

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «магістр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Система веб-аналітики вимірювання та аналізу
продуктивності веб-сайтів»**

Виконав студент групи ПЗм-23-1
КРАВЧЕНКО Роман Романович

Керівник роботи:
ЧИЖМОТРЯ Олексій Володимирович

Рецензент:
БАКАЛЮК Тетяна Анатоліївна

Житомир – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«1» жовтня 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **КРАВЧЕНКО Роман Романович**

Керівник роботи: **ЧИЖМОТРЕ Олексій Володимирович**

Тема роботи: **«Система веб-аналітики вимірювання та аналізу продуктивності веб-сайтів»**, затверджена Наказом закладу вищої освіти від «1» жовтня 2024 р., № 520/с

Вихідні дані для роботи: **системи веб-аналітики як інструмент моніторингу та аналізу продуктивності веб-додатків з метою її покращення, процеси моніторингу, аналізу та оптимізації продуктивності веб-додатків.**

Керівники / консультанти з магістерської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Керівник/ консультант	Завдання видав	Завдання прийняв
1	Чижмотря О.В.	01.10.2024р.	01.10.2024р.
2	Чижмотря О.В.	01.10.2024р.	01.10.2024р.
3	Чижмотря О.В.	01.10.2024р.	01.10.2024р.

РЕФЕРАТ

Кваліфікаційна робота освітнього ступеня «магістр» присвячена системі веб-аналітики аналізу та покращення продуктивності веб-сайтів. Пояснювальна записка складається з вступу, трьох розділів та списку використаної літератури. Пояснювальна записка до кваліфікаційної роботи містить 88 сторінок, 47 ілюстрацій, 2 таблиці, 2 додатки та 40 посилань на зовнішні джерела.

Метою роботи є створення системи веб-аналітики аналізу та покращення продуктивності веб-додатків, що поєднуватиме дані користувачів із результатами автоматизованого моніторингу, задля забезпечення глибокого аналізу та більш точного виявлення проблем продуктивності веб-додатків.

В роботі описано процес створення компонентів системи веб-аналітики, а саме системи збору та алгоритму аналізу продуктивності системи. Система проводиться моніторинг продуктивності веб-сайтів та використовує зібрані дані для виявлення проблем з продуктивністю. Досліджені способи моніторингу та метрики продуктивності веб-сайтів. Проведено експеримент з метою дослідження ефективності системи.

КЛЮЧОВІ СЛОВА: ВЕБ-АНАЛІТИКА, ПРОДУКТИВНІСТЬ, МОНІТОРИНГ ПРОДУКТИВНОСТІ, МЕТРИКИ ПРОДУКТИВНОСТІВ, ВЕБ-ДОДАТКИ.

ABSTRACT

The master's thesis is devoted to the web analytics system for analyzing and improving the performance of websites. The thesis note consists of an introduction, three sections and a list of used literature. The thesis note contains 88 pages, 47 illustrations, 2 tables, 2 appendices and 40 links to external sources.

The purpose of the work is to create a web analytics system for analyzing and improving the performance of web applications, which will combine real user monitoring with the synthetic monitoring, in order to provide in-depth analysis and more accurate detection of website performance issues.

The work describes the process of creating the components of the web analytics system, more specifically the collection system and the performance analysis algorithm. The system monitors the performance of websites and uses the collected data to identify performance issues. Methods of monitoring and metrics of the website performance have been studied. An experiment was conducted to research the effectiveness of the system.

KEYWORDS: WEB ANALYTICS, PRODUCTIVITY, PRODUCTIVITY MONITORING, PRODUCTIVITY METRICS, WEB APPLICATIONS.

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	6
ВСТУП	7
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ.....	10
1.1. Інструменти та засоби аналізу продуктивності веб-сайтів.....	10
1.2. Ключові метрики продуктивності веб-сайтів	16
1.3. Постановка задачі дослідження.....	20
Висновки до першого розділу.....	22
РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА СИСТЕМИ ВЕБ-АНАЛІТИКИ ВИМІРЮВАННЯ ТА АНАЛІЗУ ПРОДУКТИВНОСТІ	23
2.1. Визначення архітектури системи	23
2.2. Вибір засобів реалізації та вимоги до апаратного забезпечення	25
2.3. Визначення варіантів використання та структури системи	27
2.4. Проектування та розробка системи збору даних.....	34
2.5. Проектування та розробка системи аналізу даних	40
Висновки до другого розділу	47
РОЗДІЛ 3. ОПИС ПРОГРАМНОГО ПРОДУКТУ	48
3.1 Графічний інтерфейс сторінки.....	48
3.2 Покращення продуктивності веб-сайтів за допомогою системи	61
Висновки до третього розділу.....	68
ВИСНОВКИ.....	69
ЛІТЕРАТУРА	71
ДОДАТКИ.....	75

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – (англ. application programming interface) – програмний інтерфейс додатка

DOM – (англ. document object model) – об'єктна модель документа

ORM – (англ. object-relational mapping) – об'єктно-реляційне відображення

SPA – (англ. single page application) – одно сторінковий додаток

LCP – (англ. largest contentful paint) – відображення найбільшого контенту

CLS – (англ. cumulative layout shift) – кумулятивне зміщення макету

FCP – (англ. first contentful paint) – перше малювання вмісту

TBT – (англ. total blocking time) – загальний час блокування

SI – (англ. speed index) – індекс швидкості

TTFB – (англ. time to first byte) – час до першого байту

БД – база даних

СУБД – система управління базами даних

ВСТУП

Актуальність теми. Продуктивність роботи веб-додатка є важливим фактором що впливає на його успішність. Для власника веб-додатку важливо щоб цільовий користувач міг безперешкодно скористатись його додатком, адже від цього залежить і виконання бізнес цілей. Низька продуктивність веб-сайту прямо впливає на користувацький досвід. Якщо веб-сторінка завантажується повільно, більшою є вірогідність того, що користувач покине сторінку не дочекавшись повного завантаження. Також низька продуктивність може вплинути на рейтинг веб-сторінки в пошукових системах.

З огляду на визначну роль продуктивності веб-сторінок, виникає потреба в моніторингу, систематичному аналізі, підтримці та покращенні продуктивності веб-додатків.

Сучасні засоби для аналізу продуктивності веб-сторінок не завжди є достатніми для виявлення проблем з продуктивністю. Деякі з інструментів не надають достатньо детальні дані для аналізу продуктивності, хоч і використовують дані відвідувачів веб-додатку, тому веб-сторінки вимагають додаткового аналізу після виявлення проблеми в продуктивності. Інші інструменти надають більш детальні дані щодо продуктивності, при цьому аналізують лише поточну сторінку. Користувачу доводиться аналізувати кожну сторінку окремо, виникає потреба в автоматизації.

Для виявлення проблем продуктивності веб-додатка інколи недостатньо провести аналіз сторінки за допомогою наявних інструментів, адже на продуктивність впливає широкий спектр факторів, таких як час, тип пристрою та програмне забезпечення. Проблеми можуть виникати на окремих сторінках або бути характерними лише для певної групи користувачів. Окрім того, інколи проблеми з продуктивністю можуть виникати лише при певній комбінації факторів.

Актуальність даної теми зумовлена необхідністю систематизованого моніторингу, аналізу, підтримки та покращення продуктивності веб-додатків, задля утримання користувачів та покращення користувацького досвіду.

Задовільнити такі потреби може система веб-аналітики яка аналізуватиме продуктивність веб-додатків, використовуючи ключові показники продуктивності, які є достатніми для ідентифікації проблеми. Для цього системи має збирати дані щодо продуктивності веб-сторінок, а також дані зібрані з відвідувачів веб-додатку.

Метою роботи є проектування та розробка системи веб-аналітики аналізу та покращення продуктивності веб-додатків. Система поєднуватиме дані користувачів із результатами автоматизованого моніторингу, задля забезпечення глибокого аналізу та більш точного виявлення проблем продуктивності веб-додатків.

З мети можна визначити наступні завдання:

- дослідження існуючих методів аналізу продуктивності веб-додатків;
- визначення ключових метрик оцінки продуктивності;
- проектування архітектури та структури системи веб-аналітики;
- розробка механізму збору даних про продуктивність;
- впровадження механізму автоматизованого моніторингу продуктивності веб-додатків;
- реалізація системи веб-аналітики;
- створення засобів аналізу та виявлення проблем з продуктивністю веб-додатка;
- аналіз ефективності системи веб-аналітики для покращення продуктивності веб-додатків.

Об'єктом дослідження процеси моніторингу, аналізу та оптимізації продуктивності веб-додатків.

Предметом дослідження системи веб-аналітики як інструмент моніторингу та аналізу продуктивності веб-додатків з метою її покращення.

Наукова новизна роботи полягає у поєднанні підходів моніторингу справжніх користувачів та синтетичного моніторингу в системі веб-аналітики для забезпечення більш точного та всебічного аналізу продуктивності веб-сайтів, а також подальшому використанню зібраних даних в запропонованому в роботі

алгоритмі, що дозволяє ідентифікувати комбінації факторів, які негативно впливають на продуктивність.

Практичне значення полягає у можливості застосування розробленої системи веб-аналітики для покращення продуктивності веб-сайтів. Вона дозволяє проводити моніторинг, аналіз та оцінювати продуктивність веб-сайтів, формувати рекомендації щодо оптимізації та ідентифікувати проблемні місця в продуктивності.

Апробація. За темою кваліфікаційної магістерської роботи було опубліковано дві тези конференції [1, 2].

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Продуктивність веб-сайтів є критично важливим фактором, що впливає на їх успішність. Від продуктивності веб-додатку залежить його ефективність та виконання бізнес-цілей, адже низька продуктивність веб-сторінок прямо впливає на досвід користувачів [3]. Оптимізація веб-сайту та зниження швидкості завантаження можуть значно покращити показник конверсії [4]. З огляду на таку важливу роль продуктивності, виникає потреба в постійному моніторингу та вимірюванні продуктивності веб-додатків з метою підтримки та покращення.

1.1. Інструменти та засоби аналізу продуктивності веб-сайтів

Для аналізу продуктивності веб-сайтів використовуються інструменти, що забезпечують збір і обробку даних щодо продуктивності певної веб-сторінки, або веб-додатка в цілому. Аналіз зібраних даних, дозволяє оцінити продуктивність веб-сторінки, визначити чи потребує сторінка покращення, та визначити фактори, що впливають на продуктивність сторінки. Деякі інструменти аналізу продуктивності частково автоматизують аналіз продуктивності, та надають користувачу детальний звіт щодо продуктивності, що містить ключові метрики та можуть визначити фактори, що значно впливають на продуктивність.

Існують 2 основні підходи до моніторингу продуктивності веб-сторінок: синтетичний моніторинг та моніторинг реальних користувачів [5]. Дані підходи мають різні принципи роботи, способи використання, переваги та недоліки.

Для синтетичного моніторингу продуктивності веб-додатків зазвичай використовуються автоматизовані інструменти аналізу. Завантаження сторінки симулюється таким інструментом, дані справжніх відвідувачів веб-додатків не використовуються. Під час симуляції відбувається збір інформації щодо продуктивності. Оточення в якому в якому здійснюється симуляція варто тримати незмінним, для отримання найбільш точних результатів.

Синтетичний моніторинг може використовуватись для готових веб-додатків, а також розробці і тестуванні. Моніторинг продуктивності на ранніх

етапах життєвого циклу веб-додатків може допомогти виявити проблеми з продуктивністю до того як це вплине на користувацьких досвід справжніх відвідувачів веб-додатку.

Lighthouse є одним з інструментів синтетичного аналізу продуктивності веб-додатків. Lighthouse – це інструмент для аналізу та покращення продуктивності та якості веб-сторінок. Окрім продуктивності веб-додатку даний інструмент дозволяє проаналізувати доступність, оптимізацію для пошукових систем, перевірити наявність найкращих практик на веб-сайті. Для аналізу продуктивності веб-сторінки, Lighthouse завантажує її в симульованому середовищі та проводить ряд аудитів. Після чого буде створений звіт щодо продуктивності веб-сторінки.

Результати роботи інструменту можуть варіюватись, тому рекомендується проводити аналіз повторно декілька разів і використати середні значення як вихідні [6]. Також варто враховувати інші фактори, що можуть впливати на швидкість завантаження сторінки, як кеш, розширення браузера, тощо.

Lighthouse аналізує лише задану сторінку, тобто для аналізу всього веб-сайту потрібно проводити аудит кожної сторінки окремо.

Результатом аналізу завантаження сторінки інструментом Lighthouse є звіт, який включає в себе оцінки продуктивності веб-сторінки за 100-бальною шкалою та метрики за допомогою яких була вирахована оцінка, зображення в хронологічній послідовності, що відображають як користувач бачить сторінку під час завантаження, список порад та потенційних можливостей для покращення продуктивності веб-додатку.

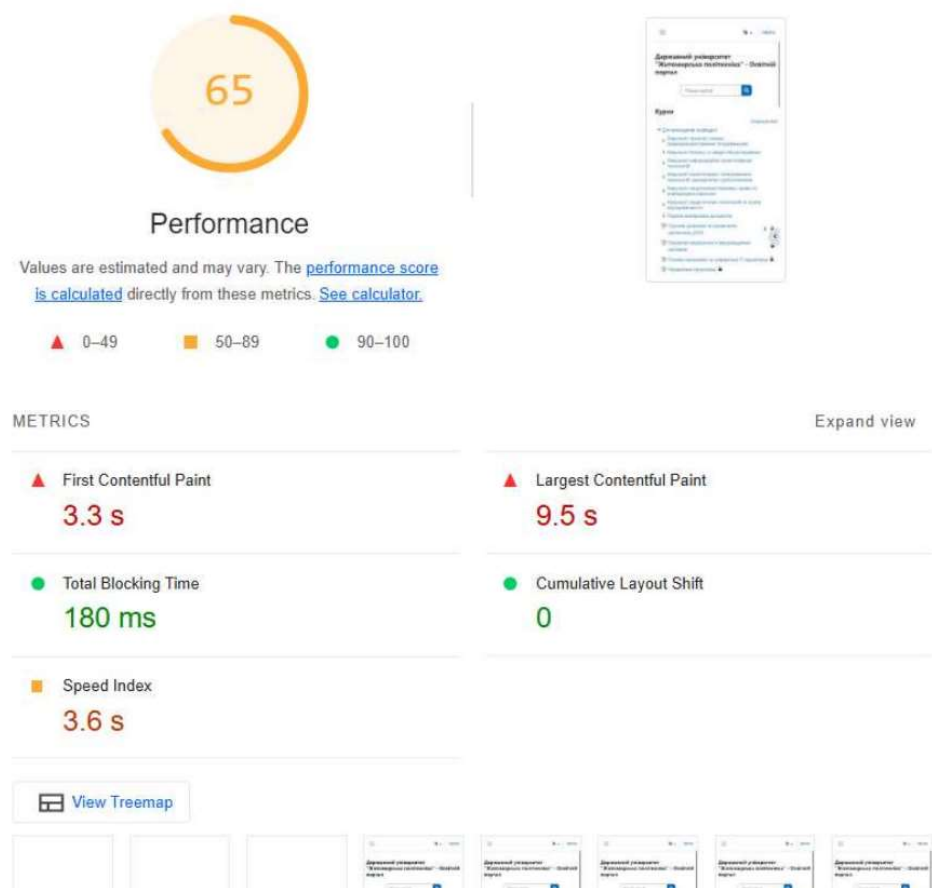


Рисунок 1.1 – Звіт щодо продуктивності веб-сторінки сформований Lighthouse

Також недоліком Lighthouse є відображення результатів що є актуальними для поточного пристрою, які можуть не завжди відображати ситуацію у справжніх користувачів. Для того щоб отримати більш схожі результати до картини у справжніх користувачів варто використати PageSpeed Insights, який використовує в собі Lighthouse для аналізу продуктивності. PageSpeed Insights більш сфокусований на продуктивності, ніж на інших метриках, і, аналізує продуктивність веб-додатку використовуючи дані справжніх користувачів з Chrome User Experience report. Тому при аналізі продуктивності веб-сторінки варто використовувати PageSpeed Insights за можливості, так як дані будуть більш точні ніж з Lighthouse, адже результати ґрунтуються на даних користувачів. Недоліком сервісу є те, що сервіс не може аналізувати веб-сайти в локальній мережі [7], на відміну від Lighthouse.

Report from Nov 22, 2024, 12:21:18 AM

<https://learn.ztu.edu.ua/>

Analyze

Mobile

Desktop

Discover what your real users are experiencing

This URL Origin

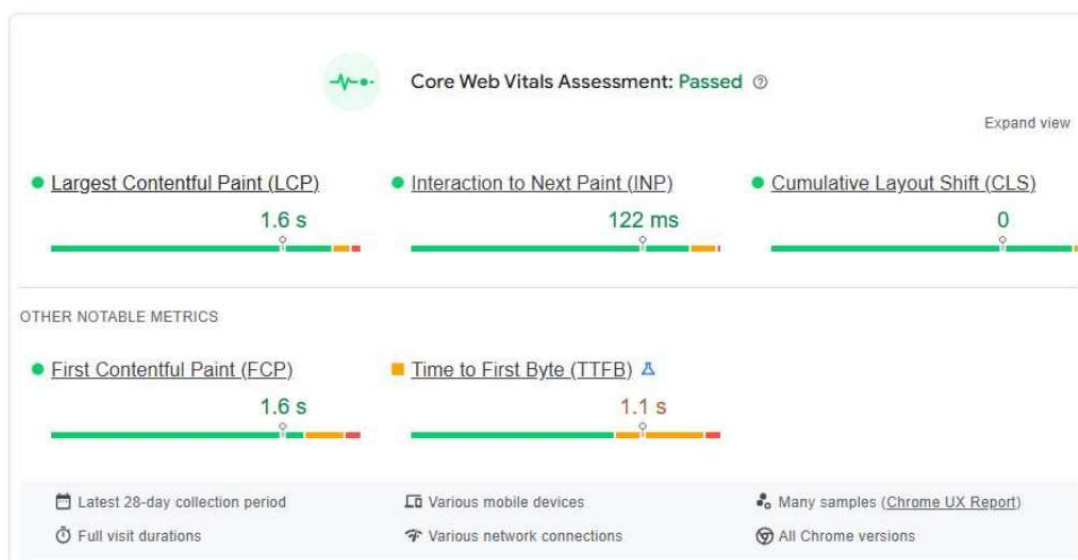


Рисунок 1.2 – Звіт щодо продуктивності веб-сторінки сформований PageSpeed Insights

Іншим підходом до моніторингу продуктивності є моніторинг справжніх користувачів. Дані в такому підході збираються з справжніх відвідувачів веб-сторінок під час їх перегляду. Зазвичай це реалізується за допомогою скрипту що інтегрується в кожну сторінку веб-додатку [8]. Такий підхід дозволяє аналізувати як додаток використовується і дозволяє виявити проблеми з продуктивністю які виникають лише за певних умов (наприклад проблема що виникає лише у певного користувача чи у групи користувачів в залежності від географічного положення). Такий підхід є корисним для комплексних сайтів з великою кількістю функціоналу. На відміну від синтетичного моніторингу, де проводиться аналіз окремих сторінок або випадків використання в даному підході аналізуються всі дії користувача, які можуть бути не передбачені під час проектування веб-сторінки.

Сервіси веб-аналітики зазвичай використовують підхід моніторингу реальних користувачів. Вони збирають дані про швидкість завантаження сторінки при кожному завантаженні сторінки одним і з користувачів. Таким чином власник веб-додатку отримує достовірні дані про швидкість завантаження сторінки та може їх проаналізувати. Сервіс веб-аналітики «Google Analytics» збирає дані про середню швидкість завантаження сторінки, швидкість з'єднання з сервером, швидкість відповіді серверу, та швидкість повного завантаження сторінки (Рис. 1.3). Дані подаються в розрізі окремих сторінок, країн та веб-браузерів. Такі дані дозволять виявити наявні проблеми з продуктивністю сторінок та приблизно визначити причину їх виникнення.

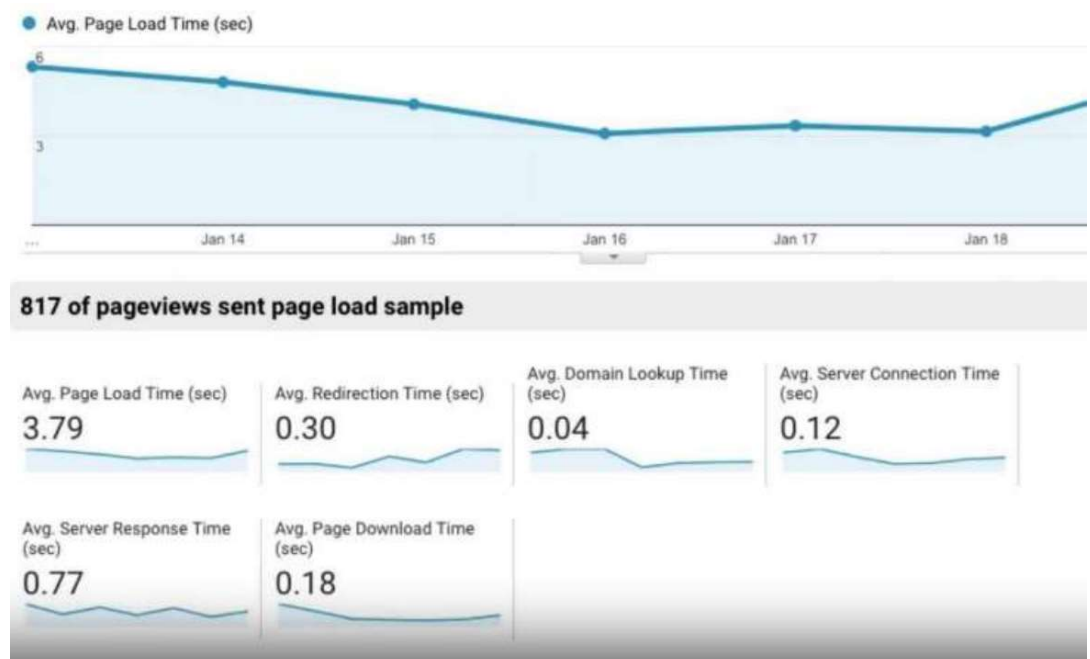


Рисунок 1.3 – Дані про швидкість завантаження сторінки в «Google Analytics»

Головним недоліком сервісів веб-аналітики при зборі даних про швидкість завантаження сторінки недостатній обсяг даних для детального аналізу продуктивності сторінки. Сервіс «Google Analytics» більш акцентований на маркетинговій аналітиці, ніж на продуктивності веб-додатків, тому дані які надає сервіс щодо продуктивності є базовими. Як наслідок, зазвичай після виявлення проблем з завантаженням, користувачу доводиться використовувати інші інструменти для отримання більш детального звіту.

Крім того, так як дані щодо продуктивності отримуються від користувачів веб-додатку, якість моніторингу та аналізу продуктивності залежить від кількості користувачів, що відвідують додаток. Деякі проблеми з продуктивністю можуть лишитись непоміченими за відсутності користувачів.

Проаналізувавши підходи до моніторингу продуктивності веб-додатків, а також існуючі інструменти аналізу продуктивності веб-додатків, було визначено їх ключові переваги та недоліки.

До переваг методу моніторингу реальних користувачів належить наближеність результатів до тих що отримує відвідувач додатку, адже вони отримані від справжніх користувачів і відображає справжню картину використання, аналізуються справжні варіанти використання, не лише визначені розробником. Також спрощується виявлення проблем, що можуть виникати лише у деяких груп користувачів.

До недоліків методу моніторингу реальних користувачів можна віднести залежність результатів від кількості користувачів, водночас за відсутності користувачів, моніторинг продуктивності не відбувається. Також, на результати впливатиме оточення користувачів, яке контролювані лише самими користувачами. З метою оптимізації, зазвичай в такому підході набір даних що збирається є базовим і є недостатнім для детального аналізу продуктивності веб-сторінки.

До переваг синтетичного методу належить передбачуваність результатів, адже збір даних щодо продуктивності відбувається в одному середовищі виконання. Також, як правило, при синтетичному аналізі набір метрик отриманих від інструмента є більшим, що забезпечує більш детальний аналіз. Аналіз продуктивності може відбувати на будь якому етапі життєвого циклу, сторінки які будуть проаналізовані і з якими параметрами буде проведена симуляція конфігурується.

Головним недоліком синтетичного методу є те, що дані не відображають картину актуальну для користувачів веб-додатку, так як дані були отримані під час симуляції. Тобто дані не враховують специфічні випадки використання, а

також середовище виконання додатку. Проблеми з продуктивністю, що виникають при комбінації декількох факторів також важче виявити з даним підходом до моніторингу.

Для проведення більш глибокого та комплексного аналізу продуктивності, а також для виявлення можливостей для оптимізації веб-сторінок, необхідно комбінувати ці підходи, використовуючи їхні переваги для забезпечення всебічної оцінки продуктивності.

1.2. Ключові метрики продуктивності веб-сайтів

Щоб зрозуміти чи потребує покращення продуктивність веб-сторінки, необхідно оцінити її продуктивність. Для оцінки продуктивності веб-сайтів використовують низку ключових показників, що дозволяють зрозуміти наскільки швидко і зручно користувачі можуть з ним взаємодіяти. Значення таких метрик дозволяють не лише оцінити продуктивність, а і визначити проблемні місця в роботі сайту. Наприклад, якщо значення метрики що відповідає за швидкість відповіді сервера перевищує норму, скоріше всього проблема знаходиться на стороні серверної частини.

До ключових показників продуктивності веб-сторінки можна віднести «Core Web Vitals» (основні веб-показники). Core Web Vitals – це набір стандартизованих показників ефективності, які були визначені компанією Google [9]. Ці метрики фокусуються на 3 основних аспектах: швидкість завантаження, швидкість реагування сторінки та візуальна стабільність. Ці фактори суттєво впливають на взаємодію користувача з сторінкою.

Largest Contentful Paint (відображення найбільшого контенту) відноситься до Core Web Vitals та відображає час візуалізації найбільшого зображення, текстового блоку або відео видимого у вікні перегляду, відносно до часу коли користувач перейшов на сторінку. Для забезпечення хорошої взаємодії з веб-сторінкою для користувача, необхідно LCP тримати на рівні 2.5 секунд і нижче [10]. Для покращення даного показника рекомендується використовувати кешування контенту веб-сторінки, оптимізувати або компресувати ресурси, в

тому числі зображення, використовувати відкладене відображення для зображень, що не знаходяться в полі зору під час завантаження, а також використовувати CDN (мережа доставки контенту) для пришвидшення завантаження.

Cumulative Layout Shift (кумулятивне зміщення макету) визначає стабільність веб-сторінки та визначає наскільки макет зміщується під час завантаження сторінки. Зміщення вмісту сторінки може бути спричинене багатьма факторами, як довантаження ресурсів, даних або реклама. Такі зміщення можуть значно впливати на досвід користувача. Наприклад, якщо користувач намагатиметься підтвердити замовлення натисканням кнопки, але через завантаження реклами вгорі макету відбувається здвиг, користувач може випадково натиснути на іншу кнопку, як «відмінити замовлення», що значно вплине як на досвід користувача, так і на виконання цілей сторінки.

Значення метрики CLS визначається сумою оцінок зсуву макета. Кожна оцінка зсуву макета оцінюється як добуток впливу зсуву на відстань зсуву. Вплив зсуву визначається в відсотковому значенні на скільки зсув вплинув на макет між 2 кадрами в області перегляду, якщо макет змінився на 75%, показник відповідатиме 0.75. Відстань зсуву визначається в відсотковому значенні відносно розміру області перегляду, на скільки макет змістився між 2 кадрами. Загальний показник CLS визначається за формулою:

$$CLS = \sum (\text{Вплив зсуву} * \text{Відстань зсуву})$$

Для забезпечення хорошого користувацького досвіду необхідно тримати показник CLS в межах від 0 до 0.1 [1139]. Для покращення даного показника рекомендується явно вказувати висоту та ширину для елементів, що завантажуються пізніше, як зображення, реклама, тощо. На значення метрики впливає структура і дизайн самої веб-сторінки, тому варто враховувати CLS при її розробці.

Ще однією метрикою з Core Web Vitals є Interaction to Next Paint (від взаємодії до наступного відображення). Дана метрика вимірює швидкість

відповіді сторінки на дії користувача, як кліки або введення з клавіатури. Для отримання значення метрики необхідно виміряти час між всіма взаємодіями користувача та до відображення вмісту в відповідь на цю дію. За значення метрики обирається найбільше з цих значень. Для забезпечення хорошого користувацького досвіду, рекомендується тримати значення метрики до 0.2 секунд [12].

First Contentful Paint (перше малювання вмісту) – метрика що визначає час з переходу користувачем на сторінку до відображення першого контенту. Контентом в даному випадку можна вважати текст, зображення, в тому числі векторні, тощо. При високому значенні FCP, користувач бачитиме порожню сторінку тривалий час під час завантаження. Значення FCP необхідно тримати нижче 1.8 секунд для забезпечення хорошого користувацького досвіду [13]. Для покращення значення даної метрики варто оптимізувати завантаження контенту, використовувати CDN, асинхронно завантажувати ресурси що блокують відображення контенту.

Total Blocking Time (загальний час блокування) вимірює час від FCP до того як основний потік буде розблокований і сторінка реагуватиме на взаємодії [14]. Тобто, це час який сторінка буде не буде реагувати на дії користувача. Дана метрика зазвичай вимірюється синтетично, не рекомендується збирати дані з користувачів, так як дії користувачі можуть впливати на значення метрики. Для покращення значення метрики рекомендується зменшити кількість операцій що виконується при завантаженні, також за можливості перенести виконання довготривалих операцій у фоновий режим.

Lighthouse Speed Index (індекс швидкості) визначає наскільки швидко вміст сторінки був відображений під час завантаження. Для визначення значення метрики записується відео завантаження сторінки, оцінюється прогрес між кадрами та вираховується даний індекс. Значення даної метрики вважається задовільним, якщо воно нижче 3.4 секунд [15]. Для покращення значення

метрики рекомендується зменшити час виконання скриптів та перенести довготривалі операції з головного потоку у фоновий режим.

Метрики SI, FCP, LCP, TBT та CLS використовуються для обрахунку ще однієї метрики Lighthouse performance score (оцінка продуктивності). Дана метрика відображається загальну продуктивність і швидкість роботи сторінки та обраховується як сума оцінок названих метрик з ваговим коефіцієнтом. Значення метрики варіюється в діапазоні від 0 до 100, де значення більші від 90 є задовільними, менші від 50 повільними, інші середніми [16]. Для Lighthouse версій 10-12 дана оцінка вираховується по формулі:

$$Score = 0.1 * FCP + 0.1 * SI + 0.25 * LCP + 0.25 * CLS + 0.3 * TBT$$

Time to First Byte (час до першого байту) вимірює час між посиланням запиту на сервер до отримання першого байту з відповіді. Значення даної метрики не включає завантаження всієї відповіді з серверу, тому її розмір є менш важливим для значення. TTFB передує LCP та FCP, тому значення метрики варто тримати чим нижчим для покращення досвіду користувача. Значення метрики варіюється від специфіки веб-додатку, але для хорошого користувацького досвіду його варто меншим ніж 0.8 секунд [17].

Час завантаження сторінки та час завантаження DOM також є метриками продуктивності сторінки. Час завантаження DOM показує час, коли HTML-документ було повністю завантажено, але в цей час не враховуються інші ресурси на сторінці. Час завантаження сторінки в свою чергу включає завантаження всіх ресурсів на сторінці [18]. Аналіз цих двох метрик допоможе визначити що саме впливає на швидкість завантаження, документ чи його залежності. Значення цих метрик залежить від специфіки веб-додатку, зменшення цього часу приводить до покращення користувацького досвіду.

Метрики продуктивності веб-додатків допомагають оцінити продуктивність, для подальшої оптимізації веб-додатків. Комбінований аналіз метрик дозволить визначити проблемні місця в роботі веб-додатку, що значно впливають на продуктивність. Моніторинг та аналіз метрик продуктивності веб-

додатків дозволить підтримувати та покращувати рівень продуктивності, що призведе до покращення досвіду користувачів веб-додатку.

1.3. Постановка задачі дослідження

Для забезпечення оптимальної продуктивності веб-додатків важливо ідентифікувати чинники, що впливають на швидкодію веб-сторінок, та визначити шляхи їх покращення. Одним з способів для досягнення такої мети та виявлення можливостей для оптимізації необхідно є збір і аналіз даних про роботу продуктивність веб-сторінок.

Проаналізувавши підходи до моніторингу продуктивності веб-додатків та інструменти для аналізу продуктивності, можна виявити недоліки та аспекти, які мають бути покращені в веб-додатку.

Синтетичний моніторинг отримує дані з контрольованого середовища, що не завжди відповідає умовам справжніх користувачів додатку. Також при синтетичному моніторингу охоплюються лише передбачені варіанти використання, справжні користувачі веб-додатку можуть взаємодіяти з додатком у непередбачуваний спосіб. Деякі проблеми з продуктивністю можуть виникати лише за комбінації певних умов, які не завжди може виявити синтетичний моніторинг, так як умови в яких відбувається моніторинг зазвичай є однаковими.

Дані при моніторингу справжніх користувачів отримуються безпосередньо від відвідувачів веб-додатків, відповідно за недостатньої кількості відвідувачів об'єм даних буде недостатнім для аналізу і виявленню проблем, деякі проблеми можуть лишитись непоміченими. Також набір які можна отримати при такому моніторингу є обмеженим, адже вимірювання деяких метрик продуктивності може вплинути на досвід користувача.

Дані методи моніторингу є взаємодоповнюючими, комбіноване використання обох підходів може забезпечити користувача достатньою кількістю даних для повного аналізу продуктивності веб-сайту.

Аналіз метрик продуктивності веб-сторінок дозволив визначити метрики, які будуть корисними для аналізу продуктивності веб-додатків. Аналіз та

порівняння таких метрик продуктивності допоможе користувачу визначити проблемні місця в продуктивності веб-додатку.

Також, існуючі сервіси веб-аналітики є більш сфокусованими на маркетинговій аналітиці, ніж на продуктивності веб-додатку. Виникає потреба в сервісі веб-аналітики, що буде орієнтований на продуктивність веб-сторінок, який забезпечить автоматизований збір та аналіз даних щодо продуктивності веб-сторінок, комбінуватиме методи моніторингу продуктивності для отримання найбільш широкої вибірки даних для аналізу, а також ідентифікуватиме фактори, що впливають на продуктивність веб-додатку.

Було сформульовано задачі, які необхідно виконати для створення такого програмного продукту:

1. Визначити та проаналізувати ключові метрики та показники продуктивності веб-сторінок, що необхідні для подальшого аналізу та визначення шляхів покращення продуктивності веб-додатків.
2. Розробити систему збору та аналізу даних щодо продуктивності веб-додатку. Система збору має забезпечити достатній збір інформації, необхідної для аналізу продуктивності веб-додатку. Інтеграція системи повинна здійснюватися ефективно.
3. Розробити інструменти для візуалізації та аналізу ключових показників продуктивності веб-додатку. Сервіс повинен збирати необхідні дані щодо продуктивності сторінок, та надавати їх в зручному для аналізу вигляді: звіти, графіки, тощо. Сервіс має визначити, чи значення метрики є оптимальним, чи вказує на необхідність покращення продуктивності.
4. Дозволити сегментацію даних. Користувач повинен мати можливість сегментувати зібрані дані в залежності від їх параметрів, таких як час, географічне розташування користувачів, тип пристрою або користувача та інше. Це дозволить здійснювати більш точний аналіз та виявлення закономірностей, що впливають на продуктивність у різних умовах.
5. Розробити алгоритм аналізу даних для ідентифікації проблем та можливостей для оптимізації веб-додатків. Система повинна надавати

користувачу сервісу дані та звіти, що допоможуть аналізувати роботу вебсторінки та визначити потенційні проблеми з продуктивністю. Сервіс веб-аналітики має сигналізувати про погіршення продуктивності певних сторінок та допомогти виявляти і проблемні для продуктивності місця

Визначений перелік завдань враховує всі аспекти необхідні для реалізації ефективного сервісу веб-аналітики для покращення продуктивності веб-сайтів. Результатом виконання завдань буде сервіс веб-аналітики з системою збору даних щодо продуктивності веб-сторінок. Сервіс здатен аналізувати зібрані дані, надавати оцінки щодо продуктивності та надавати дані користувачу у зручному для аналізу вигляді. Система виявлятиме та повідомлятиме користувача про наявні проблеми з продуктивністю веб-сторінок.

Висновки до першого розділу

В даному розділі було проведено аналіз предметної області дослідження. Були розглянуті підходи до моніторингу продуктивності веб-додатків, а саме синтетичний моніторинг та моніторинг справжніх користувачів, визначені ключові переваги та недоліки.

Проаналізовані існуючі інструменти аналізу продуктивності веб-додатків, що використовують дані підходи моніторингу продуктивності, визначені їх переваги та недоліки, а також способи застосування.

Проаналізовані ключові метрики продуктивності веб-сторінок, як вони визначаються та як застосовуються. Було визначено їх значення для роботи сайту та користувацького досвіду. Також були розглянуті основні способи покращення показників даних метрик.

Визначена задача дослідження. Був визначений перелік завдань, які необхідно виконати для реалізації сервісу веб-аналітики для аналізу та покращення продуктивності веб-сайтів.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА СИСТЕМИ ВЕБ-АНАЛІТИКИ ВИМІРЮВАННЯ ТА АНАЛІЗУ ПРОДУКТИВНОСТІ

2.1. Визначення архітектури системи

Система веб-аналітики є комплексним додатком що складається з багатьох компонентів. Основними функціями даної системи є збір, візуалізація та аналіз даних. Збір даних забезпечуватиме окремий компонент системи, який можна буде інтегрувати в бажаний веб-додаток для подальшого збору даних. За візуалізацію та аналіз даних відповідатиме інших компонент системи, який являтиме собою веб-додаток.

Як було зазначено, за збір даних відповідатиме окремий компонент. Цей компонент працюватиме шляхом вбудовування спеціальних скриптів у вебсторінки, що дозволить відстежувати ключові показники, такі як час завантаження, час отримання відповіді з серверу, тощо.

Дані будуть передаватися на сервер для подальшої обробки та збереження в базі даних. Крім того, компонент збиратиме дані про ресурси, завантаження яких займає занадто довгий час, та помилки що виникають у користувача, та повідомлятиме сервер про них.

Найбільш ефективним рішенням для реалізації даного компонента системи є використання скрипта. Скрипт можна легко інтегрувати в код веб-сторінки без необхідності змінювати основну архітектуру сайту, що робить його універсальним для різних веб-додатків. Скрипт має доступ до даних щодо продуктивності сторінки, може фіксувати поведінкові дані користувачів, такі як кліки, скролінг і час завантаження сторінок. Це дозволяє отримувати точну й актуальну інформацію для аналізу продуктивності сайту. Крім того, скрипти можуть бути налаштовані або згенеровані динамічно для збору різноманітних типів даних, в залежності від потреб користувачів, що дає гнучкість у налаштуванні метрик залежно від потреб проекту. Скрипт передаватиме дані в інші компоненти системи автоматизовано, без потреби втручання користувача.

Інший компонент системи є веб-додатком, який відповідає за візуалізацію та аналіз даних, а також конфігурацію системи. Він являє собою веб-інтерфейс (клієнтська частина), що взаємодіє із сервером для обробки та відображення зібраної інформації. Веб-інтерфейс дозволяє користувачам переглядати дані в зручному форматі, зокрема у вигляді графіків, діаграм та звітів, що полегшує аналіз продуктивності веб-сайтів. Дані будуть отримуватись з серверу, який комунікуватиме з клієнтською частиною за допомогою HTTP-запитів. Тобто архітектурою даного компоненту системи є клієнт-серверна архітектура, яка складається з двох частин – клієнт і сервер.

Модульна архітектура була обрана для серверної частини через її високу гнучкість та масштабованість. У порівнянні з монолітною архітектурою, де всі частини системи щільно пов'язані між собою, модульний підхід дозволяє розділяти функціональність на незалежні блоки [19]. Це спрощує розробку, тестування та підтримку кожного модуля окремо.

У порівнянні з монолітною архітектурою, де зміна однієї частини може вплинути на всю систему, модульний підхід дозволяє локалізувати зміни, полегшуючи підтримку та подальший розвиток проекту. Однак, на відміну від мікросервісної архітектури, всі модулі в модульній системі залишаються частиною єдиного додатка та виконуються в одному середовищі, що спрощує розгортання та знижує складність комунікації між ними.

Архітектурою клієнтської частини для системи було обрано підхід Single Page Application (SPA). Це рішення дозволяє забезпечити більш інтерактивний та швидкий користувацький досвід, оскільки SPA завантажує всі необхідні ресурси один раз на початку, а подальша навігація між сторінками відбувається без повного перезавантаження сторінки [20]. Такий підхід зменшує затримки при роботі з додатком, оскільки більшість запитів до сервера стосуються лише отримання даних, а не цілої сторінки.

SPA є оптимальним вибором для системи веб-аналітики, де користувачі активно взаємодіють з динамічними даними, такими як графіки та звіти. Постійне оновлення сторінки при роботі з великим обсягом даних могло б

створювати незручності та знижувати продуктивність. Завдяки SPA, система залишається швидкою, так як дані завантажуються швидко і швидко реагують на дії користувача.

Діаграму компонентів системи можна побачити на схемі:

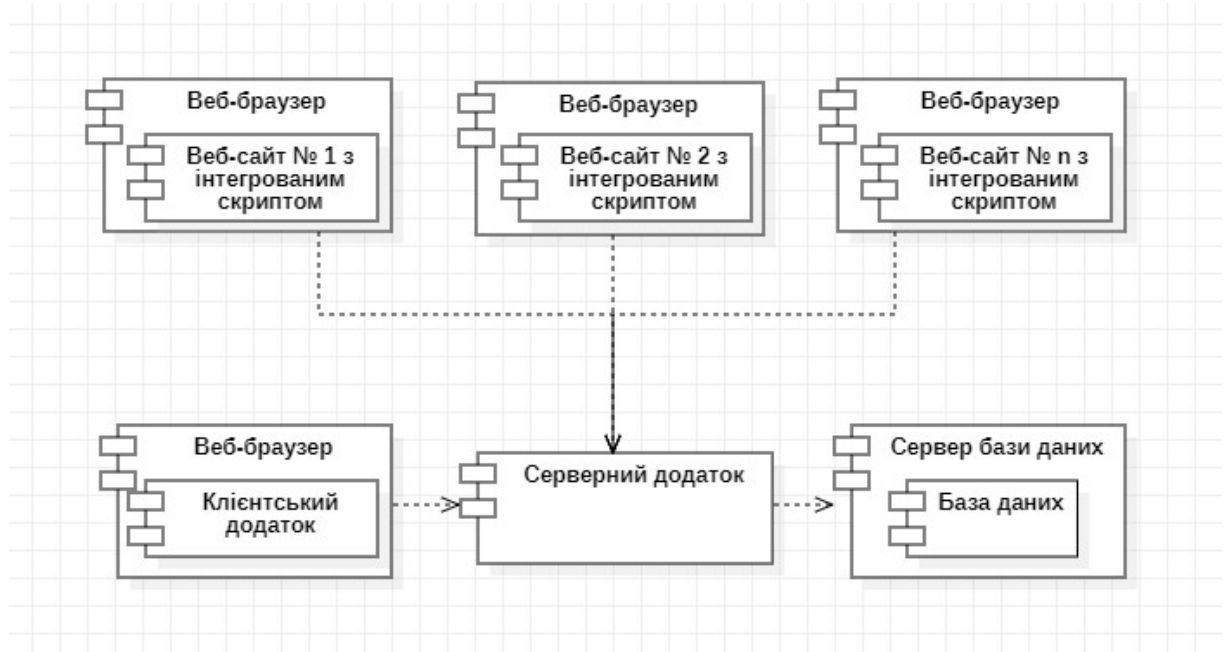


Рисунок 2.1 – Компоненти системи веб-аналітики

2.2. Вибір засобів реалізації та вимоги до апаратного забезпечення

Система веб-аналітики є комплексною системою, що складається з декількох компонентів. При виборі мов програмування для реалізації системи важливо звернути увагу на сумісність між різними частинами системи, їх взаємодію та архітектуру системи. Модульна архітектура і SPA вимагають високої взаємодії між клієнтським та серверним кодом, в такій ситуації виникає необхідність в типізації, яка присутня в мові програмування TypeScript.

Для реалізації клієнтського додатка було обрано мову програмування TypeScript, оскільки ця мова забезпечує статичну типізацію, що дозволяє зменшити кількість помилок під час розробки, особливо у великих проектах. TypeScript також полегшує читання коду і його підтримку.

Також при створенні клієнтського додатка буде використовуватись бібліотека React, яка дозволяє працювати з віртуальним DOM, замість звичайної.

Дана бібліотека була обрана, так як дозволяє створювати «компоненти», які можуть перевикористовуватись в різних місцях додатку. Також бібліотека надає можливість побудувати додаток що може працювати з динамічними даними. Ці фактори є важливими для системи веб-аналітики та обраної архітектури SPA, адже постійно відбуватиметься оновлення даних.

Важливо відзначити можливість інтеграції React з TypeScript, що забезпечить типізацію та зниження кількості помилок під час розробки, особливо у великих та складних проектах, де взаємодіють численні компоненти.

Також було вирішено використовувати бібліотеку Chart.js, яка є інструментом для побудови графіком, що має підтримку React та TypeScript. Дана бібліотека надає можливість побудови лінійних графіків з розподілом даних відносно часу, що буде корисним для побудови графіки продуктивності відносно часу [21, 22].

JavaScript використовується для реалізації скрипта збору даних, оскільки це стандартна мова для роботи з браузерами, що гарантує максимальну сумісність з різними веб-додатками з яких будуть збиратись дані та браузерами в яких вона буде використовуватись.

NestJS – це платформа для створення серверних додатків, побудована на базі Node.js і Express. Вона призначена для створення модульних додатків [23], також підтримується створення мікросервісів. Також існує багато бібліотек, що розширюють функціонал платформи, деякі з них будуть використані при розробці веб-додатку. Nest.js має повну підтримку мови програмування TypeScript та легко інтегрується з TypeORM для спрощення роботи з базою даних [23, 24].

Завдяки використанню TypeScript, серверний додаток має ті ж переваги, що й клієнтська частина, крім того спрощується типізація, адже серверна і клієнтська частина будуть написані на однакових мовах програмування.

На основі обраної архітектури, засобів реалізації та вимог користувача, було сформовано системні вимоги, за умови виконання яких, система працюватиме стабільно.

Для клієнтських пристроїв вимоги формувались за допомогою аналізу сумісності веб-браузерів з компонентами скрипта. Визначено, що мінімальними версіями веб-браузерів для виконання скрипта є Chrome 65, Firefox 60 та Edge 79.

Система побудована на платформі Node.js, одним з компонентів системи є сервіс Lighthouse та бібліотека для запуску веб-браузера Chrome. Тому вимоги для середовища виконання визначались на основі компонентів з врахуванням власних вимог. Персональний комп'ютер, на якому відбуватиметься запуск системи, для стабільної її роботи, мусить відповідати мінімальній комплектації у вигляді 2 ГБ ОЗУ, 64-розрядного (x64) процесору, операційна система Windows 10 або Ubuntu 18.

У ядрі системи повинна бути представлена СУБД MySQL, адже всі запити в базу даних у системі підтримуються MySQL.

2.3. Визначення варіантів використання та структури системи

Для визначення варіантів використання необхідно попередньо сформулювати вимоги до системи, оскільки саме вони окреслюють її функціональні можливості, технічні обмеження та характеристики, що визначають сценарії взаємодії користувачів із системою.

До функціональних вимог до системи належить можливість збору інформації щодо продуктивності сторінок веб-сайтів від справжніх користувачів. Саме ці дані в комбінації з синтетичними стануть основою для подальшого аналізу продуктивності. Тому система повинна також здійснювати синтетичний моніторинг продуктивності для забезпечення більш широкої картини продуктивності веб-додатку.

Система веб-аналітики повинна враховувати сторінки, що користувач покинув до повного завантаження, адже саме такі сторінки свідчать про низький рівень продуктивності.

Система повинна надавати можливість переглядати зібрані дані щодо продуктивності в оптимальному для аналізу вигляді. Повинна бути можливість

фільтрування даних (за періодом часу, адресою сторінки, типом пристрою, тощо).

Система повинна забезпечити користувача даними про ключові показники продуктивності сторінки, проблеми і помилки що виникають на сторінках, що допоможуть в визначенні проблемних місць в продуктивності веб-сторінки.

Також система повинна періодично створювати звіти щодо продуктивності веб-сайту для швидкого моніторингу та аналізу продуктивності. Звіти повинні містити показники ключових метрик продуктивності, проблеми продуктивності та помилки на сторінках, а також фактори, що впливають на продуктивність веб-сайту.

Окрім функціональних вимог, також були визначені нефункціональні вимоги. Інтерфейс представлення ІС повинен бути інтуїтивно зручним для цільового користувача та не вимагати від нього додаткової підготовки тривалістю більше двох годин. Враховуючи складність операцій, що виконує система, а також однопоточну архітектуру, час відповіді системи для звичайних запитів не повинен перевищувати 15 секунд, а для більш складних запитів – 90 секунд. Також веб-аналітика повинна працювати стабільно у 99% випадках використання для забезпечення позитивного досвіду користувачів системи.

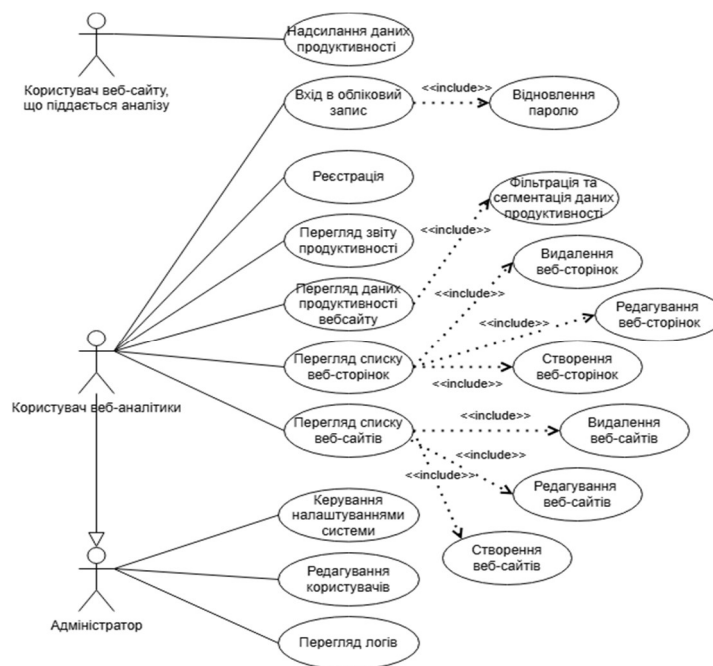


Рисунок 2.2 – Варіанти використання системи веб-аналітики

Аналіз функціональних вимог та варіантів використання системи дозволив визначити структуру системи, яка представляє собою сукупність взаємопов'язаних модулів, кожен із яких виконує визначені функції. Модулі можуть залежати один від одного, або від модулів з бібліотек NestJS. Наприклад, модуль Page залежить від модуля Website, так як Website надає функціонал доступу до даних веб-сайтів. Діаграму модулів системи, яка відображає їхню взаємодію та залежності між ними має наступний вигляд:

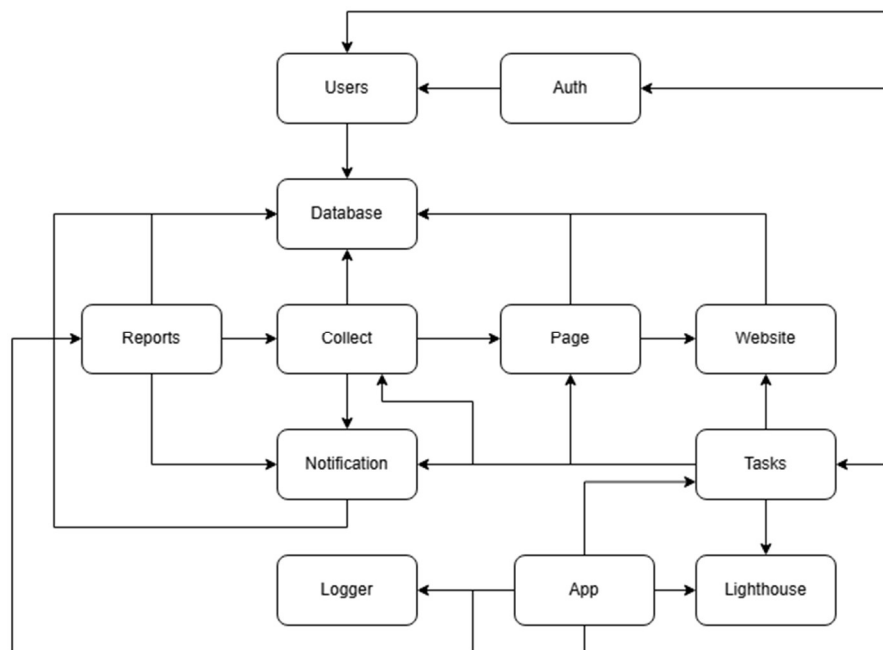


Рисунок 2.3 – Діаграма модулів системи веб-аналітики

Система складається з 12 модулів. Вихідним модулем додатку є модуль App, даний модуль ініціалізує API додатку, реєструє модулі системи, а також додаткові модулі як модуль для надсилання електронної пошти, модуль кешування, логування та налаштувань додатку. Для читання файлів логів та надання їх в зручному вигляді для користувача використовується модуль Logger. В файли логів вносяться такі записи як помилки, інформаційні повідомлення, дії користувачів, тощо.

В системі спроектована система запланованих завдань, які виконуватимуться періодично або на вимогу користувача. Це дає змогу автоматизації процесів синтетичного збору, аналізу та чистки даних.

Модулі Auth та Users необхідні для роботи з користувачами. Перший з них є модулем авторизації, в ньому реалізований функціонал для входу в обліковий запис, реєстрації, зміни паролю, тощо. Він залежить від модулю Users, який надає доступ до даних щодо користувача. Крім того, даний модуль надає функціонал для керування користувачами.

Був спроектований модуль Notification для реалізації системи оповіщень. Він необхідний для відправки повідомлень та електронних листів для користувача. Наприклад, відправка електронного листа для відновлення паролю або сповіщення при автоматичному створенні щоденного звіту щодо продуктивності.

Система веб-аналітики надаватиме можливість для аналізу продуктивності декількох веб-сайтів одночасно. Модуль Website відповідає за збереження інформації про веб-сайти та за всі операції над ними, як створення, редагування, видалення, тощо. Кожен веб-сайт містить певний набір веб-сторінок. Модуль Page відповідає за збереження інформації про веб-сторінки веб-сайтів та операції над ними. Дані модулі в основному виконують операції над даними.

Модуль Collect відповідає за збір даних щодо продуктивності. Він надає доступ до скрипта, що інтегруватиметься в веб-додатки. Також модуль надає API для збереження даних щодо переглядів сторінок, їх продуктивності, помилок, тощо. Модуль надає функціонал для отримання цих даних в зручний спосіб.

Lighthouse модуль необхідний для керування сервісом Lighthouse. Даний модуль надає функціонал для запуску аналізу продуктивності веб-сторінки за допомогою сервісу та отримання звіту що є результатом аналізу. Даний модуль використовується для синтетичного моніторингу продуктивності.

Модуль Report надає API для отримання даних щодо продуктивності веб-сайту в зручному вигляді. Наприклад дані для сторінки інформаційної панелі, дані щодо загальної продуктивності сторінок, звіти щодо продуктивності веб-сайту. Також модуль відповідає за виконання операцій над звітами.

Модуль бази даних Database відповідає за зв'язок та надсилання запитів до бази даних. Модуль необхідний для підключення до бази даних, в ньому

реєструються сутності системи. Модуль надає можливість зручної інтеграції в інші модулі системи компонентів (класи-репозиторії), що надають функціонал для зручної комунікації з базою даних.

Сутності системи це моделі в програмному коді, що відповідають записам відповідних таблиць в базі даних. На основі вимог до програмного продукту, а також структури модулів системи, було спроектовано 9 сутностей, на основі яких будуть створені таблиці в базі даних. Також була створена діаграма класів сутностей системи:

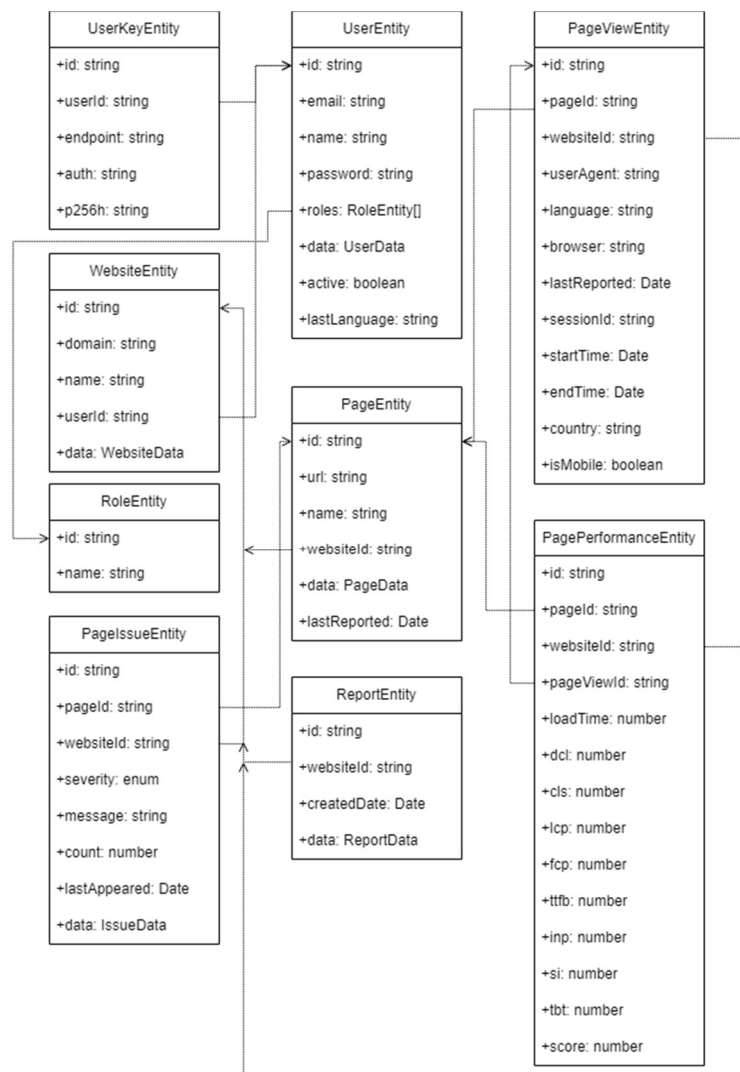


Рисунок 2.4 — Діаграма класів сутностей системи веб-аналітики

Для збереження користувачів системи веб-аналітики була створена сутність UserEntity, якій відповідає таблиця users. В даній сутності наявні такі дані про користувача як унікальний ідентифікатор, ПІБ, електронна пошта, хеш паролю, остання обрана мова вебсайту, тощо.

Доступ користувачів до певних функцій додатку визначається списком ролей користувача. Загальний список ролей зберігається в таблиці `roles`, що відповідає сутності `RoleEntity`. На даний момент присутні лише три ролі: користувач, редактор, адміністратор, список ролей може розширюватись при розширенні функціоналу системи. Зв'язок між користувачами і ролями – багато до багатьох, тому була створена допоміжна таблиця `user_to_roles`.

В системі спроектована механізм відправки «push» повідомлень, який вимагає збереження окремих даних для відправки повідомлень. Для збереження таких даних створена таблиця `usernotificationkey`, якій відповідає сутність `UserKeyEntity`. Кожен запис таблиці містить дані необхідні для відправки повідомлень, а також ідентифікатор користувача.

Сутність `WebsiteEntity` містить інформацію про веб-сайт, що піддається аналізу. До такої інформації належить назва для відображення веб-сайту, ідентифікатор власника, доменне ім'я, а також налаштування збору та аналізу даних. Дані про веб-сайти зберігаються в таблиці `websites`.

Сторінки веб-сайтів в свою чергу зберігаються в таблиці `pages`, що відповідає сутності `PageEntity`. Зв'язок з таблицею веб-сайтів багато до одного, адже один веб-сайт може мати декілька веб-сторінок. До даних сторінок що зберігаються відносяться її назва для відображення, посилання, остання дата збору даних, і ідентифікатор веб-сайту.

Дані щодо переглядів сторінок зберігаються в таблиці `page_views`, якій відповідає сутність `PageViewEntity`. До даних щодо перегляду відноситься ідентифікатори сесії перегляду, ідентифікатор веб-сайту і веб-сторінки що переглядається, а також дані щодо самого перегляду, як мова, тип веб-браузера, дати початку і кінця перегляду, чи завантажилась повністю сторінка, країна користувача і мова сторінки.

Кожен перегляд сторінки може містити до одного заміру продуктивності, дані про який зберігаються в сутності `PagePerformanceEntity`, що відповідає таблиці `page_perf`. До даних відносяться ідентифікатори сесії, веб-сайту і сторінки, а також ідентифікатор перегляду сторінки, а також значення метрик

продуктивності веб-сайту. Також обраховується оцінка продуктивності, яка також зберігається в таблиці.

Помилки та проблеми, що виникли на сторінці, а також рекомендовані покращення зберігаються в таблиці `page_issue` (сутність `IssueEntity`). До даних що зберігаються в таблиці належать кількість відтворень помилки, ідентифікатори сторінки і веб-сайту, важливість проблеми, дата останнього відтворення, а також додаткові дані, до яких належить повний текст помилки, стек помилки, тощо.

Щоденні звіти щодо продуктивності зберігаються в таблиці `reports`, що відповідають сутності `ReportEntity`. Звіт створюється щодо одного веб-сайту, тому лише містить ідентифікатор веб-сайту. Звіт містить як дані щодо продуктивності, так і виявлені фактори, що впливають на продуктивність додатку.

За допомогою сутностей було створено базу даних. Схему бази даних можна побачити на рис 2.5:

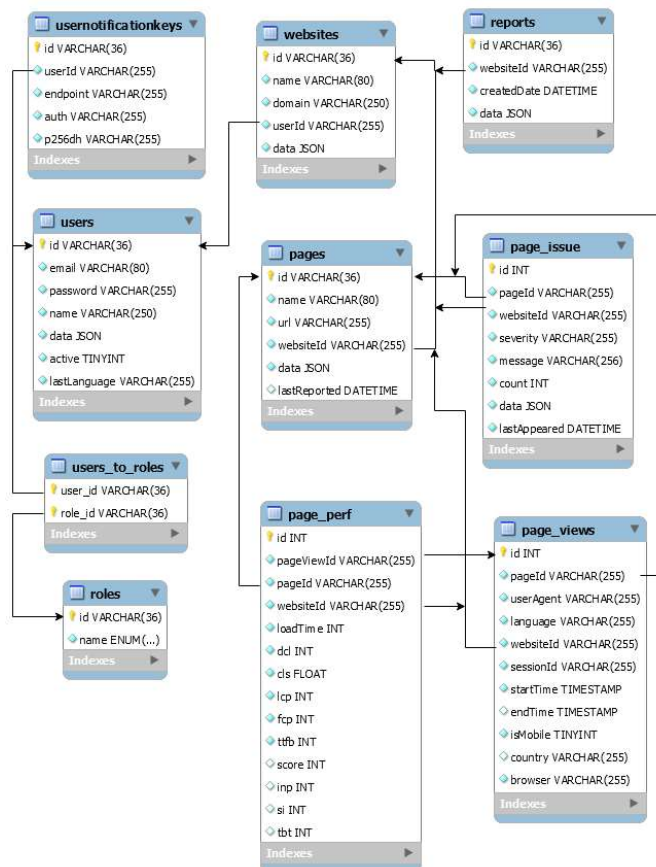


Рисунок 2.5 — Схем бази даних системи веб-аналітики

2.4. Проектування та розробка системи збору даних

Для моніторингу та аналізу продуктивності веб-сайту, необхідно здійснювати збір даних щодо його продуктивності. Як було зазначено, існують два методи моніторингу продуктивності і для отримання більш детальної картини щодо продуктивності веб-додатку, варто їх комбінувати.

В системі веб-аналітики моніторинг справжніх користувачів буде здійснюватися за допомогою скрипту, який буде вбудовуватись в веб-сайт, що піддається аналізу.

Було спроектовано алгоритм роботи скрипта для збору даних щодо продуктивності веб-сторінки:

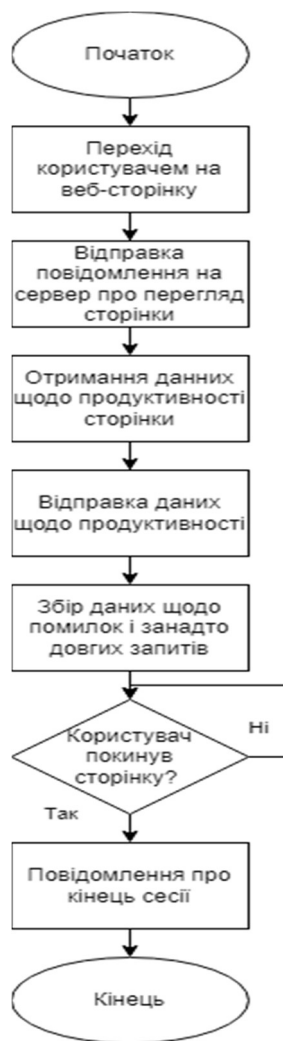


Рисунок 2.6 – Алгоритм роботи скрипту збору даних

Збір даних починає відбуватись коли користувач переходить на сторінку та завантажується скрипт збору даних, який є інтегрованим в веб-сторінку.

Скрипт розпочинає роботу з відправки запиту на сервер про початок завантаження сторінки. В відповідному запиті скрипт передає інформацію про користувача, сторінку, згенерований ідентифікатор сесії, адресу сторінки і час початку завантаження сторінки. Сервер зберігає дану інформацію та очікує дані щодо продуктивності сторінки.

Далі скрипт збирає показники щодо продуктивності веб-додатка. Сюди входить як час завантаження сторінки, так і інші метрики, що визначають продуктивність сторінки.

Як тільки скрипт отримає необхідні дані про продуктивність, він відправляє дані веб-сайту на сервер наступним запитом. Окрім даних про продуктивність, в запит містить ідентифікатор сесії згенерований раніше.

Сервер, отримавши дані про продуктивність від другого запиту, порівнює їх з даними першого запиту, перевіряє отримані дані на правильність. Якщо отримані дані проходять перевірку, сервер зберігає дані в базі даних для подальшого аналізу та відображення для користувача.

Якщо сервер отримує лише перший запит, але не отримує запит про повне завантаження, це свідчить про те що користувач покинув сторінку до повного завантаження, помилку або занадто довге завантаження.

Після відправки даних щодо продуктивності на сервер, скрипт не виконує активних дій, лише реагує на події. У разі виникнення помилки або після завантаження ресурсу, що займає надто багато часу, скрипт повідомляє сервер про це, сервер агрегує подібні помилки і повідомляє про це користувача веб-аналітики.

Користувач залишає сайт, скрипт передає серверу інформацію про завершення сесії. Це дозволяє отримати дані про тривалість сесії та визначити показник відмов за потреби.

Розглянемо детальніше процес збору даних щодо продуктивності веб-сторінок. Скрипт збирає показники метрик FCP, LCP, CLS, TTFB, час повного

завантаження та час завантаження DOM, адже саме збір цих метрик продуктивності значно не впливатиме на досвід користувача та на продуктивність веб-сторінки.

Для отримання показників цих метрик використовується Performance API [25]. Цей прикладний програмний інтерфейс використовується для вимірювання продуктивності веб-додатків. Performance API надає важливі вбудовані показники та можливість вимірювати власні показники.

Для отримання показників часу повного завантаження сторінки, часу завантаження DOM та TTFB, використовується інтерфейс PerformanceNavigationTiming [26]. Даний інтерфейс надає показники пов'язані з навігацією. На діаграмі (Рис. 2.7) можна побачити позначки часу, з яких «responseStart» відповідає TTFB [27], «domContentLoadedEventStart» – час завантаження DOM, «loadEventStart» – час повного завантаження.

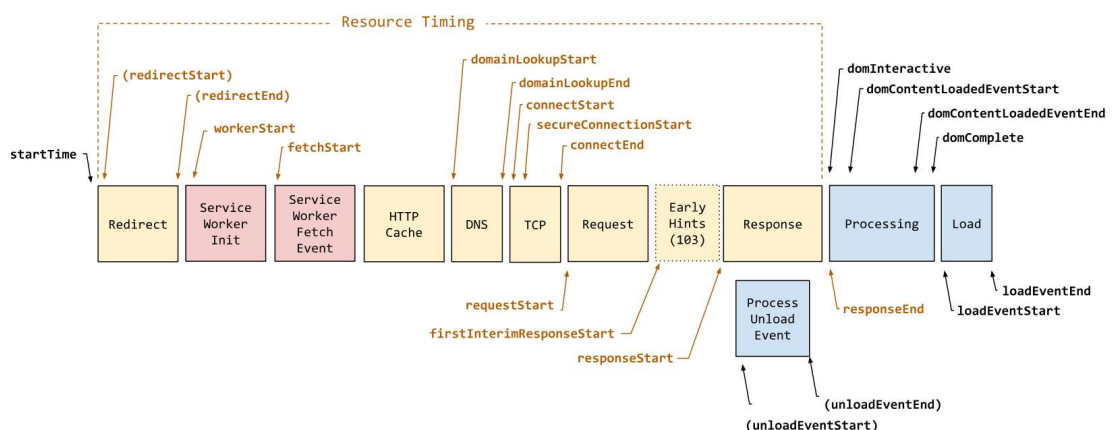


Рисунок 2.7 – Діаграма позначок часу інтерфейсу PerformanceNavigationTiming

Показники CLS, FCP та LCP можна отримати за допомогою інтерфейсу PerformanceObserver. Даний клас використовується для отримання показників продуктивності, також він сповіщає при отриманні нових записів про продуктивність веб-сторінки.

Для отримання значення метрики CLS, за допомогою PerformanceObserver збираються зміщення макету сторінки у вигляді інтерфейсу LayoutShift [2828], кожне з яких містить в собі оцінку зсуву. CLS можна визначити як суму цих

оцінок зсуву. Важливо враховувати, що зсув макету не має бути пов'язаних з дією користувача.

Приклад програмного коду для отримання значення метрики CLS наведено нижче:

```
function getCLS() {
  return new Promise(r => {
    let value = 0;
    const obs = new PerformanceObserver(list => {
      const entries = list.getEntries();
      const cls = entries.reduce((res, entry) => {
        if (!entry.hadRecentInput)
          res += entry.value;
        return res;
      }, 0)
      value += cls
    });
    setTimeout(() => { obs.takeRecords(); obs.disconnect(); r(value); },
      timeout);

    obs.observe({ type: 'layout-shift', buffered: true })
  })
}
```

Для отримання значення LCP, існує інтерфейс `LargestContentfulPaint`, значення якого і буде показником метрики. Скрипт збирає такі дані на протязі певного часу, адже відображення найбільшого елементу може відбутись не один раз під час завантаження.

Приклад програмного коду для отримання значення метрики LCP наведено нижче:

```
function getLCP() {
  return new Promise(r => {
    let value = null;
    const obs = new PerformanceObserver(list => {
      const entries = list.getEntries();
      const lcpEntry = entries[entries.length - 1];
      if (!lcpEntry)
        return;
      value = lcpEntry.renderTime || lcpEntry.loadTime;
    });
    setTimeout(() => { obs.takeRecords(); obs.disconnect(); r(value); },
      timeout);

    obs.observe({ type: 'largest-contentful-paint', buffered: true })
  })
}
```

Інтерфейс `PerformancePaintTiming` [29] надає інформацію про відображення контенту. За допомогою даного інтерфейсу можна отримати значення метрик FCP, а також FP (first paint).

Приклад програмного коду для отримання значення метрики FCP наведено нижче:

```
function getFcp() {
  return new Promise(r => {
    const clear = resolveOnTimeout(r, timeout);
    const obs = new PerformanceObserver(list => {
      const entries = list.getEntries();
      const paintEntry = entries.find(e => e.name === 'first-contentful-
paint')
      if (paintEntry) {
        clear();
        obs.disconnect();
        r(paintEntry.startTime);
      }
    })
    obs.observe({ type: 'paint', buffered: true })
  });
}
```

Синтетичний моніторинг в системі здійснюватиметься за допомогою сервісу Lighthouse. Google Lighthouse — це інструмент синтетичного аналізу, який використовується для аудиту продуктивності, якості та оптимізації веб-сторінок.

Для проведення синтетичного моніторингу, Lighthouse здійснюватиме аудит кожної сторінки. Для цього буде виконано запуску веб-браузер в «Headless» режимі, тобто без графічного інтерфейсу користувача. Далі, буде запуску аудит сторінки за допомогою Lighthouse, який завантажить сторінку, виконає аудити продуктивності і створить звіт на основі проведеного аналізу [30].

В системі буде реалізований модуль запланованих завдань, кожне з яких матиме своє призначення і виконуватиметься періодично. Одне з таких завдань відповідатиме за синтетичний моніторинг веб-додатку. Після планового запуску завдання (або якщо користувач запусить завдання за допомогою інтерфейсу користувача), система визначить список сторінок, синтетичний аналіз яких не

проводився певний період часу. В рамках завдання система проводитиме аудит кожної з сторінок по чергово за допомогою Lighthouse.

Після отримання звіту від сервісу Lighthouse, система веб-аналітики збереже дані щодо продуктивності веб-сторінки зібрані синтетичною. Такі дані будуть позначені в базі даних як зібрані сервісом Lighthouse. До метрик, що будуть отримані в ході синтетичного аналізу належать: FCP, LCP, CLS, TTFB, Speed index, TBT, час завантаження сторінки.

Також система проведе аналіз проведених сервісом аудитів та визначить список рекомендацій, можливостей для оптимізації та помилок що були помічені під час аналізу сторінки. Даний список буде збережений в базу даних як проблеми веб-сторінки. Користувач веб-аналітики зможе побачити список рекомендацій в звіті щодо продуктивності та для кожної сторінки окремо.



Рисунок 2.8 – Алгоритм проведення синтетичного моніторингу

2.5. Проектування та розробка системи аналізу даних

Після впровадження алгоритмів збору та збереження даних необхідно надати користувачам доступ до цих даних у зручний спосіб для подальшого аналізу. Дані що збираються системою можна поділити на дані щодо відвідування веб-додатку, дані щодо продуктивності, а також проблеми веб-сторінок (сюди входять як можливості для покращення, так і помилки, ресурси завантаження яких зайняло занадто довгий час, та інше.

Для швидкого і ефективного аналізу продуктивності веб-додатків важливо надати дані у зручному для користувача вигляді. При аналізі продуктивності та для оцінки загальної картини роботи веб-сайту для користувача веб-аналітики важливими значеннями будуть статистичні дані щодо продуктивності, як мінімальні, максимальні та середні значення метрик. Також корисними можуть бути значення метрик відносно часу.

Важливою для аналізу продуктивності веб-сайті є сегментація даних, що надасть можливість порівняти продуктивність веб-сайту на різних пристроях, для різних типів користувачів, браузерів чи у різні часові проміжки. Сегментація дозволяє глибше зануритися в аналіз і виявити специфічні проблеми, які можуть залишатися непомітними при загальному огляді. Наприклад, низька швидкість завантаження сторінок може бути характерною лише для мобільних пристроїв або для користувачів із певного регіону. Завдяки фільтрації даних вдасться не лише знаходити такі аномалії, але й оцінювати ефективність внесених змін у різних умовах.

Візуалізація статистичних даних та сегментація даних дозволить користувачу веб-аналітики швидко проаналізувати дані та виявити проблеми в продуктивності веб-сайту.

Для такого аналізу даних було вирішено реалізувати інформаційну панель, що включатиме статистичні дані метрик продуктивності веб-сайту, відображати зміни продуктивності відносно часу, а також надаватиме можливість сегментації даних відносно сторінки, браузера, типу користувача чи пристрою.

Для пришвидшення аналізу значень метрик, хорошою практикою буде виділення значень що перевищують норму та потребують покращення.

Щоб відобразити зміни в значеннях метрик відносно часу найкращим засобом візуалізації буде графік, де вісь X відображатиме час, а Y – значення метрики. В такому випадку для відображення графіку необхідно отримати дані метрик для кожного проміжку часу. Відображення всіх даних щодо продуктивності на графіку не є оптимальним, адже це спричинить обмін великою кількістю даних між клієнтом та сервером.

Було вирішено групувати дані за часом їх отримання. Була обрана максимальна кількість записів що буде відображатись на графіку. Якщо кількість даних є меншою за максимальну кількість, вони всі будуть відображені на графіку, інакше – групуватись.

Далі було визначено ключ групування (яким виступатиме дата та час збору) відносно проміжку часу за який будуть відображені дані. Для визначення ключа групування для проміжку часу, час ділиться на максимальну кількість записів і округлюється до найближчої логічної для сприйняття проміжку часу. Наприклад якщо запитуються дані за проміжок часу менший ніж доба, оптимально буде групувати дані за ключем що є позначкою часу з точністю в 15 хвилин, адже в такому випадку максимальна кількість записів що буде відображена – 96 (добуток 24 годин в добі на 4 позначки часу кожні 15 хвилин в годині). За значення що буде відображатись для кожної такої позначки виступатиме середнє значення для цього проміжку. Відповідним чином були визначені ключі групування і для більших проміжків часу.

Таке групування буде зрозумілим для користувача і не викликатиме обмін великою кількістю даних для візуалізації.

Алгоритм запиту та відображення даних на інформаційній панелі можна побачити на рис 2.9.



Рисунок 2.9 – Алгоритм відображення даних на інформаційній панелі

Ручний аналіз продуктивності веб-додатків не завжди є ефективним і часто залежить від людського фактору. Також ручний аналіз є ресурсозатратним процесом, тому потребує автоматизації.

Було вирішено розробити алгоритм аналізу продуктивності веб-сайту, результатом роботи якого буде звіт з показниками метрик продуктивності, факторами, що впливають на продуктивність веб-додатку, та проблемами з якими стикаються веб-сторінки.

Щоденні звіти будуть формуватись окремо для кожного веб-сайту з даних зібраних системою веб-аналітики на протязі доби.

Тобто вхідними даними до алгоритму виступатимуть дані щодо продуктивності веб-сторінок зібрані системою за певний період часу, зібрані синтетично чи від справжніх користувачів, список метрик продуктивності за якими здійснюватиметься аналіз, список фільтрів, за якими будуть групуватись дані для аналізу, а також мінімальні показники метрик, що вважаються допустимими.

Також до вхідних даних належать два коефіцієнти, які можна буде налаштувати. Зміна значень коефіцієнтів допоможе збільшити точність та кількість результатів в залежності від веб-сайту. Одним з таких коефіцієнтів є поріг відсіву комбінації (за замовчуванням було визначено 0,001), який визначає скільки разів мають бути зібрані дані з певною комбінацією факторів, щоб вона враховувалась алгоритмом. Другим є коефіцієнт відчутного зменшення продуктивності (за замовчуванням має значення 1,05), що визначає в скільки разів комбінація факторів має мати більше значення метрики, щоб враховуватись алгоритмом.

Також додатково для створення звіту використовується список проблем пов'язаних з веб-сторінкам за проміжок часу.

Алгоритм складатиметься з 3 основних етапів, підготовка даних, аналіз даних, очищення та нормалізація даних..

На етапі підготовки даних, система веб-аналітики отримує з бази даних дані щодо продуктивності веб-сайту за останню добу. Для аналізу кожного заміру продуктивності окремо потребується велика кількість ресурсів, тому при отриманні даних, вони групуються за значеннями фільтрів, вираховуються середні значення метрик та кількість записів для кожної групи. Таким чином буде отримано масив що складається з об'єктів що містять унікальний набір значень фільтрів та середнє значення метрик при таких фільтрах.

Під час наступного етапу проводиться аналіз значень кожної метрики. Згруповані записи щодо продуктивності піддаються фільтрації, обираються лише ті значення, що мають показник метрики нижче допустимого.

Для кожної такої групи проводиться пошук найбільш широкого фільтру з найгіршим середнім значенням метрики. Тобто, спочатку створюється фільтр зі значеннями поточної групи та знаходиться середнє значення метрики. Після чого з фільтру прибирається 1 параметр, відбується пошук всіх груп що підпадають під такий фільтр та обчислюється середнє значення. Якщо відношення середнього значення метрики для нової групи до значення для поточного фільтру є більшим за коефіцієнт відчутного зменшення продуктивності, то фільтр нової групи вважається більш широким з найгіршим середнім значенням. Процес повторюється рекурсивно для кожної знайденої групи, до поки фільтр не лишиться порожнім. Результуючі фільтри визначають комбінацію факторів, що негативно впливають на продуктивність веб-сайту.

Результуючі фільтри очищаються від факторів, що є однаковими для більшості груп. Далі відбуваються перевірки значень метрики для групи, чи є вони вище середніх та норми. Також перевіряється чи кількість записів щодо продуктивності, що відповідає поточному фільтру більша за добуток порогу відсіву комбінації на загальну кількість записів. У разі проходження перевірок, фільтр комбінації вважається одним з результатів роботи даного етапу та переходить в наступний.

Алгоритм роботи етапу аналізу даних можна побачити на рисунку 2.10.

Розглянемо роботу даного етапу на прикладі. Вхідними даними щодо продуктивності виступатиме вибірка даних з таблиці 2.1, з огляду на які можна визначити що користувачі стикаються з високим значенням метрики на головній сторінці при використанні мобільного пристрою:

Таблиця 2.1

Вибірка даних для демонстрації роботи алгоритму

№	Сторінка	Тип пристрою	Браузер	Значення метрики	К-сть записів
1	Головна	Мобільний	Chrome	5000	2
2	Головна	Мобільний	Firefox	6000	5
3	Головна	Настільний ПК	Chrome	2000	10
4	Новини	Мобільний	Firefox	2500	3

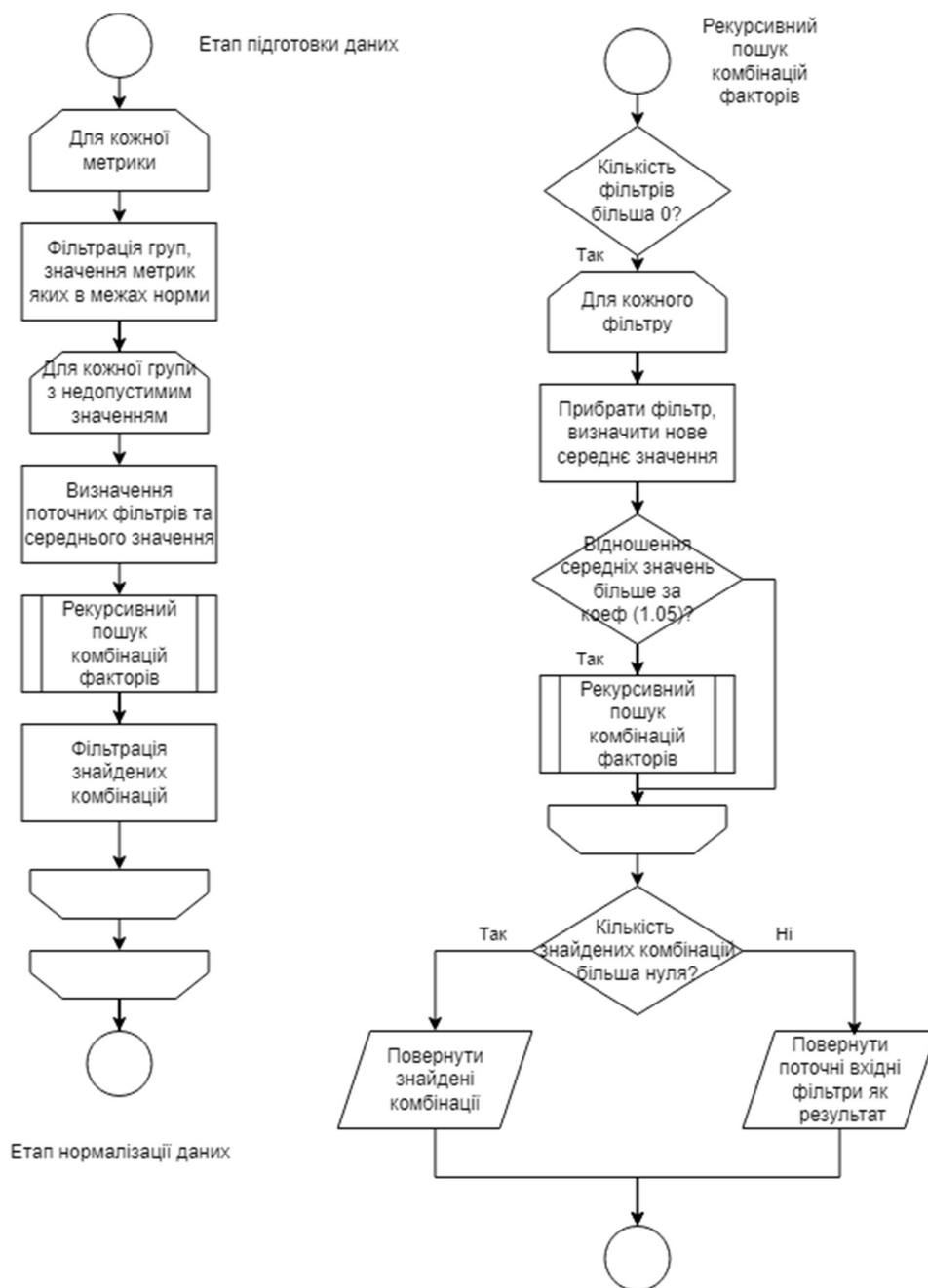


Рисунок 2.10 – Алгоритм етапу аналізу даних продуктивності груп

Поточна група №1, створюємо фільтр групи, яким є сторінка «Головна», тип пристрою «Мобільний», браузер «Chrome». Середнє значення метрики 5000 мілісекунд.

Приберемо фільтрацію по браузерам з поточного фільтру. В такому випадку фільтром є сторінка «Головна», тип пристрою «Мобільний», тепер під фільтр підпадають 2 групи – №1 та №2. Визначимо середнє значення метрики для групи яким буде відношення сум добутків значень метрик на кількість записів до сумарної кількості записів, і в даному випадку становитиме приблизно

5700 мілісекунд, відношення до поточного 1.14, що є більшим значенням за коефіцієнт відчутного зменшення продуктивності.

Відбувається рекурсія і далі з фільтру видаляється тип пристрою, тоді значенням фільтру виступатиме лише сторінка «Головна», в такому випадку під значення фільтру підпадають перші 3 записи, і середнім значенням стане приблизно 3500 мілісекунд, що є меншим за поточне найбільше значення. Подальші видалення фільтрів також будуть видавати значення менші за знайдене. Тому результатом роботи алгоритму тут стане фільтр сторінка «Головна» і тип пристрою «Мобільний». Таким чином користувач системи веб-аналітики, що аналізуватиме сформований звіт, зможе визначити, що відвідувачі головної сторінки з використанням мобільного пристрою стикаються з проблемами в продуктивності.

Процес повторюється для всіх параметрів фільтрів на різних рівнях рекурсії та для інших метрик для виявлення проблем в більш комплексних наборах даних. Знайдені комбінації факторів передаються в наступний етап очищення та нормалізації даних перед збереженням в звіт.

На етапі очищення та нормалізації даних, проводиться аналіз зібраних комбінацій фільтрів та пошук дублікатів. Також такі фільтри групуються за метриками, на які вони впливають негативно. Таким чином на даному етапі результатом роботи алгоритму буде вибірка комбінацій факторів, при яких продуктивність веб-сайту погіршуються. Вказуються метрики на які впливають такі фактори, а також значення цих метрик.

За допомогою даної вибірки визначаються проблематичні сторінки. Якщо сторінка є одним з факторів що негативно впливають на загальну продуктивність веб-сайту, вона вважається проблематичною. Для таких сторінок здійснюється пошук проблем та можливостей для покращення зібраних системою за останню добу. Така інформація в звіті допоможе значно прискорити аналіз продуктивності веб-сайту.

Також на даному етапі обчислюються значення метрик для веб-сайту в цілому, а також для кожної комбінації факторів.

Отже результатом роботи алгоритму стане вибірка інформації, що містить список комбінацій факторів, що негативно впливають на продуктивність веб-сайту та метрики на які вони впливають негативно. Також для кожної комбінації факторів та для веб-сайту в цілому додаються середні значення метрик продуктивності та список проблем пов'язаних з ними.

Такий набір інформації буде достатнім для користувача веб-аналітики для швидкого та ефективного аналізу продуктивності веб-додатку та визначення проблемних місць в продуктивності.

Висновки до другого розділу

В даному розділі була визначена архітектура системи веб-аналітики та її основні компоненти, а саме скрипт для збору даних справжніх користувачів, сервер, що надаватиме доступ до даних та веб-інтерфейс, що дозволить керувати системою та переглядати дані у зручний спосіб.

Визначені засоби реалізації системи, функціональні та нефункціональні вимоги, а також вимоги до апаратного забезпечення. Визначені варіанти використання системи, та її основні модулі, які є структурними частинами у програмному коді додатку. Спроектowana та реалізована база даних системи, а також класи сутностей системи.

Реалізована система моніторингу продуктивності веб-сайту, що комбінує підходи до моніторингу продуктивності. Створений скрипт для збору даних щодо продуктивності використовує підхід моніторингу справжніх користувачів. Реалізована система періодичного синтетичного моніторингу продуктивності за допомогою інструмента Lighthouse.

Реалізована система аналізу продуктивності веб-сайту, в тому числі інформаційна панель, що дозволяє візуалізувати дані для користувача сервісу веб-аналітики. Реалізований алгоритм, що дозволяє створити звіт щодо продуктивності веб-сторінки на основі зібраних даних, з висвітленням комбінацій факторів, що впливають на продуктивність веб-сайту.

РОЗДІЛ 3. ОПИС ПРОГРАМНОГО ПРОДУКТУ

3.1 Графічний інтерфейс сторінки

Після переходу за посилання сервісу веб-аналітики, користувач побачить головну сторінку з коротким описом системи, на сторінці присутнє посилання для зворотнього зв'язку, меню для вибору мови інтерфейсу системи, а також меню облікового запису, в якому наявні 2 посилання, на сторінку реєстрації та входу в обліковий запис.

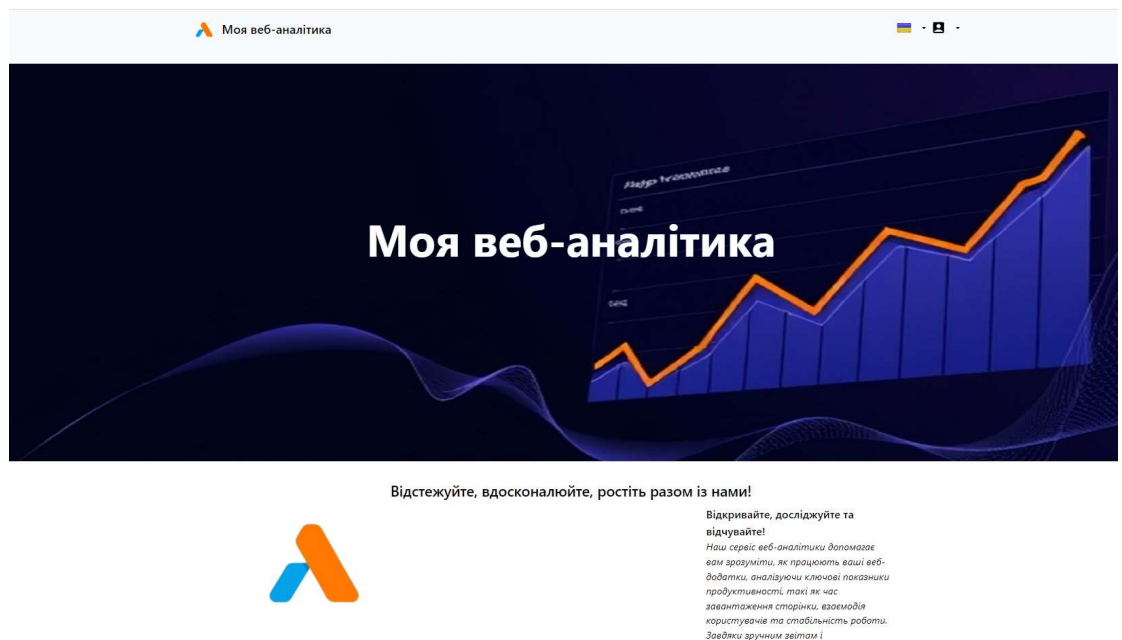


Рисунок 3.1 – Головна сторінка сервісу веб-аналітики

Для того, щоб користуватись системою, необхідно увійти в обліковий запис скориставшись формою (рис. 3.2). Якщо у користувача відсутній обліковий запис, то він може створити новий перейшовши на форму реєстрації.

Рисунок 3.2 – Сторінка реєстрації

Увійшовши в обліковий запис, користувач має можливість вийти з нього або змінити пароль. Для повноцінного використання системи, користувачу необхідно отримати доступ до панелі керування від адміністрації сервісу веб-аналітики. Після отримання доступу, у користувача з'явиться посилання на сторінку панелі керування на головній сторінці.

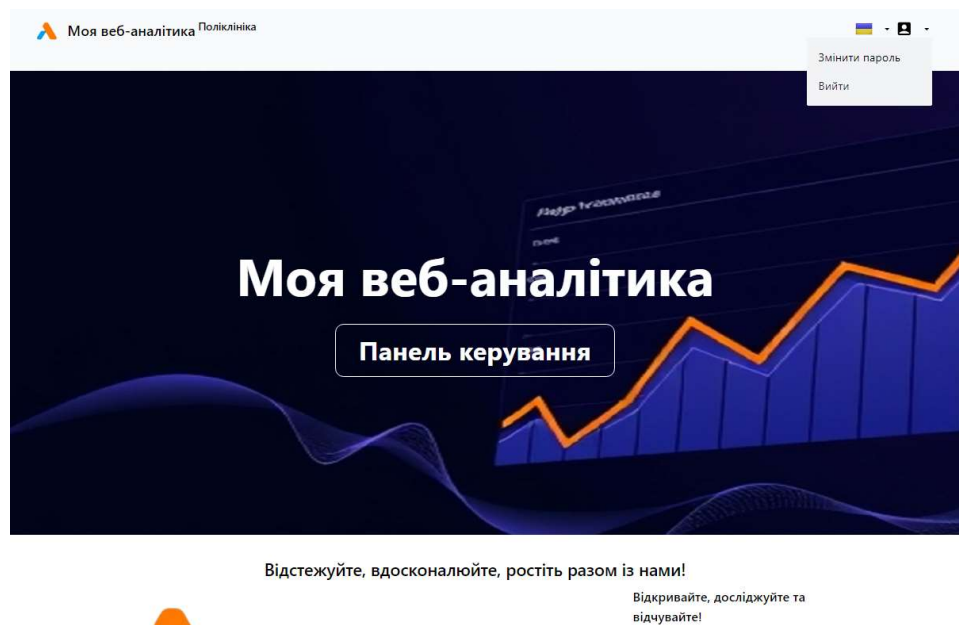


Рисунок 3.3 – Головна сторінка користувача, що увійшов в обліковий запис

Після переходу за посиланням, користувач опиниться на сторінці панелі керування, де зможе знайти меню навігації та корисну інформацію, а також скрипт для інтеграції в веб-сайти.

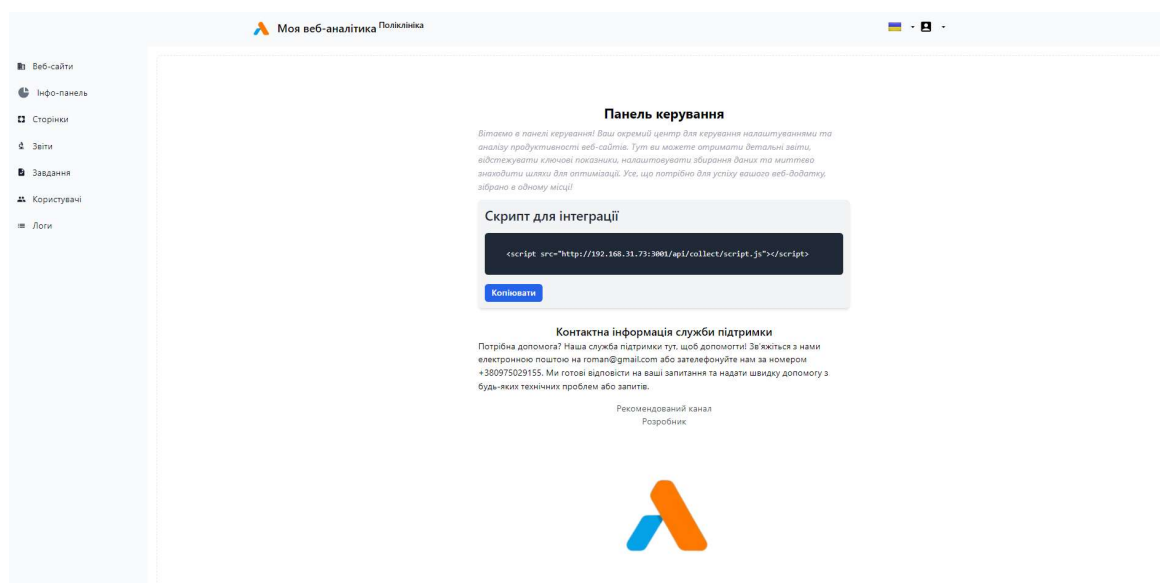


Рисунок 3.4 – Панель керування

Натиснувши на посилання «Веб-сайти» в меню навігації, користувач потрапить на сторінку списку веб-сайтів, де зможе створити, видалити, відредагувати або обрати веб-сайт, що піддається моніторингу та аналізу системою. Список веб-сайтів є унікальним для кожного користувача, тобто кожен користувач відповідає за свою групу веб-сайтів.

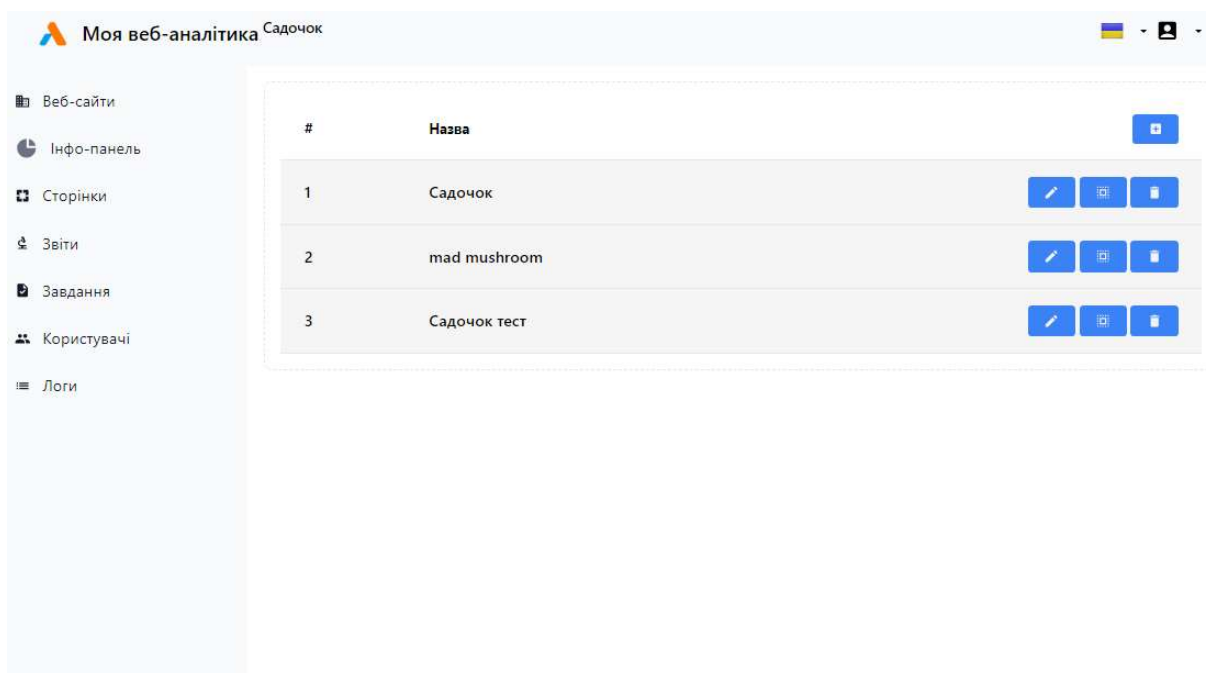


Рисунок 3.5 – Список веб-сайтів користувача

Зі списку веб-сайт можна видалити натисканням на кнопку корзини. Для операцій, що передбачають видалення даних, в веб-інтерфейсі передбачене попереджувальне повідомлення:

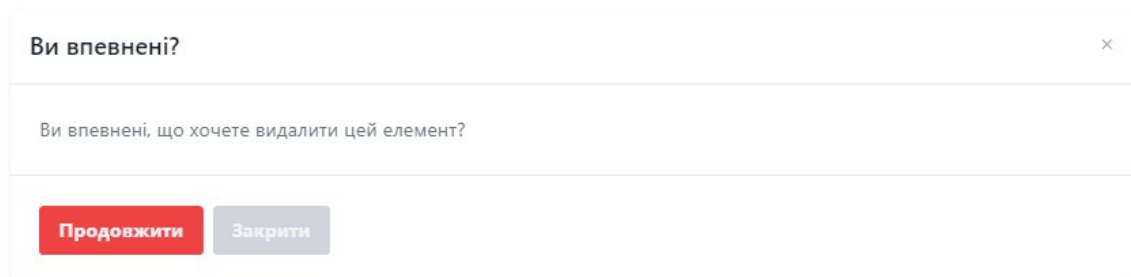


Рисунок 3.6 – Попереджувальне повідомлення при видаленні

Натиснувши на кнопку «Редагувати», користувач потрапить на сторінку редагування веб-сайту, де зможе редагувати інформацію про веб-сайт. Після

того, як веб-сайт був створений, веб-сервіс починає моніторинг його продуктивності.

The screenshot shows the 'Моя веб-аналітика' (My Web Analytics) interface. On the left is a sidebar with navigation links: Веб-сайти, Інфо-панель, Сторінки, Звіти, Завдання, Користувачі, and Логі. The main area displays a form titled 'Редагувати веб-сайти' (Edit Websites). The form has two input fields: 'Назва' (Name) with the value 'mad mushroom' and 'Домен' (Domain) with the value 'https://www.madmushroom.gg'. A blue 'Відправити' (Send) button is at the bottom right of the form.

Рисунок 3.7 – Редагування веб-сайту

Моніторинг справжніх користувачів починає проводитись після створення веб-сайту та інтеграції скрипта в розмітку веб-сторінок веб-сайту. Синтетичний моніторинг проводиться для всіх існуючих в системі сторінок веб-сайту.

Для того щоб обрати веб-сайт для аналізу, користувач має натиснути кнопку вибору веб-сайту, після чого назва веб-сайту відображатиметься в верхній частині сервісу.

The screenshot shows the 'Моя веб-аналітика' (My Web Analytics) interface. The top header bar now displays 'Моя веб-аналітика mad mushroom', with a red arrow pointing to the 'mad mushroom' text. Below the header is a table with three columns: '#', 'Назва', and a set of action icons. The table contains three rows of data.

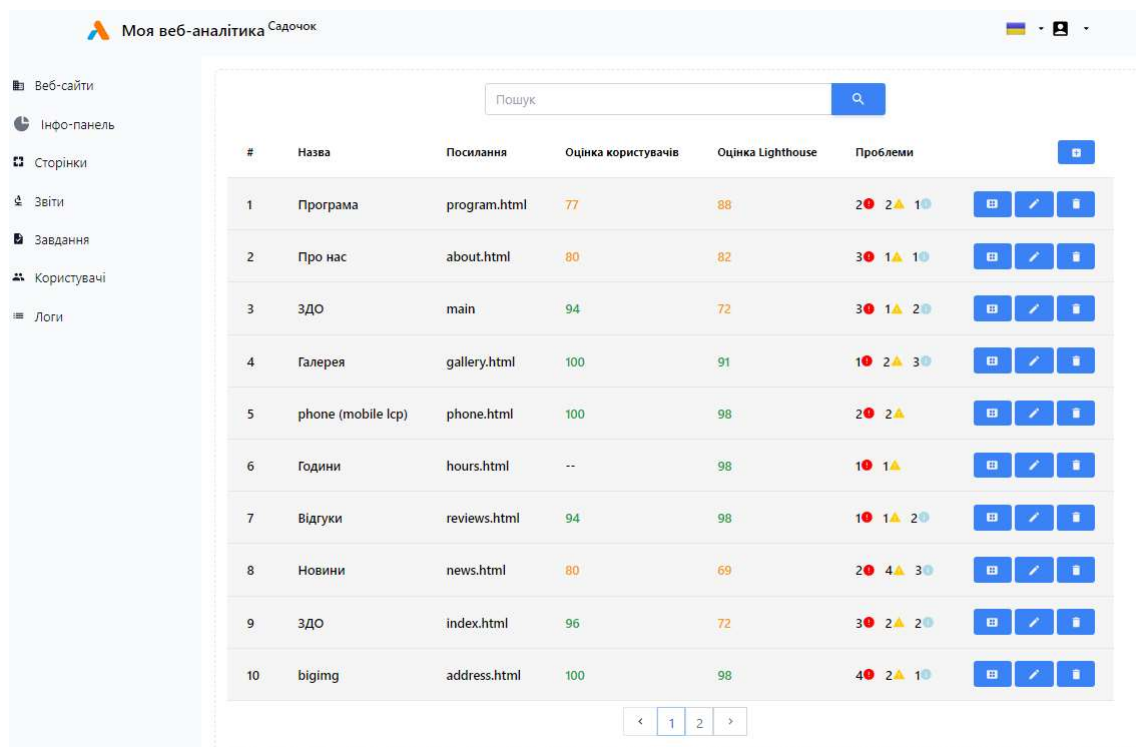
#	Назва	Дії
1	Садочок	[Edit] [Info] [Delete]
2	mad mushroom	[Edit] [Info] [Delete]
3	Садочок тест	[Edit] [Info] [Delete]

A red arrow points to the 'mad mushroom' entry in the table, specifically to the 'Info' icon.

Рисунок 3.8 – Вибір веб-сайту для аналізу

Інформація, що відображається на сторінках «Інформаційна панель», «Звіти» та «Сторінки» залежить від обраного веб-сайту і відображається лише для нього. Для того щоб переглянути інформацію для іншого веб-сайту, необхідно обрати його в списку веб-сайтів.

В списку сторінок відображається інформація про сторінки обраного веб-сайту. Сторінки можуть бути створені користувачем системи веб-аналітики або автоматично згенеровані при зборі скриптом даних щодо продуктивності веб-додатку.



#	Назва	Посилання	Оцінка користувачів	Оцінка Lighthouse	Проблеми	
1	Програма	program.html	77	88	2 ● 2 ▲ 1 ●	
2	Про нас	about.html	80	82	3 ● 1 ▲ 1 ●	
3	ЗДО	main	94	72	3 ● 1 ▲ 2 ●	
4	Галерея	gallery.html	100	91	1 ● 2 ▲ 3 ●	
5	phone (mobile lcp)	phone.html	100	98	2 ● 2 ▲	
6	Години	hours.html	--	98	1 ● 1 ▲	
7	Відгуки	reviews.html	94	98	1 ● 1 ▲ 2 ●	
8	Новини	news.html	80	69	2 ● 4 ▲ 3 ●	
9	ЗДО	index.html	96	72	3 ● 2 ▲ 2 ●	
10	bigimg	address.html	100	98	4 ● 2 ▲ 1 ●	

Рисунок 3.9 – Список сторінок веб-сайту

Для кожної сторінки відображається інформація щодо її продуктивності, а саме середній бал продуктивності отриманий при моніторингу справжніх користувачів та синтетично за останню добу, а також кількість помилок, попереджень та інформаційних повідомлень зібраних системою.

Також відображається назва та відносна адреса сторінки. При автоматичному створенні сторінки, її назва буде отримана з заголовку веб-сторінки.

Кожну з сторінок можна видалити, відредагувати або переглянути список проблем пов'язаних з нею.

Редагувати сторінку

Назва

Програма

Посилання

program.html

Відправити

Рисунок 3.10 – Форма редагування веб-сторінки в системі

Для кожної сторінки можна переглянути список проблем, що були виявлені в ході синтетичного моніторингу а також від справжніх користувачів. Система накопичує інформацію про помилки, HTTP-запити отримання відповіді яких зайняло занадто довгий час, а також рекомендації сервісу Lighthouse та відображає їх в даному списку.

Моя веб-аналітика Садочок

Веб-сайти

Інфо-панель

Сторінки

Звіти

Завдання

Користувачі

Логи

Про нас (about.html)

Проблеми

❌paq is not defined (36)

Останнє виникнення: 12/6/2024, 12:35:30 AM

Повне повідомлення

paq is not defined

Стек

ReferenceError: paq is not defined at http://192.168.31.73:3181/about.html:29:11

Заглушити

⚡Max Potential First Input Delay (24)

Останнє виникнення: 12/5/2024, 11:31:40 PM

Повне повідомлення

240 ms The maximum potential First Input Delay that your users could experience is the duration of the longest task. [Learn more](https://web.dev/lighthouse-max-potential-fid/).

Заглушити

❌Cumulative Layout Shift (24)

Останнє виникнення: 12/5/2024, 11:31:40 PM

Детальніше

Заглушити

❌Ensure text remains visible during webfont load (24)

Останнє виникнення: 12/5/2024, 11:31:40 PM

Детальніше

Заглушити

⚡Eliminate render-blocking resources (24)

Останнє виникнення: 12/5/2024, 11:31:40 PM

Детальніше

Заглушити

Рисунок 3.11 – Список проблем пов’язаних з сторінкою

Кожна проблема має короткий опис та дату останнього виникнення. Якщо проблема виникала більше ніж один раз, буде відображатись число, яке відповідає кількості виниклих проблем. Є можливість завантажити повну інформацію про кожну проблему, а також видалити проблему зі списку.

На сторінці інформаційної панелі відображаються графіки продуктивності веб-сайту відносно часу. Метрики, що вимірюються часом відображаються на головному графіку.

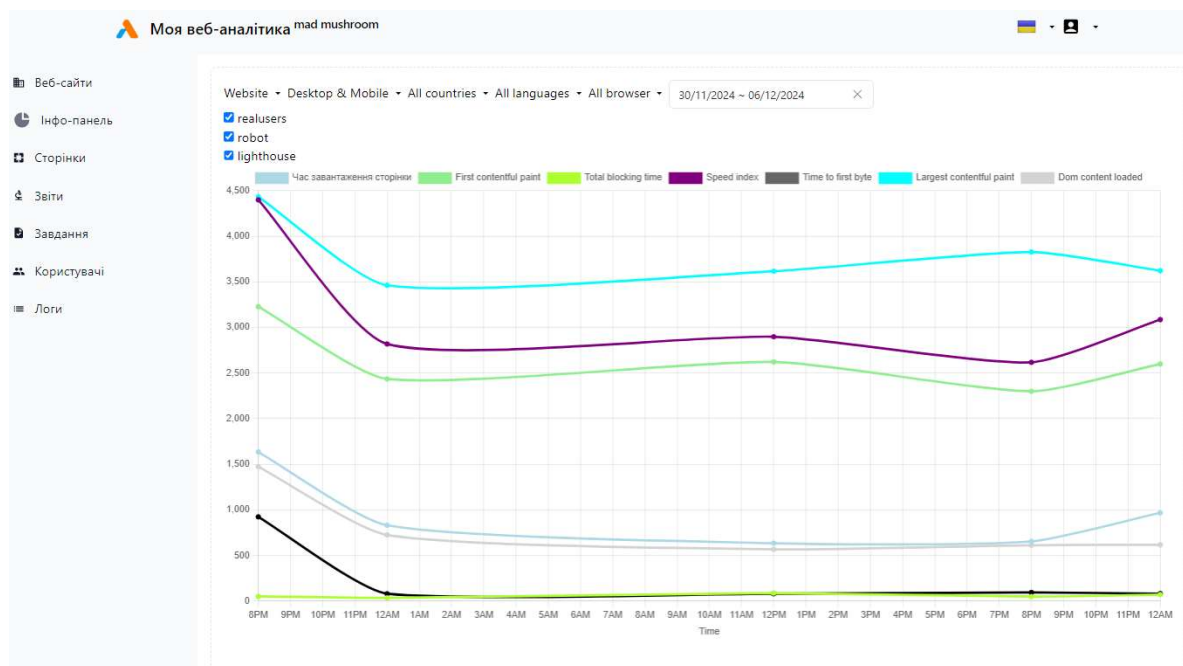


Рисунок 3.12 – Графік продуктивності веб-сайту відносно часу на сторінці «Інформаційна панель»

Оцінка продуктивності веб-додатку та CLS відображаються на окремих графіках, адже мають інші діапазони.

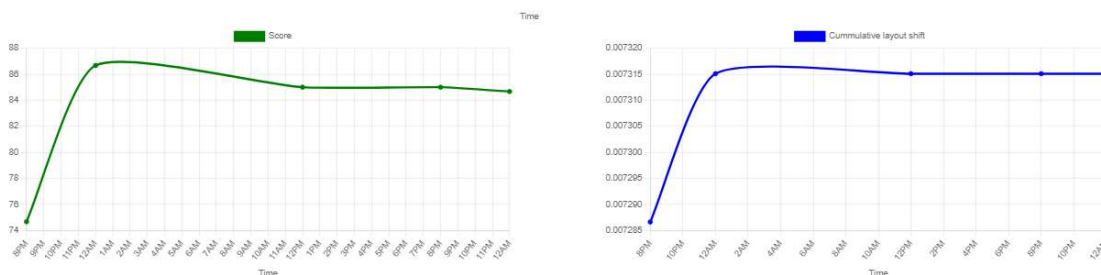


Рисунок 3.13 – Графіки оцінки продуктивності веб-додатку та CLS на сторінці «Інформаційна панель»

Також на сторінці інформаційної панелі відображаються середні, мінімальні та максимальні значення метрик продуктивності. Кожне значення підсвічується відповідним кольором в залежності від значення метрики.

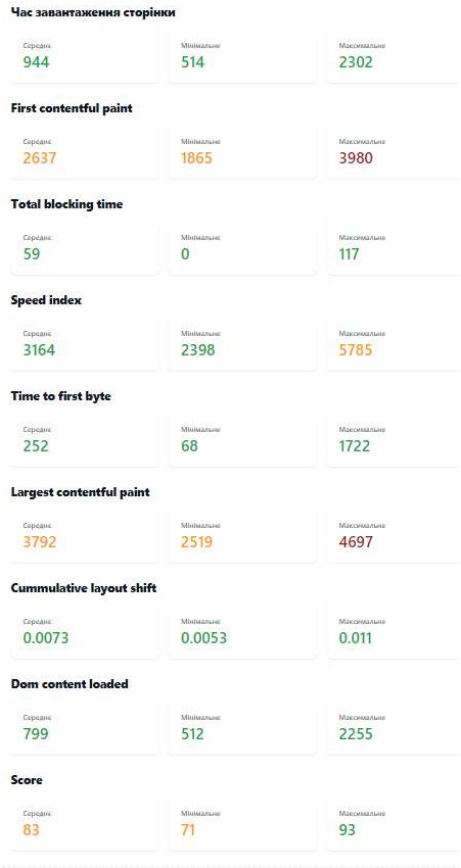


Рисунок 3.14 – Метрики продуктивності веб-сайту на сторінці «Інформаційна панель»

Всі дані в інформаційній панелі відображаються відносно фільтрів, що задаються вгорі сторінки.

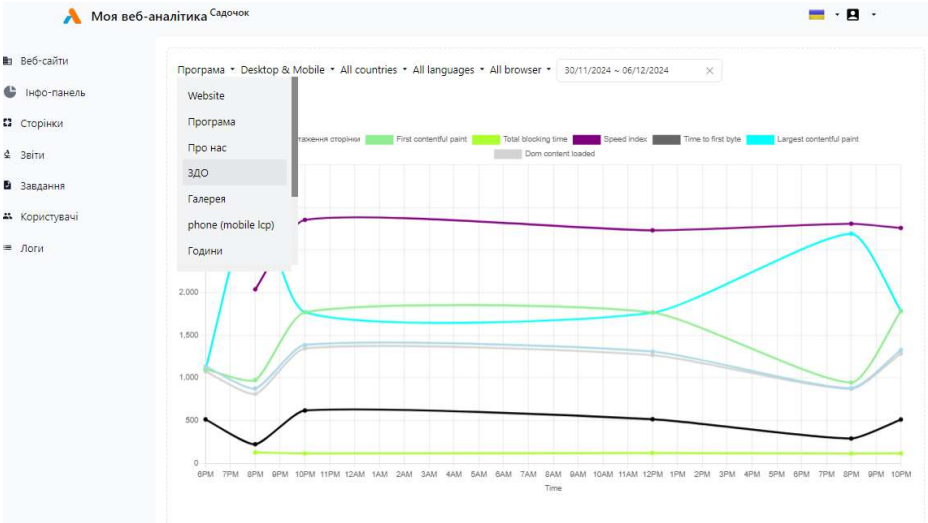


Рисунок 3.15 – Фільтрація даних за сторінкою

Також на сторінці є можливість фільтрації даних за типом пристрою, країною, мовою користувача, браузером, типом користувача та датою.

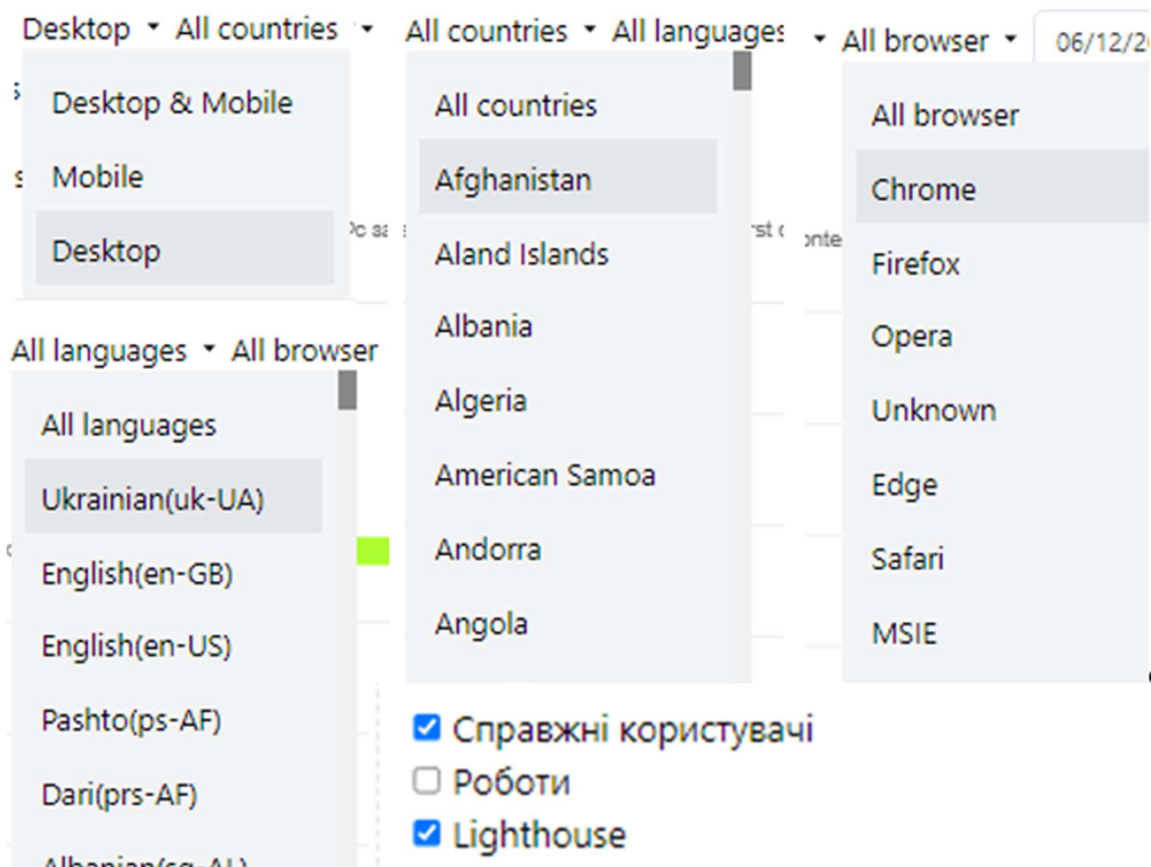


Рисунок 3.16 – Фільтрація даних за типом пристрою, країною, браузером, мовою і типом користувача

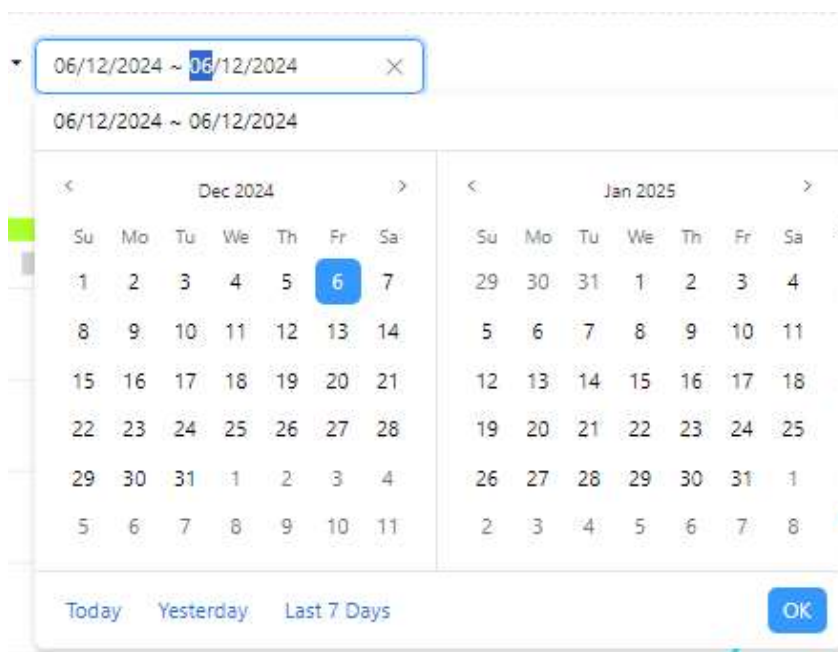


Рисунок 3.17 – Фільтрація даних за датою, коли дані були отримані

Після вибору фільтру користувачем, графіки та значення метрики продуктивності на сторінці оновляться, будуть відображатись лише дані, що підпадають під обрані фільтри.

При переході на сторінку «Завдання», користувач побачить список запланованих завдань, що виконуються періодично, або можуть бути запущені користувачем.

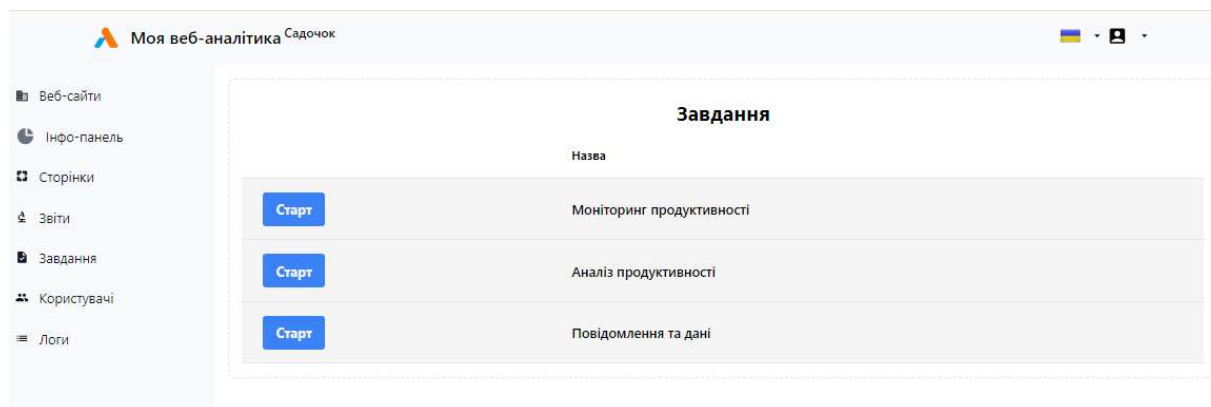


Рисунок 3.18 – Список завдань

Завдання «Моніторинг продуктивності» запускається автоматично кожні 30 хвилин та відповідає за синтетичний моніторинг веб-сайтів. Під час роботи завдання збирає дані про продуктивність набору сторінок, моніторинг продуктивності яких не здійснювався найдовше. Для кожної сторінки запускається сервіс Lighthouse, який здійснює синтетичний моніторинг сторінки і створює звіт, з якого система веб-аналітики отримує проблеми в продуктивності веб-сайту та рекомендації для покращення, а також значення метрик продуктивності.

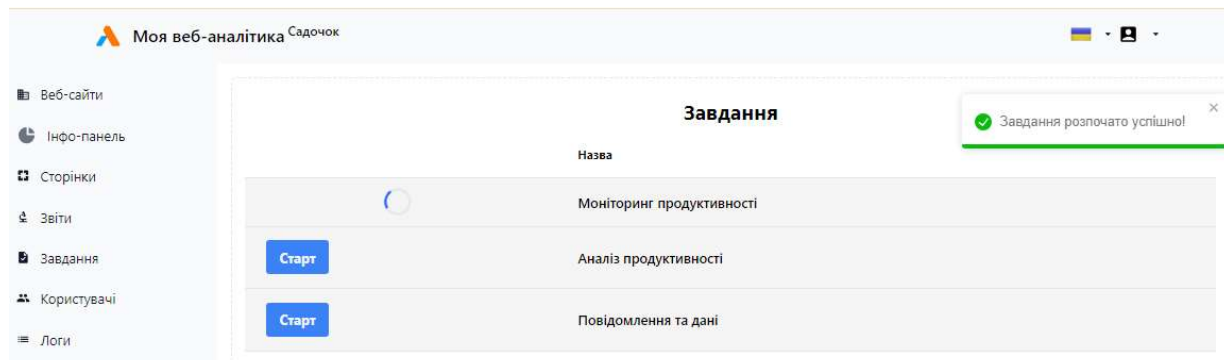


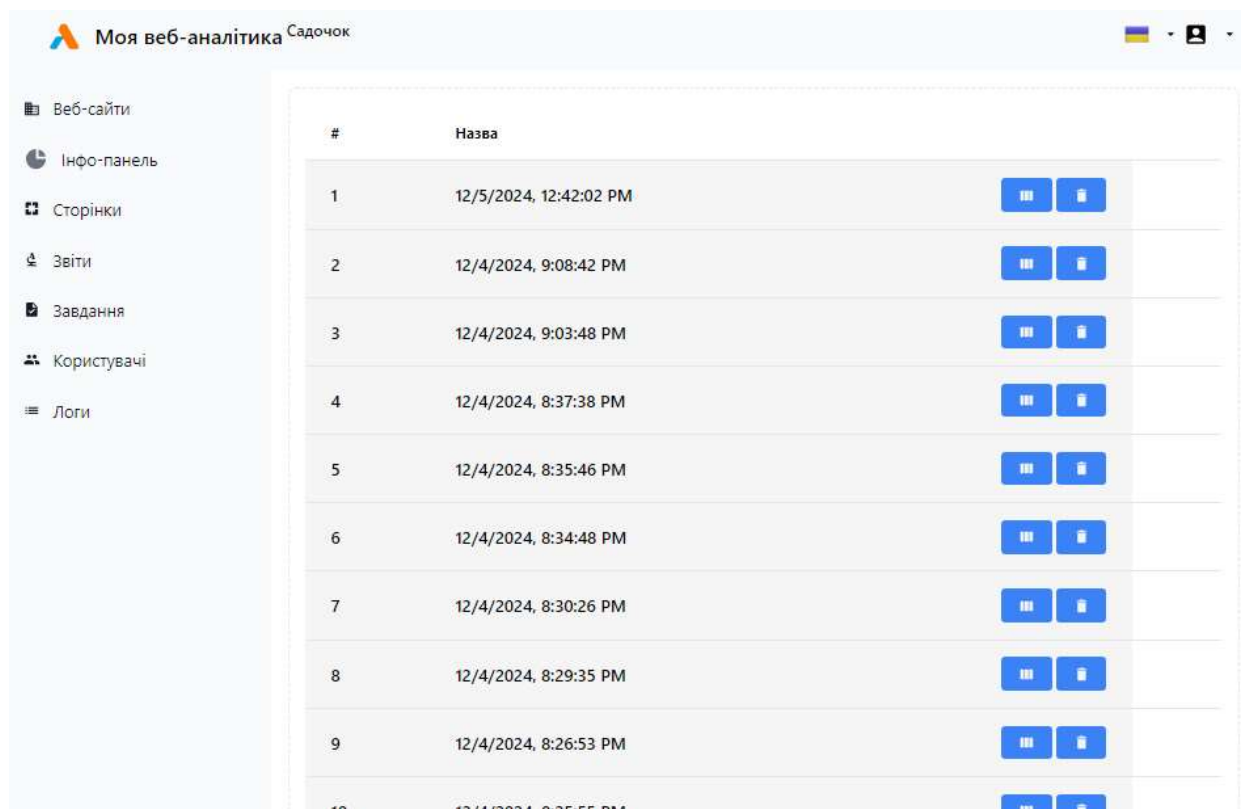
Рисунок 3.19 – Запуск завдання моніторингу продуктивності

Прогрес моніторингу сторінок, та кількість і посилання до сторінок що мають бути проаналізовані можна побачити в файлах логів системи.

```
-----
2024-12-05T22:00:00.019Z
GeneratePagesReports: task started
-----
2024-12-05T22:00:00.022Z
5: pages to be indexed
-----
2024-12-05T22:00:00.546Z
GeneratePagesReports: indexing of https://www.madmushroom.gg/submit-your-game started
-----
2024-12-05T22:00:08.785Z
GeneratePagesReports: indexing of https://www.madmushroom.gg/terms-conditions started
-----
2024-12-05T22:00:21.869Z
GeneratePagesReports: indexing of https://www.madmushroom.gg/about started
-----
```

Рисунок 3.20 – Результати роботи завдання моніторингу продуктивності в файлах логів

Завдання «Аналіз продуктивності» запускається один раз на добу або на вимогу користувача. Результатом роботи завдання є звіт, що відображається на сторінці «Звіти» в панелі керування.



#	Назва		
1	12/5/2024, 12:42:02 PM		
2	12/4/2024, 9:08:42 PM		
3	12/4/2024, 9:03:48 PM		
4	12/4/2024, 8:37:38 PM		
5	12/4/2024, 8:35:46 PM		
6	12/4/2024, 8:34:48 PM		
7	12/4/2024, 8:30:26 PM		
8	12/4/2024, 8:29:35 PM		
9	12/4/2024, 8:26:53 PM		
10	12/4/2024, 8:25:55 PM		

Рисунок 3.21 – Сторінка списку звітів щодо продуктивності

Кожен звіт щодо продуктивності веб-сайту складається з 2 частин, а саме метрики та проблеми з продуктивністю.

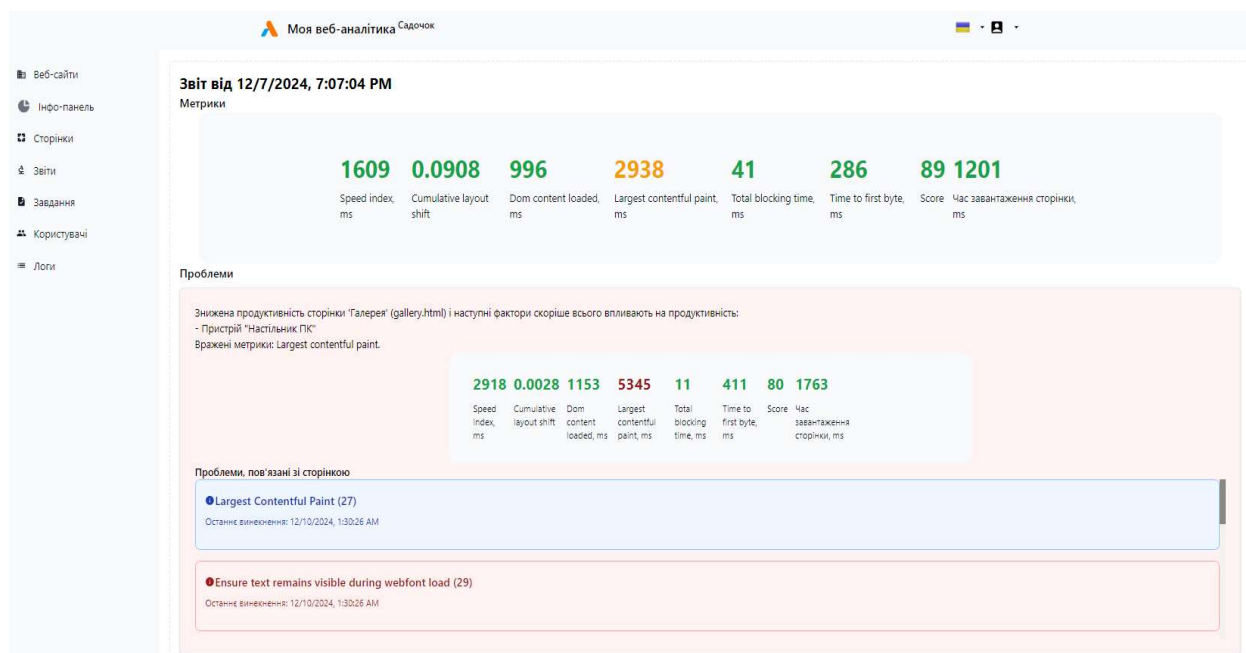


Рисунок 3.22 – Звіт щодо продуктивності веб-сайту

Проблеми в свою чергу складаються з списку факторів для котрих проблема присутня, метрик, на які ці фактори впливають та значень метрик.

Якщо проблема з продуктивністю виникає на певній сторінці, тоді для сторінки відобразиться список проблем, що виникли з сторінкою за останню добу.

На сторінці «phone.html» було розміщене не оптимізоване зображення, яке відображається лише на мобільних пристроях, що впливає на показник LCP. В даному випадку, звіт відобразить дану проблему, і вкаже на фактори, що впливають на продуктивність. На рисунку 3.23 видно, що в звіті вказана проблема з метрикою LCP на сторінці «phone.html» при використанні мобільних пристроїв, крім того, ситуація погіршується для користувачів Китаю.

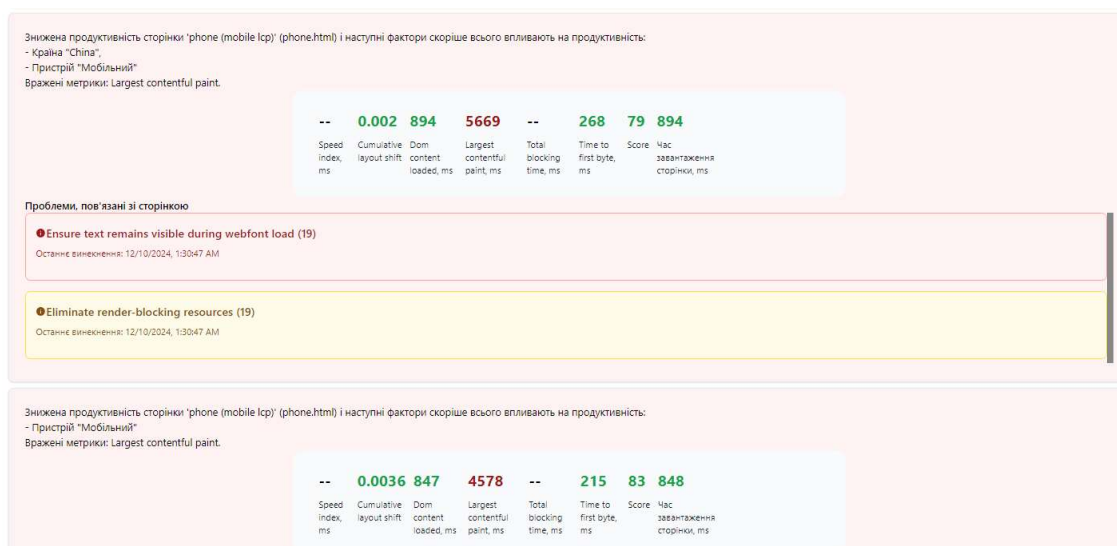


Рисунок 3.23 – Проблема з продуктивністю сторінки відображена в звіті

В системі була впроваджена система логування. Дії користувачів, інформаційні повідомлення запланованих та помилки відображаються в файлах логів, які можна побачити на сторінці «Логи». Дана сторінка доступна лише для адміністрації сервісу веб-аналітики.

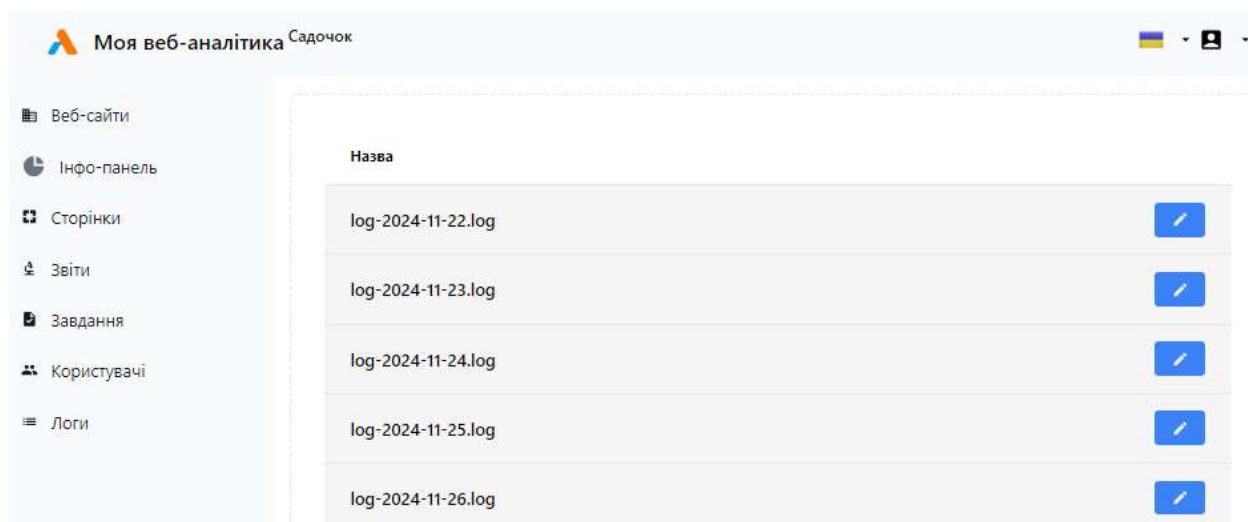


Рисунок 3.24 – Список файлів на сторінці «Логи»

Адміністрація сервісу веб-аналітики може редагувати інформацію про користувачів системи на сторінці «Користувачі». На даній сторінці відображається список всіх користувачів, а також кнопки керування.

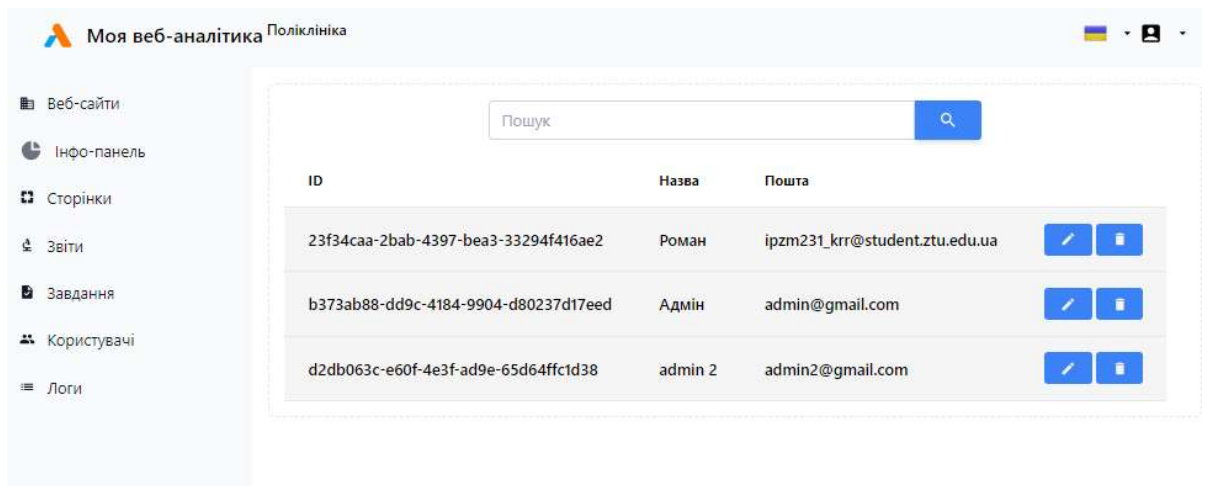


Рисунок 3.25 – Сторінка «Користувачі» в панелі управління

При редагуванні користувача, є можливість змінити його ім'я, електронну пошту та список його ролей, що визначають які дії він може виконувати в системі.

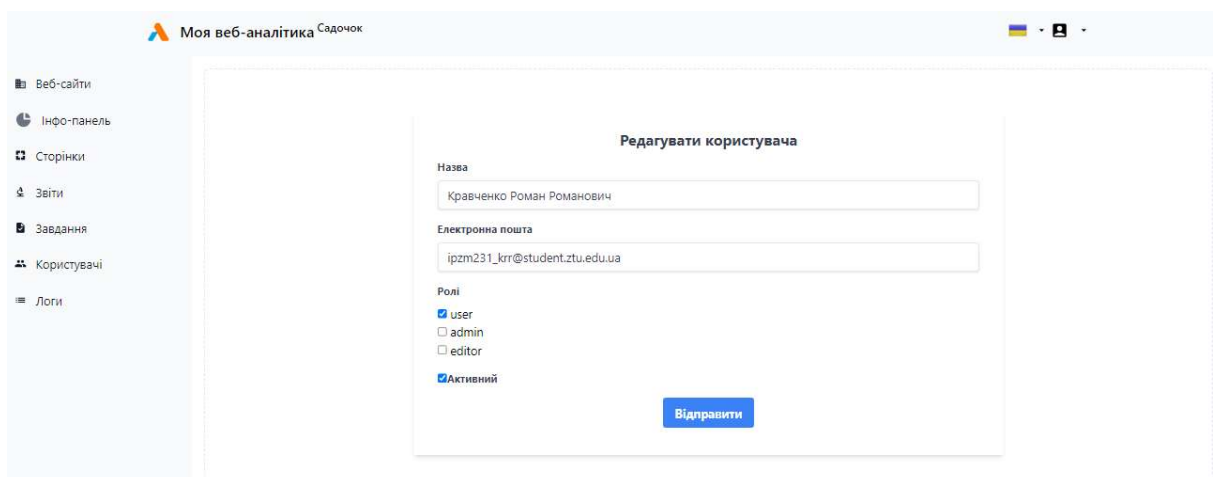


Рисунок 3.26 – Редагування користувача

3.2 Покращення продуктивності веб-сайтів за допомогою системи веб-аналітики

Було проведено експеримент з метою дослідження ефективності застосування системи веб-аналітики для аналізу та покращення продуктивності веб-сайту.

Дослідження проводилось над веб-сайтом створеним з шаблону «Klinik – Clinic Website Template» з ресурсу «html codex», який було розгорнуто на сервері побудованому за допомогою бібліотеки Express на платформі Node.js.

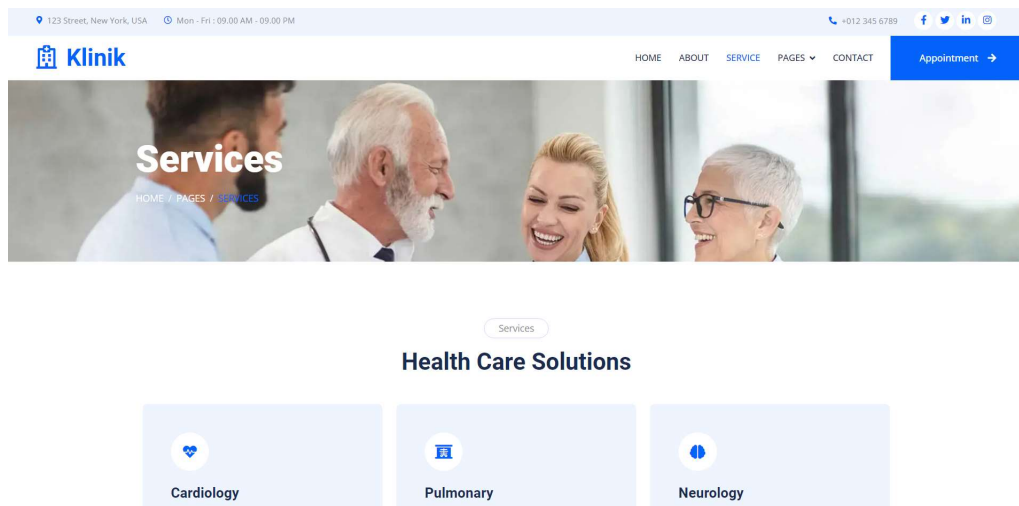


Рисунок 3.27 – Веб-сайт, використаний для проведення дослідження

В розмітку сторінок веб-сайту були додано скрипт сервісу веб-аналітики, для подальшого моніторингу його продуктивності.

Дані щодо продуктивності веб-сайту були отримані шляхом синтетичного моніторингу, а також за допомогою моніторингу справжніх користувачів. Для отримання даних користувачів під час кожного циклу вимірювання продуктивності, користувачі відвідували кожну сторінку веб-додатку з мобільного пристрою та настільного комп'ютера використовуючи веб-браузер Chrome.

Після першого збору даних щодо продуктивності, були отримані показники метрик зображені на рисунку 3.28.

Проаналізувавши показники, можна побачити, що метрики Speed Index (середнє значення 8136 мілісекунд, до 3400 мілісекунд вважається нормою) та LCP (5905 мілісекунд, до 2500 мілісекунд вважається нормою) мають високі значення, що перевищують норму. Метрика FCP також має значення, що перевищує норму в 1800 мілісекунд. Також час отримання першого байту з серверу 2268 мілісекунд є високим для локального серверу, що не містить складної логіки та обчислень.

Час завантаження сторінки

Середнє 2840	Мінімально 365	Максимально 4574
-----------------	-------------------	---------------------

First contentful paint

Середнє 2907	Мінімально 262	Максимально 6163
-----------------	-------------------	---------------------

Total blocking time

Середнє 38	Мінімально 9	Максимально 111
---------------	-----------------	--------------------

Speed index

Середнє 8136	Мінімально 4908	Максимально 9975
-----------------	--------------------	---------------------

Time to first byte

Середнє 2268	Мінімально 2	Максимально 3998
-----------------	-----------------	---------------------

Largest contentful paint

Середнє 5905	Мінімально 1417	Максимально 9597
-----------------	--------------------	---------------------

Cumulative layout shift

Середнє 0.0125	Мінімально 0	Максимально 0.0926
-------------------	-----------------	-----------------------

Dom content loaded

Середнє 2579	Мінімально 342	Максимально 4299
-----------------	-------------------	---------------------

Score

Середнє 63	Мінімально 48	Максимально 87
---------------	------------------	-------------------

Рисунок 3.28 – Показники метрик отримані після початкового збору даних щодо продуктивності

Звіт щодо продуктивності веб-сайту в системі веб-аналітики вказав на проблеми з метриками у певних сторінок. Головна сторінка і сторінка team.html мають високе значення метрики LCP. Сторінки contact.html, testimonial.html та appointment.html мають високе значення метрики Speed index. А сторінки about.html, 404.html та service.html мають високе значення як метрики LCP так і Speed index.



Рисунок 3.29 – Одна з проблем з продуктивністю на яку вказував звіт

Також сервіс веб-аналітики виявив ряд проблем з сторінками веб-сайту, що виникали під час вимірювання. Сумарна кількість проблем та рекомендацій становила 116 під час першого вимірювання.

Серед таких проблем значна кількість вказувала на проблеми сервером, наприклад система вказувала на те, що початкова відповідь з серверу зайняла занадто довгий час (рис. 3.30). Також система вказувала на повільне завантаження скриптів з серверу та інших ресурсів (дана проблема була виявлена за допомогою моніторингу справжніх користувачів, інші проблеми за допомогою синтетичного).

Крім того були виявлені проблеми і на клієнтській стороні веб-додатку, котрі можуть впливати як на продуктивність так і на досвід користувача. Наприклад те, що сторінки не використовують «Passive event listeners» для події «scroll», зображення не мають явно вказаною ширини та висоти, зображення використовують не в оптимальному форматі. Також було вказано на ресурси, що блокують відображення вмісту веб-сторінки.

Система веб-аналітики надала рекомендації, серед яких використання стиснення тексту, попереднє завантаження найбільшого контенту сторінки (рис 3.31), конвертація зображень в формати WebP або AVIF, тощо.

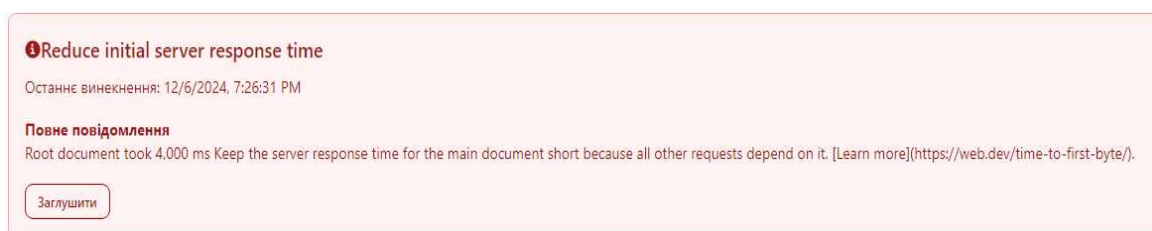


Рисунок 3.30 – Проблема з продуктивністю виявлена сервісом веб-аналітики, що вказує на занадто довгу відповідь з серверу



Рисунок 3.31 – Рекомендація отримана від сервісу веб-аналітики щодо попереднього завантаження найбільшого контенту

Аналіз продуктивності веб-сайту дозволив виявити ряд проблем з продуктивністю веб-сторінок, а також список рекомендацій, що дозволили почати процес покращення продуктивності веб-сайту.

Усунення проблем, виконання рекомендацій та покращення продуктивності веб-сайту відбувалось в 3-х етапах, після кожного з яких відбувалось вимірювання продуктивності веб-сайту.

Під час першого етапу в веб-сайт було внесені зміни, серед яких використання пасивних відловлювачів подій (passive event listeners). Також були явно вказані розміри зображень в розмітці сторінок. Була виправлена помилка на стороні серверу, що впливала на швидкість отримання відповіді з серверу, помилка була виявлена завдяки сервісу веб-аналітики.

Під час другого етапу оптимізації в веб-сайт були застосовані рекомендації щодо попереднього завантаження найбільших зображень на сторінці. Усунене блокування відображення деякими ресурсами шляхом додавання атрибуту «async», а також реалізоване стиснення тексту для файлів веб-сайту.

Під час третього етапу оптимізації в веб-сайті проводилась робота над зображеннями відповідно до рекомендацій отриманих від сервісу, а саме зображення були оптимізовані і скомпресовані, а найбільші зображення були конвертовані в формат WebP.

Результати вимірювання після кожного етапу оптимізації можна побачити в додатку А та в таблиці 3.1.

Була порахована абсолютна зміна, як відображає на скільки змінилось значення метрики після всіх застосованих покращень. Окремо була порахована

різниця в значеннях між значеннями отриманими після третього та першого етапу, адже на першому етапі була виправлена помилка в програмному коді серверного додатку, яка значно вплинула на продуктивність веб-сайту та на всі його метрики.

Таблиця 3.1

Результати вимірювання продуктивності на різних етапах оптимізації

Метрика	Початкові	I етап	II етап	III етап	Різниця між III і I	Абсолютна зміна
Час завантаження сторінки, мс	2840	651	528	516	-135	-2324
FCP, мс	2907	1141	1311	1395	254	-1512
TBT, мс	38	27	40	36	9	-2
Speed index, мс	8136	5236	3885	3981	-1255	-4155
TTFB, мс	2268	148	163	164	16	-2104
LCP, мс	5905	3363	3234	2294	-1069	-3611
CLS	0,012	0,04	0,02	0,035	-0,005	0,023
Час завантаження DOM, мс	2579	472	440	431	-41	-2148
Оцінка продуктивності	63	85	85	88	3	25
Кількість проблем	116	112	69	63	-49	-53

Проаналізувавши дані щодо продуктивності веб-сайту на різних етапах, можна побачити, що продуктивність зростає. Метрики FCP, LCP та Speed Index були не в межах норми початково, з них LCP та Speed Index теж не були в межах норми після першого етапу оптимізації, після оптимізації набули значень в межах норми. Значення метрик пов'язаних з часом завантаження сторінки значно зменшились після першого етапу оптимізації.

Кількість проблем, що виявляла система веб-аналітики зменшилась приблизно на 46%. Оцінка продуктивності зросла з 63 до 88 балів зі 100 можливих.

Значення метрик CLS та TBT значно не змінилось та знаходились в межах норми на всіх етапах.

Також на рисунку 3.32 зображені графіки зміни продуктивності веб-сайту відносно часу. З графіку видно, що загальна продуктивність веб-сайту зросла, значення метрик, що вимірюються часом зменшились, а оцінка продуктивності значно зросла. Значення метрики CLS є нестабільним, але при цьому постійно знаходиться в межах норми.

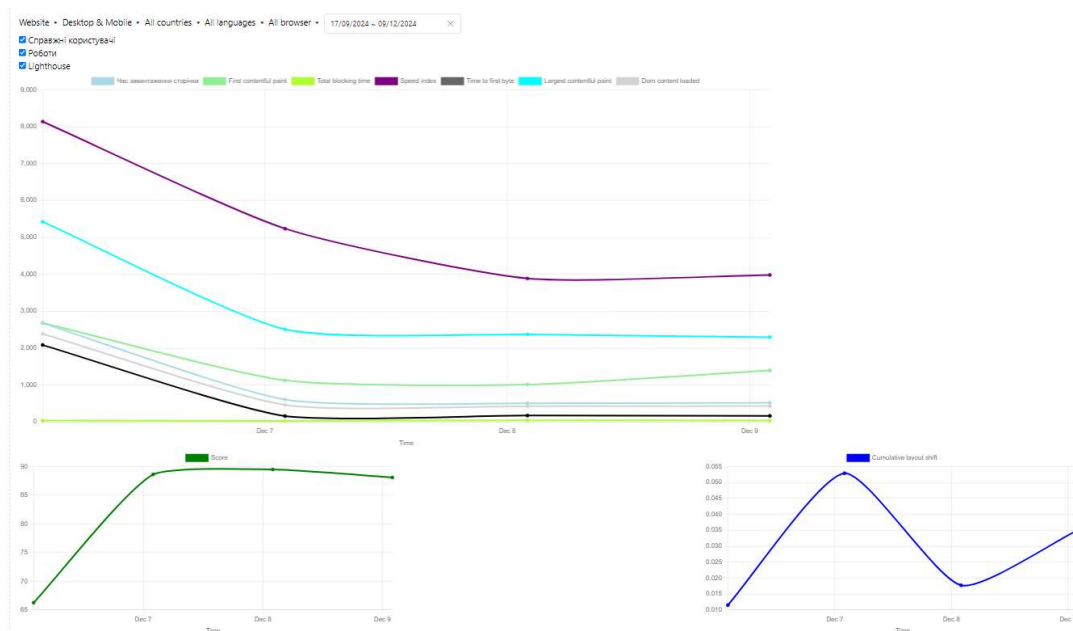


Рисунок 3.32 – Зміни значень метрик продуктивності відносно часу під час оптимізації веб-сайту

Отже, застосування системи веб-аналітики дозволило значно покращити продуктивність веб-сайту. На основі зібраних даних система дозволила виявити проблеми в продуктивності веб-сайту. Застосування рекомендацій та усунення проблем виявлених системою веб-аналітики дозволило значно покращити показники продуктивності веб-сайту, та привести їх до нормативних значень. Особливо помітним було покращення таких метрик, як Speed Index (на 104%) та LCP (на 157%).

Висновки до третього розділу

В даному розділі описано графічний інтерфейс системи веб-аналітики та наведений порядок роботи з додатком. Розглянутий функціонал системи та елементи інтерфейсу.

Проведено дослідження, в рамках якого була перевірена ефективність системи веб-аналітики при аналізі та покращенні продуктивності веб-сайтів. В результаті дослідження було покращено продуктивність веб-сайту та виявлено що створена система веб-аналітики є ефективним інструментом для покращення продуктивності веб-сайтів.

ВИСНОВКИ

Під час виконання випускної роботи було досліджено способи аналізу та покращення продуктивності веб-сайтів, спроектовано та реалізовано систему веб-аналітики аналізу та покращення продуктивності веб-сайтів.

Було проведено аналіз предметної області дослідження, розглянуті підходи до моніторингу продуктивності веб-сторінок та веб-сайтів, визначені переваги та недоліки, досліджені існуючі інструменти аналізу продуктивності, а також сервіси веб-аналітики, що є аналогами розробленої системи. Досліджені ключові метрики продуктивності веб-додатків, способи їх визначення, застосування, значення та способи покращення показників даних метрик.

Спроектowana та реалізована система веб-аналітики для покращення продуктивності веб-сайтів. Реалізована система моніторингу продуктивності веб-сайту, що комбінує підходи до моніторингу продуктивності: моніторинг справжніх користувачів та синтетичний моніторинг. Підходи були реалізовані у вигляді скрипта для моніторингу даних та сервісу Lighthouse відповідно.

Також було створено інструменти для візуалізації та аналізу даних, які спрощують інтерпретацію отриманих даних та прийняття рішень. Розроблено алгоритм, який дозволяє знаходити комбінації факторів, що впливають на продуктивність веб-сайтів. Це дало змогу створювати щоденні звіти щодо продуктивності веб-сайтів з персоналізованими рекомендаціями щодо оптимізації.

Описано графічний інтерфейс системи, його функціонал аналітики та наведений порядок роботи з додатком.

Проведено дослідження ефективності застосування системи для покращення продуктивності додатків. Проведено експеримент, в ході якого проведено аналіз продуктивності веб-сайту за допомогою системи веб-аналітики, отримані рекомендації щодо покращення веб-сайту, застосування яких призвело до покращення продуктивності, що доводить ефективність системи.

Таким чином, в результаті виконання роботи було створено ефективну систему веб-аналітики, яка може бути корисною для покращення продуктивності веб-сайтів. Отримані результати можуть стати основою для подальших досліджень у галузі веб-аналітики та оптимізації продуктивності веб-сайтів, а також для покращення існуючих сервісів веб-аналітики.

ЛІТЕРАТУРА

1. Kravchenko R. R., Vakaliuk T. A. THE USAGE AND ROLE OF WEB ANALYTICS SERVICES FOR WEBSITES // Збірник тез доповідей II Міжнар. наук.-практич. конфер. «Інновації та перспективні шляхи розвитку інформаційних технологій» (06 груд. 2023 р., м. Черкаси) [Електронний ресурс], Черкас. держ. технол. ун-т. – Черкаси : ЧДТУ, 2023. – 277 с.
2. Kravchenko R. R., Vakaliuk T. A. TOOLS FOR ANALYSIS OF WEB PAGE PERFORMANCE // Збірник тез доповідей II Міжнар. наук.-практич. конфер. «Інновації та перспективні шляхи розвитку інформаційних технологій» (06 груд. 2023 р., м. Черкаси) [Електронний ресурс], Черкас. держ. технол. ун-т. – Черкаси : ЧДТУ, 2023. – 277 с.
3. Deloitte Ireland. Milliseconds Make Millions [Електронний ресурс] – Режим доступу до ресурсу: https://www.deloitte.com/content/dam/assets-zone2/ie/en/docs/services/consulting/2023/Milliseconds_Make_Millions_report.pdf
4. Yakovliev M. EFFECTS OF LOADING SPEED ON THE SITE TRAFFIC CONVERSION [Електронний ресурс] / M. Yakovliev, K. Filonenko // Системи управління, навігації та зв'язку. Збірник наукових праць. – 2019. – Vol. 5, no. 57. – Р. 92–94. – Режим доступу до ресурсу: <https://doi.org/10.26906/sunz.2019.5.092>
5. MDN / Performance Monitoring: RUM vs. synthetic monitoring [Електронний ресурс] – Режим доступу до ресурсу: <https://developer.mozilla.org/en-US/docs/Web/Performance/Rum-vs-Synthetic>
6. Heričko T. Towards Representative Web Performance Measurements with Google Lighthouse / Šumak B., Brdник S. – 7th Student Computer Science Research Conference, 2021
7. Wagner J. Web Performance in Action: Building Faster Web Pages / Jeremy Wagner., 2017. – 376 с.
8. Kaushik A. Web Analytics: An Hour a Day / Avinash Kaushik., 2007. – 443 с.

9. Web.dev / Web Vitals [Электронный ресурс] – Режим доступа до ресурсу:
<https://web.dev/articles/vitals?hl=en>
10. Web.dev / Largest Contentful Paint [Электронный ресурс] – Режим доступа до ресурсу: <https://web.dev/articles/lcp>
11. Web.dev / Cumulative Layout Shift [Электронный ресурс] – Режим доступа до ресурсу: <https://web.dev/articles/cls>
12. Web.dev / Interaction to Next Paint [Электронный ресурс] – Режим доступа до ресурсу: <https://web.dev/articles/inp>
13. Web.dev / First Contentful Paint [Электронный ресурс] – Режим доступа до ресурсу: <https://web.dev/articles/fcp>
14. Chrome for developers / Total Blocking Time [Электронный ресурс] – Режим доступа до ресурсу:
<https://developer.chrome.com/docs/lighthouse/performance/lighthouse-total-blocking-time>
15. Chrome for developers / Speed Index [Электронный ресурс] – Режим доступа до ресурсу: <https://developer.chrome.com/docs/lighthouse/performance/speed-index>
16. Chrome for developers / Lighthouse performance scoring [Электронный ресурс] – Режим доступа до ресурсу:
<https://developer.chrome.com/docs/lighthouse/performance/performance-scoring>
17. Web.dev / Time to First Byte [Электронный ресурс] – Режим доступа до ресурсу: <https://web.dev/articles/ttfb?hl=en>
18. Web.dev / User-centric performance metrics [Электронный ресурс] – Режим доступа до ресурсу: <https://web.dev/articles/user-centric-performance-metrics>
19. Medium / Monolithic vs Modular [Электронный ресурс] – Режим доступа до ресурсу: <https://medium.com/@berto168/monolithic-vs-modular-9b6d69684a2c>
20. Medium / Single-page application vs. multiple-page application [Электронный ресурс] – Режим доступа до ресурсу: <https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58>
21. Chart.js/ Line Chart [Электронный ресурс] – Режим доступа до ресурсу:
<https://www.chartjs.org/docs/latest/charts/line.html>

- 22.Chart.js/ Time Cartesian Axis [Электронный ресурс] – Режим доступа до ресурсу: <https://www.chartjs.org/docs/latest/axes/cartesian/time.html>
- 23.Nestjs / Modules [Электронный ресурс] – Режим доступа до ресурсу: <https://docs.nestjs.com/modules>
- 24.Nestjs / TypeORM integration [Электронный ресурс] – Режим доступа до ресурсу: <https://docs.nestjs.com/techniques/database>
- 25.MDN / Performance API [Электронный ресурс] – Режим доступа до ресурсу: <https://developer.mozilla.org/en-US/docs/Web/API/Performance>
- 26.MDN / PerformanceNavigationTiming [Электронный ресурс] – Режим доступа до ресурсу: <https://developer.mozilla.org/en-US/docs/Web/API/PerformanceNavigationTiming>
- 27.MDN / Time to First Byte [Электронный ресурс] – Режим доступа до ресурсу: https://developer.mozilla.org/en-US/docs/Glossary/Time_to_first_byte
- 28.MDN / LayoutShift [Электронный ресурс] – Режим доступа до ресурсу: <https://developer.mozilla.org/en-US/docs/Web/API/LayoutShift>
- 29.MDN / PerformancePaintTiming [Электронный ресурс] – Режим доступа до ресурсу: <https://developer.mozilla.org/en-US/docs/Web/API/PerformancePaintTiming>
- 30.Lighthouse GitHub / Understanding the Results [Электронный ресурс] – Режим доступа до ресурсу: <https://github.com/GoogleChrome/lighthouse/blob/main/docs/understanding-results.md>
- 31.Peltsverger S. Web Analytics Overview [Электронный ресурс] / S. Peltsverger, J. Zheng. – 2015. – Режим доступа до ресурсу: https://www.researchgate.net/publication/272815693_Web_Analytics_Overview
- 32.Burby J. Web Analytics Definitions / J. Burby, A. Brown. – Washington DC: Web Analytics Association, 2007.
- 33.Nestjs / Configuration [Электронный ресурс] – Режим доступа до ресурсу: <https://docs.nestjs.com/techniques/configuration>

- 34.Nestjs / Logging [Электронный ресурс] – Режим доступа до ресурсу:
<https://docs.nestjs.com/techniques/logger>
- 35.HTTP Archive / Report: Loading Speed [Электронный ресурс] – Режим доступа до ресурсу: <https://httparchive.org/reports/loading-speed>
- 36.Desmos / Lighthouse Individual Metric Scoring Curve [Электронный ресурс] – Режим доступа до ресурсу: <https://www.desmos.com/calculator/oqlvmezbze>
- 37.Desmos / Normal and Log-Normal Distributions [Электронный ресурс] – Режим доступа до ресурсу: <https://www.desmos.com/calculator/2t1ugwykrl>
- 38.MDN / PerformanceEntry: entryType [Электронный ресурс] – Режим доступа до ресурсу: <https://developer.mozilla.org/en-US/docs/Web/API/PerformanceEntry/entryType>
- 39.Chrome for developers / Largest Contentful Paint [Электронный ресурс] – Режим доступа до ресурсу: <https://developer.chrome.com/docs/lighthouse/performance/lighthouse-largest-contentful-paint>
- 40.Chrome for developers / First Contentful Paint [Электронный ресурс] – Режим доступа до ресурсу: <https://developer.chrome.com/docs/lighthouse/performance/first-contentful-paint>

ДОДАТКИ

Результати вимірювання продуктивності веб-сайту на різних етапах оптимізації



Рисунок А.1 – Результати вимірювання продуктивності після першого етапу оптимізації

Час завантаження сторінки		
Середнє 528	Мінімальне 295	Максимальне 906
First contentful paint		
Середнє 1311	Мінімальне 201	Максимальне 3633
Total blocking time		
Середнє 40	Мінімальне 14	Максимальне 93
Speed index		
Середнє 3885	Мінімальне 2833	Максимальне 5286
Time to first byte		
Середнє 163	Мінімальне 2	Максимальне 259
Largest contentful paint		
Середнє 3234	Мінімальне 218	Максимальне 6526
Cumulative layout shift		
Середнє 0.0238	Мінімальне 0	Максимальне 0.3131
Dom content loaded		
Середнє 440	Мінімальне 295	Максимальне 621
Score		
Середнє 85	Мінімальне 69	Максимальне 100

Рисунок А.2 – Результати вимірювання продуктивності після другого етапу оптимізації

Час завантаження сторінки		
Середнє 516	Мінімальне 339	Максимальне 1007
First contentful paint		
Середнє 1395	Мінімальне 239	Максимальне 4465
Total blocking time		
Середнє 36	Мінімальне 9	Максимальне 104
Speed index		
Середнє 3981	Мінімальне 3270	Максимальне 5184
Time to first byte		
Середнє 164	Мінімальне 10	Максимальне 260
Largest contentful paint		
Середнє 2294	Мінімальне 277	Максимальне 6127
Cumulative layout shift		
Середнє 0.035	Мінімальне 0	Максимальне 0.3237
Dom content loaded		
Середнє 431	Мінімальне 325	Максимальне 630
Score		
Середнє 88	Мінімальне 67	Максимальне 100

Рисунок А.3 – Результати вимірювання продуктивності після третього етапу оптимізації

Програмний код системи веб-аналітики

Лістинг GeneratePagesReports.ts (клас завдання синтетичного моніторингу):

```

import { BadRequestException, Inject, Injectable, Logger } from "@nestjs/common";
import { Cron, CronExpression } from "@nestjs/schedule";
import { LighthouseService } from "../lighthouse/lighthouse.service";
import { CollectService } from "../collect/collect.service";
import { PageService } from "../pages/page.service";
import { WebsiteEntity } from "../websites/website.entity";
import { WebsitesService } from "../websites/website.service";
import { PageIssueService } from "../issues/issue.service";

@Injectable()
export class GeneratePagesReports {
  batchSize = 100;
  private readonly logger: Logger;
  constructor(
    private service: LighthouseService,
    private collectService: CollectService,
    private websiteService: WebsitesService,
    private pageService: PageService,
    private issueService: PageIssueService,
    @Inject('TASK_STATUS')
    private taskStatuses: Record<string, 'idle' | 'running' | 'failed'>,
  ) {
    this.logger = new Logger();
  }

  @Cron(CronExpression.EVERY_30_MINUTES)
  async run() {
    if (this.taskStatuses['generate-reports'] === 'running')
      throw new BadRequestException('Task is already running');

    this.logger.warn(`GeneratePagesReports: task started`);
    this.taskStatuses['generate-reports'] = 'running';

    try {
      const pages = await this.pageService.getPagesToReport();
      this.logger.warn(`${pages.length}: pages to be indexed`);
      const websites: Record<string, WebsiteEntity> = {};
      for (var page of pages) {
        await this.service.startChrome();
        const website = websites[page.websiteId] ??= await
this.websiteService.get(page.websiteId);
        const url = website.domain + '/' + page.url;
        this.logger.warn(`GeneratePagesReports: indexing of ${url} started`);
        const sessionId = uuidv4();
        const viewData = {
          language: 'en',

```

```

        pageTitle: page.name,
        startTime: Date.now(),
        url,
        sessionId,
        userAgent: 'lighthouse',
        userIp: undefined,
    };
    const pageView = await this.collectService.savePageView(viewData);
    await this.pageService.updateLastReported(page);
    const res = await this.service.runLighthouse(url);
    await this.service.killChrome();
    this.reportIssues(page.websiteId, page.id, res);
    const metrics: Record<string, number> = res.audits.metrics.details.items[0]
|| {});
    const perf = this.collectService.savePagePerformance({
        ...viewData,
        si: metrics.speedIndex,
        tbt: metrics.totalBlockingTime,
        fcp: metrics.firstContentfulPaint,
        loadTime: metrics.observedLoad,
        dcl: metrics.observedDomContentLoaded,
        cls: metrics.cumulativeLayoutShift,
        lcp: metrics.largestContentfulPaint,
        ttfb: res.audits['server-response-time'].numericValue,
        userIp: undefined,
        score: res.categories.performance.score * 100,
    })
    }

    this.taskStatuses['generate-reports'] = 'idle';

    } catch (e) {
        console.log(e);
        this.taskStatuses['generate-reports'] = 'failed';
    }
}

private reportIssues(websiteId: string, pageId: string, report: any) {
    Object.values(report.audits).filter((a: any) => a.score !== null && a.score <
0.8).forEach((a: any) => {
        this.issueService.reportIssue({
            message: a.title,
            fullMessage: a.displayValue + '\n' + a.description,
            reportDate: new Date(),
            severity: a.score > 0.5 ? 'INFO' : a.score > 0.25 ? 'WARNING' : 'ERROR',
            trace: undefined,
            pageId,
            websiteId,
        })
    })
}
}

```

Лістинг AnalyzePerformance.ts (клас завдання аналізу продуктивності):

```
import { BadRequestException, Inject, Injectable, Logger } from "@nestjs/common";
import { Cron, CronExpression } from "@nestjs/schedule";
import { LighthouseService } from "../../lighthouse/lighthouse.service";
import { CollectService } from "../../collect/collect.service";
import { PageService } from "../../pages/page.service";
import { WebsiteEntity } from "../../websites/website.entity";
import { WebsitesService } from "../../websites/website.service";
import { PageIssueService } from "../../issues/issue.service";
import { AgregatedMetrics } from "../../collect/types";
import { Issue, ReportData } from "../../reports/types";
import { ReportService } from "../../reports/report.service";
import { filterData, getCount, getSum, shallowEqual } from "../../helpers";
import { PerformanceAnalyzerV1 } from "../../PerformanceAnalyzerV1";
import { PerformanceAnalyzerV2 } from "../../PerformanceAnalyzerV2";
```

```
@Injectable()
```

```
export class AnalyzePerformance {
  private readonly logger: Logger;
  constructor(
    private collectService: CollectService,
    private websiteService: WebsitesService,
    private pageService: PageService,
    private issueService: PageIssueService,
    private reportService: ReportService,
    @Inject('TASK_STATUS')
    private taskStatuses: Record<string, 'idle' | 'running' | 'failed'>,
  ) {
    this.logger = new Logger();
  }
}
```

```
@Cron(CronExpression.EVERY_DAY_AT_6AM)
```

```
async run() {
  if (this.taskStatuses.analyze === 'running')
    throw new BadRequestException('Task is already running');

  this.logger.warn(`AnalyzePerformance: task started`);
  this.taskStatuses.analyze = 'running';

  try {
    const websites = await this.websiteService.getAll();
    this.logger.warn(`AnalyzePerformance: ${websites.length} to be analyzed`);
    for (let i = 0; i < websites.length; i++) {
      const report = await this.analyzeWebsite(websites[i]);
      await this.reportService.createReport(websites[i].id, report);
    }

    this.taskStatuses.analyze = 'idle';
  } catch (e) {
```

```

        console.log(e);
        this.taskStatuses.analyze = 'failed';
    }
}

async analyzeWebsite(website: WebsiteEntity): Promise<ReportData> {
    const res: AnalyzerIssue[] = [];
    this.logger.warn(`AnalyzePerformance: analyzing ${website.name}
($${website.domain})`);
    const data = await this.collectService.loadAgregatedMetrics(website.id);
    const filterValues = filters.reduce((r, f) => {
        r[f] = new Set(data.map(d => d[f]));
        return r;
    }, {} as Record<string, Set<string>>);
    const globalCount = data.reduce((r, d) => r + +d.count, 0);

    const metricsAvgs = [...metrics, 'score'].reduce((r, m) => {
        r[m] = data.reduce((a, d) => a + (+d.count * +d[m]), 0) / globalCount;
        return r;
    }, {} as Record<string, number>);

    for (let i = 0; i < metrics.length; i++) {
        const metric = metrics[i];
        let resComb: Record<string, string>[] = [];
        const analyzer = new PerformanceAnalyzerV2();
        const push = (value: Record<string, string>) => {
            if (!resComb.find(r => shallowEqual(r, value)))
                resComb.push(value);
        }
        analyzer.analyzeMetric(
            metric, filterValues, filters, data, push,
        )

        for (let i = 0; i < resComb.length; i++) {
            const item = resComb[i];
            const existing = res.find(r => shallowEqual(r.fields, item));
            if (existing) {
                if (!existing.metrics.includes(metric))
                    existing.metrics.push(metric);
                existing.fields = { ...item, ...existing.fields }
            } else {
                res.push({ fields: item, metrics: [metric] });
            }
        }
    }

    const uniqueFilters: Issue[] = [];

    for (let i = 0; i < res.length; i++) {
        const isDuplicate = uniqueFilters.some(uniqueFilter =>
            isSameData(data, res[i].fields, uniqueFilter.fields)

```

```

    );

    if (!isDuplicate) {
        const filtered = filterData(data, res[i].fields);
        const count = getCount(filtered);
        const values = [...metrics, 'score'].reduce((r, m) => {
            r[m] = (getSum(filtered, m) / count)
            return r;
        }, {} as Record<string, number>);
        uniqueFilters.push({...res[i], values });
    }
}

const pageIds = uniqueFilters.map(f => f.fields.pageId).filter(Boolean);
const pages = await this.pageService.getByIds(pageIds);
const issues = await this.issueService.getByPageId(pageIds);

return {
    issues: uniqueFilters,
    values: metricsAvgs,
    pageData: pages.map(p => ({ name: p.name, url: p.url, id: p.id })),
    pageIssues: issues.filter(i => i.lastAppeared.getTime() > (Date.now() - 24 *
60 * 60 * 1000)).map(i => ({
        count: i.count,
        severity: i.severity,
        pageId: i.pageId,
        message: i.message,
        lastAppeared: i.lastAppeared.toISOString(),
    })))
}
}
}

const metrics = ['loadTime', 'lcp', 'ttfb', 'si', 'tbt', 'cls', 'dcl'];

const filters = ['pageId', 'isMobile', 'language', 'country', 'browser'];

function isSameData(data: AgregatedMetrics[], a: Record<string, string>, b:
Record<string, string>): boolean {
    const filteredA = filterData(data, a);
    const filteredB = filterData(data, b);

    if (filteredA.length !== filteredB.length) return false;

    return filteredA.every(itemA =>
        filteredB.some(
            itemB => JSON.stringify(itemA) === JSON.stringify(itemB)
        )
    );
}

function getPermutations<T>(arr: T[]): T[][] {

```

```

    if (arr.length === 0) return [[]];

    const result: T[][] = [];
    for (let i = 0; i < arr.length; i++) {
        const rest = arr.slice(0, i).concat(arr.slice(i + 1));
        const restPermutations = getPermutations(rest);
        for (const perm of restPermutations) {
            result.push([arr[i], ...perm]);
        }
    }

    return result;
}

type AnalyzerIssue = { metrics: string[]; fields: Record<string, string>; };

Лістинг PerformanceAnalyzerV2.ts (клас що виконує аналіз продуктивності):
import { AgregatedMetrics } from "../collect/types";
import { config, filterData, getCount, getSum } from "./helpers";

export class PerformanceAnalyzerV2 {
    private randomDataK = 0.001;
    private keepFactorK = 1.05;
    analyzeMetric(metric: string, filterValues: Record<string, Set<string>>, filters:
string[], data: AgregatedMetrics[], push: (r: Record<string, string>) => void) {
        const allcount = getCount(data);
        const metricAvg = getSum(data, metric) / allcount;
        const randomDataCount = Math.ceil(this.randomDataK * allcount);
        const errorData = data.filter(d => d[metric] > config[metric]);
        errorData.forEach(err => {
            let filter = filters.reduce((r, f) => (r[f] = err[f], r), {});
            let filterCount = +err.count;
            let lastFilter = { ...filter };
            let avg = +err[metric];
            const resFilters = this.analyzeFilterRec(metric, filter, avg, filterCount,
data);

            resFilters.forEach(filter => {
                filter = this.clearFilter(filter, data, allcount);
                if (Object.keys(filter).length && filterCount > randomDataCount && avg >
metricAvg && avg > config[metric])
                    push(filter);
            })
        });
    }

    clearFilter(filter: Record<string, string>, data: AgregatedMetrics[], allcount:
number) {
        const keys = Object.keys(filter);
        const filtered = filterData(data, filter);

```

```

    const count = getCount(filtered);
    const res = {...filter};
    for (let index = 0; index < keys.length; index++) {
        const filterKey = keys[index];
        const allFilteredCount = getCount(filterData(data, { [filterKey]:
filter[filterKey] }));
        if (allFilteredCount / allcount > 0.95)
            delete res[filterKey];
    }

    return res;
}

analyzeFilterRec(metric: string, filter: Record<string, string>, filterAvg:
number, filterCount: number, data: AgregatedMetrics[]) {
    const keys = Object.keys(filter);
    const res = [] as Record<string, string>[];
    if (!keys.length)
        return [];

    for (let index = 0; index < keys.length; index++) {
        const filterKey = keys[index];
        const newFilter = {...filter};
        delete newFilter[filterKey];

        const newData = filterData(data, newFilter);
        const count = getCount(newData);
        const newAvg = getSum(newData, metric) / count;

        const shouldGoWider = filterAvg / newAvg <= this.keepFactorK && count !==
filterCount;
        if (!shouldGoWider)
            continue;

        res.push(...this.analyzeFilterRec(metric, newFilter, newAvg, filterCount,
data))
    }

    return res.length ? res : [filter];
}
}

```

Лістинг функції getPerfomanceData (функція для збору даних продуктивності):

```

const fcpPromise = getFcp();
const lcpPromise = getLCP();
const clsPromise = getCLS();

async function getPerfomanceData() {
    const timing = window.performance.getEntriesByType('navigation')[0];
    const result = {

```

```

    loadTime: timing.loadEventStart,
    dcl: timing.domContentLoadedEventStart,
    fcp: await fcpPromise,
    lcp: await lcpPromise,
    cls: await clsPromise,
    ttfb: timing.responseStart,
  };
  return result;
}
function getFcp() {
  return new Promise(r => {
    const clear = resolveOnTimeout(r, 3000);
    observeEntry('paint', (entries, obs) => {
      const paintEntry = entries.find(e => e.name === 'first-contentful-paint')
      if (paintEntry) {
        clear();
        obs.disconnect();
        r(paintEntry.startTime);
      }
    });
  });
}

function getLCP() {
  return new Promise(r => {
    let value = null;
    const observer = observeEntry('largest-contentful-paint', (entries, obs) => {
      const lcpEntry = entries[entries.length - 1];
      if (!lcpEntry)
        return;

      value = lcpEntry.renderTime || lcpEntry.loadTime;
    });
    resolveOnTimeout(() => {observer.takeRecords(); observer.disconnect();
r(value);}, 10000);
  });
}

function getCLS(){
  return new Promise(r => {
    let value = 0;
    const observer = observeEntry('layout-shift', (entries, obs) => {
      const cls = entries.reduce((res, entry) => {
        if (!entry.hadRecentInput)
          res += entry.value;
        return res;
      }, 0)
      value += cls
    });
    resolveOnTimeout(() => {observer.takeRecords(); observer.disconnect();
r(value);}, 5000);
  });
}

```

```

    });
  }

  function resolveOnTimeout(r, time) {
    const timeout = setTimeout(() => r(null), time);

    return () => clearTimeout(timeout);
  }
  function observeEntry(type, callback) {
    const obs = new PerformanceObserver(list => void callback(list.getEntries(),
    obs));
    obs.observe({ type, buffered: true })

    return obs;
  }

```

Лістинг lighthouse.service.ts (клас для роботи з сервісом Lighthouse):

```

import { Injectable } from '@nestjs/common';
import lighthouse from 'lighthouse';
import * as chromeLauncher from 'chrome-launcher';

@Injectable()
export class LighthouseService {
  private chrome;

  async startChrome() {
    if (this.chrome)
      await this.killChrome();

    const chrome = await chromeLauncher.launch({ chromeFlags: ['--headless'] });
    this.chrome = chrome;
  }

  async killChrome() {
    if (!this.chrome)
      return;

    await this.chrome.kill();
    this.chrome = null;
  }

  async runLighthouse(url: string): Promise<any> {
    if (!this.chrome)
      await this.startChrome();

    const options = { logLevel: 'info', output: 'json', port: this.chrome.port,
    onlyCategories: ['performance'] } as const;
    const runnerResult = await lighthouse(url, options);

    return runnerResult.lhr;
  }

```

```
}
```

Лістинг Metrics.tsx (компонент для відображення значень метрик):

```
import { useTexts } from "features/languages";
import ScoreHighlight from "primitives/ScoreHighlight";
import { metricsConfig } from "utils/helpers";

const Metrics = ({ data, small }: { data: Record<string, number>, small?:boolean })
=> {
  const T = useTexts();
  const fontSize = small ? 'text-2xl' : 'text-4xl';
  const labelSize = small ? 'text-sm' : 'text-m';
  const gap = small ? 'gap-5' : 'gap-20';
  const containerM = small ? 'py-5 px-5 xl:py-8 xl:px-10' : 'py-10 px-10 xl:py-16
xl:px-20';
  const containerL = small ? "mx-auto max-w-4xl px-2 sm:px-3 lg:px-4" : "mx-auto
max-w-8xl px-4 sm:px-6 lg:px-8";
  const center = small ? "" : " my-0 mx-auto";

  return (
    <div className={containerL}>
      <div
        className={`rounded-2xl ${containerM} bg-gray-50 flex items-center justify-
between flex-col ${gap} lg:flex-row`}
        <div className={"w-full lg:w-4/5"+center}>
          <div
            className={`flex flex-col flex-1 ${gap} lg:gap-3 lg:flex-row
lg:justify-between`}
          >
            {Object.entries(data).map(([k,v]) => (
              <div key={k} className="block">
                <div
                  className={`font-manrope font-bold ${fontSize} mb-3 text-center
lg:text-left`}
                >
                  {k !== 'cls' && v === 0 ? '--' : <ScoreHighlight
                    lowerBetter
                    good={metricsConfig[k].good}
                    medium={metricsConfig[k].medium}
                    value={k === 'cls' ? +v.toFixed(4) : Math.floor(v)}
                  />}
                </div>
                <span className={`text-gray-900 ${labelSize} text-center block
lg:text-left`}
                  >{T[`Metric_${k}`]}{k !== 'cls' && k !== 'score' && ', ms'}
                </span>
              </div>
            ))}
          </div>
        </div>
      </div>
    )
  )
}
```

```
        </div>
      </div>
    </div>
  )
};

export default Metrics
```