктування MVC та включає всі необхідні інструменти для розробки даної системи.
- Одним із ключових інструментів розробки на основі ASP.NET Core є Entity Framework. Дана технологія дозволяє представити базу даних в об'єктно-орієнтованій парадигмі.

РОЗДІЛ 3

РЕЗУЛЬТАТИ РОЗРОБКИ ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1. Моделювання варіантів використання

Для кращого розуміння розроблюваної інформаційної системи було виконано моделювання варіантів використання.

Варіант використання (use case) – це опис послідовних дій (включаючи варіації), які послідовно виконуються системою для отримання певного результату, важливого для дійової особи. Варіант використання описує набір послідовностей, кожна з яких – це взаємодія сутностей, що перебувають поза інформаційною системою (дійових осіб – актантів), із самою системою та її основними абстракціями.

Дійова особа – це логічно зв'язана множина ролей, яким слідують користувачі під час взаємодії з системою. Дійовими особами можуть бути як люди, так і автоматизовані системи, наприклад, бази даних.

Кожен варіант використання повинен виконувати певний обсяг роботи, яка має цінність з точки зору дійової особи.

UML-нотація варіантів використання зображується у вигляді еліпса, а дійові особи – «чоловічками». Це дозволяє візуалізувати один варіант використання в контексті інших і окремо від його реалізації.

Суб'єкт – це клас, який описаний повним набором варіантів використання, що складає собою систему або підсистему. Варіанти використання – це аспекти поведінки даного класу. Дійові особи – аспекти інших класів, що взаємодіють із суб'єктом. Усі варіанти використання описують повну поведінку суб'єкта.

Схема варіантів використання розробленої інформаційної системи обліку зображена на рисунку 3.1.

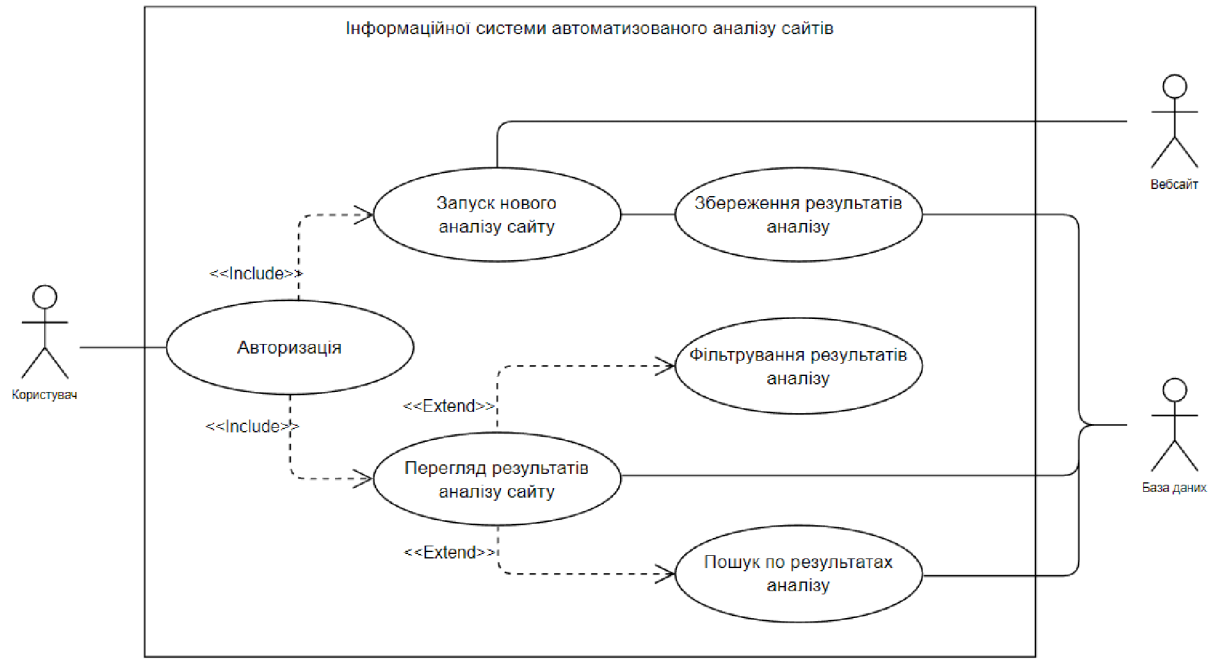


Рисунок 3.1 – Діаграма варіантів використання

Інформаційна система матиме 3 актори – користувач, база даних та вебсайт, який аналізується. Користувач матиме певний спектр дозволених дій з інформаційною системою. Після авторизації в системі користувач зможе запустити новий аналіз сайту або переглянути результати попереднього аналізу. При перегляді результатів аналізу користувачу будуть доступні опції для фільтрування та пошуку.

Перегляд результатів аналізу сайтів, пошук за результатами та збереження результатів передбачає взаємодію системи з базою даних.

3.2. Моделювання діаграми станів

Щоб уявити поведінку системи більш детально на логічному рівні, були розроблені діаграми станів основних компонентів майбутньої системи.

Діаграма станів відображає скінчений автомат у вигляді графу, вершинами якого є стани об’єкта, для якого необхідно змоделювати поведінку, а переходами – події, які переводять об’єкт, який розглядається,

з одного стану в інший. Вважається, що час перебування об’єкта в деякому стані набагато більший за час, необхідний для переходу з одного стану в інший, тобто переходи між станами відбуваються миттєво.

Діаграма станів використовується для опису станів об'єкта й умов переходу між ними. Опис станів дозволяє описати модель поведінки об'єкта при отримані різних повідомлень і взаємодії з іншими об'єктами.

Основними елементами діаграми станів є:

- Автомат (State machine) – це опис декількох станів, через які проходить об’єкт на протязі свого життя, реагуючи на події. Опис також включає реакції на події.
- Стан (State) – це момент в житті об’єкту, на протязі якого він виконує певну умову, здійснює певну роботу або очікує на якусь подію.
- Подія (Event) – це опис суттєвого факту, який виникає у процесі роботи. Подія – це стимул, здатний викликати спрацювання переходу з одного стану в інший.
- Перехід (Transition) – це відношення між двома станами, яке показує, що об’єкт, повинний виконати певну дію і перейти в інший стан.

Вершинами графа в діаграмі станів є можливі стани автомата, зображені відповідними графічними символами, а стрілки позначають його переходи зі стану в стан. Діаграми станів можуть бути вкладені одна в одну для більш детального представлення окремих елементів моделі.

Стан на діаграмі зображується прямокутником із закругленими вершинами. Прямокутник може бути розділений на дві секції горизонтальною лінією. Якщо використовується лише одна секція, то в ній вказується тільки ім'я стану. При використанні двох секцій, в першій з них вказується ім'я стану, а в другій список певних внутрішніх дій або переходів в даному стані.

На рисунку 3.2 показана діаграма станів UML, яка відображає поведінку компонента, що завантажує сторінки вебсайту, який аналізується

інформаційною системою. На діаграмі зображені різні стани, в яких може перебувати даний компонент системи.

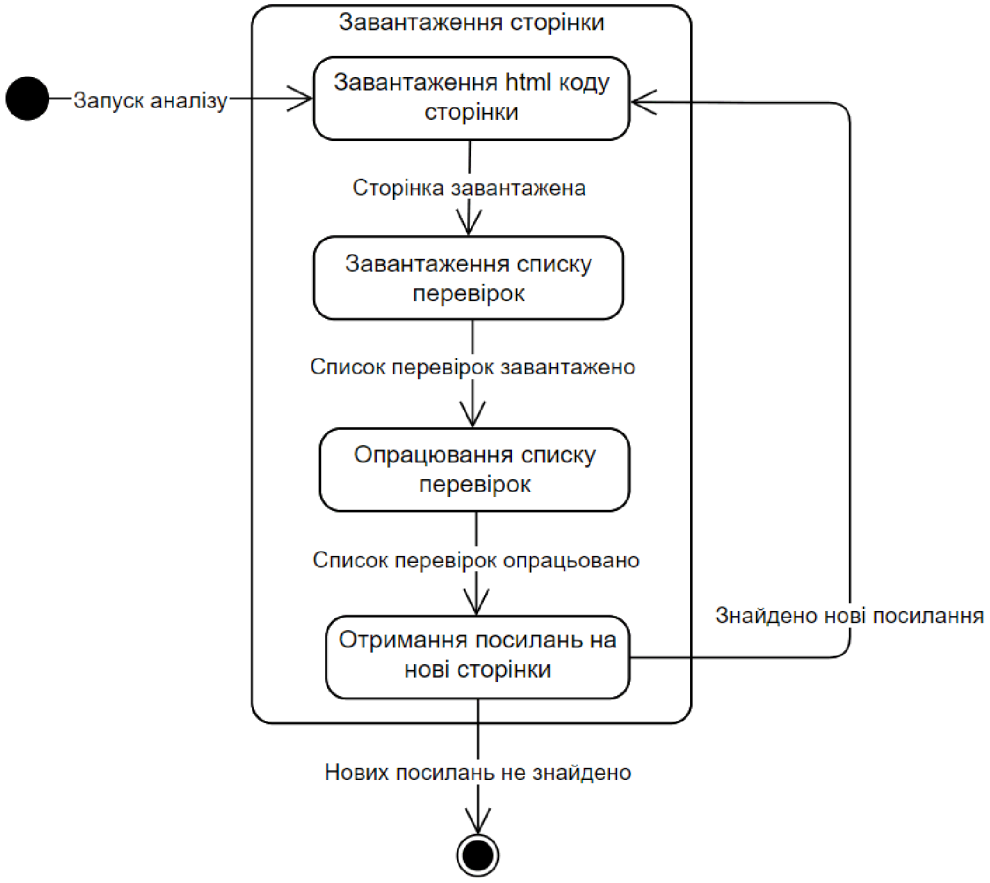


Рисунок 3.2 – Діаграма станів

Процес роботи даного компонента починається з початкової точки, далі йде найперший перехід у стан «Завантаження html коду сторінки». Після завантаження html коду сторінки система завантажує та виконує закладений список перевірок. Важливим станом компонента є стан «Отримання посилань на нові сторінки». Після цього стану система може повернутися до стану «Завантаження html коду сторінки» та почати завантажувати нову вебсторінку або закінчити свою роботу, у випадку якщо нових посилань не знайдено.

Робота компонента продовжується поки на завантажених сторінках виявляються нові посилання на сторінки сайту, які ще не перевірялися.

Таким чином, забезпечуються завантаження та аналіз всіх сторінок вебсайту, що перевіряється системою.

3.3. Моделювання діаграми класів

Одним із найбільш важливих кроків при розробці автоматизованої системи аналізу сайтів було створення UML діаграми класів для бекенд частини проєкту.

Діаграма класів – це основна діаграма, яка використовується для написання коду програми. За допомогою діаграми класів проєктується структура програми, описується унаслідування та зв'язки між класами.

Клас в мові UML використовується для опису об'єктів, які мають однакову структуру, поведінку і зв'язки з об'єктами інших класів. Для графічного зображення класу використовують прямокутник, який може бути розділений на декілька розділів за допомогою горизонтальних ліній. У даних розділах вказується ім'я класу, операції (методи) та атрибути (змінні).

Ім'я класу вказується у першій секції і має бути унікальним в межах пакету діаграм класів або однієї діаграми.

Атрибути класу або властивості вказуються у другій секції класу. Кожному атрибуту класу відповідає окремий рядок, який включає квантора видимості атрибута та інші значення:

<квантор видимості><ім'я атрибута>[кратність]: <тип

атрибута>=<початкове значення>{рядок властивості}

Ім'я атрибута – це рядок тексту, який використовується як ідентифікатор даного атрибута і тому повинен бути унікальним в межах класу.

Методи класу – вказуються у третій секції прямокутника. Кожному методу класу відповідає рядок, який складається з квантору видимості операції та інші значення:

<квантор видимості><ім’я методу>(список параметрів): <вираз типу значення, що повертається >{рядок властивості}

Крім внутрішньої структури класів, на діаграмі вказуються відносини між класами. Основними відносинами в мові UML є:

- залежності;
- асоціації;
- узагальнення.

Для даної автоматизованої системи аналізу сайтів було створено 6 класів та 1 інтерфейс, та було визначено взаємозв’язки між ними.

Основним класом в системі є клас Parser, який завантажує вебсторінки та ініціює їх аналіз. У класі Parser використовуються об’єкти розроблених класів Commands і Page. Клас Commands реалізує завантаження списку перевірок, які будуть виконуватися при аналізі вебсторінок, а клас Page описує сторінки аналізованого вебсайту.

Класи Statistic і PageStatistic реалізують зберігання зібраної під час аналізу статистики. Для додавання нових перевірок у систему аналізу сайтів розроблений інтерфейс IParseRule. Щоб додати нову перевірку, достатньо створити новий клас, який успадкує інтерфейс IParseRule, та реалізувати в ньому перевірку.

Повна UML діаграма класів зображена на рисунку 3.3.

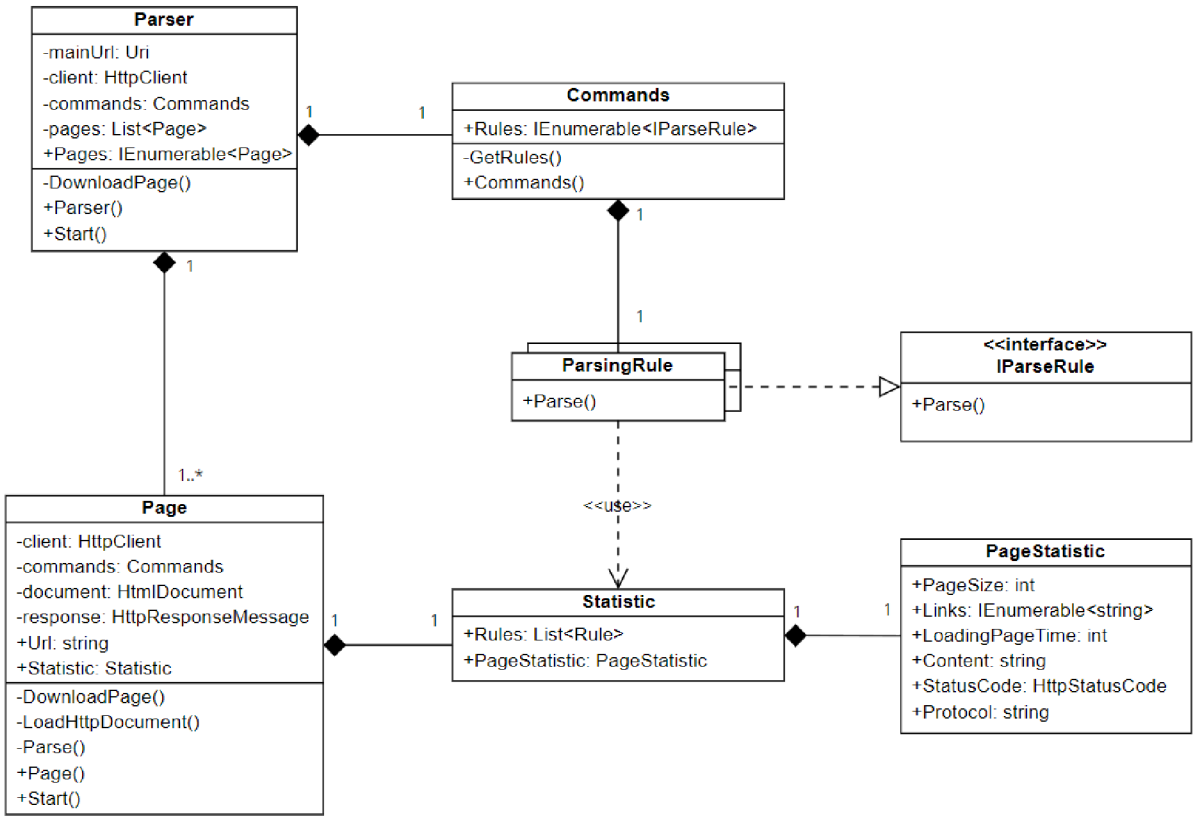


Рисунок 3.3 – Діаграма класів

3.4. Програмна реалізація проєкту

3.4.1. Розробка компонента для завантаження вебсторінок

Основним компонентом інформаційної системи є компонент, який завантажує вебсторінки та ініціює їх аналіз. Для реалізації цього компонента було створено клас Parser (рис. 3.4).

```
public class Parser
{
    private Uri _mainUrl;
    private HttpClient _client;
    private Commands _commands;
    private List<Page> _pages;
    public IEnumerable<Page> Pages => _pages;

    public Parser(Uri mainUrl)
    {…
    }

    public async Task Start()
    {…
    }

    public async Task DownloadPage(string url)
    {…
    }
}
```

Рисунок 3.4 – Клас Parser

Клас Parser включає в себе наступні поля класу:

- _mainUrl – URL-адреса сторінки, яка завантажується;
- _client – об’єкт класу HttpClient, який використовується для завантаження вебсторінок;
- _commands – об’єкт класу Commands, який використовується для завантаження списку перевірок вебсторінки;
- _pages – список об’єктів класу Page, який використовується для збереження списку сторінок вебсайту;
- Pages – оболонка поля _pages, яка доступна з зовні класу Parser (тільки для читання).

Клас Parser описує такі методи:

- Parser() – конструктор класу, в якому відбувається ініціалізація полів класу _mainUrl, _client, _commands, _pages.
- Start() – метод, який запускає завантаження сторінок сайту, використовуючи метод DownloadPage() та передаючи йому адрес першої сторінки.

– DownloadPage() – метод (рис 3.5), який виконує рекурсивне завантаження всіх сторінок вебсайту та ініціює їх аналіз. У методі DownloadPage() створюється об’єкт класу Page та запускається його метод page.Start(), який завантажує сторінку, завантажує HTML документ та запускає аналіз сторінки за списком правил.

Повний код класу Parser приведений в додатку А.

```
public async Task DownloadPage(string url)
{
    Console.WriteLine("Downloading {0}", url);
    var page = new Page(url, _client, _commands);
    _pages.Add(page);
    await page.Start();
    foreach (var link in page.Statistic.PageStatistic.Links)
    {
        if (link.StartsWith("http")) continue;
        if (_pages.Any(p => p.Url == $"{_mainUrl.Scheme}://{_mainUrl.Host}{(_mainUrl.Pc

        await DownloadPage($"{_mainUrl.Scheme}://{_mainUrl.Host}{(_mainUrl.Port != 80 ?
    }
}
```

Рисунок 3.5 – Метод DownloadPage()

У класі Parser використовуються об’єкти розроблених класів Commands і Page.

Клас Commands реалізує завантаження списку перевірок, які будуть виконуватися при аналізі вебсторінок. Завантаження відбувається за допомогою використання бібліотеки Reflection.

Клас Commands включає в себе наступні поля класу:

– Rules – список класів, які реалізують перевірки на вебсторінці та унаслідуються від інтерфейсу IParseRule.

– _namespacePath – шлях до простору імен, за яким шукаються класи, які реалізують перевірки на вебсторінці.

Клас Commands описує такі методи:

– Commands() – конструктор класу, в якому відбувається виконання методу GetRules.

– GetRules() – метод (рис. 3.6), який завантажує список перевірок вебсторінки. За допомогою бібліотеки Reflection завантажуються всі типи проєкту, з яких вибираються типи, простір імен яких починається з _namespacePath.

Повний код класу Commands приведений в додатку Б.

```
private IEnumerable<IParseRule> GetRules()
{
    List<IParseRule> objects = new List<IParseRule>();
    foreach (Type type in
        Assembly.GetExecutingAssembly().GetTypes()
        .Where(t => t.Namespace.StartsWith(_namespacePath))
        .Where(t => t.GetCustomAttribute(typeof(CompilerGeneratedAttribute)) == null))
    {
        objects.Add((IParseRule)Activator.CreateInstance(type, new object[] { }));
    }
    return objects;
}
```

Рисунок 3.6 – Метод GetRules() в класі Commands

Клас Page описує сторінки аналізованого вебсайту. Даний клас включає в себе наступні поля класу:

– Url – URL адрес сторінки.

– Statistic – об’єкт класу Statistic, який зберігає статистику для даної сторінки.

– _client – об’єкт класу HttpClient, який використовується для завантаження сторінки.

– _commands – об’єкт класу Commands, який зберігає список правил для перевірки.

– _document – об’єкт класу HtmlDocument, з бібліотеки HtmlAgilityPack, який використовується для завантаження HTML документа.

– _httpResponseMessage – об’єкт класу HtmlDocument з бібліотеки HtmlAgilityPack, який використовується для завантаження сторінки з вебсайту.

Клас Page описує такі методи:

- Page() – конструктор класу, в якому ініціалізуються поля класу.
- DownloadPage() – метод (рис. 3.7), що з завантажує сторінку з вебсайту за допомогою бібліотеки HtmlAgilityPack. Також в даному методі заміряється швидкість завантаження сторінки та записується в об'єкт Statistic.
- LoadHttpDocument() – метод, що завантажує HTML документ за допомогою бібліотеки HtmlAgilityPack.
- Parse() – метод, який запускає виконання перевірок для даної сторінки.
- Start() – метод, який починає роботу з даною сторінкою. Даний метод запускає методи DownloadPage(), LoadHttpDocument() та Parse().

Повний код класу Page приведений в додатку В.

```
private async Task DownloadPage()
{
    var watch = System.Diagnostics.Stopwatch.StartNew();
    _httpResponseMessage = await _client.GetAsync(Url);
    watch.Stop();
    Statistic.PageStatistic.LoadingPageTime = watch.ElapsedMilliseconds;
}
```

Рисунок 3.7 – Метод DownloadPage() в класі Page

Також були розроблені наступні класи:

- Statistic – клас, який відповідає за збереження статистики.
- PageStatistic – клас, який відповідає за збереження статистики одної конкретної сторінки.

Для зручного додавання нових правил перевірки вебсторінок був розроблений інтерфейс IParseRule, від якого в подальшому наслідуються класи, які описують перевірки сторінки та завантажуються в класі Commands.

3.4.2. Розробка моделі даних

При розробці даної інформаційної системи був використаний Entity Framework та спосіб проєктування бази даних Code first. При використані даного способу необхідно створити класи моделі даних, які будуть зберігатися в базі даних. Після цього Entity Framework на основі створених класів моделі автоматично створює базу даних із відповідними таблицями.

Для зберігання даних про користувачів, проаналізовані сайти, просканові сторінки та виконані перевірки були розроблені класи User, Site, Url та Check (рис 3.8).

Клас User включає в себе наступні поля:

- Id – id користувача системи.
- Name – ім’я користувача системи.
- Sites – список сайтів користувача.

Клас Site включає в себе наступні поля:

- Id – id сайту в системі.
- Name – ім’я сайту.
- URLS – список вебсторінок сайту.

Клас Url включає в себе наступні поля:

- Id – id вебсторінки в системі.
- URL – URL вебсторінки.
- Check – список перевірок для даної вебсторінки.

Клас Check включає в себе наступні поля:

- Id – id перевірки в системі.
- Name – ім’я перевірки.
- Description – опис перевірки.
- CriticalLevel – рівень критичності перевірки.

```
public class User
{
    public int Id { get; set; }
    public string Name { get; set; }
    public ICollection<Site> Sites { get; set; }
}

public class Site
{
    public int Id { get; set; }
    public string Name { get; set; }

    public ICollection<Url> URLS { get; set; }
}

public class Url
{
    public int Id { get; set; }
    public string URL { get; set; }
    public ICollection<Check> Check { get; set; }
}

public class Check
{
    public int Id { get; set; }
    public string Name { get; set; }
    public string Description { get; set; }
    public int CriticalLevel { get; set; }
}
```

Рисунок 3.8 – Класи для створення бази даних

На основі даних класів Entity Framework згенерував базу даних. Діаграма даної бази даних зображена на рисунку 3.9.

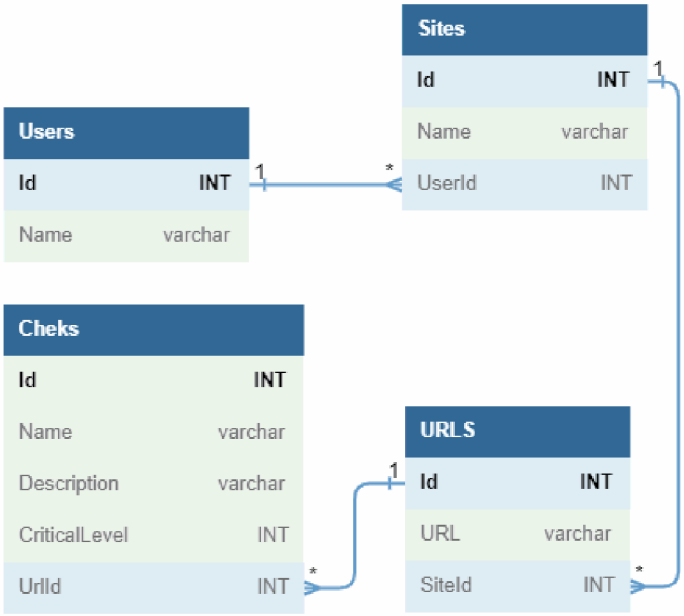


Рисунок 3.9 – Діаграма бази даних

3.4.3. Розробка контролерів та представлень

Для розробки вебінтерфейсу інформаційної системи була використана мова розмітки HTML, каскадні таблиці стилів (CSS) та вбудований в ASP.NET фронтенд фреймворк Bootstrap, який включає готові елементи для розробки вебсторінок.

Розробка вебінтерфейсу виконувалася на основі шаблону проєктування MVC (рис. 3.10). Відносини між компонентами патерну можна описати схемою, зображеною на рисунку .

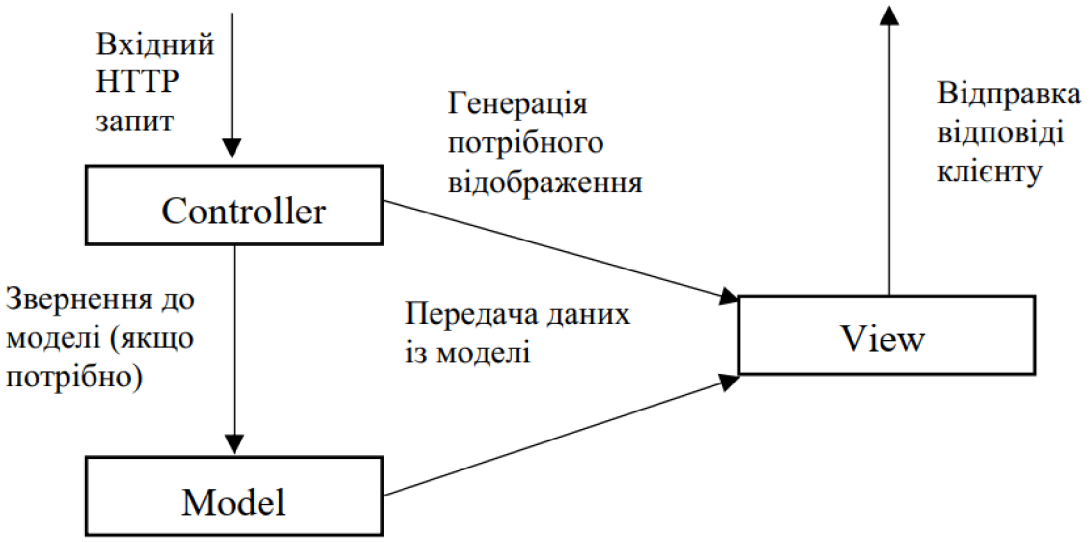


Рисунок 3.10 – Схема взаємодії компонентів MVC

У рамках цього шаблону був розроблений контролер ParserController, який реалізує логіку роботи основної частини вебінтерфейсу інформаційної системи (рис. 3.11). Даний контролер включає такі методи:

- Index() – метод, який реалізує логіку роботи головної сторінки вебінтерфейсу, на якій відображається результат аналізу вебсайту. Також метод Index відповідає за фільтрацію результатів аналізу.
- Search() – метод, який реалізує логіку роботи сторінки пошуку за результатами аналізу сайту.

– Create() – метод, який реалізує логіку роботи сторінки додавання нового сайту. При зверненні користувача за допомогою GET-запиту даний метод генерує та повертає вебсторінку.
– Create(ParserOptions options) – метод, запускає аналіз нового сайту. Даний метод спрацьовує при отриманні даних за допомогою POST запиту.

Весь код контролера ParserController наведений в додатку Г.

```
public class ParserController : Controller
{
    private readonly ApplicationDbContext _context;

    public ParserController(ApplicationDbContext context)
    {…
    }

    public ActionResult Index(string filter, int page = 1)
    {…
    }

    public ActionResult Search(string searchUrl)
    {…
    }

    public ActionResult Create()
    {…
    }

    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<ActionResult> Create(ParserOptions options)
    {…
    }
}
```

Рисунок 3.11 – Контролер ParserController

Для методів контролера ParserController були розроблені представлення Index, Search та Create. Дані представлення відповідають за формування HTML сторінок, які повертаються користувачу при зверненні до відповідних сторінок вебдодатку.

Весь вихідний код розробленої інформаційної системи автоматизованого аналізу сайтів доступний в репозиторії Github

3.5. Практичне використання інформаційної системи

Інформаційна система автоматизованого аналізу сайтів була розроблена у вигляді вебдодатку, працювати з яким можна за допомогою будь-якого браузера та платформи. Розроблена система дозволяє виконувати аналіз сайтів та переглядати його результати. При перегляді результатів користувачу доступний пошук та фільтрація результатів. Для перегляду результатів аналізу великих сайтів передбачена посторінкова навігація.

Робота з системою аналізу сайтів починається з реєстрації користувача та введення адреси сайту, який необхідно проаналізувати. Після цього запускається аналіз сайту. Система рекурсивно завантажує всі сторінки сайту та виконує перевірку на помилки відповідно до закладеного списку перевірок. Після завершення аналізу користувач може переглянути результати.

Користувачу надається загальна інформація про виконанний аналіз, яка включає в себе кількість виявлених помилок, час сканування сайту, кількість просканованих сторінок та середню швидкість завантаження сторінок, а також детальну інформацію щодо кожної сторінки (рис. 3.12).

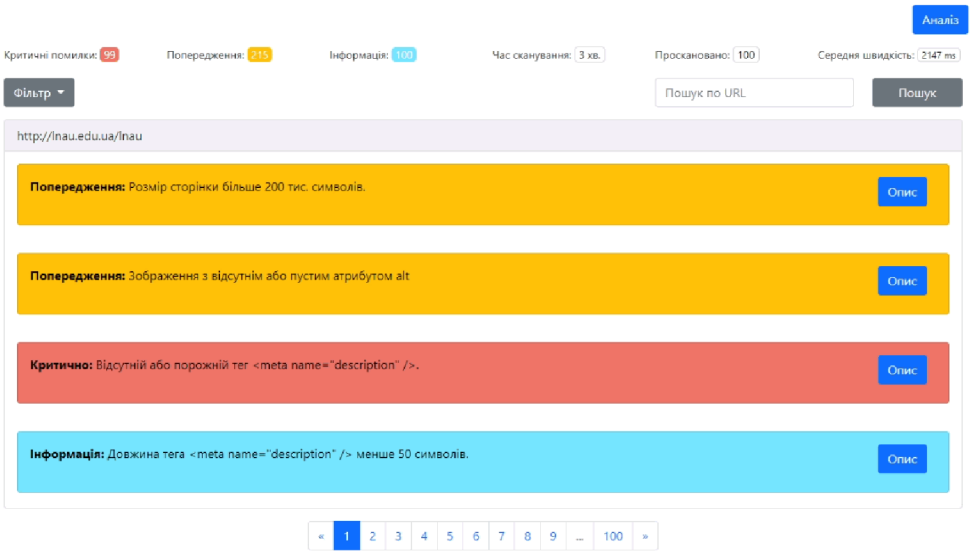


Рисунок 3.12 – Результати аналізу сайту

Працюючи з результатами аналізу, користувач може переглядати виявлені помилки окремо для кожної сторінки сайту. При цьому користувачу надається інформація про всі виявлені помилки на даній сторінці та оцінка їх критичності (критична помилка, попередження, інформація). Також оцінка критичності помилки позначається кольором (червоний, жовтий, синій).

Натиснувши на кнопку «Опис» (рис. 3. 13), користувач може отримати додаткову інформацію про помилку та посилання на документацію пошукових систем.

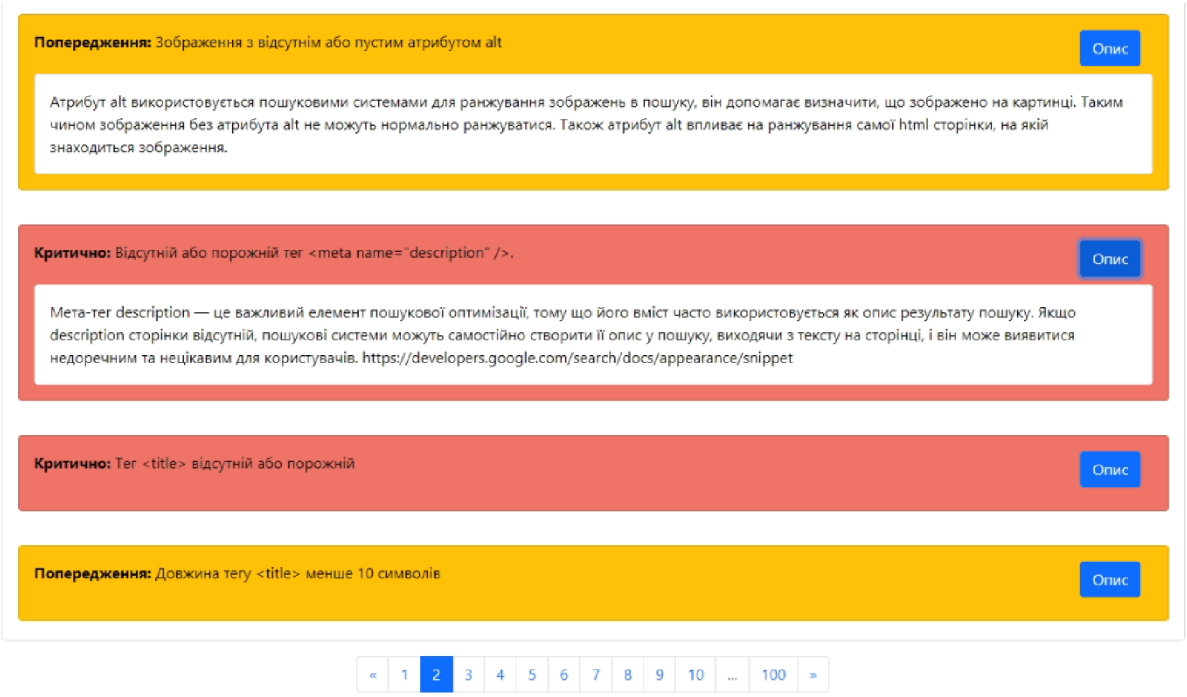


Рисунок 3.13 – Додаткова інформація при натисканні на кнопку «Опис»

При необхідності перевірки певної сторінки на сайті користувач може скористись пошуком. Для цього потрібно ввести в пошук URL сторінки, після чого система виведе результат аналізу даної сторінки (рис. 3.14).

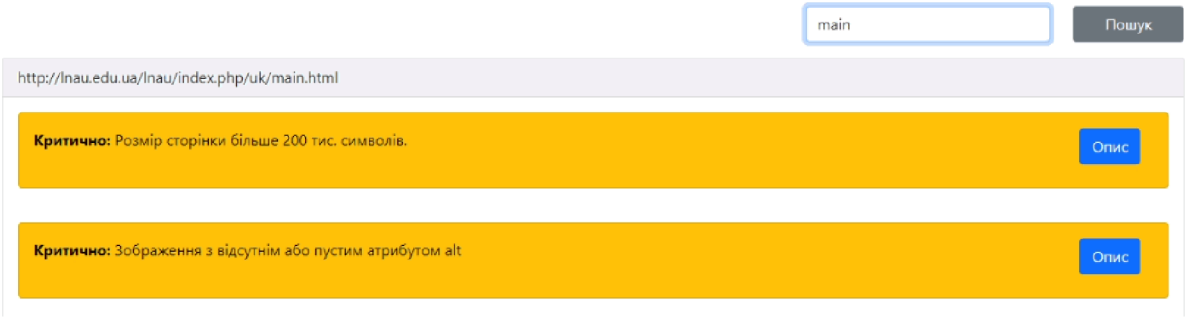


Рисунок 3.14 – Пошук сторінки по URL

Також користувач може фільтрувати виявлені помилки за оцінкою їх критичності (рис 3.15). Наприклад, можна переглянути тільки критичні помилки або попередження.

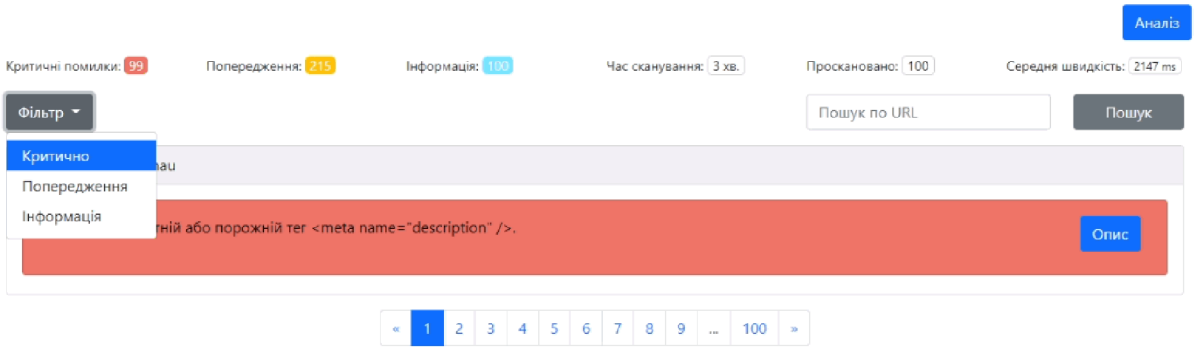


Рисунок 3.15 – Фільтрація результатів аналізу за оцінкою критичності помилки

Висновки до розділу 3

- При виконанні даного дослідження була розроблена інформаційна система автоматичного аналізу сайтів, яка дозволяє аналізувати вебсайти на присутність помилок у пошуковій оптимізації та невідповідність до рекомендацій пошукових систем.
- Перед початком розробки інформаційної системи були розроблені UML діаграми, що дозволило зрозуміти структуру майбутньої системи та уникнути проблем при розробці класів та алгоритмів.
- Використання фреймворку ASP.NET, технології Entity Framework та бібліотеки HtmlAgilityPack дозволило зменшити час розробки, оскільки відпала необхідність в реалізації вже існуючого і добре відомого іструментарію.
- Використання шаблону проєктування MVC дозволило розмежувати основну логіку інформаційної системи, модель даних та представлення.
- При розробці системи була врахована необхідність в подальшій модернізації системи, впровадженні нових функцій та додаванні нових перевірок, які будуть використовуватися при аналізі сайтів.

РОЗДІЛ 4

ОХОРОНА ПРАЦІ

4.1. Структурно-функціональний аналіз технологічного процесу

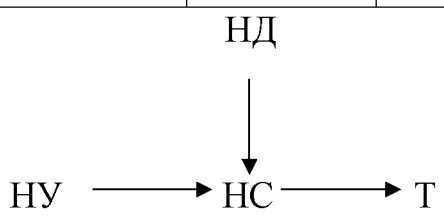
Розробка та вживання ефективних заходів запобігання аварійним і травмонебезпечним ситуаціям можливе лише при завчасному виявленні тих небезпек, з яких починаються процеси їх формування. Оскільки небезпечні умови не завжди завчасно можна виявити, а для вивчення небезпечних дій іноді потрібно багато часу, щоб зібрати статичний матеріал, то й методи виявлення цих небезпек повинні бути відповідно диференційовані (табл. 4.1).

Відповідно до аналізу небезпечних умов, які існують у виробничому процесі, виокремлено такі наступні за характером небезпеки:

- характеризують стан або рівень небезпеки обладнання, яке використовується;
- сприяють виникненню технологічних помилок обслуговуючого персоналу під час виробничого процесу;
- створення умов та можливість проникнення працівника в небезпечну зону;
- приводять до виникнення небезпечних дій (внаслідок низького рівня професійної підготовки працівників та організації навчання з охорони праці).

Схеми формування та виникнення травмонебезпечних й аварійних ситуацій у комп'ютерному кабінеті представлено у вигляді моделі формування та виникнення травмонебезпечних і аварійних ситуацій – табл. 4.1.

Таблиця 4.1. Схеми формування та виникнення травмонебезпечних і аварійних ситуацій

Види робіт, виробничий підрозділ, робоче місце, виробниче обладнання, склад агрегату	Виробнича небезпека			Можливі наслідки	Заходи запобігання небезпечним ситуаціям
	небезпечна умова (НУ)	небезпечна дія (НД)	небезпечна ситуація (НС)		
Виконання робіт із використанням електро-обладнання	Нехтування правилами ТБ	Не вимкнено живлення Відсутність заземлення	Ураження струмом	Травма (Т)	Проведення повторного інструктажу з ТБ. Розробка нових способів захисту. Встановлення заземлення
					

За умови допускання небезпечних дій в наявності небезпечних умов може утворитися небезпечна ситуація. Результатом даної небезпечної ситуації може бути відсутність наслідків, аварія, травма.

4.2. Розрахунок освітлення приміщення комп'ютерного кабінету

Освітленість виробничих приміщень може бути штучною і природною. Природне освітлення при правильному обладнанні найбільш сприятливе для людини. Основні вимоги для освітлення наступні:

- освітлення повинне бути достатнім для швидкого і легкого розпізнання об'єктів роботи;
- освітлення повинно бути рівномірне без різких тіней;
- джерело світла не повинно осліплювати працівника;
- рівень освітленості не повинен змінюватися з часом.

Перевірка природного освітлення через бокові вікна за нормами освітленості ведеться для найбільш віддаленої від вікон точки, тобто знаходять мінімальне значення коефіцієнта природної освітленості (процентне відношення фактичної освітленості Fв у будь-якій точці приміщення до освітленості Fн на відкритій місцевості):

$$e_{\min} = \frac{Fb}{Fн} \cdot 100. \qquad (4.1)$$

Значення коефіцієнта природної освітленості визначається не менш, ніж у п'яти точках (для сільськогосподарських виробничих приміщень беремо $e_{min} = 0{,}5\%$).

Розрахунок природного освітлення зводиться до визначення площі світлових променів.

Сумарну площу світлових променів $\sum F_o (м^2)$ по коефіцієнту природної освітленості для бокових променів визначаємо за формулою:

$$\sum F_o = \frac{F_н \cdot e_{\min} \cdot r_o \cdot K}{100 \cdot \tau \cdot Г_1}, \qquad (4.2)$$

де $F_н$ – площа підлоги, м^2; e_{min} – величина мінімального коефіцієнта природного освітленості; τ – загальний коефіцієнт світло пропускання віконного отвору з врахуванням його забруднення, $\tau = 0{,}25$; r_o – світлова

характеристика вікна, r_o – 9,5; $Г_1$ – коефіцієнт, який враховує підвищення освітленості за рахунок світла, яке відбивається від стін і стелі, $Г_1$ = 1,2; $К$ – коефіцієнт, який враховує затінення вікон сусідніми приміщеннями і загорожею, $К = 1$.

$$\sum F_o = \frac{36 \cdot 0,5 \cdot 9,5 \cdot 1}{100 \cdot 0,25 \cdot 1,2} = 5,7 м^2 .$$

Кількість світлових потоків визначимо:

$$N = \frac{\sum F_o}{F_o}, \qquad (4.3)$$

де F_o – площа вікна згідно зі стандартом, м^2.

$$N = \frac{5,7}{6} = 0,95.$$

Приймаємо кількість вікон – одне вікно.

При розрахунку штучного освітлення найбільш поширеним і простим є метод світлового потоку. При цьому методі розраховуємо світловий потік $F_л$ (Лк), який повинен випромінювати кожна лампа (при заданій кількості ламп).

$$F_л = \frac{k \cdot S_п \cdot E}{n_л \cdot \eta \cdot r^2}, \qquad (4.4)$$

де k– коефіцієнт запасу, $k = 1,3$; $S_п$ – площа підлоги, м^2; $S_п = 36$м^2. E – нормативна освітленість, Е = 300 Лк; $n_л$ – кількість встановлених ламп, $n_л$=6 од; η– коефіцієнт використання світлового потоку, $\eta = 0,25$; r– коефіцієнт нерівномірності освітленості, $r = 0,545$.

Коефіцієнт запасу (K) враховує можливість забруднення світильників пилом, що залежить від характеру виробництва.

Розрахунок штучного освітлення починаємо з визначення висоти розташування світильника і їх кількості. Висоту $h_н$ (м) розташування світильників над робочим місцем знаходимо за формулою:

$$h_{\text{н}} = H - (h_1 + h_2), \quad (4.5)$$

де H – висота приміщення, м; h_1 – віддаль від підлоги до освітлювальної поверхні, м; h_2 – віддаль від стелі до світильника, м.

$h_{\text{н}} = 4{,}5 - (2{,}2 + 1{,}5) = 0{,}8$м.

При симетричному розміщенні світильників по вершинах квадратів їх кількість визначається за формулою:

$$n_c = \frac{S_n}{l^2}, \quad (4.6)$$

де l – віддаль між світильниками, м.

Підставивши значення, отримаємо:

$n_c = \frac{36}{9} = 4$од.

Тоді світловий потік буде становити

$F_{\text{л}} = \frac{1{,}3 \cdot 36 \cdot 300}{4 \cdot 0{,}25 \cdot 0{,}545} = 2576{,}2$Лк.

При світловому потоці 2576,2 Лк для заданої лампи вибираємо тип і потужність.

Вибираємо тип лампи – люмінесцентну, потужністю 40Вт.

4.3. Безпека в надзвичайних ситуаціях

Надзвичайні ситуації виникають з багатьох причин – це природні катаклізми, техногенні катастрофи та аварії, соціально-політичні конфлікти. Останнім часом в Україні виникають надзвичайні ситуації, причиною яких є бойові дії, терористичні акти та диверсії.

При загрозі або виникненні надзвичайної ситуації будь-якого походження державні органи виконавчої влади зобов’язані проінформувати населення через систему оповіщення. Головним способом оповіщення про небезпеку та порядок дій населення є передача повідомлень через мережі місцевого радіомовлення та телебачення.

Перед передачею повідомлення включається протяжне звучання сирен. Після звучання сирен, через мережі місцевого радіомовлення та телебачення, передається сигнал цивільного захисту «Увага всім! Увага всім! Увага всім!», а далі повідомлення про проведення технічної перевірки системи оповіщення або інформацію про загрозу або виникнення надзвичайної ситуації та порядок дій населення. Почувши звучання сирени, необхідно:

- увімкнути радіоприймач або телевізор на місцевий канал;
- уважно прослухати повідомлення, та діяти згідно з отриманим повідомленням.

При отриманні повідомлення про загрозу хімічного ураження необхідно повідомити всіх про небезпеку, укритися в приміщенні та провести максимальну його герметизацію: закрити вікна, двері, вентиляційні канали, щілини заклеїти підручними матеріалами, наприклад клейкою стрічкою.

При отриманні повідомлення про пожежу або замінування необхідно повідомити всіх про небезпеку, терміново вимкнути електропостачання та покинути приміщення.

При отриманні сигналу «Повітряна тривога!» необхідно терміново повідомити всіх про небезпеку, вимкнути електрику та прямувати в укриття.

У випадку, якщо хтось з оточуючих отримав травму та потребує допомоги, в першу чергу необхідно викликати швидку допомогу за телефоном 103. Після цього необхідно оглянути місце пригоди та перевірити чи не загрожує небезпека вам і оточуючим. Непотрібного ризику слід уникати. Необхідно усунути вплив на організм потерпілого факторів, що загрожують його здоров'ю та життю, це дозволить безпечно для себе надавати потерпілому допомогу. Після цього можна провести огляд потерпілого та надати йому долікарську допомогу.

Висновки до розділу 4

- Розробка та вживання ефективних заходів запобігання аварійним і травмонебезпечним ситуаціям вимагає завчасного виявлення небезпек, з яких починаються процеси формування аварійних ситуацій.
- Природне освітлення найбільш сприятливе для людини. При використанні штучного освітлення необхідно провести розрахунок штучного освітлення та визначити яскравість та потужність необхідних ламп.
- Про виникнення надзвичайної ситуації повідомляється за допомогою сирени та повідомлення місцевого радіо- та телебачення. Почувши звучання сирени, необхідно увімкнути радіоприймач або телевізор, налаштувати на місцевий канал та діяти згідно з отриманим повідомленням.
- При отриманні сигналу «Повітряна тривога!» необхідно повідомити всіх про небезпеку та прямувати в укриття.

РОЗДІЛ 5

ВИЗНАЧЕННЯ ЕФЕКТИВНОСТІ ІНФОРМАЦІЙНОЇ СИСТЕМИ

Розроблена інформаційна система автоматизованого аналізу сайтів виконує пошук помилок в пошуковій оптимізації вебсайтів та складає звіт для користувача, в якому приводиться загальна інформація про аналіз вебсайту. Зокрема звіт включає інформацію про кількість проаналізованих сторінок, час сканування вебсайту, критичність виявлених помилок та середній час завантаження сторінок. Також користувачу надається інформація про кожну з завантажених сторінок вебсайту. Тут користувачу надається список помилок, виявлених на сторінці, вказується їх критичність та надається детальний опис помилки з посиланням на рекомендації пошукових систем.

Враховуючи особливості системи для дослідження ефективності інформаційної системи, було згенеровано статичний HTML-сайт, який складається з 100 сторінок. На кожній сторінці даного сайту була закладена 1 помилка. Приклад вихідного коду одної з таких сторінок наведено на рисунку 5.1.

Була проведена перевірка швидкості сканування та коректності одержаних даних при скануванні вказаних HTML-сайтів. Під час сканування тестового сайту було успішно просканово усі 100 сторінок на виявлено усі закладені помилки. Результати тестування приведені в таблиці 5.1. Час витрачений на сканування сайту – 17 секунд.

Таблиця 5.1 – Результати тестування інформаційної системи аналізу сайтів із використанням підготовлених HTML-сайтів з помилками.

Опис	Кількість
Просканові сторінки	100
Виявлені помилки	100
Критичні помилки	25
Попередження	60
Інформація	15

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>title title title title title title title title title </title>
    <meta name="robots" content="nofollow">
    <link rel="alternate" hreflang="ua-UA" href="htpps://ua.site.com" />
    <link rel="next" href="http://www.example.com/article?story=abc&page=3" />
    <link rel="prev" href="http://www.example.com/article?story=abc&page=1" />
    <link rel="canonical" href="https://example.com/dresses/green-dresses" />
    <meta name="robots" content="noindex, nofollow, max-snippet:-1,
                        max-image-preview:large, max-video-preview:-1" />
    <meta name="description" content="description description description
                                description description description
                                description description description
                                description description description ">
</head>
<body>
</body>
</html>
```

Рисунок 5.1 – Приклад вихідного HTML-коду сторінки з відсутнім тегом H1

Для дослідження ефективності інформаційної системи в реальних умовах було виконано аналіз вебсайту університету. Для отримання співставних даних кількість сторінок для сканування було обмежено на позначці 100. Після досягнення цієї кількості проаналізованих сторінок система зупиняла свою роботу. В результаті просканово 100 сторінок, на яких було виявлено 414 помилок, серед яких 99 критичні. Результати даного тестування приведені в таблиці 5.2.

Таблиця 5.2 – Результати тестування інформаційної системи аналізу сайтів при перевірці сайту університету.

Опис	Кількість
Просканові сторінки	100
Виявлені помилки	414
Критичні помилки	99
Попередження	215
Інформація	100

При використанні у реальних умовах усі перевірки, закладені в інформаційну систему, спрацювали нормально. Шляхом вибіркової ручної перевірки сторінок було визначено, що всі наявні на вебсайті помилки були зафіксовані коректно. Зокрема, серед виявлених помилок є такі:

- Сторінки з відсутнім тегом Title.
- Сторінки з відсутнім мета-тегом Description.
- Зображення без атрибуту Alt.
- Великий розмір сторінки (понад 200 тис. символів).
- Повільне завантаження основного HTML коду сторінки (більше 2 секунд).

Варто відзначити час сканування вебсайту, який при використанні в реальних умовах зріс з 17 секунд до 3 хвилин. Це пов’язано зі швидкістю завантаження сторінок, яка в середньому перевищила 2 секунди.

Таким чином, проведене дослідження ефективності інформаційної системи аналізу сайтів виявило, що в контрольованих умовах система виявляє 100% наявних помилок, перевірка яких була передбачена. При цьому, швидкість аналізу сайтів в більшій мірі залежить від швидкості завантаження вебсторінок.

Висновки до розділу 5

- У контрольованих умовах система виявляє 100% наявних помилок, перевірка яких була передбачена. При скануванні згенерованого сайту було просканово всі 100 сторінок та виявлено всі закладені помилки. При скануванні реальних сайтів система також демонструє стабільну роботу. Виявлені помилки підтверджуються при ручній перевірці вебсторінок.
- Швидкість сканування й аналізу в більшій мірі залежить від швидкості завантаження сторінок вебсайту, який аналізується. При скануванні статичного HTML-сайту швидкість сканування становить 0,17 секунд на 1 сторінку.
- Система здатна виявляти критичні помилки в пошуковій оптимізації сайтів, наприклад, такі, як відсутність тега Title чи мета-тега Description. Також система надає інформацію про середню швидкість завантаження сторінок, яка теж є важливим показником для спеціалістів із пошукової оптимізації.

ВИСНОВКИ

Результатом дипломної роботи є створена інформаційна система автоматизованого аналізу сайтів на основі ASP.NET.

На базі проведеного аналізу існуючих програм для автоматизованого аналізу сайтів було визначено, що існує потреба в створенні сучасної системи аналізу, яка б надавала користувачу максимум інформації про виявлені помилки, опираючись на рекомендації пошукових систем.

Для реалізації даної системи було обрано формат вебдодатку і фреймворк ASP.NET. Для написання програми було обрано мову програмування C# і базу даних на основі Entity Framework. Також було використано бібліотеку HtmlAgilityPack. Обрані інструменти розробки відповідають вимогам системи та можуть забезпечити високу швидкість її розробки.

Після визначення проблеми та вибору засобів реалізації за допомогою UML діаграм було розроблено структуру програми та визначено процес розробки продукту.

Система автоматизованого аналізу сайтів була розроблена з використанням об’єктно-орієнтованого підходу, що дозволяє легко підтримувати роботу системи та розширювати набір функцій та перевірок, які виконуються при аналізі вебсайту. Розроблена система дозволяє аналізувати вебсайти та виявляти помилки в їх пошуковій оптимізації. Після аналізу вебсайту користувачу системи надається інформація про сайт та кожну його сторінку, повідомляється кількість знайдених помилок, їх критичність та детальний опис. Під час роботи з результатами аналізу є можливість користуватися пошуком та фільтрацією. Доступ до інформаційної системи можливий як з настільного комп’ютера, так і з мобільних пристроїв.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What 3.25 Billion Website Visits Tell Us About the Internet's Top Traffic Sources. URL: https://growthbadger.com/traffic-study/ (дата звернення: 20.10.2022).
2. Search Engine Market Share Worldwide. URL: https://gs.statcounter.com/search-engine-market-share/all/ (дата звернення: 20.10.2022).
3. Search Engine Market Share Ukraine. URL: https://gs.statcounter.com/search-engine-market-share/all/ukraine/#monthly-202105-202205-bar (дата звернення: 20.10.2022).
4. Conversion Rate by Traffic Source. URL: https://firstpagesage.com/seo-blog/conversion-rate-by-traffic-source/ (дата звернення: 20.10.2022).
5. Google Search Essentials. URL: https://developers.google.com/search/docs/essentials?hl=en (дата звернення: 20.10.2022).
6. The Most Common Technical SEO Issues That Damage Your Site [Infographic]. URL: https://serpstat.com/blog/infographic-the-most-common-seo-errors/ (дата звернення: 20.10.2022).
7. SiteChecker Research: the Most Common SEO Errors Websites Make in 2021. URL: https://sitechecker.pro/the-most-common-seo-mistakes-you-need-to-avoid-in-2021/ (дата звернення: 20.10.2022).
8. SEO META in 1 CLICK. URL: https://chrome.google.com/webstore/detail/seo-meta-in-1-click/bjogjfinolnhfhkbipphpdlldadpnmhc (дата звернення: 20.10.2022).
9. Screaming Frog SEO Spider. URL: https://www.screamingfrog.co.uk/seo-spider/ (дата звернення: 20.10.2022).
10. Netpeak Spider: The desktop tool for everyday SEO audits. URL: https://netpeaksoftware.com/spider (дата звернення: 20.10.2022)

11. Serpstat – Growth Hacking Tool for SEO, PPC and Content Marketing. URL: https://serpstat.com/ (дата звернення: 20.10.2022).
12. Website SEO Checker for Fast & Complete Technical Audit. URL: https://sitechecker.pro/ (дата звернення: 20.10.2022).
13. Репозиторій GitHub. URL: https://github.com/GreengrayZ/WebAudit (дата звернення: 20.10.2022).
14. C# reference. URL: https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/ (дата звернення: 15.11.2022).
15. .NET documentation. URL: https://docs.microsoft.com/uk-ua/dotnet/ (дата звернення: 15.11.2022).
16. C# Coding Conventions. 2015. URL: https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/coding-style/coding-conventions (дата звернення: 15.11.2022).
17. Entity Framework documentation. URL: https://docs.microsoft.com/en-us/ef/ (дата звернення: 15.11.2022).
18. Get started with Bootstrap. URL: https://getbootstrap.com/docs/5.2/getting-started/introduction/ (дата звернення: 15.11.2022).
19. Ілляшенко Н. С., Савченко О. С. SEO-оптимізація як сучасний інструмент інтернет-маркетингу. *Маркетинг і менеджмент інновацій.* 2012. № 3. С. 63–74. URL: https://essuir.sumdu.edu.ua/bitstream/123456789/29141/1/Internet-marketing.pdf. (дата звернення: 15.05.2022).
20. ASP.NET MVC Overview. URL: https://msdn.microsoft.com/en-us/library/dd381412(v=vs.108).aspx (дата звернення: 31.05.2018).
21. Grappone J. Search Engine Optimization. An Hour a Day. Third edition. Wiley Publishing, Inc., 2011. 411 p.
22. Fleischner M. H. SEO made Simple. Strategies for Dominating the World's Largest Search Engine. Second edition. USA, 2011. 128 p.

23. Басюк Т. М. Принципи побудови системи аналізу та просування інтернет ресурсів / Т. М. Басюк. *Комп'ютерні науки та інформаційні технології.* Львів, 2012. № 784. С. 43–48.

24. Басюк Т. М. Проектування системи автоматизації процесу seo-оптимізації / Т. М. Басюк. *Комп'ютерні науки та інформаційні технології.* Львів, 2014. № 800. С. 92–97. URL: http://irbis-nbuv.gov.ua/cgi-bin/irbis_nbuv/cgiirbis_64.exe?C21COM=2&I21DBN=UJRN&P21DBN=UJRN&IMAGE_FILE_DOWNLOAD=1&Image_file_name=PDF/VNULPKNIT_2014_800_16.pdf. (дата звернення: 15.05.2018).

25. Басюк Т. М. Ранжування веб-сайтів в мережі інтернет / Т. М. Басюк, А. С. Василюк. *Інформаційні системи та мережі.* Львів, 2013. № 770. С. 3–12. URL: http://science.lp.edu.ua/sites/default/files/Papers/3_2_0.pdf. (дата звернення: 15.05.2022).

26. Терещенко В. В. Аналіз сучасних методик пошукової оптимізації (SEO) / В. В. Терещенко. *Вісник Кременчуцького національного університету імені Михайла Остроградського.* 2015. Вип. 6 (1). С. 48–54. URL: http://www.kdu.edu.ua/PUBL/statti/2015_6_48_6_2015.pdf. (дата звернення: 15.05.2022).

27. Brin S., Page L. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*. Stanford, 2004. P. 107–117.

28. Ganz A., Sieh L. Behavioral factors and SEO. *Proceedings of 24th International Conference on Computer Communications and Networks*. Las Vegas, 2015. P. 218–223.

29. Kostenko P. P., Levchenko I. V. Webservice for clarification relevant web-documents of search results of Google based on user behavior. *Inzhenerni ta osvitni tehnologii*. 2014. No 4 (8). P. 49–62.

30. Get started with ASP.NET Core. URL: https://learn.microsoft.com/en-us/aspnet/core/getting-started/ (дата звернення: 15.05.2022).
31. Overview of ASP.NET Core. URL: https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core (дата звернення: 15.05.2022).

ДОДАТОК А

Код класу Parser.

```
using Audit.Core.Pages;
using Audit.Core.Rules;

namespace Audit.Core
{
    public class Parser
    {
        private Uri _mainUrl;
        private HttpClient _client;
        private Commands _commands;
        private List<Page> _pages;
        public IEnumerable<Page> Pages => _pages;

        public Parser(Uri mainUrl)
        {
            _mainUrl = mainUrl;
            _client = new HttpClient();
            _commands = new Commands();
            _pages = new List<Page>();
        }

        public async Task Start()
        {
            await
DownloadPage($"{_mainUrl.Scheme}://{_mainUrl.Host}{(_mainUrl.Port != 80 ?
$":{_mainUrl.Port}" : "")}{_mainUrl.AbsolutePath}");
        }

        public async Task DownloadPage(string url)
        {
            Console.WriteLine("Downloading {0}", url);
            var page = new Page(url, _client, _commands);
            _pages.Add(page);
            await page.Start();
            foreach (var link in page.Statistic.PageStatistic.Links)
            {
                if (link.StartsWith("http")) continue;
                if (_pages.Any(p => p.Url ==
$"{_mainUrl.Scheme}://{_mainUrl.Host}{(_mainUrl.Port != 80 ?
$":{_mainUrl.Port}/" : "")}{link}")) continue;

                await
DownloadPage($"{_mainUrl.Scheme}://{_mainUrl.Host}{(_mainUrl.Port != 80 ?
$":{_mainUrl.Port}/" : "")}{link}");
            }
        }
    }
```

ДОДАТОК Б

Код класу Commands.

```
using System.Reflection;
using System.Runtime.CompilerServices;

namespace Audit.Core.Rules
{
    public class Commands
    {
        public IEnumerable<IParseRule> Rules { get; set; }

        private readonly string _namespacePath =
"Audit.Core.Rules.Command";

        public Commands()
        {
            Rules = GetRules();
        }

        private IEnumerable<IParseRule> GetRules()
        {
            List<IParseRule> objects = new List<IParseRule>();
            foreach (Type type in
                Assembly.GetExecutingAssembly().GetTypes()
                .Where(t => t.Namespace.StartsWith(_namespacePath))
                .Where(t =>
t.GetCustomAttribute(typeof(CompilerGeneratedAttribute)) == null))
            {

objects.Add((IParseRule)Activator.CreateInstance(type, new object[] { }));
            }
            return objects;
        }
    }
}
```

ДОДАТОК В

Код класу Page.

```
using HtmlAgilityPack;
using Audit.Core.Rules;
using Audit.Core.Statistics;

namespace Audit.Core.Pages
{
    public class Page
    {
        public string Url { get; }
        public Statistic Statistic { get; }

        HttpClient _client;
        Commands _commands;
        HtmlDocument _document;
        HttpResponseMessage _httpResponseMessage;

        public Page(string url, HttpClient client, Commands commands)
        {
            Url = url;
            Statistic = new Statistic();
            _client = client;
            _commands = commands;
            _document = new HtmlDocument();
        }

        private async Task DownloadPage()
        {
            var watch = System.Diagnostics.Stopwatch.StartNew();
            _httpResponseMessage = await _client.GetAsync(Url);
            watch.Stop();
            Statistic.PageStatistic.LoadingPageTime =
watch.ElapsedMilliseconds;
        }

        private void LoadHttpDocument()
        {

_document.Load(_httpResponseMessage.Content.ReadAsStream());
        }

        private void Parse()
        {
            foreach (var rule in _commands.Rules)
                rule.Parse(Statistic, _httpResponseMessage,
_document);
        }

        public async Task Start()
        {
            await DownloadPage();
            LoadHttpDocument();
            Parse();
        }    }    }
```

ДОДАТОК Г

Код контролера ParserController.

```
using Audit.Core;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Web.Data;
using Web.Models;

namespace Web.Controllers
{
    public class ParserController : Controller
    {
        public static Test Test = new Test() {
            URLS = new List<string> { },
            pages = new Dictionary<string,
IEnumerable<Audit.Core.Pages.Page>>()
        };

        private readonly ApplicationDbContext _context;

        public ParserController(ApplicationDbContext context)
        {
            _context = context;
        }

        private List<Audit.Core.Pages.Page>
Filter(Audit.Core.Rules.Error errorLevel)
        {
            var pages = new List<Audit.Core.Pages.Page>();
            foreach (var item in Test.pages[User.Identity.Name])
                if (item.Statistic.Rules.Where(r => r.ErrorLevel ==
errorLevel).Count() > 0)
                    pages.Add(new Audit.Core.Pages.Page(item.Url,
null, null));

            foreach (var item in Test.pages[User.Identity.Name])
                item.Statistic.Rules.Where(r => r.ErrorLevel ==
errorLevel).ToList().ForEach(rule => pages.Find(p => p.Url ==
item.Url).Statistic.Rules.Add((Audit.Core.Rules.Rule)rule.Clone()));

            return pages;
        }


        public ActionResult Index(string filter, int page = 1)
        {
            var user = _context.Users.Where(u => u.Name ==
User.Identity.Name).Include(u => u.Sites).ThenInclude(s =>
s.URLS).ThenInclude(u => u.Check).FirstOrDefault();
            if (user == null) return View(null);
            if (user.Sites == null) return View(null);
            if (user.Sites.Count == 0) return View(null);

            var site = user.Sites.ToList()[0];
            var url = site.URLS.ToList()[page - 1];
```

```
                if(url == null) return View(null);
                var p = new Audit.Core.Pages.Page(url.URL, null, null);

                p.Statistic.Rules = new List<Audit.Core.Rules.Rule>();
                foreach (var check in url.Check)
                {
                    p.Statistic.Rules.Add(new Audit.Core.Rules.Rule
                    {
                        Name = check.Name,
                        Description = check.Description,
                        ErrorLevel =
(Audit.Core.Rules.Error)check.CriticalLevel
                    });
                }
                if (filter != "" && filter != null)
                {
                    Audit.Core.Rules.Error filterError =
(Audit.Core.Rules.Error)Enum.Parse(typeof(Audit.Core.Rules.Error), filter,
true);
                    p.Statistic.Rules = p.Statistic.Rules.Where(r =>
r.ErrorLevel == filterError).ToList();
                }
                ViewBag.page = page;
                ViewBag.totalPages = site.URLS.Count;
                ViewBag.filter = filter;


                return View(p);
            }

            public ActionResult Search(string searchUrl)
            {
                var user = _context.Users.Where(u => u.Name ==
User.Identity.Name).Include(u => u.Sites).ThenInclude(s =>
s.URLS).ThenInclude(u => u.Check).FirstOrDefault();
                if (user == null) return View();
                if (user.Sites == null) return View();
                if (user.Sites.Count == 0) return View();

                var url = user.Sites.ToList()[0].URLS.Where(url =>
url.URL.Contains(searchUrl)).FirstOrDefault();
                if (url == null) return View();

                var page = new Audit.Core.Pages.Page(url.URL, null,
null);
                page.Statistic.Rules = new List<Audit.Core.Rules.Rule>();
                foreach(var check in url.Check)
                {
                    page.Statistic.Rules.Add(new Audit.Core.Rules.Rule()
                    {
                        Name = check.Name,
                        Description = check.Description,
                        ErrorLevel =
(Audit.Core.Rules.Error)check.CriticalLevel
                    });
                }

                return View(page);
            }

            public ActionResult Create()
```

```
            {
                var name = User.Identity?.Name;
                if (name != null)
                {
                    var user = _context.Users.Where(p => p.Name ==
name).FirstOrDefault();

                    if (user == null)
                    {
                        user = new User()
                        {
                            Name = name
                        };
                        _context.Users.Add(user);
                        _context.SaveChanges();
                        Console.WriteLine("Created a new user");
                    }
                }

                return View();
            }

            [HttpPost]
            [ValidateAntiForgeryToken]
            public async Task<ActionResult> Create(ParserOptions options)
            {
                var name = User.Identity?.Name;
                if (name == null)
                    return RedirectToAction("Index");

                var user = _context.Users.Where(p => p.Name ==
name).Include(p => p.Sites).FirstOrDefault();


                if (options.URL == null)
                    ModelState.AddModelError("EmptyError", "Field cannot
be empty.");
                else if (!options.URL.StartsWith("http"))
                    ModelState.AddModelError("HttpError", "Has to be a
URL.");

                if(!ModelState.IsValid) return View(options);

                //_ = Task.Run(async () =>
                {
                    var parser = new Parser(new Uri(options.URL));
                    await parser.Start();

                    var sites = new List<Site>();
                    sites.Add(new Site() { Name =
parser.Pages.First().Url });
                    sites[0].URLS = new List<Url>();

                    foreach (var page in parser.Pages)
                    {
                        sites[0].URLS.Add(new Models.Url
                        {
                            URL = page.Url,
                            Check = new List<Check>()
                        });
```

```
                            var url = sites[0].URLS.ToList().Where(url =>
url.URL == page.Url).First();
                            foreach (var rule in page.Statistic.Rules)
                            {
                                url.Check.Add(new Check
                                {
                                    Name = rule.Name,
                                    Description = rule.Description,
                                    CriticalLevel = (int)rule.ErrorLevel
                                });
                            }
                        }

                        try
                        {
                            Console.WriteLine(user.Name);
                            user.Sites = sites;

                            Console.WriteLine("update");
                            _context.Update(user);
                            Console.WriteLine("save");
                            _context.SaveChanges();
                            Console.WriteLine("saved");
                        }
                        catch (Exception e)
                        {
                            Console.WriteLine(e.Message);
                            throw;
                        }

                    }
                    return RedirectToAction("Index");
                }
            }
        }
```