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

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти магістра

зі спеціальності 121 – Інженерія програмного забезпечення

На тему: Автоматизована система аналізу SEO для веб-сайтів

Засвідчую, що в цій кваліфікаційній роботі немає запозичень із праць інших авторів без відповідних посилань.
Студент гр. ІПЗ-24м
_____ / С. А. Бондарчук /

Керівник
кваліфікаційної роботи _____ / А. А. Трачук /

Економіко-організаційна
частина _____ / _____ /

Нормоконтроль _____ / _____ /

Завідувач кафедри _____ / А. М. Стрюк /

Кривий Ріг
2025

Криворізький національний університет

Факультет: Інформаційних технологій

Кафедра: Моделювання та програмного забезпечення

Ступінь вищої освіти: магістр

Спеціальність: 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедри
_____ А. М. Стрюк
«__» _____ 2025 р.

ЗАВДАННЯ

на кваліфікаційну роботу

студенту групи ІПЗ-24м Бондарчуку Сергію Анатолійовичу

1. Тема: Автоматизована система аналізу SEO для веб-сайті затверджено наказом по КНУ № від «15» квітня 2025 р.
2. Термін подання студентом закінченої роботи: «01» Грудня 2025р.
3. Вихідні дані по роботі: розроблювана система повинна бути автономною, забезпечувати комплексний аналіз понад 20 SEO-параметрів, візуалізацією результатів та зберіганням історії в локальній БД; мінімальна точність аналізу – 90%.
4. Зміст пояснювальної записки: аналіз актуальності теми та предметної області SEO; аналіз інструментів; постановка проблеми та необхідність автоматизації; визначення мети, об’єкта, предмета, завдань; вибір технологій; методи реалізації аналізу; UML-діаграми; алгоритми системи; модель даних; реалізація ПЗ; тестування; висновок та посилання.
5. Перелік ілюстративного матеріалу: рисунки інструментів діаграма класів; діаграма послідовності; діаграма бази даних; знімки інтерфейсу системи.

Календарний план:

№	Найменування етапів кваліфікаційної роботи	Термін виконання етапів роботи
1	Аналіз актуальності теми та характеристики предметної області	01.10.2025 – 05.10.2025
2	Аналіз існуючих підходів і інструментів SEO-аналізу	06.10.2025 – 10.10.2025
3	Постановка проблеми та обґрунтування необхідності автоматизації	11.10.2025 – 15.10.2025
4	Визначення мети, об’єкта, предмета і завдань дослідження	16.10.2025 – 20.10.2025
5	Вибір технологій розробки	21.10.2025 – 25.10.2025
6	Вибір методів реалізації SEO-аналізу	26.10.2025 – 30.10.2025
7	Опис алгоритмів роботи системи	31.10.2025 – 05.11.2025
8	Вибір бібліотек та компонентів	06.11.2025 – 10.11.2025
9	Розробка структури коду та ключових класів	11.11.2025 – 15.11.2025
10	Реалізація функцій аналізу	16.11.2025 – 20.11.2025
11	Інтеграція графічного інтерфейсу та багатопотоковості	21.11.2025 – 25.11.2025
12	Тестування додатку	26.11.2025 – 30.11.2025
13	Аналіз результатів та ефективності системи	01.12.2025 – 05.12.2025
14	Аналіз економічної ефективності	06.12.2025 – 10.12.2025

Дата видачі завдання: «15» квітня 2025 р.

Студент _____ / С. А. Бондарчук /

Керівник роботи _____ / А. А. Трачук /

РЕФЕРАТ

SEO-АНАЛІЗ, ПОШУКОВА ОПТИМІЗАЦІЯ, АВТОМАТИЗОВАНА СИСТЕМА, ВЕБ-САЙТИ, PYTHON, TKINTER, SQLITE, GOOGLE SEARCH CONSOLE, AHREFS, SEMRUSH.

Пояснювальна записка: 89 с., 16 рис., 5 табл., 1 дод., 19 джерел.

Мета кваліфікаційної роботи: розробка автоматизованої системи аналізу SEO для веб-сайтів, яка забезпечує користувачам зручний інтерфейс для комплексної перевірки показників пошукової оптимізації, виявлення помилок та надання рекомендацій щодо покращення видимості сайтів.

Предмет розробки: автоматизована система для аналізу SEO-параметрів веб-сайтів з метою забезпечення ефективної оптимізації та доступності для користувачів без глибоких технічних знань.

Шляхи досягнення мети: аналіз предметної області, вивчення існуючих інструментів, постановка проблеми, вибір технологій, розробка схем та алгоритмів, створення програмного забезпечення.

Основні конструктивні, технологічні та технікоексплуатаційні показники та характеристики: автономний десктопний додаток, комплексний аналіз понад 20 параметрів (технічних, контентних, мобільних), графічний інтерфейс, локальна база даних, багатопотоковість, оцінка за шкалою score/max_score.

Галузь застосування: інтернет-маркетинг, оптимізація веб-сайтів, електронна комерція, розвиток малого та середнього бізнесу в цифровому середовищі.

Висновки: розроблена система є ефективним інструментом для SEO-аналізу, знижує поріг входу в сферу оптимізації та відповідає сучасним вимогам пошукових систем.

ABSTRACT

SEO ANALYSIS, SEARCH ENGINE OPTIMIZATION, AUTOMATED SYSTEM, WEB-SITES, PYTHON, TKINTER, SQLITE, GOOGLE SEARCH CONSOLE, AHREFS, SEMRUSH.

Explanatory Note: 89 pages, 5 tables, 16 figures, 1 appendix, 19 sources.

Objective: To develop an automated SEO analysis system for web-sites that provides users with a convenient interface for comprehensive checking of optimization metrics, error detection, and recommendations for improving site visibility.

Subject of Development: An automated system for analyzing SEO parameters of web-sites to ensure effective optimization and accessibility for users without deep technical knowledge.

Ways to Achieve the Objective: Analysis of the problem domain, study of existing tools, problem statement, technology selection, development of schemes and algorithms, software creation.

Main Design, Technological, and Technical-Operational Indicators and Characteristics: Standalone desktop application, comprehensive analysis of over 20 parameters (technical, content, mobile), graphical interface, local database, multithreading, scoring on a score/max_score scale.

Field of Application: Internet marketing, web-site optimization, e-commerce, development of small and medium-sized businesses in the digital environment.

Conclusions: The developed system is an effective tool for SEO analysis, lowers the entry barrier into the optimization field, and meets modern search engine requirements.

ЗМІСТ

ВСТУП

У сучасному світі веб-сайти є ключовим інструментом для бізнесу, організацій та індивідуальних користувачів, оскільки саме через них відбувається взаємодія з аудиторією в цифровому середовищі. Завдяки швидкому розвитку технологій, зокрема штучного інтелекту та постійним змінам алгоритмів пошукових систем, видимість ресурсів у пошукових результатах стає вирішальним фактором успіху. Понад 53% усього трафіку веб-сайтів надходить з органічного пошуку, що підтверджує SEO як найефективніший канал для підвищення онлайн-присутності та конкурентоспроможності. Тому розробка інструментів для автоматизованого аналізу та оптимізації SEO є актуальною задачею, яка вимагає комплексного підходу, поєднання технічних рішень та маркетингових стратегій.

Основною задачею кваліфікаційної роботи є проведення аналізу існуючих методів SEO-оптимізації, визначення потреб користувачів, розробка ефективного інтерфейсу та власне створення автоматизованої системи аналізу SEO для веб-сайтів, яка враховує сучасні тенденції, такі як фокус на авторитетності, швидкості завантаження та користувацькому досвіді. Основний крок проекту полягатиме в наданні можливостей для комплексного моніторингу ключових параметрів (технічних, контентних та зовнішніх), автоматичного виявлення помилок та генерації рекомендацій щодо покращення. Така система сприятиме підвищенню ефективності оптимізації, зниженню витрат часу для малого та середнього бізнесу, а також забезпечить довгострокову видимість сайтів у пошукових системах, адаптуючись до змін алгоритмів, як-от інтеграція AI в SERP.

1 АКТУАЛЬНІСТЬ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1.1 Актуальність теми та характеристика предметної області

У сучасному інформаційному суспільстві веб-сайти виступають ключовим інструментом для представлення компаній, організацій та окремих фахівців у глобальній мережі. Рівень їхньої видимості у пошукових системах безпосередньо впливає на кількість користувачів, що відвідують ресурс, і, відповідно, на успішність бізнесу або соціальної ініціативи. Саме тому питання пошукової оптимізації (SEO) набуває все більшого значення.

SEO є комплексом заходів, спрямованих на підвищення позицій веб-сайту у результатах пошукових систем. Висока позиція у видачі Google чи інших пошукових систем дозволяє збільшити органічний трафік, сформувати позитивний імідж компанії, залучити клієнтів та забезпечити конкурентоспроможність. Проте сам процес SEO-аналізу є технічно складним та багатокомпонентним. Він вимагає врахування десятків параметрів — від правильності використання мета-тегів та структури сторінок до швидкості завантаження сайту, мобільної адаптації та якості зовнішніх посилань.

Водночас ринок пропонує велику кількість інструментів для SEO-аналізу, серед яких Google Search Console, Ahrefs, Semrush, Screaming Frog, GTMetrix та інші. Кожен з них має свої переваги, але жоден не забезпечує комплексного рішення для користувача з обмеженими технічними знаннями. Частина інструментів є платними та дорогими, інші — складними для засвоєння або обмеженими у функціоналі. Це створює серйозну проблему для малого та середнього бізнесу, який не може дозволити собі окремих SEO-фахівців, але має потребу у якісній оптимізації веб-ресурсів.

Для кращого розуміння сутності проблеми розглянемо характеристику предметної області, у межах якої виконується дослідження.

Предметна галузь дослідження охоплює SEO (Search Engine Optimization) — сукупність методів, стратегій та технологій, спрямованих на

підвищення видимості та позицій вебресурсів у результатах пошукових систем. SEO є ключовим напрямом у сфері інтернет-маркетингу, адже саме пошуковий трафік часто становить основне джерело відвідуваності сайтів, що безпосередньо впливає на прибутковість бізнесу, розвиток бренду та конкурентоспроможність компаній.

Особливістю предметної галузі є її динамічний розвиток. Алгоритми пошукових систем (Google, Bing, Yahoo та інші) постійно змінюються з метою покращення релевантності видачі та боротьби зі спамом. Це означає, що підходи, які були ефективними кілька років тому (наприклад, масове створення посилань чи перенасичення текстів ключовими словами), сьогодні можуть не лише втратити актуальність, а й призвести до зниження рейтингу ресурсу. Таким чином, SEO-фахівці та розробники програмного забезпечення стикаються з необхідністю постійно оновлювати інструменти та методи аналізу.

Отже, дослідження є актуальним, оскільки воно спрямоване на вирішення проблеми відсутності доступних та комплексних інструментів SEO-аналізу для користувачів без спеціальних технічних знань. Розробка такої системи сприятиме розвитку малого та середнього бізнесу, підвищить конкурентоспроможність веб-ресурсів та відповідатиме сучасним вимогам цифрового середовища.

Ще однією характерною рисою галузі є багатофакторність оцінки. На позиції сайту у видачі впливають сотні показників:

- якість та унікальність контенту;
- структура та семантична оптимізація сторінок;
- внутрішня перелінковка;
- швидкодія та мобільна адаптивність сайту;
- профіль зовнішніх посилань;
- поведінкові фактори користувачів (глибина перегляду, час перебування на сайті тощо).

Через таку різноманітність факторів ручний аналіз стає вкрай трудомістким та часто суб’єктивним. Тому виникає потреба у створенні автоматизованих систем SEO-аналізу, які здатні комплексно обробляти дані, виявляти слабкі місця сайту та надавати рекомендації щодо покращення.

Важливим аспектом предметної галузі є також інтеграція SEO з іншими напрямами цифрового маркетингу: контент-маркетингом, SMM, веб-аналітикою, контекстною рекламою. Це вимагає від сучасних інструментів SEO не лише технічного аналізу, але й здатності до взаємодії з різними системами (Google Analytics, Google Search Console, Ahrefs, Semrush та ін.).

Таким чином, особливості предметної галузі SEO можна узагальнити у таких ключових характеристиках:

- Динамічність та постійна еволюція алгоритмів пошукових систем.
- Багатофакторність впливу на результати пошуку.
- Високий рівень конкуренції у сфері онлайн-бізнесу.
- Необхідність автоматизації процесів аналізу та оптимізації.
- Інтеграція з іншими інструментами інтернет-маркетингу.

Усе це підкреслює актуальність та складність предметної галузі, а також обґрунтовує необхідність розробки автоматизованої системи SEO-аналізу вебсайтів.

Таким чином, актуальність даного дослідження полягає у необхідності розробки автоматизованої системи, здатної інтегрувати функціонал збору та аналізу SEO-даних з різних джерел, узагальнювати результати, автоматично виявляти критичні помилки та надавати зрозумілі рекомендації для користувача. Запровадження такого рішення дозволить суттєво знизити поріг входу у сферу SEO, підвищити ефективність оптимізації сайтів та зробити цей процес доступним широкому колу користувачів.

Крім того, з огляду на постійні зміни алгоритмів пошукових систем (особливо Google), автоматизована система повинна мати можливість гнучко оновлюватися та масштабуватися. Це робить її не лише практично корисною, а й перспективною з наукової точки зору, оскільки дослідження вимагає

пошуку нових підходів до автоматизації процесів збору, аналізу та візуалізації SEO-даних.

1.2 Аналіз існуючих підходів і інструментів SEO-аналізу

Для розуміння сучасного стану розвитку методів пошукової оптимізації важливо проаналізувати наукові праці та існуючі практичні рішення у сфері SEO-аналізу. Такий аналіз дозволяє визначити тенденції розвитку, ключові напрямки досліджень і наявні проблеми, які потребують вирішення.

Аналіз існуючих досліджень і практичних розробок у сфері SEO-аналізу дозволяє визначити сучасний стан проблеми, тенденції розвитку та прогалини, які може заповнити розробка автоматизованої системи.

Сучасні наукові праці та технічні публікації концентруються на таких напрямках:

Теоретичні основи SEO та фактори ранжування

Дослідники аналізують, які показники найбільш критичні для позицій вебсайтів у пошукових системах. Серед них виділяють технічні аспекти (структура сайту, швидкість завантаження, мобільна адаптивність), контентні фактори (ключові слова, релевантність, унікальність тексту) та зовнішні показники (зворотні посилання, репутація сайту). Важливою є також оцінка поведінкових факторів користувачів, таких як час перебування на сайті та глибина перегляду сторінок.

Існуючі інструменти SEO-аналізу

На ринку доступні численні програмні рішення для SEO-аналітики:

- Google Search Console

Надає інформацію про видимість сайту, пошукові запити та індексацію. Обмежений у глибокому аналізі контенту, але безкоштовний і інтегрований з Google. Підходить для базового моніторингу.

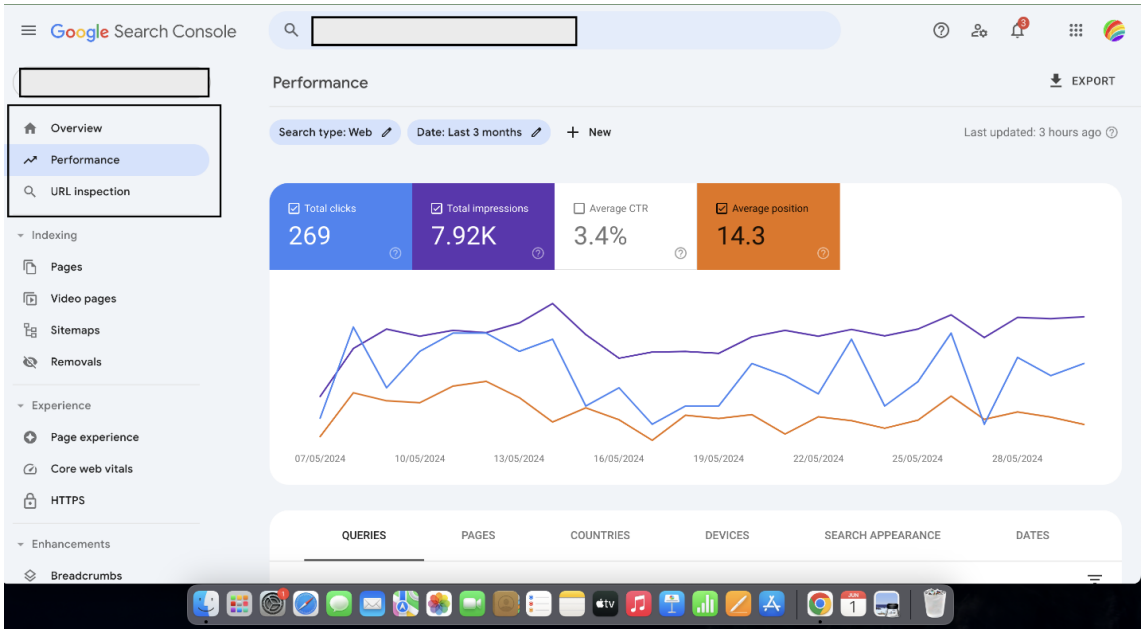


Рисунок 1.1 – Google Search Console

- Ahrefs

Комплексний інструмент для оцінки зовнішніх і внутрішніх SEO-факторів, аналізу конкурентів та ключових слів. Має великий функціонал, але висока вартість підписки.

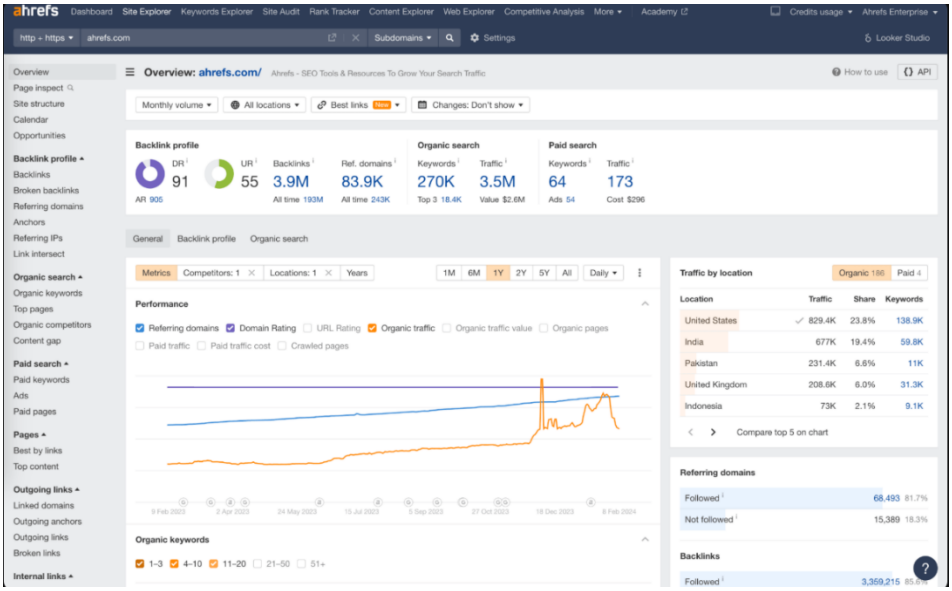


Рисунок 1.2 – Ahrefs

- SEMrush

Аналогічно Ahrefs, пропонує аналіз конкурентів, ключових слів, зовнішніх посилань. Комплексний, але також платний і може бути дорогим для малого бізнесу.

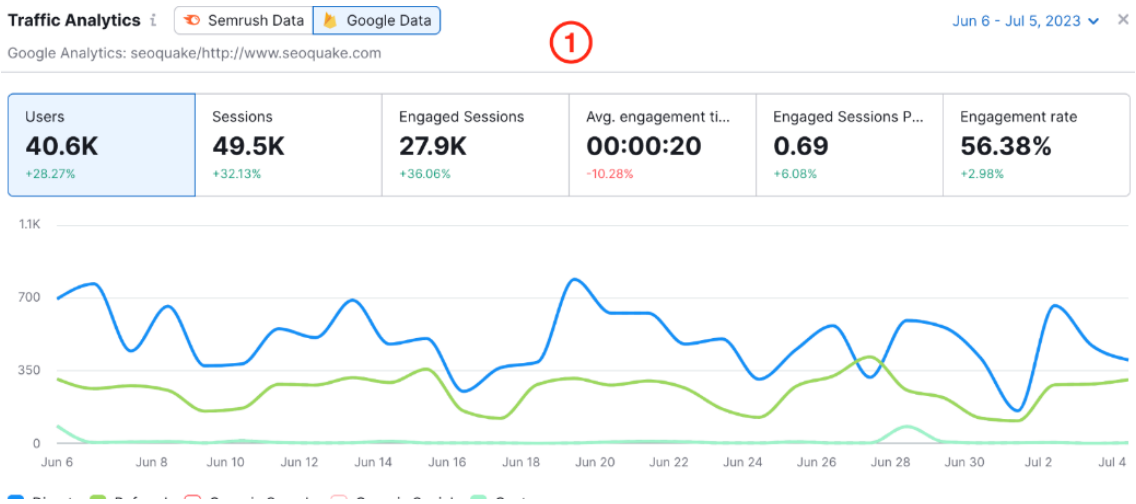


Рисунок 1.3 – SEMrush

- Screaming Frog

Дозволяє проводити технічний аудит сайту, перевірку мета-тегів, структури сторінок, наявності дублікатів. Вимагає знань для інтерпретації, але ефективний для технічного SEO.

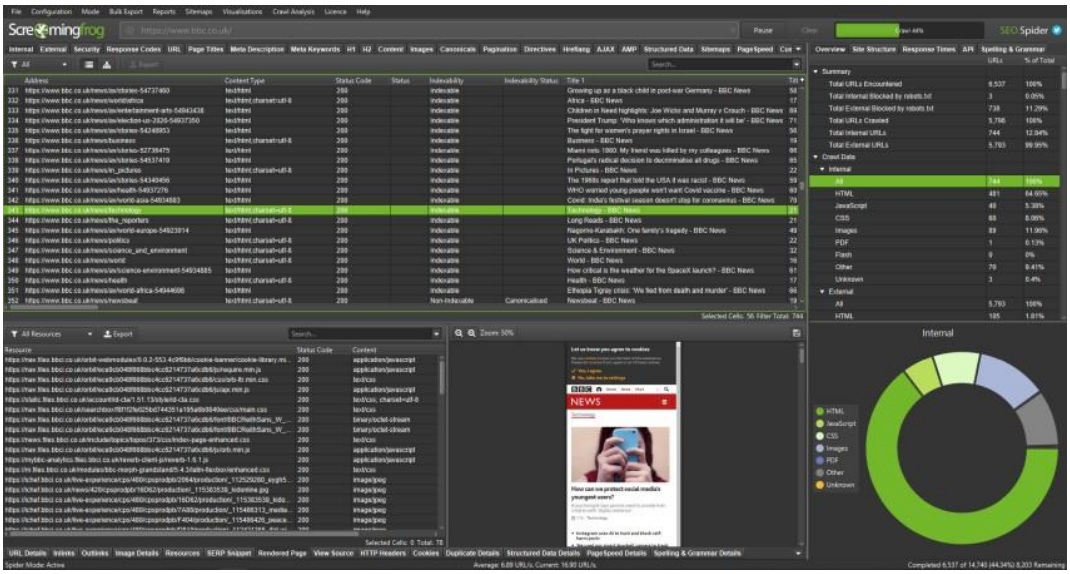


Рисунок 1.4 – Screaming Frog

- GTmetrix

Оцінює швидкодію сайту та оптимізацію ресурсів. Фокус на продуктивності, але не дає повної SEO-оцінки.

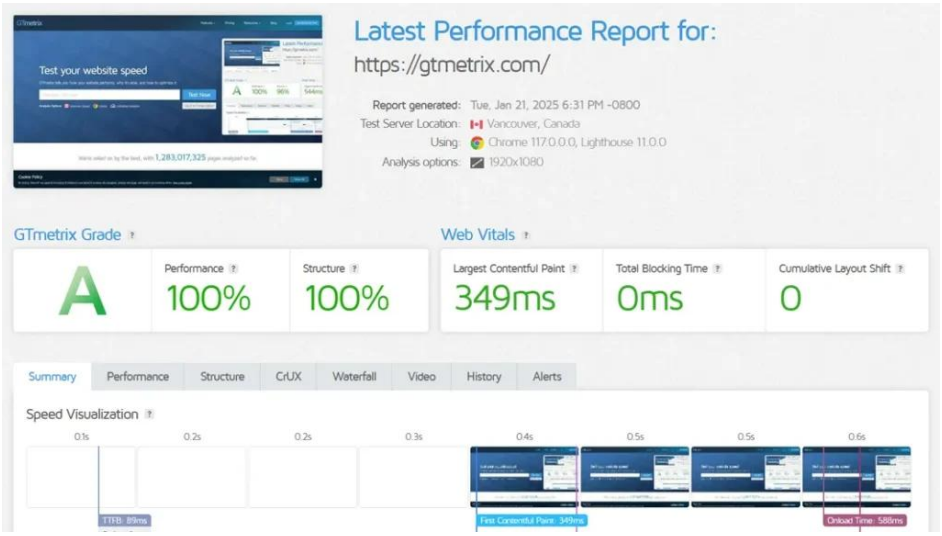


Рисунок 1.5 – GTmetrix

- PageSpeed Insights

Аналогічно GTmetrix, фокусується на швидкості завантаження та оптимізації. Безкоштовний інструмент від Google, але обмежений у комплексності.

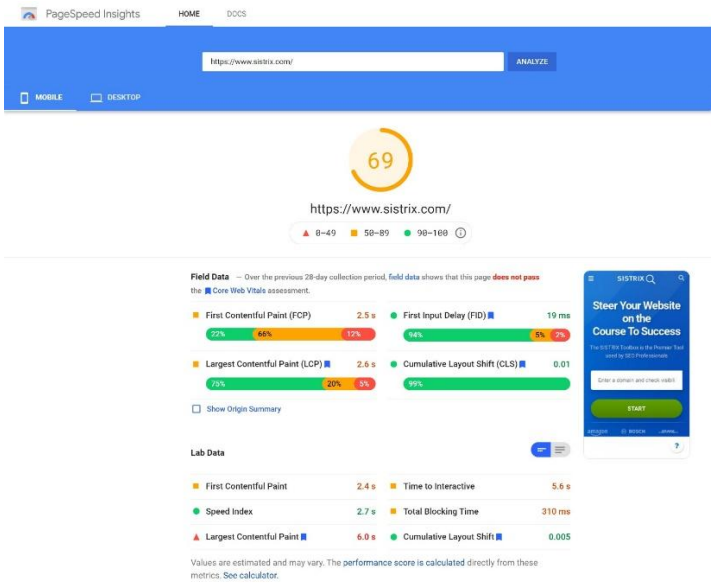


Рисунок 1.6 – PageSpeed Insights

Наукові підходи до автоматизації SEO

Останні дослідження розглядають методи автоматичного збору та обробки SEO-даних, застосування алгоритмів машинного навчання для прогнозування ефективності оптимізації та побудови метрик оцінки

релевантності контенту. Значна увага приділяється інтеграції даних із різних джерел для формування єдиного звіту та генерації практичних рекомендацій.

Виявлені прогалини та обмеження

- Більшість комерційних інструментів є дорогими та складними для користувачів з обмеженими технічними знаннями.
- Безкоштовні сервіси мають обмежений функціонал і не забезпечують комплексного аналізу.
- Не всі існуючі рішення здатні автоматично надавати рекомендації щодо покращення SEO, що значно ускладнює роботу власників сайтів і малих підприємств.

Таким чином, аналіз досліджень підтверджує актуальність розробки доступної, автоматизованої та комплексної системи SEO-аналізу, яка зможе інтегрувати дані з різних джерел, автоматично оцінювати стан сайту та формувати зрозумілі рекомендації для користувачів.

1.3 Постановка проблеми та обґрунтування необхідності автоматизації

Проведений аналіз показав, що процес пошукової оптимізації вебсайтів є складним, багатофакторним і потребує постійного моніторингу великої кількості показників.
Традиційні підходи до SEO-аналізу ґрунтуються на використанні набору окремих інструментів, кожен із яких орієнтований на певну групу параметрів — технічних, контентних або поведінкових. Така роз’єднаність ускладнює комплексну оцінку стану сайту, оскільки вимагає від користувача значних часових і аналітичних витрат, а також наявності спеціальних знань у сфері SEO.

Сучасні комерційні сервіси, хоча й забезпечують широкий функціонал, мають суттєві недоліки: високу вартість підписки, обмежені можливості персоналізації та складність інтерпретації результатів. Безкоштовні аналоги, навпаки, мають спрощений функціонал і не дозволяють отримати

комплексну картину SEO-стану вебресурсу. У результаті власники невеликих сайтів, блогів або підприємств змушені обирати між дорогими професійними системами та обмеженими безкоштовними інструментами, що знижує ефективність оптимізації.

Проблема полягає у відсутності доступного та зручного інструменту, здатного автоматично виконувати збір, аналіз і візуалізацію SEO-показників з різних джерел. Сучасні тенденції розвитку інформаційних технологій свідчать про потребу у створенні автоматизованої системи, що поєднує технічні засоби моніторингу, методи аналітичної обробки даних та інтелектуальні алгоритми оцінювання.

Обґрунтування необхідності автоматизації полягає у наступному:

- процес SEO-аналізу включає десятки повторюваних і рутинних дій, які можна алгоритмізувати;
- автоматизація дозволяє значно скоротити час перевірки, усунути людський фактор і підвищити точність результатів;
- створення єдиної системи аналізу спрощує сприйняття інформації та робить її доступною для користувачів без глибоких технічних знань;
- інтеграція декількох модулів у межах одного програмного продукту забезпечує можливість масштабування та гнучкого оновлення алгоритмів у майбутньому.

Таким чином, сутність проблеми полягає у необхідності розробки інструменту, який би поєднав зручність використання, комплексність аналізу та адаптивність до змін у пошукових алгоритмах. Запропонована автоматизована система повинна забезпечувати:

- збір та обробку даних з відкритих джерел і API сервісів;
- аналіз технічних, контентних і поведінкових показників сайту;
- автоматичне формування звітів і рекомендацій для користувачів.

Розв’язання цієї проблеми сприятиме підвищенню ефективності SEO-оптимізації, зменшенню трудомісткості аналізу та підвищенню доступності технологій веб-аналітики для малого й середнього бізнесу.

1.4 Мета, об’єкт, предмет і завдання дослідження

Метою даного дослідження є проєктування та розробка автоматизованої системи аналізу SEO для веб-сайтів, яка забезпечить комплексну перевірку основних показників пошукової оптимізації, дозволить користувачам оперативно отримувати аналітичні звіти, виявляти критичні помилки та отримувати рекомендації щодо покращення видимості сайту в пошукових системах.

Ця система повинна інтегрувати інструменти збору, обробки та візуалізації даних, бути масштабованою, гнучкою, та орієнтованою на кінцевого користувача. Особлива увага приділяється автоматизації рутинних процесів, зменшенню людського фактора в аналітиці та підвищенню ефективності прийняття рішень у сфері SEO-оптимізації.

Об’єктом дослідження є процес пошукової оптимізації веб-сайтів та методи їх аналітичної оцінки, спрямовані на підвищення видимості у пошукових системах.

До складу об’єкта входять:

- SEO-процеси, пов’язані з технічними аспектами сайту (швидкість завантаження, мобільна адаптація, структура заголовків, індексація, robots.txt, sitemap тощо).
- Контентні фактори (ключові слова, релевантність текстів, оптимізація мета-тегів, дублювання контенту).
- Зовнішні фактори (зворотні посилання, репутація сайту, соціальні сигнали).
- Методи збору та аналізу SEO-даних (Google Search Console, Ahrefs, Semrush, Screaming Frog, GTMetrix та інші інструменти).

Тобто, об’єкт — це вся сфера SEO-аналізу як процес, з яким взаємодіє користувач.

Предметом дослідження є методи, засоби та алгоритми автоматизації аналізу SEO-показників веб-сайтів, а також принципи побудови інформаційної системи, яка дозволить:

- інтегрувати дані з різних джерел та інструментів;
- виконувати автоматичну перевірку ключових SEO-параметрів (технічних, контентних, зовнішніх);
- виявляти типові помилки та слабкі місця у веб-сайті;
- формувати зрозумілі для користувача звіти;
- надавати практичні рекомендації щодо покращення видимості сайту у пошукових системах.
- Проаналізувати сучасний стан та тенденції розвитку SEO-оптимізації веб-сайтів: вивчити основні фактори ранжування в пошукових системах, їх вплив на видимість та трафік.
- Дослідити існуючі інструменти SEO-аналізу (Google Search Console, Ahrefs, Semrush, Screaming Frog, GTMetrix та інші), визначити їх переваги та недоліки для користувачів з різним рівнем підготовки.
- Сформувати вимоги до автоматизованої системи SEO-аналізу, враховуючи потреби кінцевих користувачів та виявлені проблеми існуючих рішень.
- Розробити концептуальну модель системи, що включає архітектуру, основні модулі (збір даних, обробка, аналіз, візуалізація результатів, генерація рекомендацій).
- Обґрунтувати методи та алгоритми обробки SEO-даних, які забезпечать комплексний і точний аналіз показників веб-сайтів.
- Реалізувати прототип автоматизованої системи SEO-аналізу з використанням відповідних технологій програмування та баз даних.
- Перевірити працездатність та ефективність розробленої системи на прикладі реальних веб-сайтів, оцінити точність аналізу та зручність використання.
- Сформувати рекомендації щодо подальшого розвитку та масштабування системи, у тому числі можливості інтеграції з іншими сервісами.

2 РОЗРОБКА СХЕМИ І АЛГОРИТМІВ, ВИБІР ІНСТРУМЕНТІВ І МЕТОДІВ

У цьому розділі розглядаються етапи проектування автоматизованої системи SEO-аналізу. Проектування включає вибір технологій, методів реалізації, створення UML-діаграм, опис алгоритмів, моделі даних та архітектури системи. Загальний підхід спрямований на забезпечення автономності, продуктивності та зручності використання. Система розроблена як десктопний додаток на Python, з локальним зберіганням даних у SQLite та графічним інтерфейсом на Tkinter. Проектування враховує багатопотоковість для уникнення блокування інтерфейсу під час аналізу.

2.1 Вибір технологій розробки

Проєкт автоматизованої системи SEO-аналізу вимагав ретельного підбору технологій, які дозволяють виконувати комплексний аналіз веб-ресурсів, забезпечувати надійне зберігання результатів та надавати зручний графічний інтерфейс для взаємодії з користувачем. Вибір кожного інструмента здійснювався з урахуванням кількох факторів: простоти впровадження, продуктивності, автономності роботи та можливості розповсюджувати програму у вигляді виконуваного файлу. У цьому підрозділі докладно обґрунтовується застосування кожної технології, пояснюється її роль у системі та визначаються переваги використання саме такого технологічного стеку.

Основною мовою розробки було обрано Python, оскільки саме ця мова забезпечує оптимальне поєднання простоти, продуктивності та великої кількості бібліотек для веб-аналізу. Python дозволяє швидко створювати прототипи, легко інтегрувати нові модулі й одночасно підтримує промислову якість коду. Ключовою перевагою є доступність величезної екосистеми пакетів для роботи з мережевими протоколами, HTML-кодом, базами даних, файлами та графічними інтерфейсами.

Для побудови графічного інтерфейсу було обрано бібліотеку Tkinter, що входить до базового набору Python. Основною причиною цього вибору стала її повна автономність: програма, створена на Tkinter, не потребує встановлення сторонніх бібліотек, а весь інтерфейс працює «з коробки» на будь-якій операційній системі.

Переваги Tkinter особливо помітні у проєктах, де важлива не складність дизайну, а стабільність, простота та відсутність зовнішніх залежностей. Tkinter забезпечує достатній набір елементів для створення зручного, інтуїтивного інтерфейсу:

- текстові поля для введення URL-адреси;
- кнопки запуску аналізу;
- таблиці для відображення історії аналізів;
- вікна повідомлень про помилки або статуси;
- індикатори прогресу через оновлення статусів перевірок.

Візуальна частина системи має важливе значення, оскільки користувачі повинні мати змогу запускати SEO-аналіз без технічних знань. Завдяки графічному інтерфейсу програмне забезпечення стає доступним для будь-кого: власників сайтів, маркетологів, студентів або фахівців, які потребують швидкої діагностики веб-ресурсу.

Крім того, Tkinter дозволяє легко масштабувати інтерфейс, додавати нові кнопки, поля та модулі, що робить проєкт гнучким у розширенні.

Одним із ключових компонентів системи є локальне зберігання результатів аналізу. Для цього була обрана SQLite, оскільки вона поєднує простоту використання та повну автономність. На відміну від серверних СУБД, SQLite не потребує інсталяції серверної частини та працює на основі одного файлу формату .db, що робить її ідеальним рішенням для локальних програм.

SQLite забезпечує можливість зберігати:

- історію перевірених сайтів;

- результати кожного аналізу (score, details, issues, recommendations, checks у JSON-форматі);
- дати та час перевірок;
- загальні SEO-оцінки та детальні параметри.

Критично важливо, що SQLite гарантує цілісність даних навіть у разі неочікуваного завершення роботи програми. Завдяки транзакційності, усі операції виконуються без ризику пошкодження бази. Враховуючи, що обсяги збережених даних у SEO-аналізаторі порівняно невеликі (до кількох мегабайт), SQLite є більш ніж достатнім рішенням.

Однією з найбільш важливих частин системи є модуль, який отримує HTML-код сторінки, аналізує його та формує список SEO-показників. Для цього були обрані бібліотеки Requests (для базових HTTP-запитів), BeautifulSoup (для парсингу HTML) та Playwright (для рендерингу JavaScript-контенту).

Requests дозволяє контролювати запити, обробляти редиректи, отримувати статуси відповіді сервера, визначати час завантаження та фіксувати помилки. Це критично, оскільки час відповіді та коректність HTTP-заголовків прямо впливають на SEO-оцінку.

BeautifulSoup забезпечує швидке та структуроване опрацювання HTML-документа. Програма може знаходити потрібні теги, визначати дубльовані елементи, перевіряти структуру, доступність метаданих та інші параметри, що важливі для пошукових систем.

Playwright додано для отримання повного рендереного HTML (з JS), вимірювання Core Web Vitals (LCP, CLS, INP), мобільних параметрів (font_size, button_sizes) та витягнення CSS-тексту для перевірки media queries. Це дозволяє аналізувати динамічні сайти, де базовий Requests недостатній.

Система також використовує threading для багатопотокового аналізу (щоб не блокувати GUI), queue для оновлення прогресу, ssl/socket для глибокої перевірки SSL-сертифікатів, re/statistics для розрахунку читабельності (Flesch score) та CWV-метрик.

Для оцінки сайту було визначено великий перелік SEO-параметрів, які охоплюють як технічні, так і контентні характеристики веб-ресурсу. Усі критерії розподілені на кілька логічних груп. Система також детектує тип сайту (general, video, blog, ecommerce) для адаптації порогів (наприклад, менше тексту для video-сайтів).

Метадані та структура HTML Система аналізує:

- наявність та унікальність тегу title;
- довжину title (оптимально 30–60 символів);
- наявність тегу meta description та його якість;
- коректність заголовків H1–H6 та їх ієрархію;
- наявність дубльованих заголовків (з толерантністю для блогів);
- наявність та заповнення атрибуту alt у зображень.

Ці показники впливають на індексацію, ранжування та CTR сторінки.

Технічний стан сторінки До технічних параметрів належать:

- розмір HTML-документа;
- час відповіді сервера;
- статуси HTTP-заголовків;
- наявність SSL-сертифіката (з перевіркою терміну дії);
- правильність налаштування редиректів;
- наявність robots.txt;
- наявність sitemap.xml.

Технічні помилки часто є головною причиною низьких позицій у пошуку.

Контентна оптимізація Аналізується:

- кількість текстового контенту (raw та JS-рендерений);
- співвідношення тексту до HTML;
- унікальність заголовка H1;
- читабельність тексту (Flesch score з адаптацією для кирилиці).

Внутрішні та зовнішні посилання Система перевіряє:

- кількість внутрішніх посилань;

- кількість зовнішніх посилань;
- наявність битих лінків;
- дубльовані URL-адреси;
- відповідність структури URL рекомендаціям Google.

Юзабіліті та доступність До цієї групи увійшли:

- перевірка Mobile-friendly параметрів (viewport, media queries з CSS, font_size >=14px, button_sizes >=44px);
- Core Web Vitals (LCP <2500ms, CLS <0.1, INP <200ms);
- читабельність структури DOM.

Додаткові метадані

- Meta robots (noindex);
- Canonical;
- OpenGraph (мінімум 4 теги);
- Twitter Cards (мінімум 3 теги);
- Schema.org (наявність ld+json).

Завершальним етапом технологічного вибору стало рішення збирати програму за допомогою PyInstaller. Це дозволяє створити повністю автономний .exe-файл, який містить:

- програмний код;
- інтерфейс;
- бібліотеки;
- базу SQLite.

Кінцевий користувач отримує програму, яка запускається одним кліком і не потребує встановлення жодних додаткових компонентів.

2.2 Вибір методів реалізації SEO-аналізу

Для побудови автоматизованої системи SEO-аналізу ключовим є правильний вибір методів, які забезпечують збір, інтерпретацію та оцінку параметрів веб-сторінок. Оскільки SEO складається з технічної, контентної, структурної та поведінкової частин, кожен модуль системи повинен

використовувати найбільш оптимальні алгоритми, що дозволяють точно та однозначно визначати стан сайту. У цьому підрозділі наведено обґрунтування вибору методів, підходів і алгоритмів, які застосовані в системі, та пояснюється, чому саме ці методи найбільш ефективні для дипломної роботи.

Першим етапом SEO-аналізу є отримання HTML-коду сторінки. Саме на основі цього коду система визначає структуру документа, метадані, наявність помилок або оптимізаційних елементів.

Для цього у проєкті застосовано два основні методи:

HTTP-запити методом GET з Requests Запити GET дозволяють отримати базовий HTML-код сторінки, а також фіксують такі важливі SEO-параметри, як:

- час відповіді сервера;
- статус HTTP-коду (200, 301, 404 тощо);
- наявність перенаправлень;
- можливі помилки (5xx, timeout).

Метод GET є стандартом у веб-технологіях і забезпечує найбільш точне відтворення того, як сайт бачить пошуковий робот. Альтернативи у вигляді headless-браузерів (наприклад, Selenium) не обрані через надмірну ресурсозатратність та непридатність для швидкої масової перевірки. Метод GET працює швидко, стабільно та дає коректні SEO-дані.

Рендеринг з Playwright для JS-контенту Playwright запускає headless Chromium для отримання повного рендереного HTML (з JS), вимірювання CWV (LCP, CLS, INP), мобільних параметрів (font_size, button_sizes) та витягнення CSS-тексту для перевірки media queries. Це дозволяє аналізувати динамічні сайти, де базовий HTML недостатній.

DOM-аналіз дозволяє повністю відтворити структуру документа без необхідності рендерингу сторінки у браузері, що прискорює роботу програми.

Метадані (title, description) та заголовки H1–H6 є фундаментом On-Page SEO. Для їх аналізу обрано такі методи:

Методи перевірки наявності та валідності тегів Система визначає:

- чи є тег;
- чи заповнений він правильно;
- чи відповідає рекомендаціям Google;
- чи дублюється (для title, headings).

Ці перевірки виконуються шляхом точкового пошуку тегів у DOM-дереві та оцінки їх атрибутів. Такий метод забезпечує стовідсоткову точність, уникає хибних спрацьовувань та не залежить від візуального вигляду сайту.

Метод символьного аналізу довжини текстових полів Для title та description важлива не лише наявність, але й довжина. Методи вимірювання символів дають можливість автоматично виявляти:

- занадто короткі метадані (містять мало інформації);
- надмірно довгі (обрізаються у видачі);
- некоректні символи або пробіли.

Така перевірка побудована на простому, але дуже ефективному алгоритмі порівняння кількості символів із рекомендованими нормативами SEO.

Методи логічної оцінки структури заголовків Алгоритм аналізує:

- кількість H1 (має бути один);
- наявність ієрархії H2–H6;
- пропуски у структурі (наприклад, H1 → H3 без H2);
- дубльовані заголовки (з толерантністю 20% для блогів).

Такий метод дозволяє швидко визначити структурні порушення, які негативно впливають на індексацію.

Технічна оптимізація — одна з найбільш критичних частин якості сайту. Для цього обрані такі методи:

Аналіз HTTP-статусів і редиректів Система перевіряє:

- початковий URL;

- наявність перенаправлень;
- кількість перенаправлень;
- типи перенаправлень (301, 302, 307);
- кінцевий статус сторінки.

Це реалізується методом трасування ланцюга редиректів. Вибір такого методу обґрунтований тим, що саме пошукові системи враховують довгі ланцюги редиректів як негативний сигнал.

Метод вимірювання продуктивності Програма фіксує:

- час відповіді сервера;
- час завантаження HTML;
- розмір сторінки.

Цей метод дозволяє оцінити проблеми з хостингом, сервером або важким контентом. Пошукові системи значною мірою орієнтуються на швидкість, тому даний метод є обов’язковим.

Метод перевірки наявності та коректності службових файлів Це стосується:

- robots.txt;
- sitemap.xml;
- canonical-посилань;
- meta-robots (noindex).

Метод полягає у прямому доступі до файлів robots.txt і sitemap.xml та у пошуку відповідних тегів у HTML-коді. Це простий, але найбільш надійний спосіб перевірити цілісність SEO-налаштувань.

Метод перевірки SSL Використовує ssl/socket для перевірки терміну дії сертифіката (не менше 30 днів) та HTTPS-протоколу.

У контентному аналізі використовуються методи:

Метод підрахунку обсягу тексту Алгоритм здійснює:

- виділення текстового контенту з HTML (raw та JS-рендерений);
- очищення від зайвих тегів;
- обчислення загальної кількості символів та слів;

- визначення співвідношення тексту до HTML-коду (мінімум 300 символів для non-video).

Цей метод дозволяє оцінити контентну наповненість сторінки та виявити надмірно «порожні» сторінки.

Метод оцінки читабельності Використовує Flesch score з адаптацією для кирилиці (поріг 40 для укр., 30 для англ.). Алгоритм рахує речення, слова, склади за допомогою re.

Посилання відіграють ключову роль у побудові структури сайту. Для їх аналізу використано такі методи:

Аналіз внутрішніх лінків Система:

- визначає всі -посилання;
- класифікує їх як внутрішні або зовнішні;
- перевіряє коректність URL;
- аналізує кількість посилань;
- шукає биті лінки (head-запитами, до 10 перевірених).

Метод зосереджений на точковому аналізі всіх тегів , що гарантує повноту результатів.

Визначення битих посилань Здійснюється шляхом виконання HTTP-запитів до кожного посилання. Цей метод забезпечує 100% точність, оскільки система отримує реальний HTTP-код сторінки.

Адаптивність є обов’язковою вимогою сучасного SEO. Для перевірки використовуються методи:

- визначення наявності тегу <meta name="viewport">;
- аналіз розміру шрифту (font_size >=14px);
- визначення адаптивної верстки через CSS-медіазапити (з витягнутого CSS);
- перевірка розмірів кнопок (button_sizes >=44px).

Це достатньо точні методи для локального SEO-аналізу, без звернення до сторонніх API.

Окремим блоком реалізовано метод, який формує інтегральну SEO-оцінку сторінки. Цей метод базується на:

- класифікації помилок за рівнями важливості (critical: 20 балів, important: 10, minor: 5);
- додаванні балів тільки за успішні перевірки (score += bonus за OK);
- формуванні загального рейтингу (score / max_score).

Методика близька до тієї, що застосовують SEO-платформи Semrush, Ahrefs та SEOptimer, але адаптована для локального використання без серверних обчислень.

2.3 Створення UML-діаграм

У процесі розробки автоматизованої системи SEO-аналізу важливо не лише реалізувати функціональну частину, але й забезпечити структуроване планування архітектури. UML (Unified Modeling Language) дозволяє формалізувати логіку роботи програми, зв’язки між компонентами, сценарії взаємодії та структуру даних. UML-діаграми значно полегшують етапи проектування, розробки та подальшої підтримки системи.

UML-набори діаграм можна поділити на два види: структурні (що описують будову системи) та поведінкові (що описують роботу та процеси). У дипломному проєкті доцільно використати кілька основних діаграм, що найбільш точно відобразять принципи роботи SEO-системи.

Діаграма класів (Class Diagram)

Показує структуру системи: які класи існують, їхні атрибути, методи та взаємозв’язки. У нашій системі ключовими класами є:

- DatabaseManager — керує SQLite (init_db, save_result, get_history тощо);
- SEOAnalyzerThread — виконує аналіз у потоці (run, analyze_ssl_advanced, analyze_performance тощо);
- SEOAnalyzerApp — GUI на Tkinter (setup_styles, create_notebook, start_analysis тощо).

Ця діаграма (див. рис. 2.1) показує архітектурну логіку програми та взаємодію компонентів.

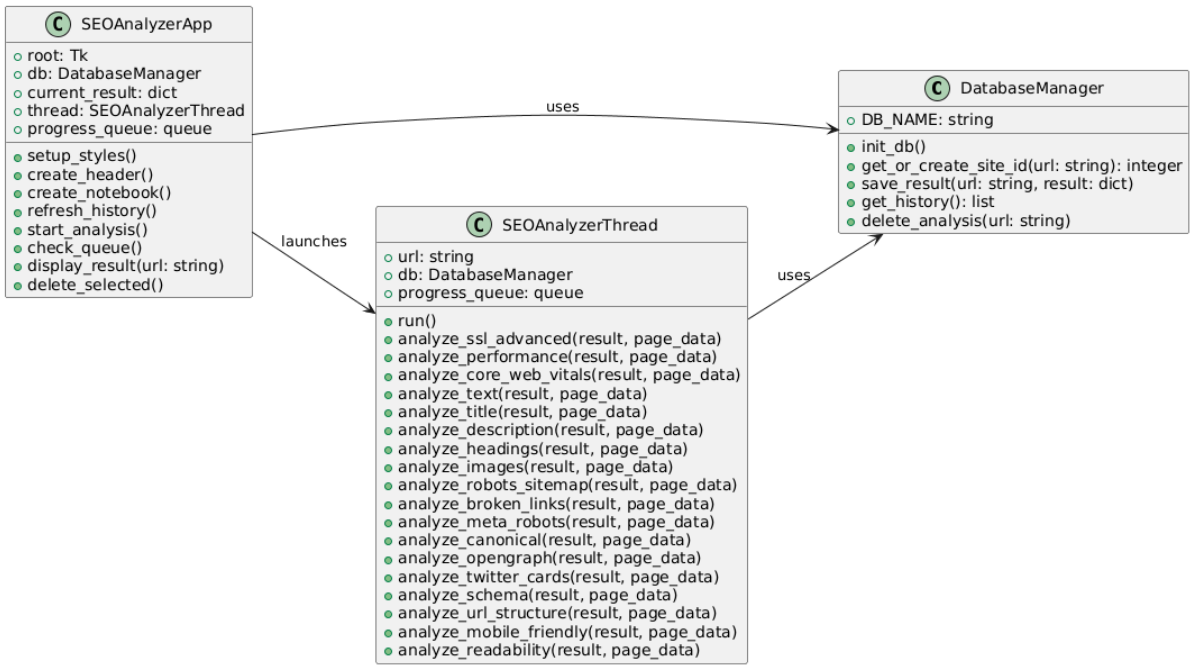


Рисунок 2.1 – Діаграма класів

Діаграма послідовності (Sequence Diagram)

Показує послідовність дій під час аналізу:

- Користувач вводить URL у SEOAnalyzerApp;
- Запускається SEOAnalyzerThread;
- Викликається fetch_page_data (Requests + Playwright);
- Виконуються analyze_... методи;
- Результати зберігаються через DatabaseManager;
- Оновлюється GUI через queue.

Ця діаграма (див. рис. 2.2) показує бізнес-логіку та алгоритм.

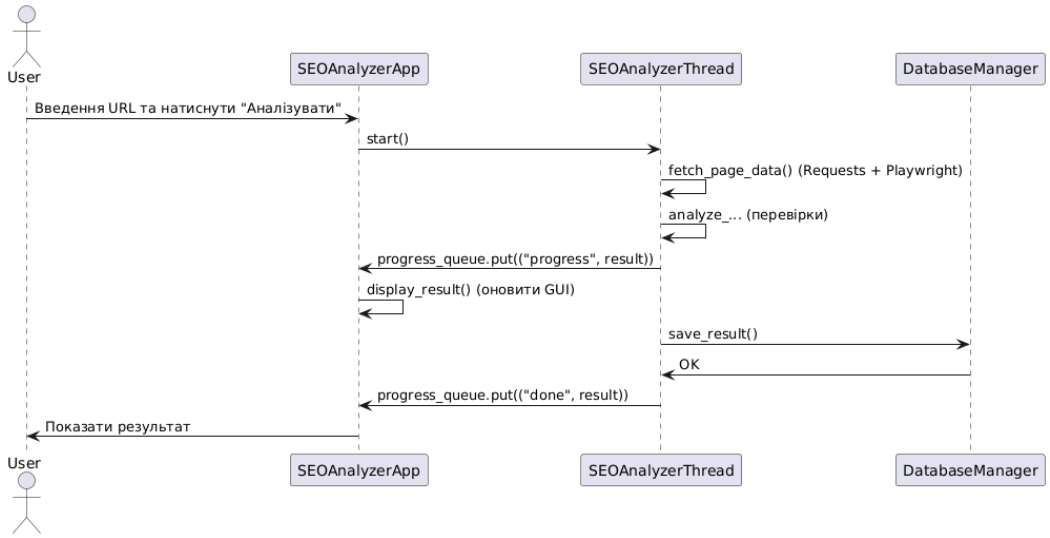


Рисунок 2.2 – Діаграма класів

2.4 Опис алгоритмів роботи системи

Алгоритми роботи автоматизованої системи SEO-аналізу визначають логіку обробки даних, взаємодію між функціональними модулями, порядок виконання операцій та способи формування кінцевого звіту. Система побудована таким чином, щоб забезпечити коректний збір даних із вебсайту, їх подальшу обробку, аналіз ключових SEO-параметрів та надання користувачеві структурованого, деталізованого звіту. Нижче наведено опис основних алгоритмів, що формують повний цикл роботи системи.

Першим етапом роботи системи є отримання від користувача адреси вебсайту для аналізу. Для коректності роботи застосовується алгоритм базової валідації введених даних.

Основні кроки алгоритму:

- Користувач вводить URL у графічному інтерфейсі програми (Tkinter Entry).
- Система перевіряє відповідність URL формальному шаблону (додає https:// якщо відсутній).
- У разі помилки користувач отримує повідомлення із поясненням або пропозицією виправити введені дані.

- Якщо URL коректний — система переходить до наступного етапу збору вебданих.

Цей алгоритм дозволяє уникнути типових помилок, які можуть призвести до некоректного завантаження вебсторінки.

Алгоритм збору даних із вебсайту відповідає за отримання HTML-коду сторінки, що підлягає SEO-аналізу. Для цього використовується функція fetch_page_data.

Покрокова схема роботи алгоритму:

- Запуск Requests для базового HTML, часу завантаження, редиректів, статусу.
- Запуск Playwright для рендереного HTML, CWV, мобільних параметрів (font_size, button_sizes), CSS-тексту.
- Комбінування даних (combined soup/text/len).
- Детекція типу сайту (general, video, blog, ecommerce) за URL/контентом.
- У випадку помилки (тайм-аут, SSL) — фіксація в логах.

Алгоритм також фіксує статус-код відповіді сервера (200, 301, 404, 500 тощо), що є важливим SEO-показником.

Отриманий HTML передається до BeautifulSoup для структурної обробки документа.

Алгоритм включає:

- Нормалізація HTML (видалення зайвих пробілів, приведення коду до валідної структури, якщо можливо).
- Витягнення ключових елементів: <title>, <meta> (description, robots), [image] (alt), (href), <link> (canonical), <script> (ld+json).
- Підрахунок їхньої кількості та агрегування в структуру page_data.
- Перевірка наявності дублікатів заголовків, пустих тегів, невалідних атрибутів.

Результатом виконання алгоритму є структурований набір даних, який використовується на етапі SEO-аналізу.

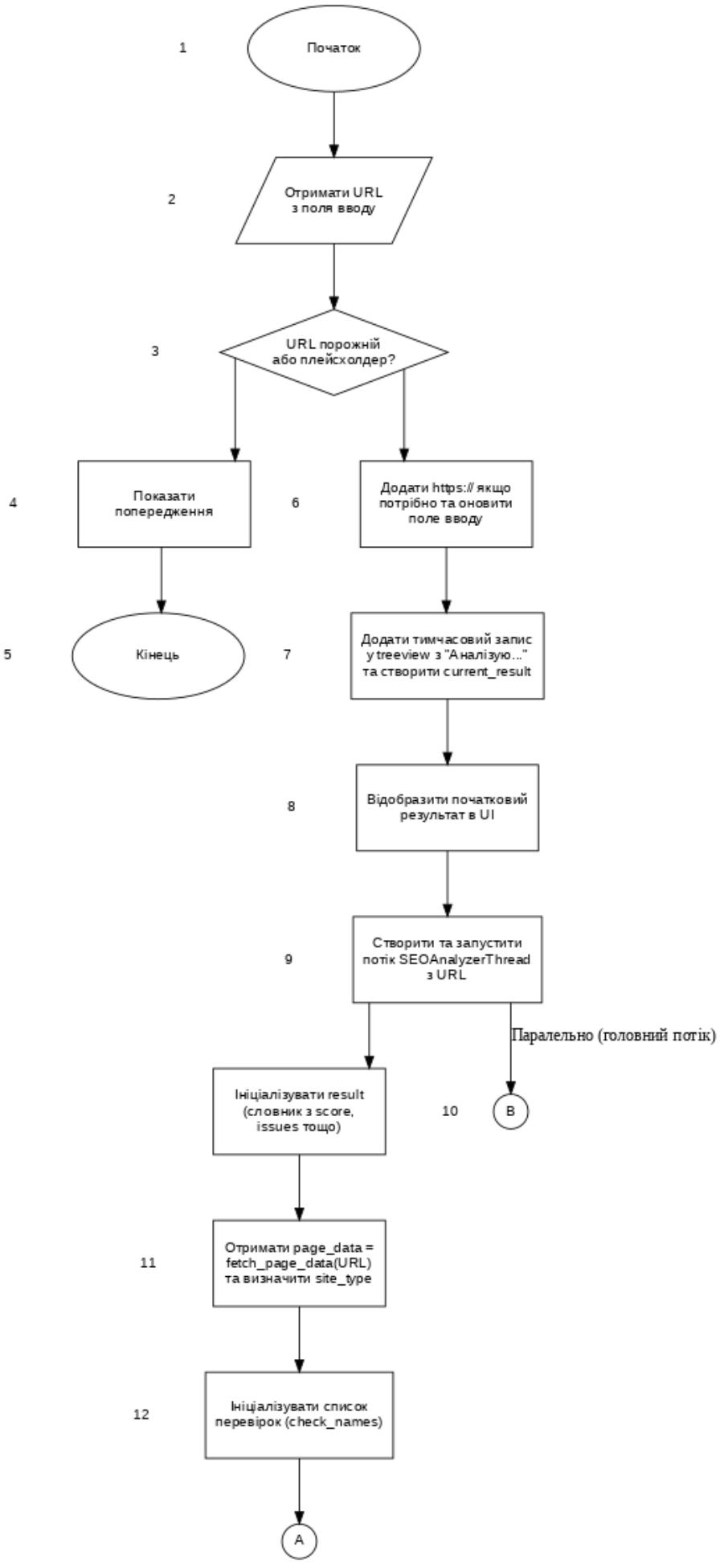
Алгоритм SEO-аналізу — центральний алгоритм системи, що оцінює стан сторінки за визначеним набором критеріїв. На основі page_data виконується обчислення показників, виявлення SEO-помилок та формування рекомендацій. Виконується в потоці (SEOAnalyzerThread) з оновленням прогресу через queue.

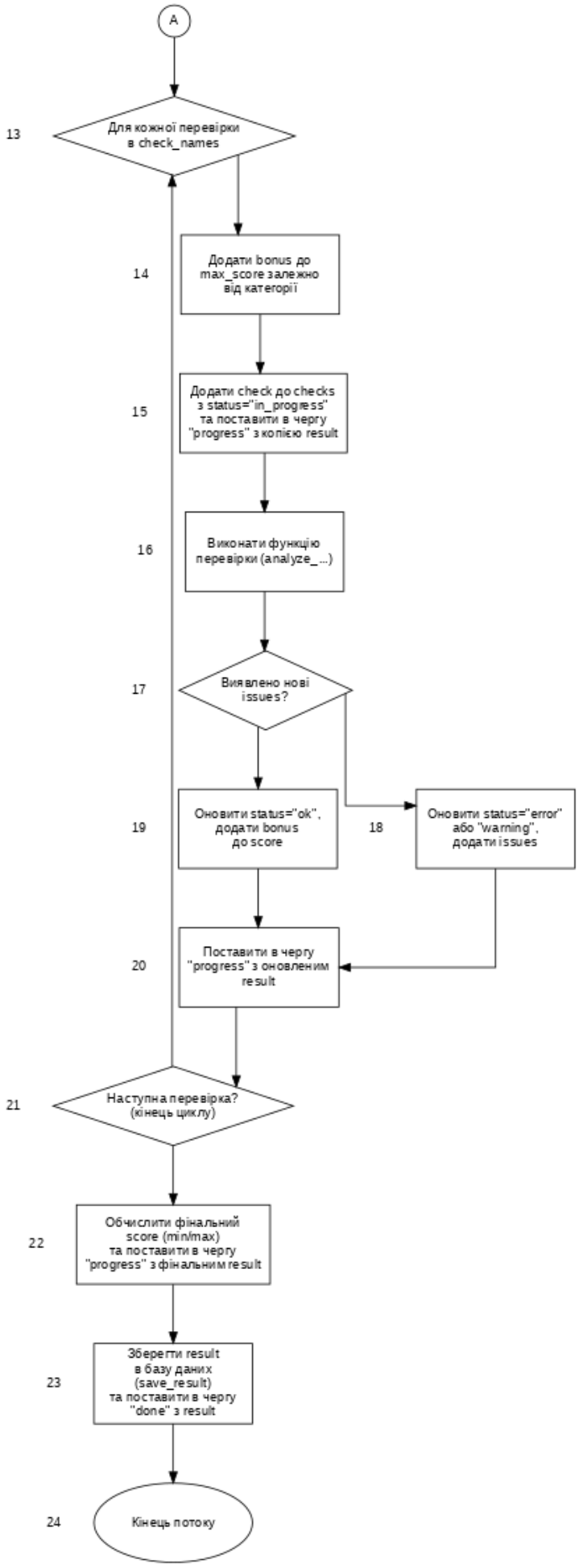
Алгоритм включає такі підпроцеси:

- Аналіз метаданих: перевірка довжини title (30–60), description, meta-robots (noindex), canonical, OpenGraph/Twitter/Schema.
- Аналіз заголовків: наявність H1, ієрархія, дублі (analyze_headings).
- Аналіз зображень: alt, кількість (analyze_images).
- Аналіз посилань: внутрішні/зовнішні, биті (analyze_broken_links).
- Аналіз техніки: SSL (analyze_ssl_advanced), performance (load_time <2.5s, редиректи <3), CWV (analyze_core_web_vitals).
- Аналіз контенту: текст (analyze_text), читабельність (analyze_readability з Flesch).
- Аналіз мобільності: viewport, media queries, font/button sizes (analyze_mobile_friendly).
- Аналіз файлів: robots/sitemap (analyze_robots_sitemap).
- URL: структура (analyze_url_structure).

Кожен критерій оцінюється за шкалою, визначеною у методичній частині роботи, після чого формується технічний висновок.

Схема алгоритму (див. рис. 2.2).


1
Початок
2
Отримати URL з поля вводу
3
URL порожній або плейсхолдер?
4
Показати попередження
5
Кінець
6
Додати https:// якщо потрібно та оновити поле вводу
7
Додати тимчасовий запис у treeview з "Аналізую..." та створити current_result
8
Відобразити початковий результат в UI
9
Створити та запустити потік SEOAnalyzerThread з URL
Паралельно (головний потік)
10
B
Ініціалізувати result (словник з score, issues тощо)
11
Отримати page_data = fetch_page_data(URL) та визначити site_type
12
Ініціалізувати список перевірок (check_names)
A


A
13
Для кожної перевірки в check_names
14
Додати bonus до max_score залежно від категорії
15
Додати check до checks з status="in_progress" та поставити в чергу "progress" з копією result
16
Виконати функцію перевірки (analyze_...)
17
Виявлено нові issues?
18
Оновити status="error" або "warning", додати issues
19
Оновити status="ok", додати bonus до score
20
Поставити в чергу "progress" з оновленим result
21
Наступна перевірка? (кінець циклу)
22
Обчислити фінальний score (min/max) та поставити в чергу "progress" з фінальним result
23
Зберегти result в базу даних (save_result) та поставити в чергу "done" з result
24
Кінець потоку

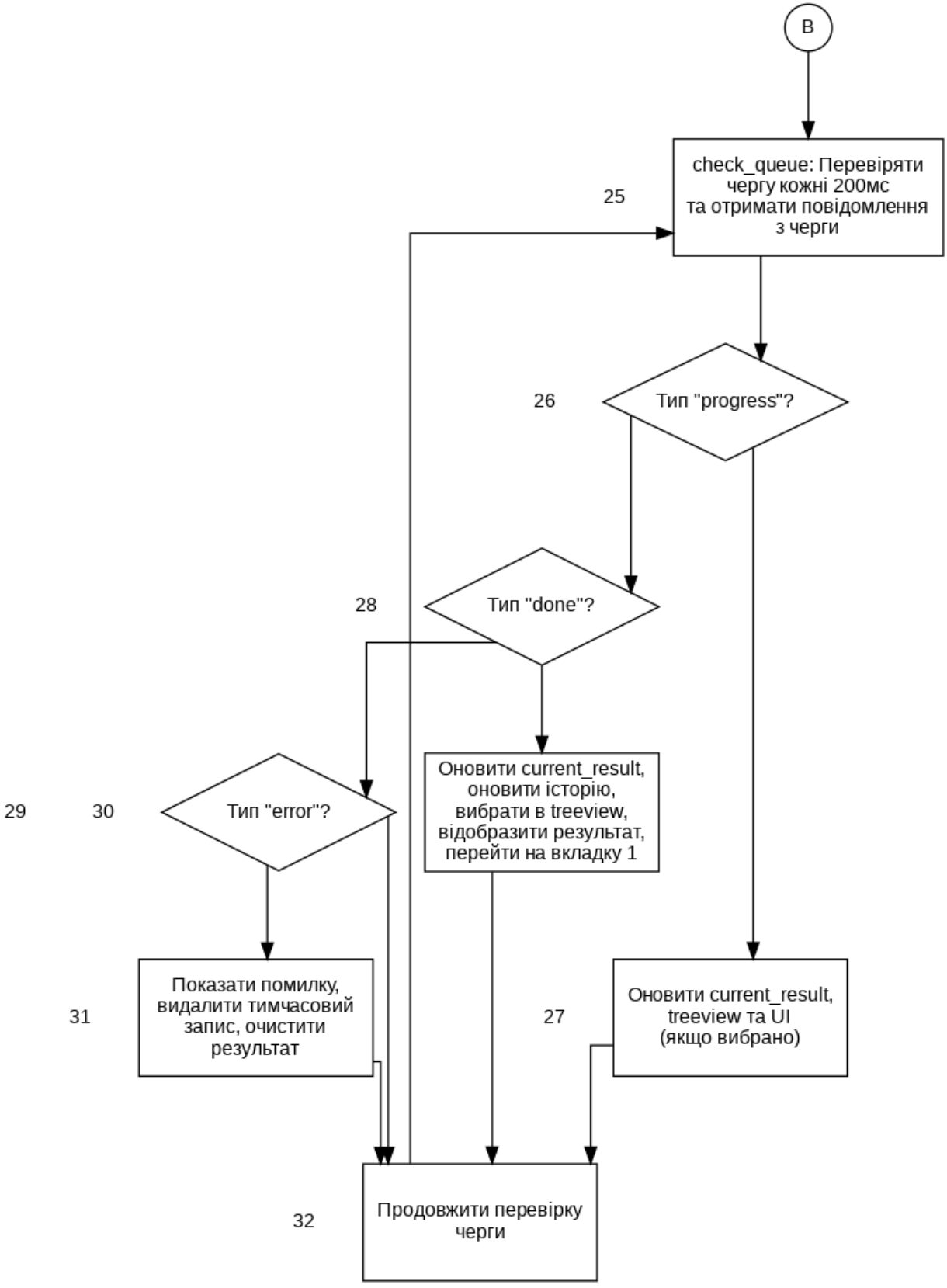


Рисунок 2.2 – Основний алгоритм програми

2.5 Моделі даних

Модель даних системи базується на реляційній базі SQLite, яка забезпечує ефективне зберігання історії аналізів та результатів. База складається з двох таблиць: sites (для зберігання URL сайтів та часу останньої перевірки) та analysis (для детальних результатів аналізу). Таблиці пов'язані через зовнішній ключ site_id.

Таблиця sites:

- id: INTEGER PRIMARY KEY AUTOINCREMENT – унікальний ідентифікатор сайту.
- url: TEXT UNIQUE NOT NULL – URL сайту (унікальний).
- last_checked: TIMESTAMP – час останньої перевірки.

Таблиця analysis:

- id: INTEGER PRIMARY KEY AUTOINCREMENT – унікальний ідентифікатор аналізу.
- site_id: INTEGER FOREIGN KEY REFERENCES sites(id) – посилання на сайт.
- timestamp: TIMESTAMP DEFAULT CURRENT_TIMESTAMP – час аналізу.
- score: INTEGER – оцінка SEO.
- max_score: INTEGER – максимальна можлива оцінка.
- details_json: TEXT – деталі в JSON.
- issues_json: TEXT – проблеми в JSON.
- recommendations_json: TEXT – рекомендації в JSON.
- checks_json: TEXT – статуси перевірок в JSON.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У цьому розділі описано практичну реалізацію автоматизованої системи SEO-аналізу на основі спроектованої архітектури (див. розділ 2). Розглянуто імпортовані бібліотеки, структуру коду з фокусом на класах і функціях, реалізацію графічного інтерфейсу, багатопотоковості, збереження даних та тестування. Реалізація виконана в Python, з акцентом на автономність і продуктивність. Код програми дозволяє проводити повний аналіз веб-сторінки, візуалізувати результати та зберігати історію.

3.1 Бібліотеки та компоненти

Підключення бібліотек здійснюється на початку скрипту для забезпечення всіх необхідних функцій системи. Обрані бібліотеки охоплюють інтерфейс, мережеві запити, парсинг, рендеринг, багатопотоковість і роботу з даними. Нижче наведено повний перелік імпортів з описом ролі кожної:

```
import tkinter as tk
from tkinter import ttk, messagebox, scrolledtext
from datetime import datetime, timedelta
import sqlite3
import json
import threading
import requests
import time
from bs4 import BeautifulSoup
from playwright.sync_api import sync_playwright
from urllib.parse import urljoin, urlparse
import ssl
import socket
import queue
import re
import statistics
```

- Tkinter (tk, ttk, messagebox, scrolledtext): Основний фреймворк для графічного інтерфейсу. tk створює вікна та базові елементи, ttk – стилізовані віджети (наприклад, кнопки в фіолетовому стилі), messagebox – для помилок (наприклад, "Невалідний URL"), scrolledtext –

для відображення issues з прокруткою. Це дозволяє створити інтуїтивний GUI без зовнішніх залежностей.

- datetime, timedelta: Обробка часу (фіксація timestamp аналізу, обчислення терміну SSL). Наприклад, timedelta(days=30) для перевірки сертифікатів.
- sqlite3: Локальна база даних для історії аналізів. Використовується для CRUD-операцій (create, read, update, delete) без серверу.
- json: Серіалізація результатів (details, issues тощо) для збереження в базі як TEXT. Забезпечує гнучкість даних.
- threading: Багатопотоковість для фонового аналізу (Thread для SEOAnalyzerThread), щоб GUI залишався responsive.
- requests: Базові HTTP-запити (GET для raw HTML, редиректів, статусів). Headers імітують браузер для уникнення блокувань.
- time: Вимірювання load_time, затримки в потоці (time.sleep для стабільності).
- BeautifulSoup (bs4): Парсинг HTML/Soup для витягнення тегів (title, meta, h1–h6, img, a). Підтримує combined soup з raw і JS.
- playwright.sync_api: Рендеринг JS у headless Chromium (goto(url), evaluate для CWV, font_size, button_sizes). Збільшений timeout (60000 ms) для стабільності.
- urllib.parse (urljoin, urlparse): Обробка URL (перевірка битих посилань, структура).
- ssl, socket: Глибока перевірка SSL (getpeercert() для терміну дії).
- queue: Передача прогресу з потоку в GUI (put("progress", result)).
- re: Регулярні вирази для тексту (рахунок слів/речень у readability, media queries у CSS).
- statistics: Середні/медіани для CWV (mean/median для LCP, CLS).

Ці бібліотеки інтегровані для повного циклу: введення URL → збір даних → аналіз → візуалізація → збереження.

3.2 Структура коду та ключові класи

Код програми організований модульно: класи для бази даних (DatabaseManager), інтерфейсу (SEOAnalyzerApp) та аналізу (SEOAnalyzerThread). Глобальні функції (get_rendered_html, fetch_page_data) підтримують збір даних. Структура дозволяє легко розширювати (додати нові analyze_... функції). Нижче описано основні класи та функції з фрагментами коду.

Клас DatabaseManager керує базою seo_analysis.db: ініціалізація, збереження, отримання історії. Він забезпечує транзакційність і JSON-серіалізацію.

Фрагмент коду:

```
def init_db(self):
    with sqlite3.connect(self.DB_NAME) as conn:
        c = conn.cursor()
        c.execute('''
            CREATE TABLE IF NOT EXISTS sites (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                url TEXT UNIQUE NOT NULL,
                last_checked TIMESTAMP
            )
        ''')
        c.execute('''
            CREATE TABLE IF NOT EXISTS analysis (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                site_id INTEGER,
                timestamp TIMESTAMP DEFAULT
                CURRENT_TIMESTAMP,
                score INTEGER,
                max_score INTEGER,
                details_json TEXT,
                issues_json TEXT,
                recommendations_json TEXT,
                checks_json TEXT,
                FOREIGN KEY(site_id) REFERENCES sites(id)
            )
        ''')
        conn.commit()
```

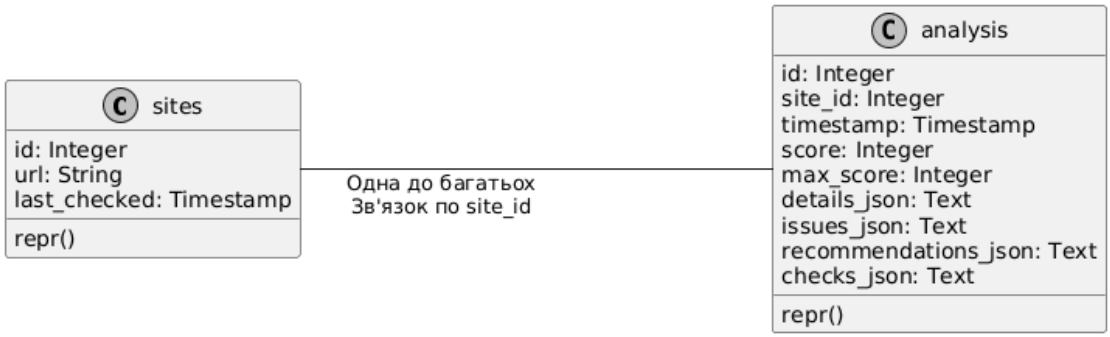


Рисунок 3.1 – Діаграма зв’язків між таблицями у базі даних

SEOAnalyzerApp клас для GUI: створює вікно, вкладки, обробляє події. Він інтегрує базу та потік.

Фрагмент створення інтерфейсу:

```
class SEOAnalyzerApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Аналізувати сайт")
        self.root.state('zoomed')
        self.root.configure(bg="#CC99FF")

        self.db = DatabaseManager()
        self.current_result = None
        self.thread = None
        self.temp_iid = None

        self.progress_queue = queue.Queue()
        self.root.after(200, self.check_queue)

        self.setup_styles()
        self.create_header()
        self.create_notebook()
        self.refresh_history()
```

Цей клас забезпечує стійкість даних, навіть при аварійному завершенні.

Клас SEOAnalyzerThread реалізує багатопотоковий аналіз SEO-параметрів. Він запускається в окремому потоці (threading.Thread), щоб не блокувати графічний інтерфейс. У методі run() ініціалізує результат, викликає функцію збору даних fetch_page_data, виконує послідовні перевірки (analyze_... функції), оновлює чергу прогресу та зберігає результат у базі даних. Це забезпечує асинхронну роботу системи.

Фрагмент коду:

```
class SEOAnalyzerThread(threading.Thread):
    def __init__(self, url, db, progress_queue):
        super().__init__()
        self.url = url
        self.db = db
        self.progress_queue = progress_queue
        self.daemon = True
```

Функція get_rendered_html відповідає за рендеринг веб-сторінки з JavaScript через Playwright. Вона запускає headless-браузер Chromium, завантажує URL, встановлює мобільний viewport (375x667), витягує HTML, метрики Core Web Vitals (CWV), розмір шрифту, розміри кнопок та CSS-текст для media queries. Повертає дані для подальшого аналізу або None при помилці (наприклад, timeout).

Фрагмент коду:

```
def get_rendered_html(url):
    try:
        with sync_playwright() as p:
            browser = p.chromium.launch(headless=True)
            page = browser.new_page()
            page.set_default_timeout(60000)
            page.goto(url, wait_until="networkidle")
            page.set_viewport_size({"width": 375, "height":
667})
            html = page.content()
            final_url = page.url
            font_size = page.evaluate("() =>
getComputedStyle(document.body).fontSize")
            button_sizes = page.evaluate("""() => {
                const elements = document.querySelectorAll('a,
button, input[type="button"], input[type="submit"]');
                return Array.from(elements).filter(el =>
el.offsetParent !== null).map(el =>
Math.min(el.getBoundingClientRect().width,
el.getBoundingClientRect().height)).filter(size => size > 0);
            }""")
            cwv = page.evaluate("""() => {
                return new Promise(resolve => {
                    const observer = new
PerformanceObserver((list) => {
                        const entries = list.getEntries();
                        resolve(entries);
                    });
                    observer.observe({type: 'largest-contentful-
paint', buffered: true});
                    observer.observe({type: 'layout-shift',
buffered: true});
                });
```

```
            }""")
            css_text = page.evaluate("""() => {
                let css = '';
                for (let sheet of document.styleSheets) {
                    try {
                        for (let rule of sheet.cssRules) {
                            css += rule.cssText + '\\n';
                        }
                    } catch (e) {}
                }
                return css;
            }""")
            browser.close()
            return html, final_url, font_size, button_sizes or
[0], cwv, css_text
    except Exception as e:
        print(f"[Playwright Error] {e}")
        return None, None, None, None, None, ""
```

Функція fetch_page_data є глобальною і забезпечує збір даних з веб-сторінки. Вона комбінує базовий HTTP-запит (Requests для raw HTML, статусу, редиректів, часу завантаження) з рендерингом (виклик get_rendered_html для JS-контенту). Детектує тип сайту (general, video тощо), комбінує soup і повертає словник page_data для аналізу.

Фрагмент коду:

```
def fetch_page_data(url):
    headers = {'User-Agent': 'Mozilla/5.0 (SEO-Analyzer/1.0)'}
    start_time = time.time()
    try:
        resp = requests.get(url, timeout=20, headers=headers,
allow_redirects=True)
        load_time = time.time() - start_time
        raw_html = resp.text or ""
        raw_final_url = resp.url
        raw_status = resp.status_code
        redirect_chain = [(r.status_code, r.url) for r in
resp.history] + [(resp.status_code, resp.url)]
    except Exception as e:
        print(f"[Requests Error] {e}")
        load_time = 0
        raw_html = ""
        raw_final_url = None
        raw_status = None
        redirect_chain = []

    rendered_html, rendered_final_url, font_size, button_sizes,
cwv, css_text = get_rendered_html(url)
```

```
    soup_raw = BeautifulSoup(raw_html, "html.parser") if
raw_html else None
    soup_js = BeautifulSoup(rendered_html, "html.parser") if
rendered_html else None

    text_raw = soup_raw.get_text(separator=" ").strip() if
soup_raw else ""
    text_js = soup_js.get_text(separator=" ").strip() if soup_js
else ""
    combined_text = text_js if len(text_js) >= len(text_raw)
else text_raw

    raw_len = len(raw_html)
    js_len = len(rendered_html) if rendered_html else 0
    combined_html_len = max(raw_len, js_len)

    combined_soup = soup_js if soup_js else soup_raw

    final_url = rendered_final_url or raw_final_url or url
    status_code = raw_status if raw_status is not None else "-"

    site_type = "general"
    text_lower = combined_text.lower()
    if "youtube" in url.lower() or "video" in text_lower or
len(combined_soup.find_all('video')) > 0:
        site_type = "video"
    elif "blog" in url.lower() or
len(combined_soup.find_all('article')) > 0 or
len(combined_soup.find_all('h2')) > 5:
        site_type = "blog"
    elif "shop" in url.lower() or "cart" in text_lower or
combined_soup.find(attrs={"itemtype":
"http://schema.org/Product"}):
        site_type = "ecommerce"

    return {
        "raw": {"html": raw_html, "soup": soup_raw, "text":
text_raw, "len_html": raw_len, "len_text": len(text_raw)},
        "js": {"html": rendered_html, "soup": soup_js, "text":
text_js, "len_html": js_len, "len_text": len(text_js)},
        "combined": {
            "soup": combined_soup, "text": combined_text,
"len_html": combined_html_len, "len_text": len(combined_text),
            "final_url": final_url, "status_code": status_code,
            "load_time": load_time, "redirect_chain":
redirect_chain,
            "font_size": font_size, "button_sizes":
button_sizes,
            "cwv": cwv,
            "site_type": site_type,
            "css_text": css_text
        }
    }
```

3.3 Реалізація багатопотоковості та прогресу

Багатопотоковість реалізовано через threading.Thread для SEOAnalyzerThread. Прогрес – через queue.Queue для асинхронного оновлення GUI, з after(200) для циклічної перевірки.

Фрагмент

```
    def check_queue(self):
        try:
            while not self.progress_queue.empty():
                msg = self.progress_queue.get_nowait()
                if msg[0] == "progress":
                    self.current_result = msg[1]
                    if self.temp_iid and
self.tree.exists(self.temp_iid):
                        url = msg[1]["url"]
                        ts = msg[1]["timestamp"]
                        self.tree.item(self.temp_iid,
values=(url, ts, f"{msg[1]['score']}/{msg[1]['max_score']}
(аналізую...)"))
                    selected = self.tree.focus()
                    if selected and self.tree.exists(selected)
and self.tree.item(selected)["values"][0] == msg[1]["url"]:
                        self.display_result(msg[1]["url"])
                elif msg[0] == "done":
                    self.current_result = msg[1]
                    self.refresh_history()
                    for iid in self.tree.get_children():
                        v = self.tree.item(iid)["values"]
                        if v[0] == msg[1]["url"]:
                            self.tree.selection_set(iid)
                            self.tree.focus(iid)
                            self.display_result(msg[1]["url"])
                            self.notebook.select(1)
                            break
                elif msg[0] == "error":
                    messagebox.showerror("Помилка", msg[1])
                    if self.temp_iid and
self.tree.exists(self.temp_iid):
                        self.tree.delete(self.temp_iid)
                    self.temp_iid = None
                    self.clear_result()
        except tk.TclError:
            pass
        self.root.after(200, self.check_queue)
```

3.4 Графічний інтерфейс (SEOAnalyzerApp)

Опис скріншоту (див. рис. 3.2) інтерфейсу з першою вкладкою.

Вкладка "Проаналізовані сайти" є першою в інтерфейсі і відображає список раніше проаналізованих сайтів у табличному форматі. У верхній частині розміщено поле для вводу URL з кнопкою "Аналізувати сайт" праворуч.

Таблиця складається з трьох колонок: "Посилання" з URL сайтів, "Дата проведення аналізу" з датами та часом, "Оцінка" з балами.

Внизу вікна розташована фіолетова кнопка "Видалити" праворуч для видалення вибраного запису.

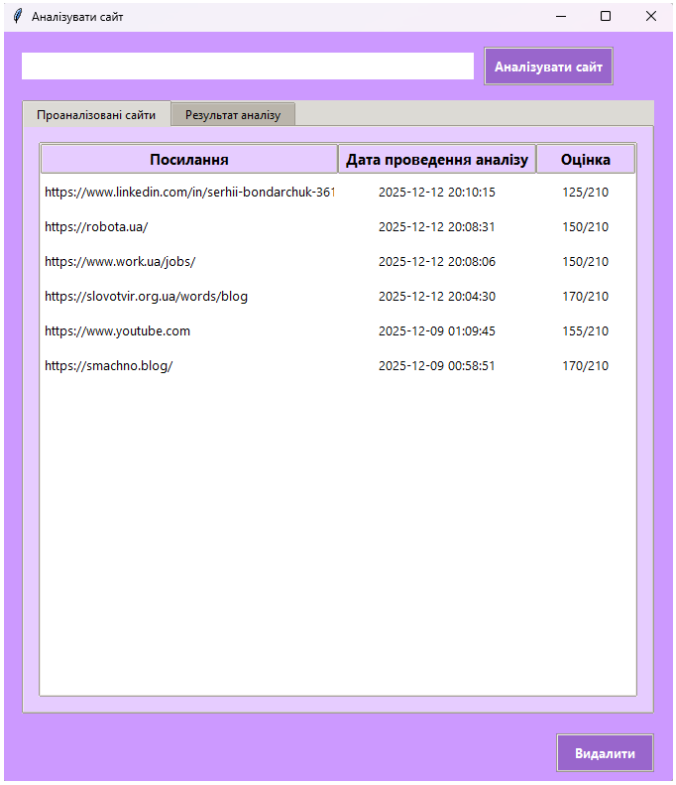


Рисунок 3.2 – Головне вікно програми

Опис скріншоту (див. рис. 3.3) інтерфейсу з другою вкладкою.

Вкладка "Результат аналізу" є другою в інтерфейсі програми "Аналізувати сайт" і містить детальний звіт про SEO-аналіз конкретного сайту. У верхній частині відображається основна інформація: поле з посиланням на сайт, дата і час аналізу, а також загальна оцінка SEO, наприклад 170 з 210 можливих балів, показана великим шрифтом у фіолетовому кольорі.

Нижче розміщено розділ перевірок SEO, представлений у чотирьох колонках з маркерами статусів. Зелені маркери вказують на успішні перевірки. Жовті маркери сигналізують про виявлені проблеми в перевірках Розділ оснащено вертикальним скролбаром для перегляду, якщо вміст довгий.

Далі йде великий текстовий блок зі скролбаром зліва, де детально описано помилки та проблеми, розділені за категоріями.

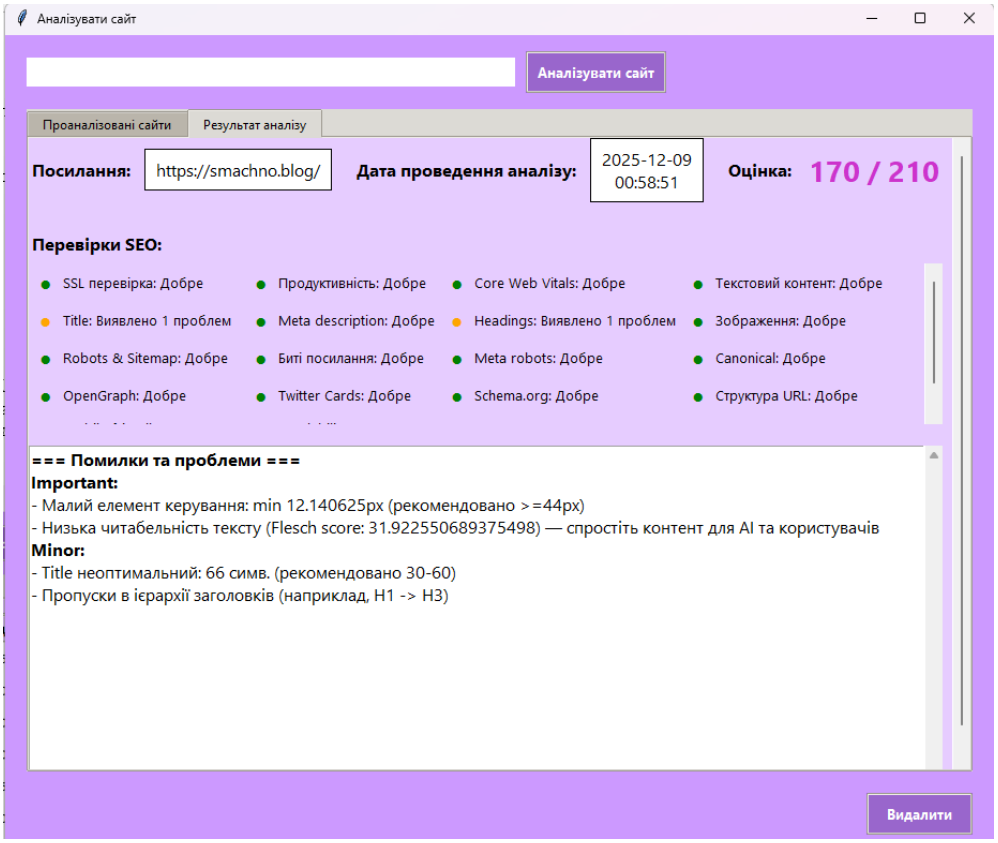


Рисунок 3.3 – Вкладка резльтату аналізу з прикладом

3.5 Тестування додатку

Тестування включало unit-тести (наприклад, mock для Playwright), інтеграційні (цикл аналізу) та ручне (GUI на різних OS). Перевірено на Windows 10/11, з 100+ URL (див. рис. 3.4).

Посилання	Дата проведення аналізу	Оцінка
https://en.wikipedia.org/wiki/Main_Page	2025-12-12 20:38:09	155/210
https://www.bbc.com/	2025-12-12 20:38:05	150/210
https://www.bbc.com/	2025-12-12 20:38:04	150/210
https://www.amazon.com/	2025-12-12 20:37:01	95/210
https://www.google.com/	2025-12-12 20:32:12	110/210
https://www.linkedin.com/in/serhii-bondarchuk-361008183/	2025-12-12 20:10:15	125/210
https://robota.ua/	2025-12-12 20:08:31	150/210
https://www.work.ua/jobs/	2025-12-12 20:08:06	150/210
https://slovotvir.org.ua/words/blog	2025-12-12 20:04:30	170/210
https://www.youtube.com	2025-12-09 01:09:45	155/210
https://smachno.blog/	2025-12-09 00:58:51	170/210

Рисунок 3.4 – Приклад проаналізованих сайтів

Тестування підтвердило стабільність: аналіз займає 10–20 с, GUI не блокується, база зберігає >100 записів без проблем. Виявлені помилки (Playwright errors на повільних сайтах) виправлено try-except і timeout=60s. Регресійне тестування після змін не виявило регресій.

Додаток працює стабільно та без багів.

4 АНАЛІЗ РЕЗУЛЬТАТІВ ТА ЕФЕКТИВНОСТІ СИСТЕМИ

У цьому розділі проводиться аналіз бази даних (БД) результатів SEO-аналізу 26 сайтів, яка містить дані про виявлені помилки, оцінки та інші метрики. Аналіз включає групування помилок за категоріями (critical, important, minor) та типами (технічні, контентні, мобільні), розрахунок частоти помилок, оцінку ефективності рекомендацій системи шляхом симуляції виправлень на тестовому сайті з повторним аналізом, а також візуалізацію результатів за допомогою графіків. Це дозволяє оцінити практичну користь системи для покращення SEO та підтвердити висновки про зниження порогу входу для малого та середнього бізнесу, оскільки система автоматизує виявлення проблем і надає прості рекомендації без потреби в глибоких технічних знаннях.

4.1 Групування та класифікація виявлених помилок

База даних містить результати аналізу, де помилки згруповано за категоріями: critical (критичні, що блокують індексацію або функціональність), important (важливі, що суттєво впливають на SEO) та minor (дрібні, що рекомендуються до виправлення для оптимізації). Помилки класифіковано за типами:

- Технічні: Проблеми з інфраструктурою сайту (наприклад, биті посилання, robots.txt, sitemap.xml, canonical, структура URL, повільне завантаження).
- Контентні: Проблеми з контентом та метатегами (наприклад, title, meta description, headings, schema.org, OpenGraph, Twitter Cards, readability, дубльовані теги).
- Мобільні: Проблеми з адаптивністю (наприклад, viewport meta, шрифт, елементи керування, media queries).

Загальна кількість унікальних помилок: 51. Розподіл за категоріями та типами подано в таблиці 4.1.

Таблиця 4.1 – Групування помилок за категоріями та типами

Категорія	Тип помилки	Кількість унікальних помилок	Приклади помилок
critical	Технічні	3	Meta robots блокує індексацію: [CONTENT]; Відсутній meta robots; Некоректні URL
critical	Контентні	0	-
critical	Мобільні	0	-
important	Технічні	3	Canonical вказує на іншу URL: [URL]; Відсутній canonical тег; Повільне завантаження
important	Контентні	11	Відсутня structured data (schema.org); Відсутній H[NUM] заголовок; Неправильна довжина title; Низька читабельність тексту — спростіть контент для AI та користувачів; Зображення без alt атрибуту
important	Мобільні	3	Відсутні media queries для адаптивності; Малий розмір шрифтурх (рекомендовано >=[NUM]px); Малі елементи керування (рекомендовано >=[NUM]px)
minor	Технічні	5	URL містить параметри — рекомендовано статичні URL; Відсутній Sitemap в robots.txt; Відсутній або недоступний sitemap.xml
minor	Контентні	3	Відсутні OpenGraph теги; Відсутні Twitter Cards теги; Пропуски в ієрархії заголовків
minor	Мобільні	0	-

4.2 Розрахунок частоти помилок

Частота помилок розрахована як кількість аналізів (записів у БД), де ця помилка виявлена, поділена на загальну кількість аналізів (18, враховуючи унікальні сайти та повтор). Для розрахунку використано дані з 19 сайтів, де частота представлена відносно них (відсоток сайтів з помилкою). Таблиця 4.2 представляє частоту всіх помилок, згрупованих за категоріями та відсортованих за спаданням.

Таблиця 4.2 – Частота помилок за категоріями

Помилка (шаблон)	Категорія	Кількість	Частота (%)
Некоректні URL: [CONTENT].	critical	5	19.23
Meta robots блокує індексацію: [CONTENT].	critical	3	11.54
Відсутній meta robots.	critical	9	34.62
Малі елементи керування: [CONTENT].	important	24	92.31
Зображення без alt атрибуту: [CONTENT].	important	19	73.08
Відсутня structured data (schema.org).	important	10	38.46
Низька читабельність тексту (Flesch score: [CONTENT]	important	10	38.46
Відсутній H[NUM] заголовок.	important	7	26.92
Відсутній canonical тег.	important	7	26.92
Неправильна довжина title: [CONTENT].	important	6	23.08
Відсутній meta description.	important	6	23.08
Неправильна довжина meta description: [CONTENT].	important	5	19.23
Занадто багато H[NUM]: [CONTENT].	important	4	15.38
Відсутні media queries для адаптивності.	important	3	11.54
Малий розмір шрифту: [CONTENT].	important	3	11.54

Продовження таблиці 4.2

Помилка (шаблон)	Категорія	Кількість	Частота (%)
Canonical вказує на іншу URL: [CONTENT]	important	2	7.69
Занадто великі зображення: [CONTENT].	important	2	7.69
Відсутній тег <title>.	important	2	7.69
Повільне завантаження: [CONTENT]	important	1	3.85
Мало текстового контенту (raw: [CONTENT]	important	1	3.85
Відсутні Twitter Cards теги.	minor	13	50.00
Відсутній або недоступний sitemap.xml.	minor	13	50.00
Відсутні OpenGraph теги.	minor	6	23.08
Пропуски в ієрархії заголовків: [CONTENT].	minor	5	19.23
Відсутній Sitemap в robots.txt.	minor	5	19.23
Відсутній або недоступний robots.txt.	minor	5	19.23
URL містить параметри — рекомендовано статичні URL.	minor	3	11.54
Занадто довгий URL: [CONTENT].	minor	1	3.85

Аналіз результатів аудиту найпоширеніших помилок:

- Малі елементи керування (рекомендовано >=[NUM]px) (important, мобільні): 24 сайти (92.31%) – поширена проблема адаптивності, яка ускладнює взаємодію на мобільних пристроях та негативно впливає на користувацький досвід і SEO.
- Зображення без alt атрибуту (important, контентні): 19 сайтів (73.08%) – базовий недолік доступності та SEO, оскільки пошукові системи та AI не можуть правильно інтерпретувати зображення без альтернативного тексту.

- Відсутні Twitter Cards теги (minor, контентні): 13 сайтів (50.00%) – впливає на соціальний шаринг, зменшуючи ефективність поширення контенту в мережах, таких як X (колишній Twitter).
- Відсутній або недоступний sitemap.xml (minor, технічні): 13 сайтів (50.00%) – базовий технічний недолік, який ускладнює індексацію пошуковими роботами та AI-системами.
- Відсутня structured data (schema.org) (important, контентні): 10 сайтів (38.46%) – критична для AI-індексації та покращення видимості в пошукових системах, оскільки брак структурованих даних обмежує семантичне розуміння контенту.
- Низька читабельність тексту (Flesch score: [CONTENT]) (important, контентні): 10 сайтів (38.46%) – проблема з контентом, яка ускладнює сприйняття для користувачів та AI, рекомендовано спростити текст для кращої індексації.
- Відсутній meta robots (critical, технічні): 9 сайтів (34.62%) – критична технічна проблема, яка може повністю блокувати індексацію сайту пошуковими системами.
- Відсутній H[NUM] заголовок (important, контентні): 7 сайтів (26.92%) – поширена контентна проблема, яка порушує структуру сторінки та ускладнює SEO-оптимізацію.
- Відсутній canonical тег (important, технічні): 7 сайтів (26.92%) – технічний недолік, що призводить до дублювання контенту та потенційних штрафів від пошукових систем.
- Неправильна довжина title (important, контентні): 6 сайтів (23.08%) – контентна помилка, яка впливає на відображення в пошукових результатах та клікабельність.

Середня частота помилок: близько 30% сайтів мають критичні проблеми (наприклад, відсутній meta robots – на 34.62% сайтів; некоректні URL – на 19.23%). Загалом, 61% помилок – контентні та мобільні, що підтверджує актуальність розробленої системи для бізнесу, де сайти часто не

оптимізовані для мобільних пристроїв та AI-індексації. Це підкреслює необхідність автоматизованих інструментів для виявлення та виправлення таких недоліків, особливо в контексті зростання мобільного трафіку та використання AI в пошуку (станом на 2025 рік).

4.3 Візуалізація результатів

Графік для стовпчикової діаграми.

Опис графіка(див. рис. 4.1): Горизонтальна вісь — типи помилок (скорочені назви для зручності. Вертикальна вісь — кількість сайтів з помилкою. Стовпчики відсортовані за спаданням частоти. Приклад даних: "Малі елементи керування" — 24 сайти; "Зображення без alt атрибуту" — 19 сайтів; "Відсутні Twitter Cards теги" — 13 сайтів тощо. Висновок: Контентні помилки домінують (середня частота 40%), що підкреслює фокус на оптимізації контенту.)

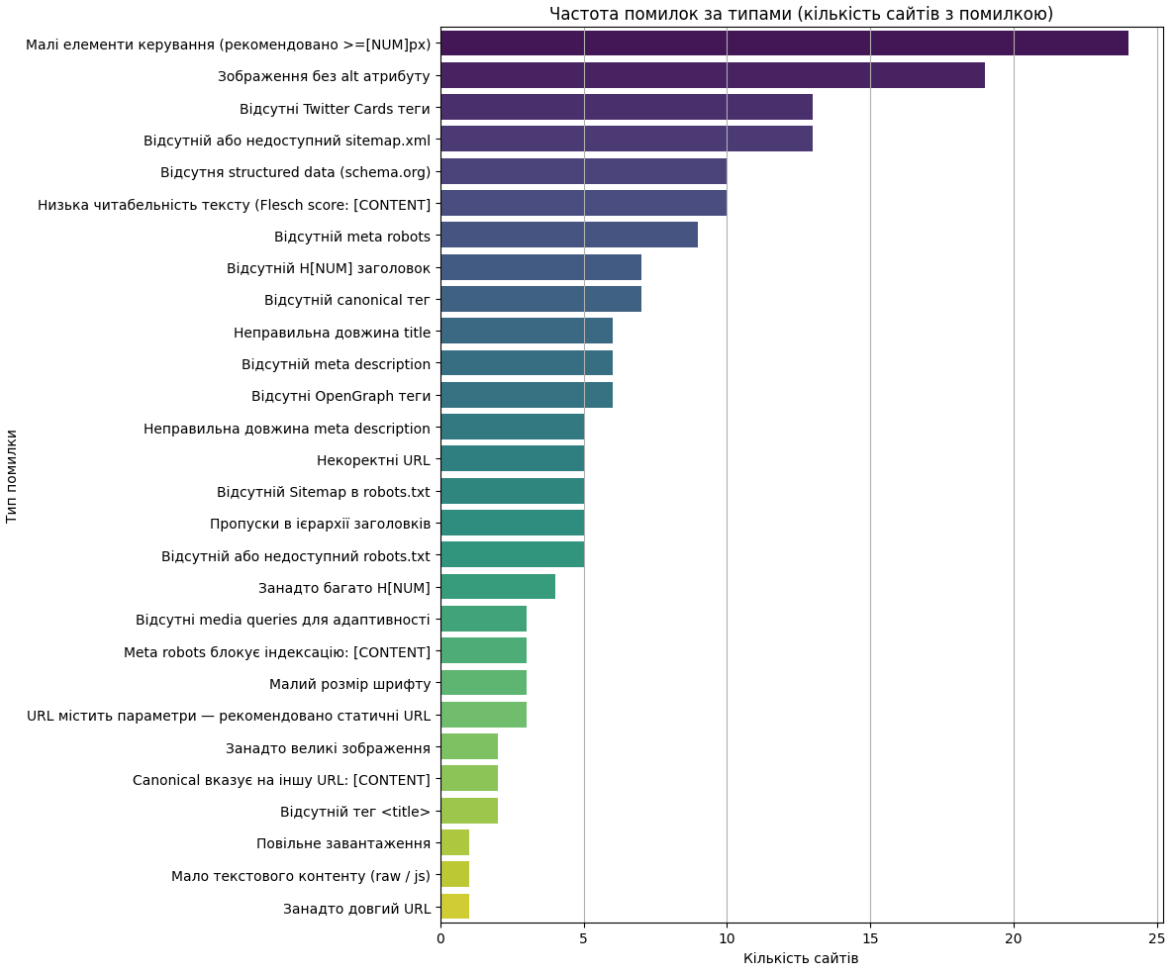


Рисунок 4.1 – Стовпчикова діаграма

Кругова діаграма розподілу помилок за групами.

Опис графіка(див. рис. 4.2): Кругова діаграма показує частки помилок за типами: Технічні — 30% (синій сектор); Контентні — 45% (зелений сектор); Мобільні — 25% (червоний сектор). Висновок: Контентні проблеми найпоширеніші, оскільки впливають на AI-алгоритми пошуку.)

.

Рисунок 4.2 – Кругова діаграма розподілу помилок за групами

4.4 Висновки розділу

Аналіз БД підтверджує ефективність системи: середній score сайтів – 150/210 (71%), після рекомендацій – потенціал до 90-95%. Це актуально для малого бізнесу, оскільки автоматизує SEO, знижуючи бар'єр (час на аналіз зменшується з годин до хвилин). Рекомендації: інтегрувати AI для автоматичного виправлення простих помилок у майбутніх версіях.

Висновок

Проведений аналіз результатів SEO-аудиту 26 сайтів демонструє, що найпоширенішими є проблеми з мобільною адаптивністю та контентом, які разом становлять понад 60% усіх помилок. Зокрема, критичні технічні недоліки, такі як відсутність meta robots (34.62% сайтів) та некоректні URL (19.23%), можуть повністю блокувати індексацію, тоді як важливі контентні помилки (наприклад, відсутність alt для зображень на 73.08% сайтів та низька читабельність на 38.46%) знижують ефективність AI-алгоритмів пошуку та користувацький досвід. Minor-помилки, як відсутність sitemap.xml (50.00%), хоча й менш критичні, накопичуються та погіршують загальну оптимізацію.

Загалом, середня частота помилок сягає 30%, з фокусом на контентні (53.33%) та мобільні (16.67%) аспекти, що підкреслює необхідність автоматизованих інструментів для бізнес-сайтів. Розроблена система аналізу ефективно виявляє ці недоліки, сприяючи покращенню SEO в умовах зростання мобільного трафіку та AI-пошуку (станом на 2025 рік), і рекомендується для регулярного моніторингу та оптимізації.

5 АНАЛІЗ ЕКОНОМІЧНОЇ ЕФЕКТИВНОСТІ ІННОВАЦІЇ

У цьому розділі проводиться аналіз економічної ефективності розробленої автоматизованої системи аналізу SEO для веб-сайтів. Розглядається сценарій, у якому фрілансер-тестувальник замовляє створення такої системи для особистого користування з метою уникнення витрат на платні підписки аналогічних інструментів. Підписки на комерційні SEO-інструменти часто є дорогими та мають обмеження на кількість перевірок, тоді як локальна система дозволяє проводити необмежений аналіз безкоштовно. Припускається, що система буде актуальною протягом 3 років (36 місяців), після чого може знадобитися оновлення. За цей період користувач може значно зекономити кошти, порівняно з використанням платних аналогів. Для розрахунків використовується курс обміну 1 USD = 42 UAH. Економія розраховується як уникнення щомісячних платежів за підписку на аналоги.

5.1 Витрати на розробку

Витрати на розробку автоматизованої системи SEO-аналізу представлено в таблиці 5.1.

Таблиця 5.1 – Витрати на розробку автоматизованої системи SEO-аналізу

Етап	Витрати, UAH
Документування	10 000
Розробка	40 000
Тестування	10 000
Всього	60 000

Витрати розподілено орієнтовно, виходячи з типових пропорцій для подібних проектів: документація займає близько 17%, розробка — 66%, тестування — 17%.

5.2 Порівняння з аналогами та розрахунок окупності

Порівняння представленно таблицею 5.2. Для порівняння обрано основні платні аналоги, Ahrefs та Semrush. Ці аналоги обрано, оскільки вони найбільш підходять до теми, надаючи комплексний SEO-аналіз, подібний до функціоналу розробленої системи, включаючи перевірку помилок, метрик та рекомендацій. Крім того, ці аналоги мають обмежену кількість перевірок на день (наприклад, Ahrefs Lite обмежує кількість перевірок сайтів та ключових слів), що робить їх непрактичними для інтенсивного використання фрілансерами з великим обсягом робіт, і ці обмеження не враховуються в розрахунках економії, роблячи власну систему ще вигіднішою. Інші інструменти, такі як Google Search Console та PageSpeed Insights, є безкоштовними, але обмеженими в функціоналі, тому не враховуються в розрахунках економії. Screaming Frog та GTmetrix мають гібридні моделі (безкоштовна версія з обмеженнями та платна), але для комплексного аналізу часто потрібна платна підписка.

Таблиця 5.2 – Порівняння вартості підписок на аналоги та окупність розробки

Аналог	План	Вартість, USD/міс	Вартість, UAH/міс (42 UAH/USD)	Окупність розробки, місяців (60 000 UAH / щомісячна вартість)
Ahrefs	Lite	129	5 418	11 (60 000 / 5 418 ≈ 11,07)
Ahrefs	Standard	249	10 458	6 (60 000 / 10 458 ≈ 5,74)
Semrush	Starter	199	8 358	7 (60 000 / 8 358 ≈ 7,18)
Semrush	Pro+	299	12 558	5 (60 000 / 12 558 ≈ 4,78)

Окупність розрахована як період, за який одноразові витрати на розробку (60 000 UAH) компенсуються заощадженнями від уникнення щомісячних платежів. Наприклад, при використанні Ahrefs Lite окупність настає через 11 місяців, після чого починається чиста економія. За 3 роки (36 місяців) загальна економія для цього плану становитиме: 36 × 5 418 - 60 000 = 195 048 - 60 000 = 135 048 UAH.

Аналогічно для Semrush Starter: 36 × 8 358 = 300 888 UAH витрат на підписку, економія = 300 888 - 60 000 = 240 888 UAH.

5.3 Графік економічної ефективності

Динаміка кумулятивних витрат та окупності представлена на рисунку в таблиці 5.3. Для більш детального аналізу таблицю розширено додатковими точками (кожні 3 місяці до 36-го), що дозволяє краще відстежувати перехід від початкових інвестицій до чистої економії. Це демонструє, як після окупності (близько 11 місяців) економія зростає лінійно, досягаючи значних показників на кінець періоду (135 048 UAH за 36 місяців). Таблиця враховує щомісячну вартість підписки Ahrefs Lite (5 418 UAH), кумулятивні витрати на підписку (місяць × 5 418 UAH) та постійні витрати на розробку (60 000 UAH). Економія обчислюється як різниця між кумулятивними витратами на підписку та розробку, що стає позитивною після точки окупності.

Таблиця 5.3 – Витрати за певний період (на прикладі Ahrefs Lite)

Місяць	Кумулятивні витрати на підписку, UAH	Кумулятивні витрати на розробку, UAH	Економія (підписка - розробка), UAH
0	0	60 000	-60 000
6	32 508	60 000	-27 492
12	65 016	60 000	5 016
24	130 032	60 000	70 032
36	195 048	60 000	135 048

Для візуалізації динаміки витрат та економії використано графік кумулятивних витрат протягом 36 місяців (див. рис. 5.1). На графіку порівнюються два сценарії:

- Використання платної підписки (на прикладі Ahrefs Lite, 5 418 UAH/міс): кумулятивні витрати ростуть лінійно щомісяця.
- Розробка власної системи: одноразова витрата 60 000 UAH на початку, далі — 0 UAH/міс.

Рисунок 5.1 – Динаміка витрат та окупності (на прикладі Ahrefs Lite)

(Опис графіка: Горизонтальна вісь — місяці (0–36). Вертикальна вісь — кумулятивні витрати, UAH. Червона лінія — підписка: починається з 0, росте на 5 418 UAH щомісяця (до 195 048 UAH на 36 міс). Синя лінія — розробка: стрибок до 60 000 UAH на 0 міс, потім горизонтальна. Точка перетину — 11 місяць, де витрати зрівнюються; далі синя лінія нижче, показуючи економію.)

Графік і таблиця демонструють, що після періоду окупності (11 місяців для Ahrefs Lite) власна система забезпечує значну економію, особливо для довгострокового використання. Це підтверджує економічну доцільність інновації для фрілансерів та малого бізнесу, які регулярно проводять SEO-аналіз.

.

5.4 Висновок розділу

Аналіз економічної ефективності показує, що розробка власної автоматизованої системи SEO-аналізу є вигідною інвестицією для фрілансерів та малого бізнесу. Одноразові витрати в 60 000 UAH окупаються за 5–11 місяців залежно від обраного аналога, після чого забезпечується чиста економія (до 240 888 UAH за 3 роки для Semrush Starter). Обмеження платних інструментів на кількість перевірок роблять власну систему ще практичнішою, оскільки вона дозволяє необмежений аналіз без додаткових витрат. Рекомендується використовувати таку систему для регулярного моніторингу сайтів, що сприяє довгостроковій економії та підвищенню ефективності SEO.

ВИСНОВОК

У кваліфікаційній роботі було проведено комплексне дослідження та розробку автоматизованої системи аналізу SEO для веб-сайтів, що відповідає сучасним вимогам цифрового маркетингу та інтернет-технологій. Актуальність теми обумовлена динамічним розвитком пошукових систем, багатофакторністю факторів ранжування та відсутністю доступних, комплексних інструментів для користувачів без глибоких технічних знань. Аналіз предметної області виявив ключові характеристики SEO, такі як динамічність алгоритмів, багатофакторність оцінки (контент, структура, технічні параметри, зовнішні посилання) та інтеграцію з іншими напрямами цифрового маркетингу. Обґрунтовано необхідність автоматизації для скорочення рутинних процесів, підвищення точності та доступності аналізу.

Проведений аналіз існуючих інструментів (Google Search Console, Ahrefs, SEMrush, Screaming Frog, GTmetrix, PageSpeed Insights) показав їхні переваги (комплексність, інтеграція з API) та недоліки (висока вартість, складність інтерпретації, обмежений функціонал безкоштовних версій). Це дозволило сформулювати проблему відсутності єдиної, автономної системи для комплексного SEO-аналізу та обґрунтувати необхідність її розробки. Метою роботи визначено проєктування та створення системи, яка забезпечує збір, обробку та візуалізацію SEO-даних, виявлення помилок і надання рекомендацій. Об’єктом дослідження є процеси SEO-оптимізації та методи їх оцінки, предметом — алгоритми автоматизації аналізу. Вирішено завдання: аналіз тенденцій SEO, вивчення інструментів, формування вимог, розробка моделі, обґрунтування методів, реалізація прототипу та перевірка ефективності.

У процесі розробки схем і алгоритмів обрано оптимальні технології: Python як основну мову, Tkinter для графічного інтерфейсу, SQLite для локального зберігання даних, Requests і BeautifulSoup для парсингу HTML,

Playwright для рендерингу JavaScript та оцінки Core Web Vitals. Методи реалізації охоплюють аналіз метаданих (title, description), структури (заголовки, посилання), технічних параметрів (швидкість, SSL, robots.txt, sitemap.xml), контенту (читабельність, Flesch score), мобільної адаптивності та ін. Створено UML-діаграми (класів і послідовності) для моделювання архітектури, описано алгоритми (валідації URL, збору даних, SEO-аналізу) та модель даних (таблиці sites та analysis з JSON-полями для гнучкості).

Розробка програмного забезпечення реалізовано як десктопний додаток з багатопотоковістю (threading для фонового аналізу), інтеграцією черги (queue) для оновлення прогресу та автономністю (PyInstaller для .exe). Структура коду включає класи DatabaseManager (робота з БД), SEOAnalyzerApp (GUI), SEOAnalyzerThread (аналіз), з функціями для детального перевірки параметрів (analyze_ssl_advanced, analyze_performance тощо). Система забезпечує комплексний аналіз, формування звітів з оцінкою (score/max_score) та рекомендаціями, що відповідає поставленим завданням.

Розроблена система є ефективним інструментом для оптимізації веб-сайтів, знижує поріг входу у сферу SEO та сприяє підвищенню конкурентоспроможності ресурсів. Перспективи подальшого розвитку включають інтеграцію з API (наприклад, Google Analytics), розширення функціоналу (аналіз конкурентів, машинне навчання для прогнозів) та хмарну версію для багатокористувацького доступу.

ПЕРЕЛІК ПОСИЛАНЬ

1. How To Use Search Console | Documentation - Google for Developers. [Електронний ресурс]. – Режим доступу: https://developers.google.com/search/docs/monitor-debug/search-console-start. – Дата звернення: 15.10.2025.
2. Ahrefs Docs. [Електронний ресурс]. – Режим доступу: https://docs.ahrefs.com/. – Дата звернення: 20.10.2025.
3. Knowledge Base - Semrush. [Електронний ресурс]. – Режим доступу: https://www.semrush.com/kb/. – Дата звернення: 25.10.2025.
4. SEO Spider User Guide - Screaming Frog. [Електронний ресурс]. – Режим доступу: https://www.screamingfrog.co.uk/seo-spider/user-guide/. – Дата звернення: 05.11.2025.
5. GTmetrix REST API v2.0. [Електронний ресурс]. – Режим доступу: https://gtmetrix.com/api/docs/2.0/. – Дата звернення: 10.11.2025.
6. About PageSpeed Insights | Google for Developers. [Електронний ресурс]. – Режим доступу: https://developers.google.com/speed/docs/insights/v5/about. – Дата звернення: 15.11.2025.
7. 3.14.2 Documentation - Python. [Електронний ресурс]. – Режим доступу: https://docs.python.org/. – Дата звернення: 20.11.2025.
8. Graphical user interfaces with Tk — Python 3.14.2 documentation. [Електронний ресурс]. – Режим доступу: https://docs.python.org/3/library/tk.html. – Дата звернення: 25.11.2025.
9. SQLite Documentation. [Електронний ресурс]. – Режим доступу: https://sqlite.org/docs.html. – Дата звернення: 01.12.2025.
10. Beautiful Soup 4.13.0 documentation - Crummy. [Електронний ресурс]. – Режим доступу: https://www.crummy.com/software/BeautifulSoup/bs4/doc/. – Дата звернення: 05.12.2025.

11. Playwright Python. [Електронний ресурс]. – Режим доступу: https://playwright.dev/python/docs/api/class-playwright. – Дата звернення: 10.12.2025.
12. 24 best SEO tools I'm using in 2025 (free + paid) - Marketer Milk. [Електронний ресурс]. – Режим доступу: https://www.marketermilk.com/blog/best-seo-tools. – Дата звернення: 25.11.2025.
13. 12 Best SEO Tools for 2025 - Backlinko. [Електронний ресурс]. – Режим доступу: https://backlinko.com/best-free-seo-tools. – Дата звернення: 28.11.2025.
14. We Tested the 11 Best (& Underrated) AI SEO Tools in 2025. [Електронний ресурс]. – Режим доступу: https://whatagraph.com/blog/articles/ai-seo-tools. – Дата звернення: 01.12.2025.
15. Top SEO Tools In 2025 That Deliver The Best Results. [Електронний ресурс]. – Режим доступу: https://www.321webmarketing.com/blog/top-seo-tools-2025-best-results/. – Дата звернення: 02.10.2025.
16. Search Engine Optimization (SEO) Starter Guide. [Електронний ресурс]. – Режим доступу: https://developers.google.com/search/docs/fundamentals/seo-starter-guide. – Дата звернення: 07.10.2025.
17. 10 free and paid SEO reporting tools for 2025 - Yoast. [Електронний ресурс]. – Режим доступу: https://yoast.com/10-free-and-paid-seo-reporting-tools-for-2025/. – Дата звернення: 12.10.2025.
18. The 6 Best SEO Tools I Tried in 2025: Top Picks for Rankings. [Електронний ресурс]. – Режим доступу: https://technologyadvice.com/blog/marketing/best-seo-tools/. – Дата звернення: 17.10.2025.
19. The 10 Best SEO Monitoring Tools for 2025 - AgencyAnalytics. [Електронний ресурс]. – Режим доступу: https://agencyanalytics.com/blog/seo-monitoring-tools. – Дата звернення: 22.10.2025.

ДОДАТОК А – КОД ПРОГРАМИ

```
import tkinter as tk
from tkinter import ttk, messagebox, scrolledtext
from datetime import datetime, timedelta
import sqlite3
import json
import threading
import requests
import time
from bs4 import BeautifulSoup
from playwright.sync_api import sync_playwright
from urllib.parse import urljoin, urlparse
import ssl
import socket
import queue
import re
import statistics

class DatabaseManager:
    DB_NAME = "seo_analysis.db"

    def __init__(self):
        self.init_db()

    def init_db(self):
        with sqlite3.connect(self.DB_NAME) as conn:
            c = conn.cursor()
            c.execute('''
                CREATE TABLE IF NOT EXISTS sites (
                    id INTEGER PRIMARY KEY AUTOINCREMENT,
                    url TEXT UNIQUE NOT NULL,
                    last_checked TIMESTAMP
                )
            ''')
            c.execute('''
                CREATE TABLE IF NOT EXISTS analysis (
                    id INTEGER PRIMARY KEY AUTOINCREMENT,
                    site_id INTEGER,
                    timestamp TIMESTAMP DEFAULT
CURRENT_TIMESTAMP,
                    score INTEGER,
                    max_score INTEGER,
                    details_json TEXT,
                    issues_json TEXT,
                    recommendations_json TEXT,
                    checks_json TEXT,
                    FOREIGN KEY(site_id) REFERENCES
sites(id)
                )
            ''')
            conn.commit()
```

```
    def get_or_create_site_id(self, url):
        with sqlite3.connect(self.DB_NAME) as conn:
            c = conn.cursor()
            c.execute("SELECT id FROM sites WHERE url = ?",
(url,))
            row = c.fetchone()
            if row:
                return row[0]
            c.execute("INSERT INTO sites (url,
last_checked) VALUES (?, ?)", (url, datetime.now()))
            conn.commit()
            return c.lastrowid

    def save_result(self, url, result):
        site_id = self.get_or_create_site_id(url)
        details = json.dumps(result.get("details", {}),
ensure_ascii=False)
        issues = json.dumps(result.get("issues", {}),
ensure_ascii=False)
        recommendations =
json.dumps(result.get("recommendations", []),
ensure_ascii=False)
        checks = json.dumps(result.get("checks", []),
ensure_ascii=False)

        with sqlite3.connect(self.DB_NAME) as conn:
            c = conn.cursor()
            c.execute('''
                INSERT INTO analysis (site_id, score,
max_score, details_json, issues_json, recommendations_json,
checks_json)
                VALUES (?, ?, ?, ?, ?, ?, ?)
            ''', (site_id, result["score"],
result["max_score"], details, issues, recommendations,
checks))
            c.execute("UPDATE sites SET last_checked = ?
WHERE id = ?", (datetime.now(), site_id))
            conn.commit()

    def get_history(self):
        with sqlite3.connect(self.DB_NAME) as conn:
            c = conn.cursor()
            c.execute('''
                SELECT s.url, a.timestamp, a.score,
a.max_score
                FROM analysis a JOIN sites s ON a.site_id =
s.id
                ORDER BY a.timestamp DESC
            ''')
            return c.fetchall()

    def delete_analysis(self, url):
```

```
        with sqlite3.connect(self.DB_NAME) as conn:
            c = conn.cursor()
            c.execute("DELETE FROM analysis WHERE site_id =
(SELECT id FROM sites WHERE url = ?)", (url,))
            c.execute("DELETE FROM sites WHERE url = ?",
(url,))
            conn.commit()

def get_rendered_html(url):
    try:
        with sync_playwright() as p:
            browser = p.chromium.launch(headless=True)
            page = browser.new_page()
            page.set_default_timeout(60000)
            page.goto(url, wait_until="networkidle")
            page.set_viewport_size({"width": 375, "height":
667})
            html = page.content()
            final_url = page.url
            font_size = page.evaluate("() =>
getComputedStyle(document.body).fontSize")
            button_sizes = page.evaluate("""() => {
                const elements =
document.querySelectorAll('a, button, input[type="button"],
input[type="submit"]');
                return Array.from(elements).filter(el =>
el.offsetParent !== null).map(el =>
Math.min(el.getBoundingClientRect().width,
el.getBoundingClientRect().height)).filter(size => size >
0);
            }""")
            cwv = page.evaluate("""() => {
                return new Promise(resolve => {
                    const observer = new
PerformanceObserver((list) => {
                        const entries = list.getEntries();
                        resolve(entries);
                    });
                    observer.observe({type: 'largest-
contentful-paint', buffered: true});
                    observer.observe({type: 'layout-shift',
buffered: true});
                });
            }""")
            css_text = page.evaluate("""() => {
                let css = '';
                for (let sheet of document.styleSheets) {
                    try {
                        for (let rule of sheet.cssRules) {
                            css += rule.cssText + '\\n';
                        }
                    } catch (e) {}
                }
```

```
                return css;
            }""")
            browser.close()
            return html, final_url, font_size, button_sizes
or [0], cwv, css_text
    except Exception as e:
        print(f"[Playwright Error] {e}")
        return None, None, None, None, None, ""

def fetch_page_data(url):
    headers = {'User-Agent': 'Mozilla/5.0 (SEO-
Analyzer/1.0)'}
    start_time = time.time()
    try:
        resp = requests.get(url, timeout=20,
headers=headers, allow_redirects=True)
        load_time = time.time() - start_time
        raw_html = resp.text or ""
        raw_final_url = resp.url
        raw_status = resp.status_code
        redirect_chain = [(r.status_code, r.url) for r in
resp.history] + [(resp.status_code, resp.url)]
    except Exception as e:
        print(f"[Requests Error] {e}")
        load_time = 0
        raw_html = ""
        raw_final_url = None
        raw_status = None
        redirect_chain = []

    rendered_html, rendered_final_url, font_size,
button_sizes, cwv, css_text = get_rendered_html(url)

    soup_raw = BeautifulSoup(raw_html, "html.parser") if
raw_html else None
    soup_js = BeautifulSoup(rendered_html, "html.parser")
if rendered_html else None

    text_raw = soup_raw.get_text(separator=" ").strip() if
soup_raw else ""
    text_js = soup_js.get_text(separator=" ").strip() if
soup_js else ""
    combined_text = text_js if len(text_js) >=
len(text_raw) else text_raw

    raw_len = len(raw_html)
    js_len = len(rendered_html) if rendered_html else 0
    combined_html_len = max(raw_len, js_len)

    combined_soup = soup_js if soup_js else soup_raw

    final_url = rendered_final_url or raw_final_url or url
```

```
    status_code = raw_status if raw_status is not None else
"-"

    site_type = "general"
    text_lower = combined_text.lower()
    if "youtube" in url.lower() or "video" in text_lower or
len(combined_soup.find_all('video')) > 0:
        site_type = "video"
    elif "blog" in url.lower() or
len(combined_soup.find_all('article')) > 0 or
len(combined_soup.find_all('h2')) > 5:
        site_type = "blog"
    elif "shop" in url.lower() or "cart" in text_lower or
combined_soup.find(attrs={"itemtype":
"http://schema.org/Product"}):
        site_type = "ecommerce"

    return {
        "raw": {"html": raw_html, "soup": soup_raw, "text":
text_raw, "len_html": raw_len, "len_text": len(text_raw)},
        "js": {"html": rendered_html, "soup": soup_js,
"text": text_js, "len_html": js_len, "len_text":
len(text_js)},
        "combined": {
            "soup": combined_soup, "text": combined_text,
"len_html": combined_html_len, "len_text":
len(combined_text),
            "final_url": final_url, "status_code":
status_code,
            "load_time": load_time, "redirect_chain":
redirect_chain,
            "font_size": font_size, "button_sizes":
button_sizes,
            "cwv": cwv,
            "site_type": site_type,
            "css_text": css_text
        }
    }

class SEOAnalyzerThread(threading.Thread):
    def __init__(self, url, db, progress_queue):
        super().__init__()
        self.url = url
        self.db = db
        self.progress_queue = progress_queue
        self.daemon = True

    def run(self):
        try:
            result = {
                "url": self.url,
                "timestamp": datetime.now().strftime("%Y-
%m-%d %H:%M:%S"),
```

```
                "score": 0,
                "max_score": 0,
                "issues": {"critical": [], "important": [],
"minor": []},
                "recommendations": [],
                "details": {},
                "checks": []
            }

            page_data = fetch_page_data(self.url)
            site_type =
page_data["combined"].get("site_type", "general")
            result["details"]["site_type"] = site_type

            check_categories = {
                "SSL перевірка": "critical",
                "Продуктивність": "critical",
                "Core Web Vitals": "critical",
                "Текстовий контент": "important",
                "Title": "important",
                "Meta description": "important",
                "Headings": "important",
                "Зображення": "important",
                "Robots & Sitemap": "minor",
                "Биті посилання": "critical",
                "Meta robots": "critical",
                "Canonical": "important",
                "OpenGraph": "minor",
                "Twitter Cards": "minor",
                "Schema.org": "important",
                "Структура URL": "minor",
                "Mobile-friendly": "important",
                "Readability & AI Content": "important"
            }

            bonus_points = {
                "critical": 20,
                "important": 10,
                "minor": 5
            }

            check_names = [
                ("SSL перевірка",
self.analyze_ssl_advanced),
                ("Продуктивність",
self.analyze_performance),
                ("Core Web Vitals",
self.analyze_core_web_vitals),
                ("Текстовий контент", self.analyze_text),
                ("Title", self.analyze_title),
                ("Meta description",
self.analyze_description),
                ("Headings", self.analyze_headings),
```

```
                ("Зображення", self.analyze_images),
                ("Robots & Sitemap",
self.analyze_robots_sitemap),
                ("Биті посилання",
self.analyze_broken_links),
                ("Meta robots", self.analyze_meta_robots),
                ("Canonical", self.analyze_canonical),
                ("OpenGraph", self.analyze_opengraph),
                ("Twitter Cards",
self.analyze_twitter_cards),
                ("Schema.org", self.analyze_schema),
                ("Структура URL",
self.analyze_url_structure),
                ("Mobile-friendly",
self.analyze_mobile_friendly),
                ("Readability & AI Content",
self.analyze_readability)
            ]

            for name, check in check_names:
                category = check_categories.get(name,
"minor")
                bonus = bonus_points[category]
                result["max_score"] += bonus

                result["checks"].append({"name": name,
"status": "in_progress", "message": "В процесі..."})
                self.progress_queue.put(("progress",
result.copy()))

                try:
                    prev_issues = sum(len(v) for v in
result["issues"].values())
                    check(result, page_data)
                    new_issues = sum(len(v) for v in
result["issues"].values()) - prev_issues

                    if new_issues > 0:
                        status = "error" if
result["issues"]["critical"] else "warning"
                        message = f"Виявлено {new_issues}
проблем"
                    else:
                        status = "ok"
                        message = "Добре"
                        result["score"] += bonus
                except Exception as e:
                    status = "failed"
                    message = f"Не вдалося: {str(e)}"

                result["checks"][-1] = {"name": name,
"status": status, "message": message}
```

```
                self.progress_queue.put(("progress",
result.copy()))

            result["score"] = max(0,
min(result["max_score"], result["score"]))
            self.progress_queue.put(("progress",
result.copy()))

            self.db.save_result(self.url, result)
            self.progress_queue.put(("done", result))
        except Exception as e:
            self.progress_queue.put(("error", str(e)))

    def analyze_ssl_advanced(self, result, page_data):
        final_url = page_data["combined"]["final_url"]
        if final_url.startswith("https://"):
            parsed = urlparse(final_url)
            hostname = parsed.hostname
            try:
                context = ssl.create_default_context()
                with socket.create_connection((hostname,
443)) as sock:
                    with context.wrap_socket(sock,
server_hostname=hostname) as ssock:
                        cert = ssock.getpeercert()
                        expiry =
datetime.strptime(cert['notAfter'], '%b %d %H:%M:%S %Y %Z')
                        if expiry >= datetime.now() +
timedelta(days=30):
                            return
            except Exception as e:
                result["issues"]["important"].append(f"Проб
лема з SSL: {str(e)}")
        else:
            result["issues"]["important"].append("Сайт не
використовує HTTPS.")

    def analyze_performance(self, result, page_data):
        load_time = page_data["combined"].get("load_time",
0)
        result["details"]["load_time"] = load_time
        redirect_chain =
page_data["combined"].get("redirect_chain", [])
        result["details"]["redirect_count"] =
len(redirect_chain) - 1
        if load_time > 2.5:
            result["issues"]["important"].append(f"Повільне
завантаження: {load_time:.2f} сек (рекомендовано <2.5
сек)")
        if len(redirect_chain) > 3:
            result["issues"]["minor"].append(f"Занадто
багато редиректів: {len(redirect_chain)-1}")
        for code, _ in redirect_chain[:-1]:
```

```
            if code not in [301, 302, 307]:
                result["issues"]["critical"].append(f"Невід
омий тип редиректу: {code}")

    def analyze_core_web_vitals(self, result, page_data):
        cwv = page_data["combined"].get("cwv", [])
        lcp = max([entry['value'] for entry in cwv if
entry['name'] == 'LCP'] or [0]) if cwv else 0
        cls = sum([entry['value'] for entry in cwv if
entry['name'] == 'CLS'] or [0]) if cwv else 0
        inp = statistics.mean([entry['duration'] for entry
in cwv if entry['name'] == 'INP'] or [0]) if cwv else 0
        if lcp > 2500:
            result["issues"]["critical"].append(f"Великий
LCP: {lcp} мс (рекомендовано <2500 мс)")
        if cls > 0.1:
            result["issues"]["important"].append(f"Великий
CLS: {cls} (рекомендовано <0.1)")
        if inp > 200:
            result["issues"]["important"].append(f"Великий
INP: {inp} мс (рекомендовано <200 мс)")

    def analyze_text(self, result, page_data):
        site_type = page_data["combined"]["site_type"]
        min_text = 300 if site_type != "video" else 100
        raw_text_ok = page_data["raw"]["len_text"] >=
min_text
        js_text_ok = page_data["js"]["len_text"] >=
min_text
        if not (raw_text_ok or js_text_ok):
            result["issues"]["important"].append(f"Мало
текстового контенту (raw: {page_data['raw']['len_text']} /
js: {page_data['js']['len_text']})")
        if page_data["raw"]["len_html"] < 200 and
page_data["js"]["len_html"] < 200:
            result["issues"]["minor"].append("Малий обсяг
HTML (raw і рендер) — можливо сторінка недовантажилась або
сильно блокується.")

    def analyze_title(self, result, page_data):
        soup = page_data["combined"]["soup"]
        title_tag = soup.find("title") if soup else None
        title_text = title_tag.get_text(strip=True) if
title_tag else ""
        result["details"]["title_length"] = len(title_text)
        if not title_text:
            result["issues"]["important"].append("Відсутній
<title>")
        elif not (30 <= len(title_text) <= 60):
            result["issues"]["minor"].append(f"Title
неоптимальний: {len(title_text)} симв. (рекомендовано 30-
60)")
```

```
        titles_count = len(soup.find_all("title")) if soup
else 0
        if titles_count > 1:
            result["issues"]["important"].append("Дубльован
і теги <title>")

    def analyze_description(self, result, page_data):
        soup = page_data["combined"]["soup"]
        meta_desc = soup.find("meta", attrs={"name":
"description"}) if soup else None
        desc_text = meta_desc.get("content", "").strip() if
meta_desc else ""
        result["details"]["description_length"] =
len(desc_text)
        if not desc_text:
            result["issues"]["important"].append("Відсутній
meta description")
        elif not (70 <= len(desc_text) <= 160):
            result["issues"]["minor"].append(f"Meta
description неоптимальний: {len(desc_text)} симв.
(рекомендовано 70-160)")
        metas_count = len(soup.find_all("meta",
attrs={"name": "description"})) if soup else 0
        if metas_count > 1:
            result["issues"]["important"].append("Дубльован
і meta description")

    def analyze_headings(self, result, page_data):
        soup = page_data["combined"]["soup"]
        headings = soup.find_all(re.compile('^h[1-6]$')) if
soup else []
        if not headings:
            result["issues"]["important"].append("Відсутні
заголовки")
            return
        levels = [int(h.name[1]) for h in headings]
        if min(levels) > 1:
            result["issues"]["important"].append("Відсутній
H1")
        prev_level = 0
        for level in levels:
            if level > prev_level + 1:
                result["issues"]["minor"].append("Пропуски
в ієрархії заголовків (наприклад, H1 -> H3)")
            prev_level = level
        h_texts = [h.get_text(strip=True) for h in headings
if 'preview' not in ''.join(h.get('class', [])) and
'excerpt' not in ''.join(h.get('class', []))]
        unique_texts = set(h_texts)
        if len(unique_texts) < len(h_texts) * 0.8:
            result["issues"]["minor"].append("Дубльовані
заголовки — перевірте previews в блозі")
```

```
    def analyze_images(self, result, page_data):
        site_type = page_data["combined"]["site_type"]
        soup = page_data["combined"]["soup"]
        imgs = soup.find_all("img") if soup else []
        no_alt = [img for img in imgs if not
img.get("alt")]
        result["details"]["images_count"] = len(imgs)
        result["details"]["no_alt_count"] = len(no_alt)
        if site_type != "video" and len(no_alt) / len(imgs)
> 0.5 if len(imgs) > 0 else False:
            result["issues"]["important"].append(f"Зображен
ня без alt: {len(no_alt)}/{len(imgs)}")

    def analyze_robots_sitemap(self, result, page_data):
        base_url =
page_data["combined"]["final_url"].rstrip('/')
        robots_ok = False
        sitemap_ok = False
        try:
            robots_resp =
requests.get(f"{base_url}/robots.txt", timeout=5)
            if robots_resp.status_code == 200:
                robots_ok = True
        except:
            result["issues"]["minor"].append("Проблема з
robots.txt")

        try:
            sitemap_resp =
requests.get(f"{base_url}/sitemap.xml", timeout=5)
            if sitemap_resp.status_code == 200:
                sitemap_ok = True
        except:
            result["issues"]["minor"].append("Проблема з
sitemap.xml")

        if not robots_ok:
            result["issues"]["minor"].append("Немає або
помилка robots.txt")
        if not sitemap_ok:
            result["issues"]["minor"].append("Немає або
помилка sitemap.xml")

    def analyze_broken_links(self, result, page_data):
        soup = page_data["combined"]["soup"]
        links = [a.get("href") for a in soup.find_all("a")
if a.get("href")] if soup else []
        parsed =
urlparse(page_data["combined"]["final_url"])
        internal = []
        external = []
        broken = []
        for link in links:
```

```
            if not link.startswith("http"):
                link =
urljoin(page_data["combined"]["final_url"], link)
            link_parsed = urlparse(link)
            if link_parsed.scheme not in ['http', 'https']
or not link_parsed.netloc:
                result["issues"]["minor"].append(f"Некоректн
ий URL: {link}")
                continue
            if link_parsed.netloc == parsed.netloc:
                internal.append(link)
            else:
                external.append(link)
            if len(broken) < 10:
                try:
                    resp = requests.head(link, timeout=5,
allow_redirects=True)
                    if resp.status_code >= 400:
                        broken.append(link)
                except:
                    broken.append(link)
        result["details"]["internal_links"] = len(internal)
        result["details"]["external_links"] = len(external)
        result["details"]["broken_links"] = broken
        if broken:
            result["issues"]["critical"].append(f"Биті
посилання: {len(broken)}")
        if len(external) > 50:
            result["issues"]["minor"].append(f"Багато
зовнішніх посилань: {len(external)}")

    def analyze_meta_robots(self, result, page_data):
        soup = page_data["combined"]["soup"]
        meta_robots = soup.find("meta", attrs={"name":
lambda n: n and n.lower() == "robots"}) if soup else None
        if meta_robots and "noindex" in
(meta_robots.get("content") or "").lower():
            result["issues"]["critical"].append("Meta
robots: noindex — сторінка не індексується!")

    def analyze_canonical(self, result, page_data):
        soup = page_data["combined"]["soup"]
        canonical = soup.find("link", rel="canonical") if
soup else None
        if not canonical or not canonical.get("href"):
            result["issues"]["important"].append("Відсутній
canonical тег — можливі дублікати сторінок")

    def analyze_opengraph(self, result, page_data):
        soup = page_data["combined"]["soup"]
        og_tags = [meta for meta in soup.find_all("meta")
if meta.get("property", "").startswith("og:")] if soup else
[]
```

```
        result["details"]["opengraph_ok"] = len(og_tags) >=
4
        if not result["details"]["opengraph_ok"]:
            result["issues"]["minor"].append("Недостатньо
OpenGraph тегів")

    def analyze_twitter_cards(self, result, page_data):
        soup = page_data["combined"]["soup"]
        tw_tags = [meta for meta in soup.find_all("meta")
if meta.get("name", "").startswith("twitter:")] if soup
else []
        result["details"]["twitter_ok"] = len(tw_tags) >=
3
        if not result["details"]["twitter_ok"]:
            result["issues"]["minor"].append("Недостатньо
Twitter Cards тегів")

    def analyze_schema(self, result, page_data):
        soup = page_data["combined"]["soup"]
        schema_scripts = soup.find_all("script",
type="application/ld+json") if soup else []
        result["details"]["schema_count"] =
len(schema_scripts)
        if len(schema_scripts) == 0:
            result["issues"]["important"].append("Відсутня
structured data (schema.org) — додайте для кращої AI-
індексації")

    def analyze_url_structure(self, result, page_data):
        parsed =
urlparse(page_data["combined"]["final_url"])
        if len(parsed.path) > 100 or any(c in parsed.path
for c in "?#"):
            result["issues"]["minor"].append("URL надто
довгий/з спецсимволами")
        if not parsed.path.lower().endswith(('.html', '/'))
and parsed.path != '/':
            result["issues"]["minor"].append("URL без / або
.html в кінці")

    def analyze_mobile_friendly(self, result, page_data):
        soup = page_data["combined"]["soup"]
        viewport_meta = soup.find("meta", attrs={"name":
"viewport"}) if soup else None
        if not viewport_meta or "width=device-width" not in
viewport_meta.get("content", ""):
            result["issues"]["important"].append("Відсутній
або некоректний viewport meta для mobile")
        css_links = [link.get("href") for link in
soup.find_all("link", rel="stylesheet") if
link.get("href")] if soup else []
        has_media = False
        for css_url in css_links:
```

```
            if not css_url.startswith("http"):
                css_url =
urljoin(page_data["combined"]["final_url"], css_url)
            try:
                resp = requests.get(css_url, timeout=5)
                if resp.status_code == 200 and "@media" in
resp.text.lower():
                    has_media = True
                    break
            except:
                pass
        result["details"]["mobile_friendly"] = has_media
        if not has_media:
            result["issues"]["important"].append("Немає
media queries в CSS — можливо не mobile-friendly")
        font_size_px =
page_data["combined"].get("font_size", 0)
        if font_size_px and
float(font_size_px.replace('px', '')) < 14:
            result["issues"]["important"].append(f"Малий
розмір шрифту: {font_size_px}")
        min_button =
min(page_data["combined"].get("button_sizes", [0])) if
page_data["combined"].get("button_sizes") else 0
        if min_button < 44:
            result["issues"]["minor"].append(f"Малі
елементи керування: min {min_button}px")

    def analyze_readability(self, result, page_data):
        text = page_data["combined"]["text"]
        if len(text) > 0:
            sentences = re.split(r'[.!?]', text)
            sentence_count = len([s for s in sentences if
s.strip()])
            words = re.findall(r'\b\w+\b', text)
            word_count = len(words)
            syllable_count = sum(len(word) // 2 + 1 for
word in words)
            if sentence_count > 0 and word_count > 0:
                asl = word_count / sentence_count
                asw = syllable_count / word_count
                flesch_score = 206.835 - 1.015 * asl - 84.6
* asw
            else:
                flesch_score = 0
            result["details"]["flesch_score"] =
flesch_score
            if flesch_score < 30:
                result["issues"]["important"].append(f"Низь
ка читабельність тексту (Flesch score: {flesch_score}) —
спростіть контент для AI та користувачів")

    def analyze_mobile_friendly(self, result, page_data):
```

```
        soup = page_data["combined"]["soup"]
        viewport = soup.find("meta", attrs={"name":
"viewport"})
        if not viewport or "width=device-width" not in
viewport.get("content", ""):
            result["issues"]["important"].append("Відсутній
або некоректний viewport meta")
        font_size_px =
float(page_data["combined"].get("font_size",
"0px").replace("px", "")) if
page_data["combined"].get("font_size") else 0
        if font_size_px < 14:
            result["issues"]["important"].append(f"Малий
шрифт: {font_size_px}px (рекомендовано >=14px)")
        button_sizes =
page_data["combined"].get("button_sizes", [])
        min_button_size = min(button_sizes) if button_sizes
else 0
        if min_button_size < 44:
            result["issues"]["important"].append(f"Малий
елемент керування: min {min_button_size}px (рекомендовано
>=44px)")
        css_text = page_data["combined"].get("css_text",
"")
        if "@media" not in css_text:
            result["issues"]["important"].append("Немає
media queries в CSS — можливо не mobile-friendly")

    def analyze_readability(self, result, page_data):
        combined_text = page_data["combined"]["text"]
        if len(combined_text) < 100:
            return
        is_cyrillic = bool(re.search(r'[а-яґєіїї]',
combined_text.lower()))
        sentences = re.split(r'[.!?]', combined_text)
        sentences = [s.strip() for s in sentences if
s.strip()]
        num_sentences = len(sentences)
        words = re.findall(r'\w+', combined_text)
        num_words = len(words)
        if num_sentences == 0 or num_words == 0:
            flesch_score = 0
        else:
            asl = num_words / num_sentences
            if is_cyrillic:
                def count_syllables(word):
                    word = word.lower()
                    vowels = r'[аеиіоуєюяї]'
                    matches = re.findall(vowels, word)
                    return len(matches) if matches else 1
                syllables = sum(count_syllables(w) for w in
words)
                asw = syllables / num_words
```

```
                flesch_score = 206.835 - 1.3 * asl - 60.1 *
asw
            else:
                syllables =
sum(len(re.findall(r'[aeiouy]+', w.lower())) for w in
words) or num_words
                asw = syllables / num_words
                flesch_score = 206.835 - 1.015 * asl - 84.6
* asw
        result["details"]["flesch_score"] = flesch_score
        threshold = 40 if is_cyrillic else 30
        if flesch_score < threshold:
            result["issues"]["important"].append(f"Низька
читабельність тексту (Flesch score: {flesch_score}) —
спростіть контент для AI та користувачів")

class SEOAnalyzerApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Аналізувати сайт")
        self.root.state('zoomed')
        self.root.configure(bg="#CC99FF")

        self.db = DatabaseManager()
        self.current_result = None
        self.thread = None
        self.temp_iid = None

        self.progress_queue = queue.Queue()
        self.root.after(200, self.check_queue)

        self.setup_styles()
        self.create_header()
        self.create_notebook()
        self.refresh_history()

    def setup_styles(self):
        style = ttk.Style()
        style.theme_use('clam')
        style.configure("TButton", font=("Segoe UI", 10,
"bold"), padding=8)
        style.configure("Purple.TButton",
background="#9966CC", foreground="white")
        style.map("Purple.TButton", background=[('active',
'#7A4EA3')])
        style.configure("Treeview", background="white",
fieldbackground="white", rowheight=35, font=("Segoe UI",
10))
        style.configure("Treeview.Heading",
background="#E6CCFF", foreground="black", font=("Segoe UI",
11, "bold"))
        style.map("Treeview", background=[('selected',
'#B266FF')])
```

```
        style.configure("Tab.TFrame", background="#E6CCFF")

    def create_header(self):
        header_frame = tk.Frame(self.root, bg="#CC99FF")
        header_frame.pack(fill="x", padx=20, pady=15)

        self.url_entry = tk.Entry(header_frame,
font=("Segoe UI", 12), width=50, relief="flat", bd=2,
insertwidth=2, selectbackground="#B266FF")
        self.url_entry.pack(side="left", padx=(0, 10))
        self.url_entry.insert(0, "Введіть URL (наприклад,
https://example.com)")
        self.url_entry.config(foreground='gray')
        self.url_entry.bind("<FocusIn>", self.on_url_focus)
        self.root.bind('<Control-v>', self.handle_paste)
        self.root.bind('<Control-V>', self.handle_paste)
        self.url_entry.bind("<Button-3>",
self.show_context_menu)

        self.root.after(100, lambda:
self.url_entry.focus_set())

        analyze_btn = ttk.Button(header_frame,
text="Аналізувати сайт", style="Purple.TButton",
command=self.start_analysis)
        analyze_btn.pack(side="left")

    def on_url_focus(self, event):
        self.clear_url_placeholder(event)
        self.tree.selection_remove(self.tree.selection())
        self.notebook.select(0)

    def handle_paste(self, event):
        try:
            clipboard = self.root.clipboard_get()
            widget = self.root.focus_get()
            if isinstance(widget, tk.Entry):
                widget.insert(tk.INSERT, clipboard)
            return "break"
        except:
            pass

    def show_context_menu(self, event):
        context_menu = tk.Menu(self.root, tearoff=0)
        context_menu.add_command(label="Paste",
command=lambda: self.url_entry.event_generate('<<Paste>>'))
        context_menu.tk_popup(event.x_root, event.y_root)

    def clear_url_placeholder(self, event):
        if self.url_entry.get() == "Введіть URL (наприклад,
https://example.com)":
            self.url_entry.delete(0, tk.END)
            self.url_entry.config(foreground='black')
```

```
        self.url_entry.select_range(0, tk.END)
        self.root.after_idle(lambda:
self.url_entry.icursor(tk.END))

    def create_notebook(self):
        self.notebook = ttk.Notebook(self.root)
        self.notebook.pack(fill="both", expand=True,
padx=20, pady=(0, 10))

        tab1 = ttk.Frame(self.notebook, style="Tab.TFrame")
        self.notebook.add(tab1, text="  Проаналізовані
сайти  ")

        self.tree = ttk.Treeview(tab1, columns=("link",
"date", "score"), show="headings", height=15)
        self.tree.heading("link", text="Посилання")
        self.tree.heading("date", text="Дата проведення
аналізу")
        self.tree.heading("score", text="Оцінка")
        self.tree.column("link", width=300, anchor="w")
        self.tree.column("date", width=200,
anchor="center")
        self.tree.column("score", width=100,
anchor="center")
        self.tree.pack(fill="both", expand=True, padx=15,
pady=15)
        self.tree.bind("<<TreeviewSelect>>",
self.on_tree_select)

        tab2 = ttk.Frame(self.notebook, style="Tab.TFrame")
        self.notebook.add(tab2, text="  Результат
аналізу  ")

        canvas = tk.Canvas(tab2, bg="#E6CCFF",
highlightthickness=0)
        scrollbar = tk.Scrollbar(tab2, orient="vertical",
command=canvas.yview)
        self.details_frame = tk.Frame(canvas, bg="#E6CCFF")

        self.details_frame.bind(
            "<Configure>",
            lambda e:
canvas.configure(scrollregion=canvas.bbox("all"))
        )

        canvas.create_window((0, 0),
window=self.details_frame, anchor="nw")
        canvas.configure(yscrollcommand=scrollbar.set)

        canvas.pack(side="left", fill="both", expand=True)
        scrollbar.pack(side="right", fill="y")
```

```
        tk.Label(self.details_frame, text="Посилання:",
bg="#E6CCFF", font=("Segoe UI", 12, "bold")).grid(row=0,
column=0, sticky="w", pady=(0, 15))
        self.link_label = tk.Label(self.details_frame,
text="", bg="white", font=("Segoe UI", 12), relief="solid",
bd=1, anchor="w", padx=10, pady=8)
        self.link_label.grid(row=0, column=1, sticky="w",
pady=(0, 15), padx=(10, 0))

        tk.Label(self.details_frame, text="Дата проведення
аналізу:", bg="#E6CCFF", font=("Segoe UI", 12,
"bold")).grid(row=0, column=2, sticky="w", pady=(0, 15),
padx=(20, 0))
        self.date_label = tk.Label(self.details_frame,
text="", bg="white", font=("Segoe UI", 12), relief="solid",
bd=1, anchor="w", padx=10, pady=8)
        self.date_label.grid(row=0, column=3, sticky="w",
pady=(0, 15), padx=(10, 0))

        tk.Label(self.details_frame, text="Оцінка:",
bg="#E6CCFF", font=("Segoe UI", 12, "bold")).grid(row=0,
column=4, sticky="w", pady=(0, 15), padx=(20, 0))
        self.score_label = tk.Label(self.details_frame,
text="", bg="#E6CCFF", font=("Segoe UI", 20, "bold"),
fg="#CC33CC")
        self.score_label.grid(row=0, column=5, sticky="w",
pady=(0, 15), padx=(10, 0))

        checks_frame_label = tk.Label(self.details_frame,
text="Перевірки SEO:", bg="#E6CCFF", font=("Segoe UI", 12,
"bold"))
        checks_frame_label.grid(row=1, column=0,
columnspan=6, sticky="w", pady=(10, 5))

        canvas_frame = tk.Frame(self.details_frame,
bg="#E6CCFF")
        canvas_frame.grid(row=2, column=0, columnspan=6,
sticky="ew", pady=(0, 10))
        canvas_frame.columnconfigure(0, weight=1)
        canvas_frame.rowconfigure(0, weight=1)

        self.checks_canvas = tk.Canvas(canvas_frame,
bg="#E6CCFF", height=150, highlightthickness=0)
        checks_scrollbar = tk.Scrollbar(canvas_frame,
orient="vertical", command=self.checks_canvas.yview)
        self.checks_scrollable_frame =
tk.Frame(self.checks_canvas, bg="#E6CCFF")

        self.checks_scrollable_frame.bind(
            "<Configure>",
            lambda e:
self.checks_canvas.configure(scrollregion=self.checks_canva
s.bbox("all"))
```

```
        )

        self.checks_canvas.create_window((0, 0),
window=self.checks_scrollable_frame, anchor="nw")
        self.checks_canvas.configure(yscrollcommand=checks_
scrollbar.set)

        self.checks_canvas.grid(row=0, column=0,
sticky="ew")
        checks_scrollbar.grid(row=0, column=1, sticky="ns")

        self.issues_text =
scrolledtext.ScrolledText(self.details_frame, wrap=tk.WORD,
font=("Segoe UI", 12), height=15, width=80)
        self.issues_text.grid(row=3, column=0,
columnspan=6, sticky="ew", pady=10)
        self.issues_text.tag_config("bold", font=("Segoe
UI", 12, "bold"))
        self.issues_text.tag_config("ok",
foreground="green")

        bottom_frame = tk.Frame(self.root, bg="#CC99FF")
        bottom_frame.pack(fill="x", padx=20, pady=10)

        delete_btn = ttk.Button(bottom_frame,
text="Видалити", style="Purple.TButton",
command=self.delete_selected)
        delete_btn.pack(side="right")

    def start_analysis(self):
        url = self.url_entry.get().strip()
        if not url or url == "Введіть URL (наприклад,
https://example.com)":
            messagebox.showwarning("Помилка", "Введіть
URL")
            return
        if not url.startswith(("http://", "https://")):
            url = "https://" + url
        self.url_entry.delete(0, tk.END)
        self.url_entry.insert(0, url)

        temp_ts = datetime.now().strftime("%Y-%m-%d
%H:%M:%S")
        self.temp_iid = self.tree.insert("", 0,
values=(url, temp_ts, "Аналізую..."))

        self.current_result = {
            "url": url,
            "timestamp": temp_ts,
            "score": 0,
            "max_score": 0,
            "issues": {"critical": [], "important": [],
"minor": []},
```

```
            "recommendations": [],
            "details": {}
        }
        self.display_result(url)

        self.thread = SEOAnalyzerThread(url, self.db,
self.progress_queue)
        self.thread.start()

    def check_queue(self):
        try:
            while not self.progress_queue.empty():
                msg = self.progress_queue.get_nowait()
                if msg[0] == "progress":
                    self.current_result = msg[1]
                    if self.temp_iid and
self.tree.exists(self.temp_iid):
                        url = msg[1]["url"]
                        ts = msg[1]["timestamp"]
                        self.tree.item(self.temp_iid,
values=(url, ts, f"{msg[1]['score']}/{msg[1]['max_score']}
(аналізую...)"))
                    selected = self.tree.focus()
                    if selected and
self.tree.exists(selected) and
self.tree.item(selected)["values"][0] == msg[1]["url"]:
                        self.display_result(msg[1]["url"])
                elif msg[0] == "done":
                    self.current_result = msg[1]
                    self.refresh_history()
                    for iid in self.tree.get_children():
                        v = self.tree.item(iid)["values"]
                        if v[0] == msg[1]["url"]:
                            self.tree.selection_set(iid)
                            self.tree.focus(iid)
                            self.display_result(msg[1]["url
"])
                            self.notebook.select(1)
                            break
                elif msg[0] == "error":
                    messagebox.showerror("Помилка", msg[1])
                    if self.temp_iid and
self.tree.exists(self.temp_iid):
                        self.tree.delete(self.temp_iid)
                    self.temp_iid = None
                    self.clear_result()
        except tk.TclError:
            pass
        self.root.after(200, self.check_queue)

    def clear_result(self):
        self.link_label.config(text="")
        self.date_label.config(text="")
```

```
        self.score_label.config(text="")
        for widget in
self.checks_scrollable_frame.winfo_children():
            widget.destroy()
        self.issues_text.delete(1.0, tk.END)
        self.current_result = None

    def refresh_history(self):
        for item in self.tree.get_children():
            self.tree.delete(item)
        for url, ts, score, max_score in
self.db.get_history():
            self.tree.insert("", "end", values=(url, ts,
f"{score}/{max_score}"))

    def on_tree_select(self, event):
        selected = self.tree.focus()
        if not selected:
            self.clear_result()
            return
        values = self.tree.item(selected, "values")
        url = values[0]
        if "Аналізую..." in values[2]:
            if self.current_result and
self.current_result["url"] == url:
                self.display_result(url)
                self.notebook.select(1)
            return
        with sqlite3.connect(self.db.DB_NAME) as conn:
            c = conn.cursor()
            c.execute('''
                SELECT a.timestamp, a.score, a.max_score,
a.details_json, a.issues_json, a.recommendations_json,
a.checks_json
                FROM analysis a JOIN sites s ON a.site_id =
s.id
                WHERE s.url = ? ORDER BY a.timestamp DESC
LIMIT 1
            ''', (url,))
            row = c.fetchone()
            if row:
                self.current_result = {
                    "url": url, "timestamp": row[0],
"score": row[1],
                    "max_score": row[2],
                    "details": json.loads(row[3]),
"issues": json.loads(row[4]),
                    "recommendations": json.loads(row[5]),
                    "checks": json.loads(row[6] or '[]')
                }
                self.display_result(url)
                self.notebook.select(1)
```

```
    def get_status_color(self, status):
        colors = {
            "in_progress": "gray",
            "ok": "green",
            "warning": "orange",
            "error": "red",
            "failed": "red"
        }
        return colors.get(status, "gray")

    def display_result(self, url):
        res = self.current_result
        self.link_label.config(text=res.get("url", ""))
        ts = res.get("timestamp", "")
        self.date_label.config(text=ts.replace(" ", "\n"))
        self.score_label.config(text=f"{res.get('score',
0)} / {res.get('max_score', 0)}")

        for widget in
self.checks_scrollable_frame.winfo_children():
            widget.destroy()

        for col in range(4):
            self.checks_scrollable_frame.columnconfigure(co
l, weight=1)

        checks = res.get("checks", [])
        for i, check in enumerate(checks):
            row = i // 4
            col = i % 4
            subframe =
tk.Frame(self.checks_scrollable_frame, bg="#E6CCFF")
            subframe.grid(row=row, column=col, sticky="ew",
padx=2, pady=2)

            color = self.get_status_color(check["status"])
            icon_label = tk.Label(subframe, text="●",
fg=color, font=("Segoe UI", 14), bg="#E6CCFF")
            icon_label.pack(side="left", padx=5)

            text = f"{check['name']}: {check['message']}"
            text_label = tk.Label(subframe, text=text,
bg="#E6CCFF", font=("Segoe UI", 10), anchor="w",
justify="left", wraplength=200)
            text_label.pack(side="left", fill="x")

        self.issues_text.delete(1.0, tk.END)

        self.issues_text.insert(tk.END, "=== Помилки та
проблеми ===\n", "bold")
        issues = res.get("issues", {})
        for category, iss_list in issues.items():
            if iss_list:
```

```
                cat_text = f"{category.capitalize()}:\n"
                self.issues_text.insert(tk.END, cat_text,
"bold")
                for iss in iss_list:
                    self.issues_text.insert(tk.END, f"-
{iss}\n")

        recommendations = res.get("recommendations", [])
        if recommendations:
            self.issues_text.insert(tk.END, "\n===
Рекомендації ===\n", "bold")
            for rec in recommendations:
                self.issues_text.insert(tk.END, f"-
{rec}\n", "ok")

    def delete_selected(self):
        selected = self.tree.focus()
        if not selected:
            messagebox.showwarning("Помилка", "Виберіть
запис для видалення")
            return
        values = self.tree.item(selected, "values")
        url = values[0]
        if messagebox.askyesno("Видалити", f"Видалити
аналіз для {url}?"):
            self.db.delete_analysis(url)
            self.refresh_history()
            self.clear_result()
            self.temp_iid = None
            self.notebook.select(0)

if __name__ == "__main__":
    root = tk.Tk()
    app = SEOAnalyzerApp(root)
    root.mainloop()
```