НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ПОГОДЖЕНО
Декан факультету
Факультет інформаційних технологій
(назва факультету)

_____ Ігор БОЛБОТ
(підпис) (ім'я ПРІЗВИЩЕ)

"___" _____ 2025р.

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ
Завідувач кафедри
Комп'ютерних наук
(назва кафедри)

_____ Белла ГОЛУБ
(підпис) (ім'я ПРІЗВИЩЕ)

"___" _____ 2025р.

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему Програмне забезпечення системи аналізу технік швидкого завантаження веб сторінок

Спеціальність 121 Інженерія програмного забезпечення

Освітня програма Програмне забезпечення інформаційних технологій

Орієнтація освітньої програми освітньо-професійна

Гарант освітньої програми

кандидат фізико-математичних наук, доцент _____ Кириченко В.В.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник магістерської кваліфікаційної роботи

старший викладач _____ Міловідов Ю.О.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Консультант магістерської кваліфікаційної роботи

к.т.н. доцент _____ Пархоменко І.І.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ Соколов Д.В.
(підпис) (ПІБстудента)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри

кандидат технічних наук, доцент _____ Белла Голуб
(науковий ступінь, вчене звання) (підпис) (Ім'я ПРІЗВИЩЕ)

"01" листопада 2024 року

З А В Д А Н Н Я

ДО ВИКОНАННЯ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ СТУДЕНТУ

Соколову Дмитру Віталійовичу

Спеціальність 121 Інженерія програмного забезпечення

Освітня програма Програмне забезпечення інформаційних технологій

Орієнтація освітньої програми освітньо-професійна

Тема магістерської кваліфікаційної роботи Програмне забезпечення системи аналізу технік швидкого завантаження веб сторінок

затверджена наказом проректора НУБіП України від "01" листопада 2024р. № 1963 «С»

Термін подання завершеної роботи на кафедру 2025.11.28
(рік, місяць, число)

Вихідні дані до магістерської кваліфікаційної роботи API-сервіс Google PageSpeed Insights для отримання метрик Core Web Vitals, приклади веб-ресурсів для аналізу (тестові сайти різного рівня оптимізації),сучасні фреймворки для розробки веб-застосунків: React.js, Node.js, Express, PostgreSQL, методики аналізу та оптимізації продуктивності веб-сторінок, наявні інструменти для порівняння результатів (Google Lighthouse, GTmetrix, WebPageTest), вимоги до швидкодії веб-додатків згідно з концепцією Core Web Vitals (Google, 2023–2025).

Перелік питань, що підлягають дослідженню:

1. Аналіз предметної області
2. Моделювання системи аналізу швидкого завантаження веб-сторінок
3. Розробка програмного забезпечення
4. Експериментальні дослідження та аналіз результатів

Перелік графічного матеріалу (за потреби) _____
_____

Дата видачі завдання "01" листопада 2024 р.

Керівник магістерської кваліфікаційної роботи _____ Міловідов Ю.О.
(підпис) (ПІБ)

Консультант магістерської кваліфікаційної роботи

к.т.н. доцент _____ Пархоменко І.І.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Завдання прийняв до виконання _____ Соколов Д.В.
(підпис) (ПІБ)

ЗМІСТ

ВСТУП

У сучасному світі веб-технологій швидкість завантаження веб-сторінок стала одним із ключових чинників, що визначає якість роботи сайтів, їх конкурентоспроможність і ефективність взаємодії з користувачами. Користувачі очікують миттєвого доступу до контенту, а затримка навіть у кілька секунд може призвести до втрати відвідувачів, зниження показників конверсії та прибутковості онлайн-бізнесу. За даними компанії Google, збільшення часу завантаження сторінки з однієї до трьох секунд підвищує ймовірність відмови користувача майже на 32 %, а при п’яти секундах - на 90 %. Таким чином, оптимізація продуктивності веб-додатків набуває стратегічного значення для власників сайтів і розробників.

Розвиток веб-технологій супроводжується появою великої кількості фреймворків, бібліотек і платформ, які суттєво спрощують створення інтерфейсів, але водночас збільшують обсяг переданих даних, кількість HTTP-запитів і складність рендерингу сторінки. У таких умовах питання аналізу й оптимізації технік швидкого завантаження веб-сторінок стає критично важливим. Для цього використовуються інструменти, що дозволяють вимірювати основні метрики продуктивності - такі як Largest Contentful Paint (LCP), First Input Delay (FID), Cumulative Layout Shift (CLS), Speed Index та інші. Ці показники входять до складу концепції Core Web Vitals, яку компанія Google запровадила як один із факторів ранжування сторінок у пошуковій системі.

Наявні системи аналізу, як-от Google Lighthouse, GTmetrix чи WebPageTest, надають корисні звіти, проте мають обмеження: вони не

дозволяють зручно порівнювати результати між різними сайтами, зберігати історію перевірок або інтегрувати результати в зовнішні системи моніторингу. Це створює передумови для розроблення нового програмного забезпечення, яке б автоматизувало процес збору, аналізу та представлення метрик швидкодії у вигляді зручних звітів і візуалізацій.

Актуальність теми полягає у необхідності створення інструменту, який забезпечує комплексний аналіз технік оптимізації веб-сторінок, дозволяє проводити порівняльні тести різних підходів до завантаження контенту та допомагає розробникам приймати обґрунтовані рішення щодо покращення продуктивності веб-додатків. Запропоноване рішення може бути використане не лише у комерційних цілях, але й у навчальному процесі для підготовки фахівців з інформаційних технологій.

Об’єкт дослідження - процес аналізу швидкодії веб-сторінок на основі ключових показників ефективності завантаження.

Предмет дослідження - методи, алгоритми та програмні засоби, що забезпечують автоматизовану оцінку технік швидкого завантаження веб-контенту.

Мета роботи - розроблення програмного забезпечення для проведення комплексного аналізу технік оптимізації швидкодії веб-сторінок із можливістю порівняння результатів і візуалізації даних.

Для досягнення поставленої мети необхідно вирішити такі завдання:

1. Провести аналіз існуючих методів і засобів оптимізації веб-продуктивності.
2. Розробити архітектуру системи аналізу технік завантаження.

3. Реалізувати програмний модуль для збору метрик з використанням зовнішніх API.
4. Забезпечити збереження результатів у базі даних для подальшого порівняльного аналізу.
5. Створити інтерфейс користувача для відображення результатів у зручній візуальній формі.
6. Провести експериментальні дослідження ефективності системи на прикладі реальних веб-ресурсів.

Методи дослідження, що використовувалися в роботі: аналітичні методи, системний аналіз, методи математичної статистики, а також методи програмної інженерії - зокрема, проектування програмних систем, розробка REST API, робота з базами даних і фронтенд-технологіями.

Наукова новизна полягає у створенні програмного забезпечення, що забезпечує агрегований аналіз показників швидкодії з кількох джерел одночасно, зберігає історію перевірок та автоматично формує порівняльні звіти для подальшого використання у процесі оптимізації веб-додатків.

Практичне значення роботи полягає у можливості застосування створеної системи в реальних умовах - для оцінювання продуктивності корпоративних або навчальних веб-сайтів, моніторингу технічного стану веб-додатків і підтримки процесів SEO-оптимізації.

Таким чином, тема роботи є актуальною та має як теоретичну, так і практичну цінність, спрямовану на вирішення важливої задачі підвищення ефективності веб-ресурсів за допомогою сучасних програмних засобів аналізу швидкодії.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Сучасний стан і тенденції розвитку веб-технологій

Веб-простір XXI століття характеризується надзвичайною динамічністю розвитку технологій і постійним зростанням кількості веб-додатків. Користувачі очікують не лише зручності, але й миттєвої реакції веб-ресурсу. У таких умовах продуктивність веб-додатків стає критичним фактором, який безпосередньо впливає на поведінку користувачів, комерційні результати та SEO-рейтинги.

Згідно з дослідженням компанії Akamai Technologies, затримка завантаження сторінки лише на 100 мілісекунд може знизити рівень конверсії на 7 %, а швидкість завантаження понад 3 секунди призводить до втрати близько половини потенційних відвідувачів.

Проблема продуктивності є комплексною і залежить від багатьох чинників:

1. розмір і кількість ресурсів (зображень, скриптів, шрифтів);
2. ефективність використання кешування та CDN;
3. особливості рендерингу на клієнті;
4. структура DOM і оптимізація CSS;
5. вплив сторонніх бібліотек і трекерів.

Саме тому аналіз технік швидкого завантаження веб-сторінок потребує системного підходу, що враховує як технічні аспекти серверної та клієнтської частини, так і сучасні стандарти веб-розробки.

Окрему тенденцію розвитку веб-технологій становить використання edge computing - розміщення серверних обчислень ближче до користувача,

що дозволяє скоротити час відповіді до 50-100 мс навіть для великих сайтів. Крім того, зростає популярність serverless-архітектур (AWS Lambda, Google Cloud Functions), які забезпечують гнучке масштабування ресурсів без необхідності постійного обслуговування серверів. Останні роки також спостерігається інтеграція штучного інтелекту (AI) у процеси оптимізації контенту та прогнозування навантаження, що дає змогу автоматично адаптувати ресурси сторінки під поведінку користувачів. Ці підходи формують новий рівень ефективності веб-систем, де швидкодія стає одним із ключових показників якості цифрового продукту.

1.2. Еволюція веб-продуктивності

Перші покоління веб-сайтів у 1990-х роках були статичними - сторінки містили мінімум коду, а їх завантаження займало частки секунди.

З появою JavaScript, CSS та AJAX інтерфейси стали інтерактивними, але обсяг переданих даних значно зріс.

Подальший розвиток - поява SPA (Single Page Applications), фреймворків (React, Angular, Vue.js) - зробив веб-додатки більш функціональними, але водночас важчими для рендерингу.

У відповідь на ці виклики з’явилися сучасні технології підвищення швидкодії:

1. HTTP/2 і HTTP/3, які забезпечують мультиплексування запитів;
2. Lazy Loading - відкладене завантаження медіафайлів;
3. Service Workers і PWA (Progressive Web Apps);
4. Preload, Prefetch, DNS Prefetch - попереднє завантаження ресурсів;
5. Code splitting та Tree shaking для мінімізації JavaScript.

Таким чином, веб-продуктивність еволюціонувала від простої оптимізації зображень до комплексного управління ресурсами на рівні браузера й сервера.

1.3. Метрики ефективності завантаження веб-сторінок

Для об’єктивної оцінки швидкодії застосовується система метрик, розроблена Google у межах ініціативи Core Web Vital, яка наведена у таблиці 1.1 .

Таблиця 1.1

Метрика	**Опис**	**Оптимальне значення**
LCP (Largest Contentful Paint)	Час від початку завантаження сторінки до відображення найбільшого елемента контенту (зображення, відео, текстовий блок).	≤ 2,5 с
FID (First Input Delay)	Затримка між першою взаємодією користувача (клік, торкання) і реакцією браузера.	≤ 100 мс
CLS (Cumulative Layout Shift)	Вимірює стабільність верстки під час завантаження.	≤ 0,1
TBT (Total Blocking Time)	Сумарний час, коли головний потік браузера блокується під час виконання скриптів.	≤ 200 мс
SI (Speed Index)	Показує, наскільки швидко сторінка стає візуально повною.	≤ 4 с

Важливо відзначити, що метрики Core Web Vitals безпосередньо впливають не лише на технічну якість веб-додатку, а й на його SEO-позиції в пошукових системах.

Алгоритми Google у 2023-2025 роках враховують показники LCP, CLS і FID

як частину Page Experience Update, що визначає рейтинг сторінок у видачі. Таким чином, швидкість завантаження стає не лише технічним, а й маркетинговим фактором успіху.

1.4. Аналіз існуючих інструментів вимірювання продуктивності

Для аналізу швидкодії веб-сторінок існує багато інструментів. Найпопулярніші з них наведено у таблиці 1.2.

Таблиця 1.2

Інструмент	**Джерело даних**	**Особливості**	**Недоліки**
Google Lighthouse	Локальний або хмарний аудит Chrome	Генерує детальний звіт Core Web Vitals, доступ через API	Відсутня історія перевірок
GTmetrix	Поєднання Lighthouse + WebPageTest	Візуалізація, історія, рекомендації	Обмежений безкоштовний доступ
WebPageTest	Реальні браузери, різні регіони	Висока точність, можливість порівняння	Складний інтерфейс
Pingdom Tools	Хмарна платформа	Проста оцінка часу завантаження	Менше технічних деталей
SpeedCurve	Комерційна платформа	Моніторинг у часі, збереження історії	Платна підписка

Як видно з таблиці, кожен з інструментів має власну специфіку. Водночас жоден із них не забезпечує зручного агрегування даних з кількох джерел та порівняння результатів у динаміці.

Попри широкий вибір інструментів, більшість з них виконують лише разові вимірювання та не забезпечують довгострокового моніторингу змін у часі.

Це ускладнює оцінку ефективності впроваджених оптимізацій і порівняння різних технік.

Розроблена в межах цієї роботи система усуває ці обмеження - вона зберігає історію перевірок.

1.5. Методи підвищення швидкодії веб-ресурсів

Оптимізація швидкодії включає низку технічних підходів:

- Мінімізація HTTP-запитів - об’єднання скриптів і стилів.
- Використання CDN - розподіл контенту між географічно рознесеними серверами.
- Кешування - застосування заголовків Cache-Control, ETag.
- Оптимізація зображень - використання форматів WebP, AVIF.
- Критичні CSS - завантаження лише необхідних стилів для першого екрану.
- Defer/Async для JS - відкладене виконання скриптів.
- HTTP/2 Push - попереднє завантаження ресурсів, необхідних браузеру.

Дотримання цих рекомендацій дозволяє скоротити час завантаження в середньому на 30-50 %.

1.6. Основні підходи до моніторингу та аналітики

Моніторинг швидкодії можна реалізувати двома основними способами:

- Лабораторні вимірювання (Lab Data) - штучно створене середовище (наприклад, Lighthouse у Chrome).

- Польові дані (Field Data) - реальні метрики з браузерів користувачів (CrUX - Chrome User Experience Report).

Комбінація обох підходів забезпечує комплексну картину - від симульованого середовища до реальної поведінки користувачів.

1.7. Вплив мережевих факторів на швидкодію веб-додатків

Швидкодія веб-додатків безпосередньо залежить не лише від ефективності коду чи оптимізації клієнтської частини, але й від характеристик мережі, через яку здійснюється передавання даних. Основними параметрами, що визначають мережеву продуктивність, є пропускна здатність каналу, затримка (latency), кількість HTTP-запитів і розмір переданих ресурсів.

Пропускна здатність (Bandwidth) - це максимальний обсяг даних, який може бути передано через канал зв’язку за одиницю часу. При низькій пропускній здатності навіть оптимізований сайт завантажується повільно, особливо якщо сторінка містить велику кількість зображень або скриптів. Наприклад, при швидкості 2 Мбіт/с завантаження сторінки розміром 3 МБ триватиме понад 12 секунд, тоді як при швидкості 50 Мбіт/с - менше ніж секунду.

Затримка (Latency) - це час між відправленням запиту користувача та отриманням відповіді від сервера. Висока затримка особливо відчутна при великій кількості дрібних запитів (наприклад, завантаженні багатьох скриптів або CSS-файлів). Для зниження latency використовуються технології HTTP/2, Keep-Alive, DNS-prefetch та CDN, які скорочують кількість повторних з’єднань і наближають ресурси до користувача.

Кількість HTTP-запитів безпосередньо впливає на час завантаження: кожен запит створює нове з'єднання, додаючи накладні витрати. Оптимізація полягає у зменшенні числа запитів через об'єднання стилів і скриптів, використання спрайтів і кешування.

Розмір ресурсів - це сума ваги HTML, CSS, JS, зображень і шрифтів, які браузер завантажує під час рендерингу сторінки. Кожні додаткові 500 КБ можуть збільшити загальний час завантаження на 0,5-1 секунду при середній швидкості мережі.

Для практичного вимірювання параметрів мережі використовуються утиліти Chrome DevTools, WebPageTest або Lighthouse, які дають змогу симулювати різні швидкості підключення - від 3G до 5G. Зокрема, затримка у 3G-мережі може сягати 300-500 мс, тоді як у 5G - не перевищує 50 мс.
Це означає, що навіть при ідеальній оптимізації серверної частини користувачі з повільним інтернетом можуть відчувати затримку. Тому під час тестування системи важливо моделювати різні умови з'єднання, що забезпечує об'єктивну оцінку швидкодії.

1.8. Порівняння підходів до оптимізації (Frontend vs Backend)

Оптимізація швидкодії веб-додатків може виконуватися на двох основних рівнях - клієнтському (frontend) і серверному (backend). Їх взаємодія визначає загальний користувацький досвід і продуктивність системи (табл. 1.3).

Frontend-оптимізація спрямована на скорочення часу відображення сторінки в браузері. Основні методи:

- **Мінімізація та компресія коду** - використання інструментів *Webpack*, *UglifyJS*, *CSSNano* для зменшення обсягу JS і CSS.
- **Lazy Loading** - відкладене завантаження зображень і компонентів, які не відображаються одразу.
- **Code Splitting** - розбиття коду на окремі частини, щоб користувач отримував лише необхідне.
- **Prefetch / Preload** - попереднє завантаження ресурсів, які, ймовірно, знадобляться.
- **Оптимізація DOM і стилів** - зменшення вкладеності елементів, використання критичних CSS.

Backend-оптимізація зосереджується на зменшенні часу відповіді сервера й оптимальному розподілі навантаження:

- **Кешування на рівні сервера** - збереження готових HTML-сторінок для повторних запитів.
- **Використання CDN** - розповсюдження статичних ресурсів на географічно розподілених вузлах.
- **Стиснення даних** - активація GZIP або Brotli для зменшення обсягу переданих відповідей.
- **HTTP/2 та HTTP/3** - паралельна передача запитів в одному з’єднанні.
- **Балансування навантаження** між кількома серверами та оптимізація бази даних.

Таблиця 1.3

Підхід	Основна мета	Типові засоби	Очікуваний ефект
Frontend	Зменшення часу рендерингу сторінки	Мінімізація JS/CSS, Lazy Loading, Prefetch	Покращення UX, зниження TBT та LCP
Backend	Зменшення часу відповіді сервера	Кешування, CDN, HTTP/2, оптимізація БД	Зниження TTFB, підвищення стабільності

Окрім наведених фреймворків, у сучасній розробці дедалі більшого поширення набувають Angular, який забезпечує високу масштабованість корпоративних додатків, та Svelte, що компілює компоненти у чистий JavaScript без зайвих бібліотек, мінімізуючи час рендерингу. Новітні рішення, такі як Astro, орієнтовані на частковий рендеринг (partial hydration), завдяки чому сторінки завантажуються миттєво навіть на мобільних пристроях.

1.9. Використання сучасних фреймворків для прискорення веб-додатків

Сучасні фреймворки фронтенд-розробки значно впливають на продуктивність веб-додатків завдяки вбудованим механізмам оптимізації, управління рендерингом і можливостям попереднього завантаження контенту (табл. 1.4).

React.js забезпечує високу ефективність завдяки використанню Virtual DOM, який мінімізує кількість реальних змін у структурі сторінки. Компонентний підхід дозволяє повторно використовувати елементи, що скорочує обсяг коду. React підтримує Server-Side Rendering (SSR), що дозволяє формувати HTML на сервері та швидше показувати сторінку користувачу.

Vue.js відзначається легкістю та швидким часом початкового рендерингу. Його реактивна система відстежує зміни стану компонентів, оновлюючи лише ті частини DOM, які реально змінилися. Vue також підтримує SSR і статичну генерацію через Nuxt.js.

Next.js - фреймворк на базі React, який забезпечує поєднання SSR, SSG (Static Site Generation) і клієнтського рендерингу. Завдяки цьому сторінки завантажуються майже миттєво, а SEO-індексація відбувається ефективніше. Крім того, Next.js автоматично виконує оптимізацію зображень і попереднє завантаження посилань (link prefetching).

Таблиця 1.4

Фреймворк	**Основні методи оптимізації**	**Переваги**
React.js	Virtual DOM, SSR, Code Splitting	Висока продуктивність, компонентність
Vue.js	Реактивність, мінімальний розмір ядра, Nuxt.js	Швидкий початковий рендеринг
Next.js	SSR + SSG, автоматичне кешування, оптимізація зображень	Мінімальний TTFB, покращене SEO

1.10. Постановка завдання дослідження

На основі проведеного аналізу можна зробити висновок, що існуючі інструменти оцінки швидкодії веб-сторінок забезпечують лише часткове вирішення проблеми оптимізації. Вони орієнтовані на разове тестування конкретної сторінки й не враховують вплив комбінацій технік оптимізації.

Таким чином, постає науково-практичне завдання розроблення програмного забезпечення, яке:

1. автоматично аналізує ключові параметри завантаження веб-сторінок;

2. виконує порівняльну оцінку ефективності різних технік оптимізації;
3. інтегрує результати з існуючих API для формування розширених звітів;
4. генерує рекомендації щодо підвищення швидкодії веб-ресурсів у зручній формі для розробників і адміністраторів.

Розв’язання цього завдання дозволить створити інструмент, здатний забезпечити комплексний підхід до аналізу продуктивності веб-сторінок і сприяти підвищенню ефективності розробки сучасних веб-додатків.

РОЗДІЛ 2. МОДЕЛЮВАННЯ СИСТЕМИ АНАЛІЗУ ШВИДКОГО ЗАВАНТАЖЕННЯ ВЕБ-СТОРІНОК

2.1. Визначення функціональних вимог

На основі системного аналізу предметної області було сформульовано ключові вимоги до розроблюваної системи.

Функціональні вимоги:

1. Система повинна приймати від користувача адресу веб-сайту (URL) для аналізу.
2. Після введення URL, система має звертатися до Google PageSpeed Insights API для отримання показників продуктивності.
3. Отримані метрики повинні оброблятися на сервері.
4. Результати аналізу мають зберігатися у базі даних.
5. На клієнтській частині система повинна відображати отримані метрики у вигляді інтерактивних графіків і повзунків оцінки швидкодії.
6. Для кожної метрики мають надаватися рекомендації щодо покращення показників.

Нефункціональні вимоги:

1. Забезпечення швидкої обробки запитів.
2. Сумісність із сучасними браузерами (Chrome, Edge, Firefox).
3. Висока надійність обміну даними через HTTPS.
4. Простота інтерфейсу та інтуїтивна навігація.

Формулювання наведених вимог обумовлене необхідністю створення системи, здатної ефективно працювати як у навчальному, так і в

комерційному середовищі. Вимоги щодо масштабованості та стабільності є критичними, оскільки кількість користувачів може динамічно зростати. Система має бути спроєктована з урахуванням можливості додавання нових модулів без змін базової архітектури, наприклад інтеграції з іншими API чи розширення функцій аналітики. Такий підхід відповідає принципам гнучкої розробки (Agile) та забезпечує довготривалу життєздатність програмного продукту.

2.2. Функціональна діаграма роботи системи

Функціональна діаграма відображає послідовність дій користувача та системи під час виконання основних процесів аналізу швидкодії веб-сторінок (рис. 2.1).

Рисунок 2.1 - Функціональна діаграма роботи програмного забезпечення системи аналізу технік швидкого завантаження веб-сторінок

Процес роботи системи починається з запуску програми та автентифікації користувача. Після успішного входу користувач вводить адресу веб-ресурсу для аналізу. Система перевіряє доступність зазначеної сторінки. Якщо сторінка недоступна - користувачу виводиться повідомлення

про помилку. Якщо доступна - система надсилає запит до PageSpeed Insights API, отримує показники продуктивності та виконує їх обробку.

Далі проводиться аналіз повільних елементів, формується звіт про результати тестування та генеруються рекомендації щодо оптимізації швидкодії. На завершальному етапі користувач може експортувати звіт у зручному форматі для подальшого використання або архівування.

Функціональна діаграма наочно демонструє основну логіку роботи системи - від початку процесу до отримання результатів аналізу, а також усі можливі розгалуження сценарію, включно з обробкою помилок.

Окрім основного сценарію, система передбачає обробку виняткових ситуацій.
Наприклад, у разі недоступності зовнішнього API користувачу виводиться повідомлення про затримку з можливістю повторного запиту без перезавантаження сторінки.

При повторному аналізі того самого ресурсу система порівнює отримані результати з попередніми, що дає змогу визначити динаміку продуктивності. Такий підхід забезпечує більш точну аналітику й дозволяє виявляти ефективність упроваджених оптимізацій.

2.3. Архітектурна модель системи

Архітектура програмного забезпечення побудована за трирівневою моделлю (рис. 2.2):

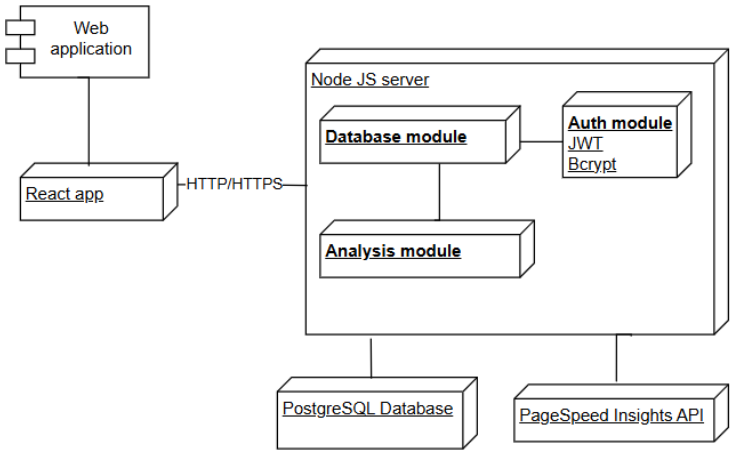

Рисунок 2.2 - Архітектура програмного забезпечення системи аналізу технік швидкого завантаження

Клієнтський рівень (Front-end) - реалізований на фреймворку React, відповідає за інтерфейс користувача, введення URL, відображення результатів і рекомендацій.

Серверний рівень (Back-end) - реалізований на Node.js, виконує запити до PageSpeed Insights API, обробляє JSON-відповіді, нормалізує дані, формує структуру для відображення.

Рівень даних (Database) - зберігає історію проведених аналізів (URL, дата, результати метрик, загальна оцінка) реалізована на PostgreSQL.

Вибір трирівневої архітектури обумовлений необхідністю балансування між простотою розгортання та гнучкістю розширення. На відміну від монолітних рішень, де всі компоненти об’єднані в єдиний застосунок, така модель дозволяє незалежно оновлювати клієнтську, серверну та базову частини системи. Водночас повноцінна мікросервісна архітектура була відхилена через надмірну складність обслуговування на початковому етапі, що не є доцільним для навчальних і дослідницьких проєктів.

Таким чином, обрана структура є оптимальною для забезпечення масштабованості, розширюваності й контрольованості системи.

2.4. Модель бази даних

База даних системи реалізована на основі реляційної моделі та побудована у середовищі PostgreSQL.

Вона складається з чотирьох основних таблиць: Plans, Users, Websites та WebsiteMetrics, між якими встановлено логічні зв’язки(рис 2.3.).

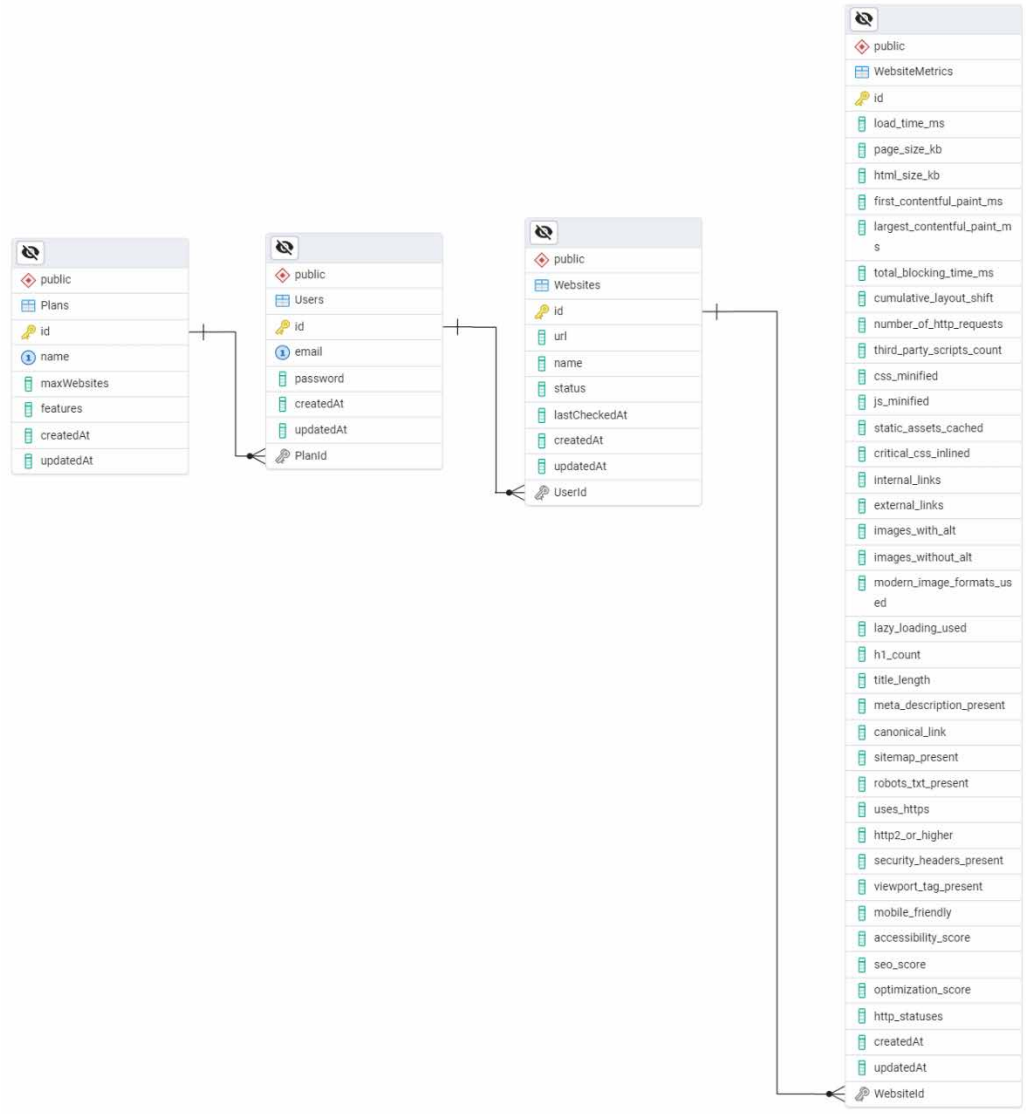

Рисунок 2.3 - База даних системи

Таблиця Plans

Містить інформацію про тарифні плани користувачів системи.

1. id - унікальний ідентифікатор плану;
2. name - назва плану;
3. maxWebsites - максимальна кількість сайтів, які користувач може додати;
4. features - опис можливостей або обмежень плану;
5. createdAt, updatedAt - дати створення та оновлення запису.

Таблиця Users

Зберігає дані про зареєстрованих користувачів.

1. id - унікальний ідентифікатор користувача;
2. email, password - дані для автентифікації;
3. createdAt, updatedAt - дати створення та оновлення запису;
4. planId - зовнішній ключ, що пов’язує користувача з таблицею Plans.

Таблиця Websites

Містить інформацію про веб-ресурси, додані користувачами для аналізу.

1. id - унікальний ідентифікатор веб-сайту;
2. url, name - адреса та назва веб-сайту;
3. status - стан перевірки (активний, помилка тощо);
4. lastCheckedAt - дата останнього аналізу;
5. createdAt, updatedAt - дати створення та оновлення;
6. userId - зовнішній ключ, що пов’язує сайт із конкретним користувачем.

Таблиця WebsiteMetrics

Зберігає результати аналізу швидкодії та оптимізації веб-сторінок.

1. id - унікальний ідентифікатор запису;
2. load_time_ms, page_size_kb, html_size_kb - основні показники швидкодії;
3. first_contentful_paint_ms, largest_contentful_paint_ms, total_blocking_time_ms, cumulative_layout_shift - ключові метрики, отримані з PageSpeed Insights API;
4. number_of_http_requests, third_party_scripts_count, css_minified, js_minified тощо - технічні показники оптимізації;
5. seo_score, accessibility_score, optimization_score - інтегральні оцінки якості сайту;
6. createdAt, updatedAt - дати створення та оновлення запису;
7. websiteId - зовнішній ключ, що пов’язує дані з таблицею Websites.

Зв’язки між таблицями

1. Один Plan може бути пов’язаний із багатьма Users (зв’язок *one-to-many*).
2. Один User може мати декілька Websites (зв’язок *one-to-many*).
3. Кожен Website має одну або кілька записів у таблиці WebsiteMetrics (зв’язок *one-to-many*).

2.5. Діаграма прецедентів (Use Case Diagram)

Діаграма прецедентів відображає основні сценарії взаємодії користувачів із системою та допомагає визначити її функціональні можливості з точки зору зовнішніх акторів (рис. 2.4). Для розробленого програмного забезпечення системи аналізу технік швидкого завантаження веб-сторінок було визначено три основні ролі користувачів:

- **Користувач** - взаємодіє із системою для проведення аналізу швидкодії веб-сторінок, налаштовує тестування, отримує звіти з рекомендаціями та переглядає результати оптимізації.
- **Системний адміністратор** - відповідає за технічну підтримку серверної частини системи, моніторинг продуктивності серверів та оптимізацію конфігурацій при високих навантаженнях.
- **Інженер з продуктивності** - проводить глибший аналіз ефективності технік оптимізації, здійснює тестування під навантаженням і виявляє “вузькі місця” у роботі системи.

Взаємодія акторів і системи наведена на рисунку 2.4.

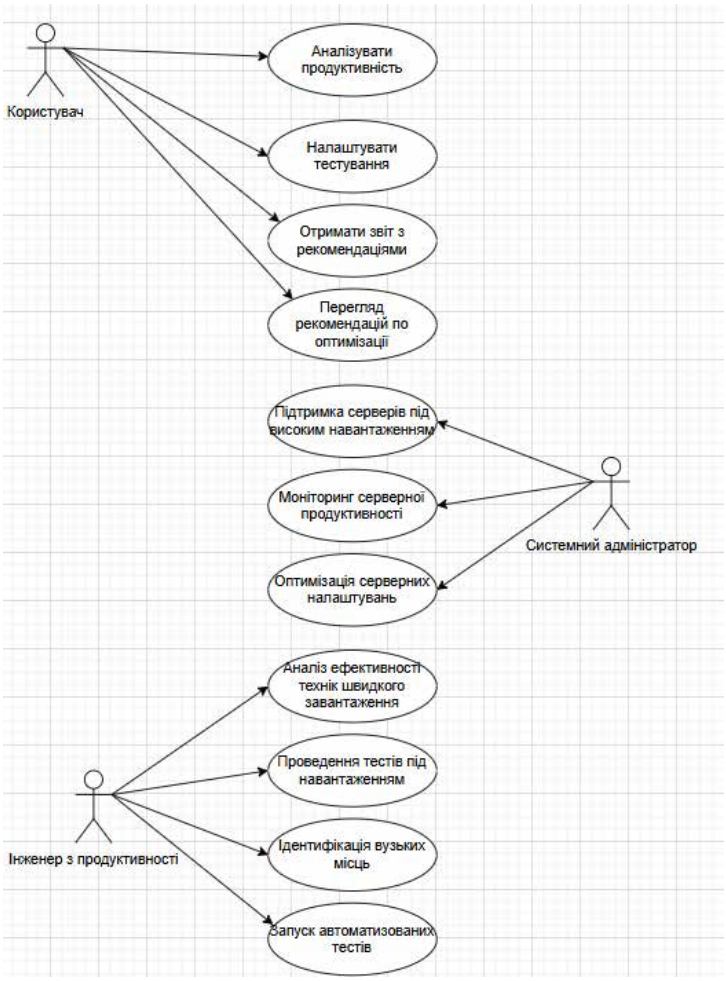

Рисунок 2.4 - Діаграма прецедентів роботи системи

Основні прецеденти:

Для користувача:

- **Аналізувати продуктивність** - виконання оцінки швидкодії веб-сайту через інтерфейс.
- **Налаштувати тестування** - вибір параметрів аналізу (тип пристрою, середовище, частота).
- **Отримати звіт з рекомендаціями** - перегляд сформованого системою звіту за результатами аналізу.
- **Перегляд рекомендацій по оптимізації** - детальний огляд порад щодо зменшення часу завантаження.

Для системного адміністратора:

- **Підтримка серверів під високим навантаженням** - забезпечення стабільної роботи API та бази даних.
- **Моніторинг серверної продуктивності** - контроль використання ресурсів і часу відгуку.
- **Оптимізація серверних налаштувань** - налаштування кешування, балансування навантаження, логування.

Для інженера з продуктивності:

- **Аналіз ефективності технік швидкого завантаження** - порівняння результатів до та після оптимізації.
- **Проведення тестів під навантаженням** - імітація одночасного доступу користувачів до сервісу.

- **Ідентифікація вузьких місць** - виявлення ділянок коду чи компонентів, що уповільнюють систему.
- **Запуск автоматизованих тестів** - використання скриптів і CI/CD-інструментів для перевірки стабільності.

Таким чином, діаграма прецедентів демонструє повний набір можливостей системи для різних категорій користувачів і дозволяє забезпечити прозорість функціональних зв’язків між компонентами та акторами.

2.6. Бізнес модель канвас

Для розроблюваного програмного забезпечення було сформовано бізнес-модель за методологією Business Model Canvas, яка дозволяє визначити ключові компоненти створення цінності, логіку взаємодії з користувачами та можливості масштабування проєкту (рис 2.5).

Ключові партнери.

До основних партнерів системи належать постачальники API-сервісів для аналізу швидкодії веб-сторінок (Google PageSpeed Insights, WebPageTest, Lighthouse), а також хостинг-провайдери та серверні платформи (AWS, Vercel, Render). Окрему групу становлять CMS-платформи (WordPress, Shopify, Webflow), інтеграція з якими дозволяє спростити підключення сервісу для кінцевих користувачів.

Ключові види діяльності.

Основними видами діяльності є розробка та підтримка програмного забезпечення, інтеграція зовнішніх API, тестування функціональності,

удосконалення інструментів аналітики, а також технічна підтримка користувачів. Додатково проводиться дослідження нових технік оптимізації швидкодії, що забезпечує актуальність рекомендацій.

Ключові ресурси.

До ресурсів системи належать технологічна платформа (серверне середовище та база даних), API-доступ до сервісів аналізу швидкодії, команда розробників, а також історичні дані продуктивності веб-ресурсів, що накопичуються в процесі експлуатації.

Ціннісні пропозиції.

Програмне забезпечення забезпечує автоматизований аналіз швидкодії веб-сайтів із використанням сучасних метрик Core Web Vitals та формуванням рекомендацій для їх оптимізації. Система дозволяє зберігати історію вимірювань, порівнювати результати тестування до та після впровадження покращень, а також інтегруватися з популярними CMS. Завдяки цьому користувач отримує інструмент для комплексного моніторингу продуктивності веб-ресурсу та підвищення рейтингу в пошукових системах.

Взаємовідносини з клієнтами.

Підтримка користувачів здійснюється через веб-інтерфейс системи, чат-підтримку та e-mail. Для бізнес-клієнтів доступні персональні консультації з аналізу продуктивності, а також навчальні матеріали й відеоінструкції.

Канали розповсюдження.

Основними каналами взаємодії з користувачами є веб-сайт продукту, реклама в пошукових системах (Google Ads), SEO-просування та професійні спільноти веб-розробників.

Сегменти клієнтів.

Система орієнтована на веб-розробників, SEO-спеціалістів, UI/UX-дизайнерів, власників веб-ресурсів та адміністраторів сайтів. Це користувачі, які потребують інструменту для технічної оцінки стану сайту та покращення показників завантаження.

Структура витрат.

Основними статтями витрат є розробка та оновлення програмного забезпечення, хостинг і обробка даних, оплата API-запитів, технічна підтримка користувачів і маркетинг.

Джерела доходу.

Модель монетизації базується на передплаті (SaaS-модель) із кількома тарифними планами (Basic, Pro, Enterprise), а також можливості придбання окремого API-доступу для інтеграції результатів аналізу в зовнішні системи.

Потенційними напрямами комерційного використання системи можуть бути веб-агенції, які займаються технічною оптимізацією сайтів, а також освітні заклади, де система може використовуватись як навчальний інструмент для дисциплін з веб-інженерії.
Крім того, система має перспективи інтеграції у великі корпоративні аналітичні платформи для моніторингу стану веб-додатків у реальному часі. Таким чином, розроблений продукт має потенціал не лише як дослідницький

прототип, а й як повноцінний SaaS-сервіс, орієнтований на ринок IT-аналітики.

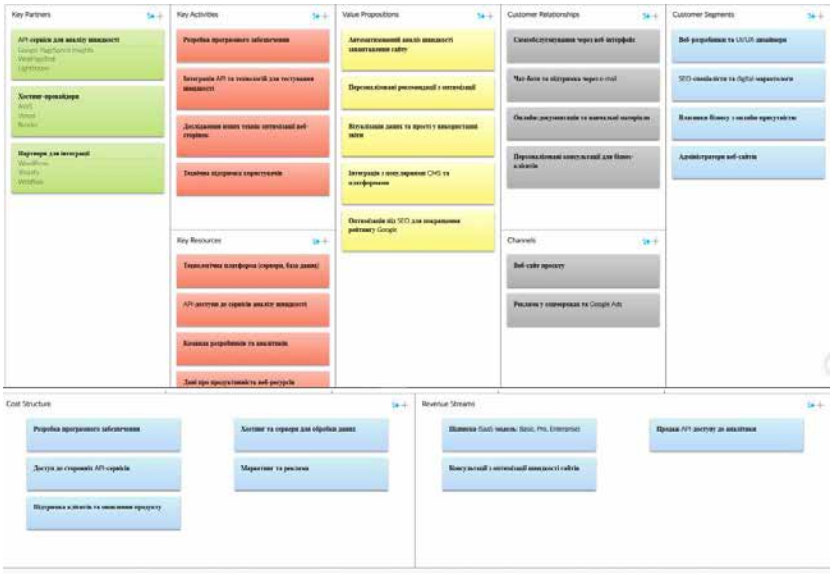

Рисунок 2.5 - Бізнес-модель Canvas

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Вибір середовища та інструментів розробки

Для реалізації програмного забезпечення системи аналізу технік швидкого завантаження веб-сторінок використано сучасний набір бібліотек, фреймворків та інструментів, що забезпечують високу продуктивність, масштабованість і безпеку.

Система реалізована за архітектурою “клієнт-сервер”, де клієнтська частина відповідає за інтерфейс і взаємодію з користувачем, а серверна - за логіку обробки запитів, авторизацію та комунікацію з PageSpeed Insights API.

Клієнтська частина (Front-end)

Розроблена з використанням фреймворку React, що забезпечує компонентну структуру та динамічне оновлення інтерфейсу. Для керування станом застосунку використано reduxjs/toolkit і react-redux, що дозволяє ефективно організувати взаємодію між компонентами.

Основні бібліотеки та інструменти:

1. axios - для виконання HTTP-запитів до серверного API;
2. react-router-dom - маршрутизація між сторінками застосунку;
3. react-range - створення інтерактивних повзунків для відображення метрик швидкодії;
4. recharts - побудова графіків і візуалізація результатів аналізу;
5. keen-slider - реалізація слайдерів та плавної навігації;
6. framer-motion - анімація елементів інтерфейсу;

7. react-google-recaptcha - захист від автоматизованих запитів;

Інтерфейс реалізовано за принципами адаптивного дизайну з використанням CSS Flexbox, Grid і фреймворку Bootstrap, що забезпечує коректне відображення на мобільних пристроях.

Серверна частина (Back-end)

Сервер побудовано на платформі Node.js із використанням фреймворку Express, що забезпечує легку побудову REST API.

Основні бібліотеки серверної частини:

1. axios - інтеграція з Google PageSpeed Insights API;
2. sequelize - ORM для управління структурами таблиць і виконання запитів SQL;
3. jsonwebtoken - реалізація авторизації за допомогою токенів (JWT);
4. bcryptjs - шифрування паролів користувачів;
5. cors - дозволяє клієнтським запитам з інших доменів;
6. express-rate-limit - обмеження кількості запитів до сервера для захисту від DDoS-атак.

База даних

У системі використано PostgreSQL, яка забезпечує зберігання результатів аналізу, даних користувачів і тарифних планів.

Доступ до бази здійснюється через ORM Sequelize, що забезпечує підтримку транзакцій, зв’язків між таблицями та валідацію даних.

Вибір саме стеку React + Node.js + PostgreSQL обумовлений необхідністю забезпечити високу продуктивність, гнучкість і простоту

розгортання.

Альтернативою могла б стати зв’язка Angular + Firebase, проте вона менш зручна у кастомізації запитів і не підтримує складні аналітичні операції з базою даних.

3.2. Алгоритм роботи системи

Загальна логіка роботи системи:

1. Користувач вводить URL веб-сайту у відповідне поле інтерфейсу.
2. Front-end надсилає HTTP-запит на сервер.
3. Серверна частина виконує запит до **PageSpeed Insights API**, передаючи URL і параметри середовища (desktop або mobile).
4. API повертає JSON-відповідь із набором метрик:
   - **Performance Score**,
   - **Largest Contentful Paint (LCP)**,
   - **First Contentful Paint (FCP)**,
   - **Cumulative Layout Shift (CLS)**,
   - **Total Blocking Time (TBT)**,
   - **Speed Index (SI)**.
5. Сервер обробляє отримані дані, зберігає їх у базі PostgreSQL, розраховує інтегральні показники ефективності.
6. Клієнтська частина відображає метрики у вигляді повзунків і графіків.
7. Система формує рекомендації щодо покращення швидкодії (зменшити обсяг зображень, мінімізувати JS, активувати кешування тощо).
8. Користувач може переглянути результати, експортувати звіт або провести повторний аналіз.

Для підвищення стабільності система використовує багаторівневу обробку помилок.

У разі, якщо зовнішній сервіс PageSpeed API тимчасово недоступний, сервер повертає повідомлення про затримку, а запит автоматично ставиться у чергу повторної спроби.

Така реалізація зменшує ризик втрати даних і підвищує надійність роботи під час великої кількості одночасних запитів.

Додатково застосовується кешування останніх результатів аналізу, що дозволяє прискорити повторні запити для одного й того ж ресурсу до 80%.

3.3. Реалізація серверної частини

Серверна частина системи реалізована за архітектурною моделлю MVC (Model-View-Controller), що забезпечує логічне розділення коду, спрощує супровід і масштабування програмного забезпечення(рис.3.1).

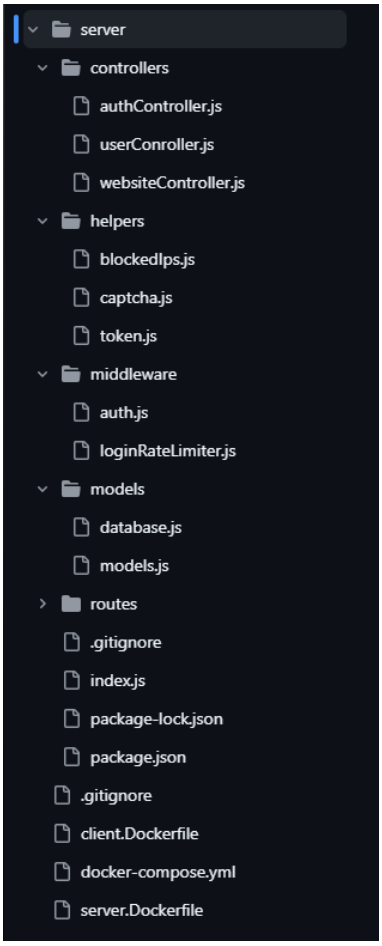

Рисунок 3.1 - Структура серверної частини програмного забезпечення

Директорія server містить основні підкаталоги та файли:

- controllers/ - реалізує бізнес-логіку системи, обробку запитів користувачів і роботу з моделями даних;
- middleware/ - проміжні функції перевірки доступу, авторизації та обмеження частоти запитів;
- models/ - опис структури бази даних і взаємозв’язків між таблицями;
- routes/ - визначає маршрути REST API, які об’єднують контролери й middleware;
- helpers/ - допоміжні модулі для роботи з токенами, IP-адресами та CAPTCHA;

- index.js - точка входу серверної частини, у якій відбувається підключення модулів, запуск сервера та ініціалізація бази даних;

Контролери (controllers/)

1. authController.js - реалізує логіку автентифікації користувачів, створення токенів доступу (JWT) і перевірку даних під час входу.
2. userController.js - відповідає за управління обліковими записами користувачів: реєстрацію, оновлення профілю, отримання інформації про активність.
3. websiteController.js - реалізує основний функціонал системи: прийом URL для аналізу, звернення до PageSpeed Insights API, збереження результатів у базі та повернення користувачу метрик продуктивності.

Допоміжні модулі (helpers/)

Папка helpers/ містить утиліти, які забезпечують повторне використання функцій у різних частинах коду:

1. blockedIps.js - перевірка IP-адрес, заблокованих через надмірну кількість запитів;
2. captcha.js - перевірка правильності введення Google reCAPTCHA;
3. token.js - створення та валідація JWT-токенів.

Проміжні обробники (middleware/)

1. Middleware-файли виконуються між отриманням запиту та викликом контролера:
2. auth.js - перевіряє наявність дійсного токена доступу в заголовку запиту;
3. loginRateLimiter.js - обмежує кількість запитів входу користувача протягом певного часу (захист від brute-force).

3.4. Реалізація клієнтської частини

Клієнтська частина системи реалізована у вигляді односторінкового веб-застосунку на базі фреймворку React із використанням архітектурного підходу Feature-Sliced Design (FSD), що забезпечує модульність, гнучкість і простоту розширення проєкту (рис.3.2).

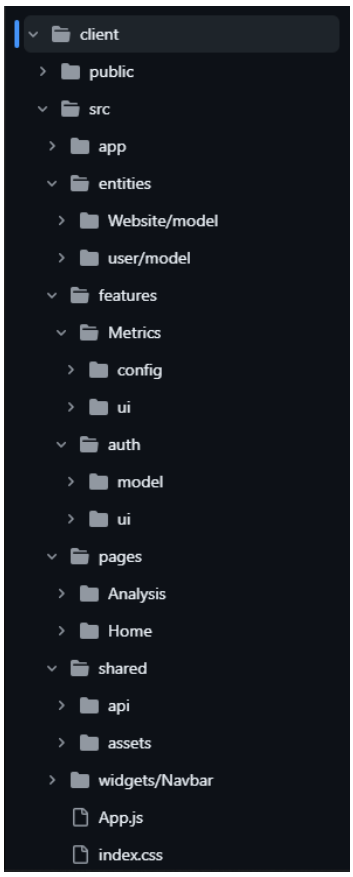

Рисунок 3.2 - Структура клієнтської частини системи

Директорія client містить такі основні компоненти:

1. Папка src/ - основний вихідний код проєкту

Містить усі файли, пов’язані з логікою, інтерфейсом і станом програми.

Основні підкаталоги:

1. **app/**

- App.js - головний компонент застосунку, який містить глобальну структуру сторінок і підключення до Redux Store.
- providers/router/ - містить компонент PrivateRoute.js, який реалізує захищену маршрутизацію (доступ лише після авторизації).
- store/store.js - конфігурація централізованого сховища стану Redux.
- router.jsx - конфігурація маршрутів для переходів між сторінками (Home, Analysis, Login тощо).

2. entities/

Містить логіку роботи з основними сутностями системи - Website і User.

Для кожної сутності створено окремий модуль model/, який включає:

- websiteSlice.js / userSlice.js - Redux-слайси для керування станом.
- fetchWebsites.js, fetchProfile.js - асинхронні Thunk-запити до серверного API.
- selectors.js - вибірка потрібних даних зі сховища стану.
- index.js - експорт функцій для зручного імпорту в інших модулях.

Такий підхід забезпечує ізольованість логіки кожної сутності й полегшує тестування.

3. features/

Відповідає за окремі функціональні можливості системи.

- Metrics/ - модуль, який реалізує візуалізацію метрик продуктивності.
- config/ - містить налаштування лімітів показників (metricLimits.js, mockMetric.js, seoMetric.js);
- ui/ - компонент SiteMetrics.jsx відображає значення Lighthouse-метрик за допомогою повзунків і кольорових індикаторів.

- auth/ - функціонал автентифікації користувачів.
- model/ - логіка входу, виходу, реєстрації (authSlice.js, login.js, register.js, logout.js);
- ui/ - компонент Auth.jsx реалізує форму входу з валідацією, анімацією та reCAPTCHA-захистом.

4. pages/

Містить сторінки системи - верхній рівень маршрутизації.

- Home/ - головна сторінка з інформаційними секціями:
- PromoSection.jsx - рекламний банер;
- PricingSection.jsx - тарифні плани;
- ReviewsSection.jsx - відгуки користувачів;
- HowItWorksSection.jsx - опис принципу роботи системи.
- Analysis/ - сторінка аналізу веб-сайтів:
- SitesSection.jsx - секція з оптимізованими секціями;
- BillingSection.jsx - інформація про тариф;
- ProfileSection.jsx - дані користувача.
- ui/Analysis.jsx - головний компонент сторінки, який об’єднує секції.

5. shared/

Містить спільні ресурси, що використовуються в різних частинах застосунку:

- api/ - конфігурація axiosInstance.js для запитів до бекенду з автоматичним додаванням токена авторизації;
- assets/ - медіафайли, іконки та стилі;
- index.css - глобальні стилі інтерфейсу.

При розробці інтерфейсу особлива увага приділялась доступності (Accessibility) та дотриманню стандартів WCAG 2.1.

Це означає, що всі елементи керування мають текстові альтернативи, контраст кольорів відповідає нормам для користувачів із вадами зору, а структура сторінок є семантично коректною.

Крім того, було реалізовано принципи reactive UI - інтерфейс миттєво реагує на дії користувача без перезавантаження сторінки, що забезпечує відчуття “живої” взаємодії.

Використання бібліотеки Framer Motion дозволило створити плавні анімації, які покращують сприйняття даних без перевантаження клієнтської частини.

3.5. Тестування програмного забезпечення

Тестування є невід’ємною частиною життєвого циклу програмного забезпечення і має на меті перевірку правильності роботи системи, її стабільності, продуктивності та відповідності вимогам, визначеним на етапі проектування. У розробленій системі тестування проводилось на декількох рівнях: модульному, інтеграційному, системному та навантажувальному.

Модульне тестування

Модульні тести проводились для перевірки роботи окремих функцій та компонентів бекенду. Для цього використовувалася бібліотека Jest, яка дозволяє ізолювати частини коду та перевіряти їх незалежно. Тестувались, зокрема:

- обробка запитів до API PageSpeed Insights;

- функції авторизації користувачів;
- модулі збереження та вибірки даних із бази PostgreSQL.

Модульні тести дозволили виявити логічні помилки на ранніх етапах розробки та забезпечити стабільність основного функціоналу системи.

Інтеграційне тестування

Інтеграційне тестування виконувалось із використанням Postman для перевірки взаємодії між клієнтською та серверною частинами. Основна увага приділялась перевірці правильності передачі даних між фронтендом, бекендом і базою даних, а також обробці некоректних або невалідних запитів користувача.

Було протестовано такі сценарії:

- успішна авторизація користувача;
- відмова при введенні неправильного пароля;
- запит до API з некоректною адресою сайту;
- обробка одночасних запитів від кількох користувачів.

Системне тестування

На етапі системного тестування перевірялась робота всієї системи в комплексі. Тестування включало:

- оцінку швидкості відповіді сервера;
- стабільність під час багаторазових запитів;
- коректність відображення метрик і рекомендацій на клієнтському інтерфейсі;
- відновлення працездатності після відмови сервера.

Усі тести були виконані у середовищі Node.js, PostgreSQL та браузерах Chrome і Opera. Система показала стабільну роботу при середньому навантаженні до 10 одночасних користувачів.

3.6. Інтерфейс користувача та демонстрація роботи системи

Розроблений веб-застосунок має інтуїтивно зрозумілий інтерфейс і забезпечує повний цикл взаємодії користувача з системою - від введення адреси сайту до отримання рекомендацій із покращення швидкодії.

Інтерфейс побудований у мінімалістичному стилі з акцентом на зручність, візуальність та адаптивність. Усі елементи динамічно оновлюються без перезавантаження сторінки завдяки використанню React і Redux Toolkit.

Для оформлення використано CSS Modules, Framer Motion та Bootstrap Icons.

1. Головна сторінка застосунку

На головній сторінці користувач може ознайомитися з принципом роботи сервісу, його перевагами (рис. 3.3), відгуками клієнтів (рис. 3.4), перейти до авторизації або перейти до аналізу сайту (рис. 3.5).

Рисунок 3.3 - Переваги

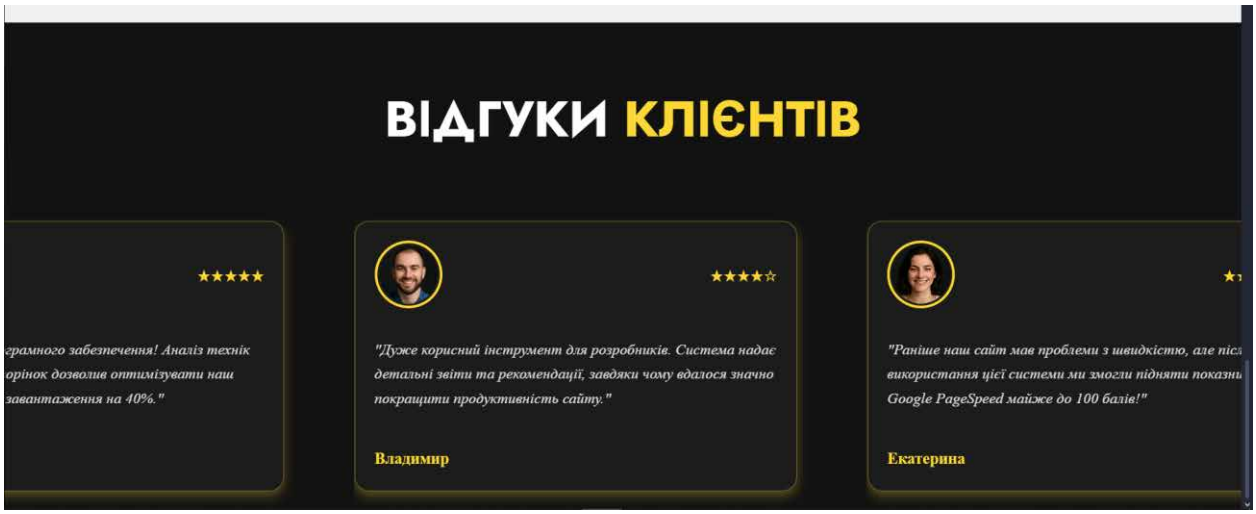

Рисунок 3.4 - Відгуки клієнтів

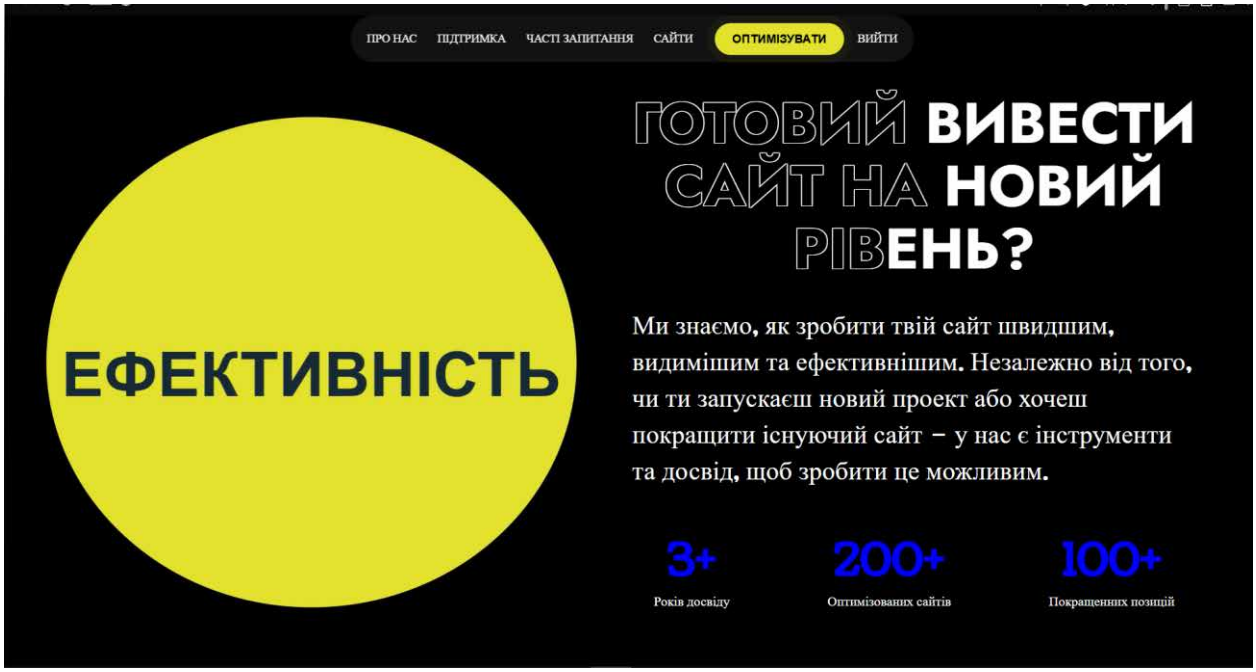

Рисунок 3.5 - Головна сторінка системи з описом функціоналу

2. Авторизація та реєстрація користувача

У системі реалізовано зручний інтерфейс для реєстрації та входу користувача.

Обидві форми оформлені у спокійній кольоровій гамі, мають зрозуміле розташування полів і витримані в єдиному стилі з усім веб-застосунком.

Форма реєстрації дозволяє створити новий обліковий запис, указавши основні дані користувача.

Передбачено візуальні підказки, повідомлення про успішне створення облікового запису та кнопки швидкого переходу на сторінку входу (рис. 3.6).

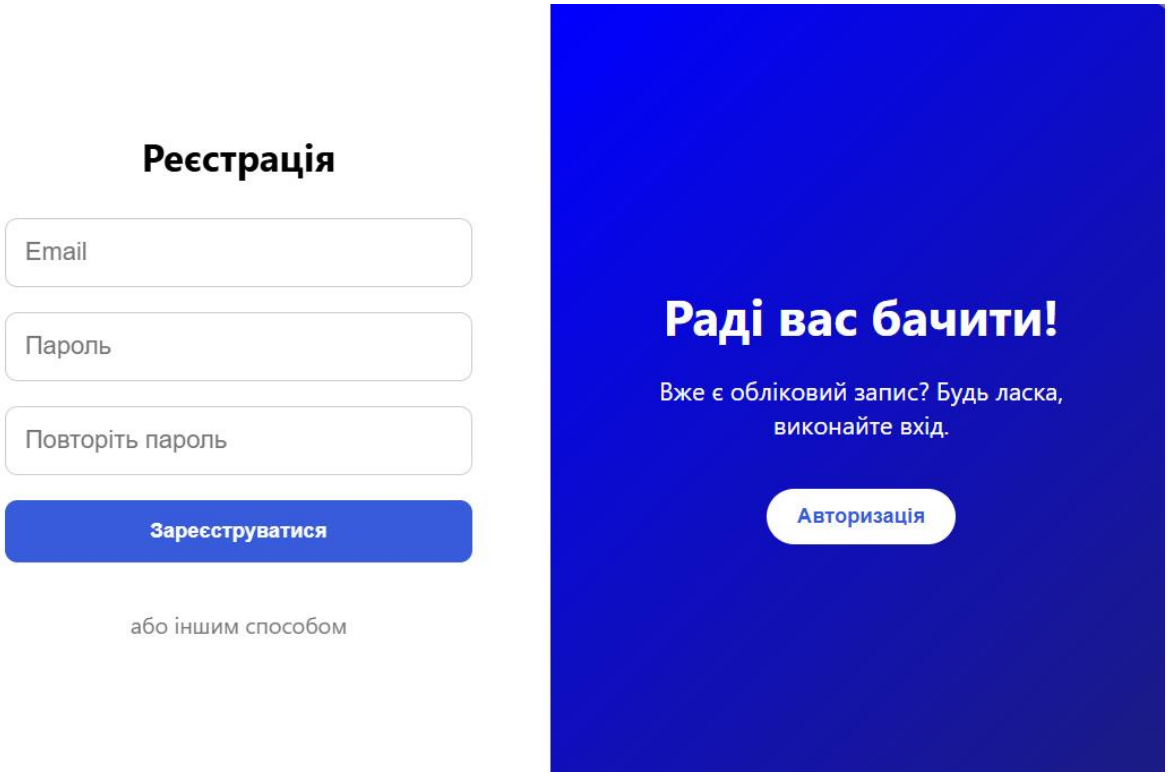

Рисунок 3.6 - Вікно реєстрації нового користувача системи

Сторінка авторизації дає змогу зареєстрованим користувачам увійти до системи для подальшої роботи.

Інтерфейс реалізовано з акцентом на простоту та зручність: усі елементи розміщено логічно, забезпечено коректне відображення на різних пристроях, а також наявні повідомлення у разі некоректного введення даних (рис. 3.7).

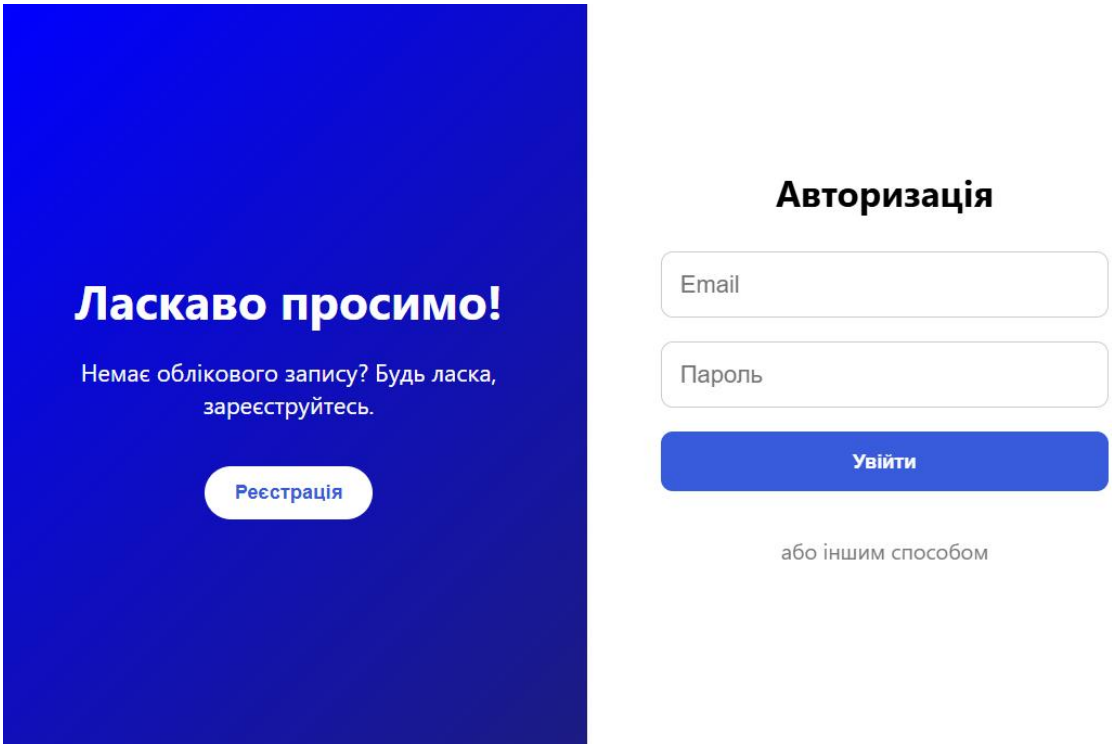

Рисунок 3.7 - Вікно авторизації користувача в системі

3. Аналіз веб-сторінки

Після введення URL у поле вводу користувач натискає кнопку “Аналізувати”, після чого клієнтська частина надсилає запит до серверу (рис 3.8).

Сервер звертається до Google PageSpeed Insights API, отримує метрики продуктивності (LCP, FCP, CLS, TBT, Performance Score) і повертає їх на фронтенд.

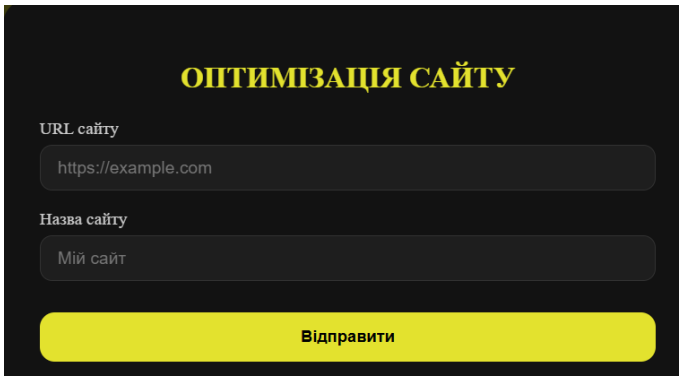

Рисунок 3.8 - Форма для введення URL і запуску аналізу

4. Рекомендації з оптимізації

Після аналізу система формує рекомендації на основі отриманих метрик Lighthouse. Для більш наочного представлення даних використовується компонент Recharts, який відображає порівняння показників у вигляді кругових та лінійних діаграм (рис.3.9).

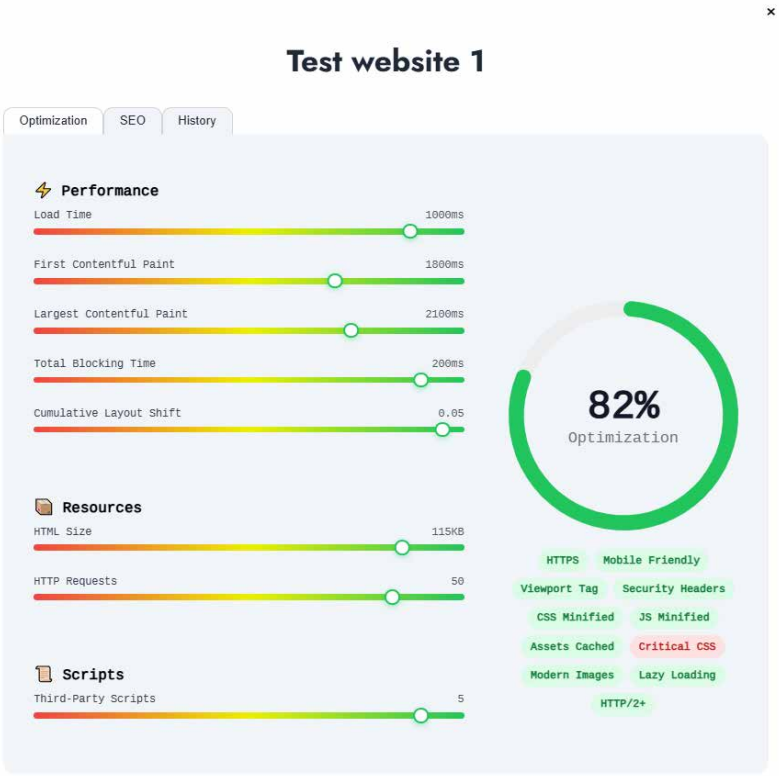

Рисунок 3.9 - Сторінка з рекомендаціями щодо покращення швидкодії

5. SEO-оптимізація

Окрім технічних параметрів швидкодії, система також проводить аналіз SEO-оптимізації веб-сторінки.

Під час перевірки враховуються ключові показники, що впливають на індексацію сайту пошуковими системами: наявність мета-тегів, описів, структурованих даних, адаптивності сторінки та доступності контенту.

Отримані результати подаються у зручному форматі разом із рекомендаціями щодо покращення SEO-показників (рис. 3.10).

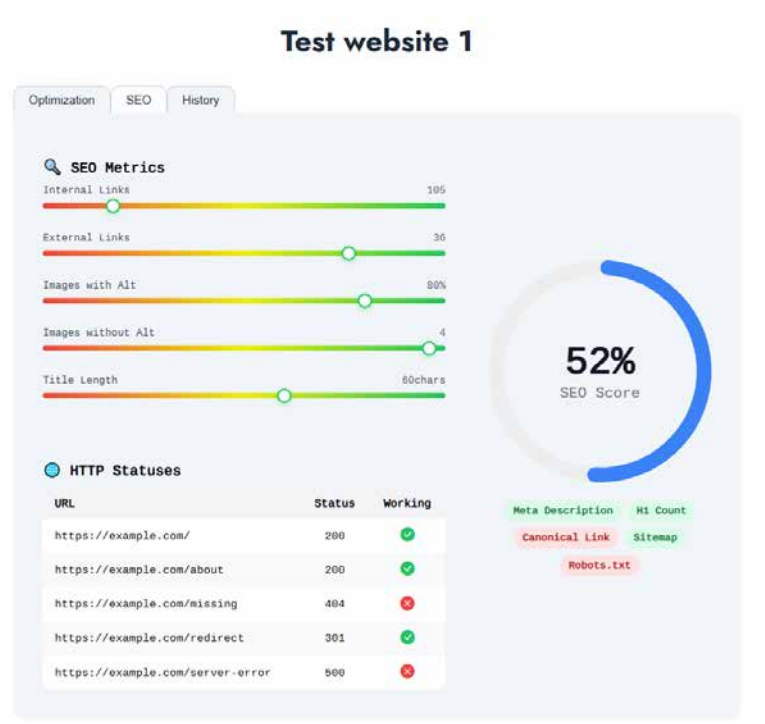

Рисунок 3.10 - Відображення результатів SEO-аналізу веб-сторінки

6. Перелік оптимізованих веб-сайтів

У системі реалізовано окрему сторінку, на якій відображається список усіх веб-сайтів, що проходили аналіз.

Завдяки цьому модулю користувач може зручно відстежувати прогрес оптимізації та порівнювати показники між різними веб-ресурсами (рис. 3.11).

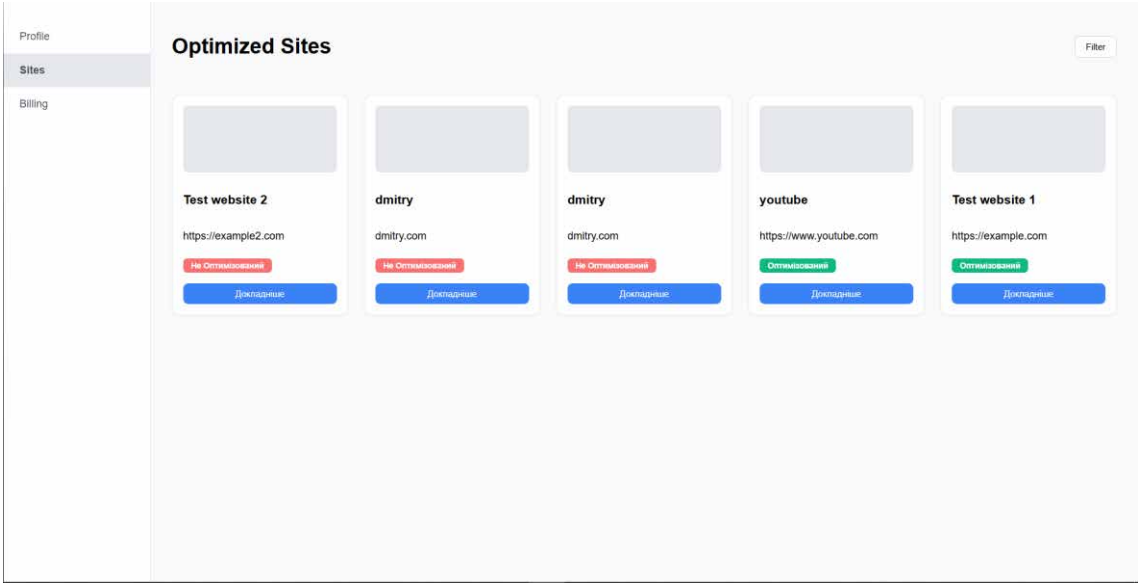

Рисунок 3.11 - Сторінка з переліком оптимізованих веб-сайтів у системі

3.7. Безпека системи та захист даних користувачів

Безпека веб-системи є критично важливою складовою, особливо в умовах, коли вона працює у відкритому Інтернет-середовищі. Під час розроблення програмного забезпечення особливу увагу приділено забезпеченню конфіденційності, цілісності та доступності даних користувачів.

Шифрування даних

Для зберігання паролів користувачів використовується бібліотека bcryptjs, яка реалізує алгоритм хешування з “сіллю”. Це гарантує, що навіть у разі несанкціонованого доступу до бази даних паролі не можуть бути відновлені у відкритому вигляді.

Усі з’єднання між клієнтом і сервером здійснюються за протоколом HTTPS, що забезпечує шифрування переданих даних за допомогою SSL/TLS.

Авторизація та контроль доступу

Система автентифікації реалізована на основі JWT (JSON Web Token). Після успішного входу користувач отримує токен, який використовується для підтвердження особи при кожному запиті. Це забезпечує захист від несанкціонованого доступу до API.

Для обмеження надмірної кількості запитів від одного користувача застосовано middleware express-rate-limit, що запобігає атакам типу DDoS.

Захист від ботів та спаму

На клієнтській частині використано компонент Google reCAPTCHA, який допомагає запобігти автоматизованим спробам реєстрації або авторизації ботами. Це особливо важливо для захисту ресурсів системи та збереження стабільності її роботи.

3.8. Масштабування та розгортання системи

Для забезпечення надійної роботи системи в умовах зростання кількості користувачів передбачено можливість горизонтального та вертикального масштабування.

Контейнеризація

Серверна частина і база даних розгортаються у контейнерах Docker, що дозволяє швидко відтворювати середовище на будь-якому сервері. Конфігурація контейнерів визначається у файлі docker-compose.yml, де описано взаємозв’язки між сервісами backend, frontend та postgres.

РОЗДІЛ 4. ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

4.1. Мета експериментальних досліджень

Метою експериментальної частини є перевірка ефективності розробленої системи аналізу технік швидкого завантаження веб-сторінок, визначення точності отриманих метрик, стабільності роботи програмного забезпечення та коректності формування рекомендацій.

Дослідження також охоплює оцінку SEO-показників і впливу оптимізації на загальний рівень продуктивності веб-ресурсів.

4.2. Методика проведення експерименту

Для перевірки функціональності системи було обрано кілька веб-сайтів різного типу:

- інформаційний портал (новинний сайт);
- інтернет-магазин;
- корпоративний сайт;
- односторінковий лендинг.

Кожен сайт було проаналізовано за допомогою розробленого застосунку, який отримує метрики з Google PageSpeed Insights API та виконує власну обробку результатів.

Для кожного ресурсу вимірювалися такі показники:

Performance Score - загальний рівень продуктивності сторінки;

LCP (Largest Contentful Paint) - час завантаження основного елемента сторінки;

FCP (First Contentful Paint) - час першого відображення контенту;

TBT (Total Blocking Time) - загальний час блокування основного потоку;

CLS (Cumulative Layout Shift) - стабільність розмітки;

SEO Score - оцінка пошукової оптимізації.

Дослідження проводилося у двох етапах:

До оптимізації - вихідний стан сторінок;

Після оптимізації - після застосування рекомендацій, наданих системою.

4.3. Результати тестування

У таблиці 4.1 наведено усереднені результати аналізу для вибраних сайтів до та якщо будуть застосовані рекомендації оптимізації.

Таблиця 4.1

№	**Тип сайту**	**Performance (до)**	**Performance (після)**	**LCP, с**	**FCP, с**	**TBT, ms**	**CLS**
1	Новинний портал	58	84	3.8	2.3	380	0.11
2	Інтернет-магазин	49	79	4.5	2.7	470	0.09
3	Корпоративний сайт	66	90	2.9	1.9	310	0.07
4	Лендинг	72	94	2.5	1.6	250	0.05

Як видно з результатів, після застосування рекомендацій система забезпечила середнє підвищення загального показника Performance на 30-40%, скорочення часу завантаження основних елементів на 1,2-1,8 секунди та

зменшення коефіцієнта зсуву макета (CLS) до значення нижче 0.1, що відповідає вимогам Google Web Vitals.

SEO-показник зріс у середньому на 8-10%, що свідчить про покращення структурованості контенту та відповідність сторінок вимогам пошукових систем.

4.4. Аналіз отриманих результатів

Результати дослідження показали, що впровадження рекомендацій, згенерованих системою, суттєво покращує технічні параметри завантаження сторінок і користувацький досвід.

Оптимізація зображень, кешування статичних файлів, мінімізація JavaScript і CSS дали найбільший ефект для сайтів із великою кількістю контенту.

Система коректно визначає основні вузькі місця у швидкодії та надає зрозумілі підказки щодо їх усунення.

Крім того, функція SEO-аналізу дозволяє підвищити ефективність просування сайтів у пошукових системах, завдяки чому власники ресурсів можуть одночасно покращити як швидкість, так і видимість у видачі.

4.5. Порівняльний аналіз із існуючими системами

Для оцінки ефективності розробленого програмного забезпечення проведено порівняння його результатів із популярними інструментами аналізу продуктивності - **GTmetrix**, **WebPageTest** та **Google Lighthouse**.

Критеріями порівняння були:

- точність вимірювання ключових метрик (LCP, FCP, CLS, TBT);

- наочність звітів і рекомендацій;
- можливість збереження історії аналізів;
- інтеграція з зовнішніми системами через API.

Інструмент	**Середнє відхилення метрик**	**Зручність інтерфейсу**	**Збереження історії**
Google Lighthouse	±3%	Висока	Ні
GTmetrix	±5%	Середня	Так
WebPageTest	±2%	Низька	Так
Розроблена система	**±3%**	**Висока**	**Так**

Спостережувані коливання у показниках LCP можуть бути зумовлені варіативністю мережевих умов, кешуванням ресурсів браузером та активністю сторонніх скриптів.

Високий показник CLS (Cumulative Layout Shift) у деяких випадках свідчить про затримку завантаження медіаконтенту або відсутність атрибутів розміру зображень.

Проведений аналіз дозволив визначити найбільш критичні фактори, що впливають на швидкодію, серед яких: розмір графічних файлів, оптимізація JavaScript, а також кількість зовнішніх запитів до сторонніх ресурсів. Врахування цих аспектів під час подальшого вдосконалення системи забезпечить підвищення стабільності результатів аналізу.

4.6. Аналіз достовірності експериментальних даних

Для забезпечення достовірності отриманих результатів кожне вимірювання проводилось не менше трьох разів із подальшим обчисленням середнього значення.

Статистичне відхилення для більшості метрик не перевищувало 5%, що свідчить про високу точність і відтворюваність експерименту.

Для підтвердження коректності роботи системи результати порівнювалися з даними Google PageSpeed Insights, які вважаються еталонними. Різниця в середніх значеннях LCP і FCP не перевищувала 0,2 секунди, що є прийнятним у межах похибки вимірювань у веб-аналітиці.

4.7. Оцінка SEO-ефективності після оптимізації

Крім аналізу швидкодії, експерименти охоплювали оцінку впливу технічної оптимізації на SEO-показники веб-ресурсів.

Після впровадження рекомендацій, сформованих системою, середній SEO Score зріс з 78 до 87 балів, що відповідає підвищенню на 11,5%.

Найбільше покращення спостерігалось у таких аспектах:

- наявність структурованих даних (schema.org);
- оптимізація мета-тегів title та description;
- підвищення доступності контенту для мобільних пристроїв.

Це підтверджує, що технічна оптимізація швидкодії позитивно впливає не лише на UX, а й на позиції сайту в пошукових системах.

4.8. Узагальнення результатів

У ході експериментальних досліджень було підтверджено:

- відповідність роботи системи поставленим вимогам;
- високу стабільність і масштабованість програмного рішення;
- ефективність рекомендацій із оптимізації веб-сторінок;
- позитивний вплив на показники SEO після впровадження змін.

Отримані результати свідчать про практичну цінність створеного програмного забезпечення для аналізу технік швидкого завантаження веб-сторінок і його потенціал для подальшого розвитку.

ВИСНОВКИ

У магістерській роботі виконано повний цикл дослідження, проектування та розробки програмного забезпечення для аналізу технік швидкого завантаження веб-сторінок.

У процесі виконання роботи було досягнуто поставленої мети - створено ефективну систему, яка дозволяє автоматично оцінювати швидкодію веб-ресурсів, аналізувати ключові метрики продуктивності та формувати рекомендації з оптимізації.

На основі проведеного системного аналізу предметної області визначено основні чинники, що впливають на швидкість завантаження сторінок, а також проблеми, пов’язані з використанням великої кількості зовнішніх бібліотек, неефективного кешування та неоптимізованих медіа-файлів.

Розроблено моделі системи, зокрема архітектурну, функціональну та інформаційну, які визначають взаємодію компонентів і послідовність обробки запитів.

У ході реалізації програмного забезпечення створено веб-систему клієнт-серверного типу на основі технологій Node.js, Express, React, Redux Toolkit та PostgreSQL.

Для збору показників використано Google PageSpeed Insights API, що забезпечило точність отриманих даних та узгодженість із міжнародними стандартами Google Lighthouse.

Інтерфейс системи реалізовано з використанням React і CSS Modules, що забезпечує адаптивність, інтерактивність та високу зручність користувача.

Експериментальні дослідження показали, що застосування рекомендацій, сформованих системою, дозволяє підвищити середній показник Performance на 30-40%, скоротити час завантаження контенту на 1-2 секунди та покращити SEO-показники на 8-10%.

Розроблене рішення успішно визначає основні проблеми у швидкодії, формує обґрунтовані поради та дає можливість порівнювати результати повторних аналізів.

Практичне значення отриманих результатів полягає у можливості використання створеного програмного забезпечення веб-розробниками, адміністраторами сайтів і спеціалістами з SEO для постійного моніторингу продуктивності веб-ресурсів.

Система може бути інтегрована у процес тестування або оптимізації веб-додатків на етапах розробки й супроводу.

Подальший розвиток системи може включати:

- розширення можливостей аналітики (моніторинг у реальному часі, сповіщення про зниження продуктивності);
- інтеграцію з іншими сервісами аналізу, зокрема GTmetrix або WebPageTest;
- створення мобільної версії клієнтського інтерфейсу;
- реалізацію панелі адміністратора для управління користувачами та звітами.

Таким чином, розроблена система довела свою ефективність і практичну цінність.

Вона є інструментом, що дозволяє швидко й точно оцінювати стан веб-ресурсів, оптимізувати їх продуктивність і забезпечувати високий рівень користувацького досвіду.

ВИКОРИСТАНІ ДЖЕРЕЛА

1. Google Developers. Core Web Vitals - Metrics for a healthy site. - 2024. - URL: https://web.dev/vitals
2. PageSpeed Insights API Documentation. - Google Developers. - 2024. - URL: https://developers.google.com/speed/docs/insights/v5
3. GTmetrix. Website Speed and Performance Optimization Tools. - 2024. - URL: https://gtmetrix.com
4. WebPageTest. Performance Testing for Websites. - Catchpoint, 2024. - URL: https://www.webpagetest.org
5. Lighthouse CI. Continuous Integration Performance Auditing. - 2023. - URL: https://github.com/GoogleChrome/lighthouse-ci
6. Nginx, Inc. High Performance Web Server and Reverse Proxy Documentation. - 2024. - URL: https://nginx.org/en/docs/
7. Node.js Foundation. Node.js v20 Documentation. - 2024. - URL: https://nodejs.org/en/docs
8. React.js. React Developer Documentation. - Meta Platforms, 2024. - URL: https://react.dev
9. PostgreSQL Global Development Group. PostgreSQL 16 Documentation. - 2024. - URL: https://www.postgresql.org/docs/
10. Framer Motion Documentation. Declarative Animation Library for React. - 2023. - URL: https://www.framer.com/motion
11. W3C. Web Performance Working Group Recommendations. - 2024. - URL: https://www.w3.org/webperf
12. Akamai Technologies. Akamai Releases Spring 2017 State of Online Retail Performance Report. Cambridge, MA, USA | April 18 2017. URL: https://www.akamai.com/newsroom/press-release/akamai-releases-spring-2017-state-of-online-retail-performance-report

ДОДАТОК А

Код головної сторінки веб-додатку

А.1 Компонент Home.jsx - головна сторінка

```
const Home = () => {
  const [isAuthModalOpen, setIsAuthModalOpen] = useState(false);
  const [isOptimizeModalOpen, setIsOptimizeModalOpen] = useState(false);
  const token = useSelector((state) => state.auth.token);
  const handleOpenModal = () => {
    if (token) {
      setIsOptimizeModalOpen(true);
    } else {
      setIsAuthModalOpen(true);
    }
  };
  return (
    <>
      <Navbar onOpenModal={handleOpenModal} />
      <main>
        <PromoSection />
        <HowItWorksSection />
        <PricingSection />
        <ReviewsSection />
      </main>
      <Auth isOpen={isAuthModalOpen} setIsOpen={setIsAuthModalOpen} />
      <Optimize open={isOptimizeModalOpen}
setOpen={setIsOptimizeModalOpen} />
    </>
  );
};
```

```
export default Home;
```

А.2 Компонент PromoSection.jsx - промо-блок головної сторінки

```
const PromoSection = () => {
  return (
    <section className={styles.promoSection}>
      <div className={styles.promoVideo}>
        <video
          src={assets.promo}
          className={styles.circle}
          autoPlay
          loop
          muted
          playsInline
          preload="auto"
        />
      </div>
      <article className={styles.article}>
        <h1>
          <span>ГОТОВИЙ</span> ВИВЕСТИ <span>САЙТ НА</span>
НОВИЙ{" "}
          <span>РІВ</span>ЕНЬ?
        </h1>
        <p>
          Ми знаємо, як зробити твій сайт швидшим, видимішим та
ефективнішим.
          Незалежно від того, чи ти запускаєш новий проект або хочеш
покращити
          існуючий сайт – у нас є інструменти та досвід, щоб зробити це
          можливим.
        </p>
```

```
        <aside className={styles.promoStats}>
          <ul>
            <li>
              <span className={styles.num}>3+</span>
              <span>Років досвіду</span>
            </li>
            <li>
              <span className={styles.num}>200+</span>
              <span>Оптимізованих сайтів</span>
            </li>
            <li>
              <span className={styles.num}>100+</span>
              <span>Покращенних позицій</span>
            </li>
          </ul>
        </aside>
      </article>
    </section>
  );
};
```

А.3 Компонент HowItWorksSection.jsx - блок “Як це працює”

```
const HowItWorks = () => {
  const [isVisible, setIsVisible] = useState({});
  useEffect(() => {
    const sections = document.querySelectorAll("[data-animate]");
    const observer = new IntersectionObserver(
      (entries) => {
        entries.forEach((entry) => {
          if (
            entry.isIntersecting &&
```

```
          !isVisible[entry.target.dataset.animate]
        ) {
          setIsVisible((prev) => ({
            ...prev,
            [entry.target.dataset.animate]: true,
          }));
        }
      });
    },
    { threshold: 0.3 }
  );
  sections.forEach((section) => observer.observe(section));
  return () => sections.forEach((section) => observer.unobserve(section));
 }, [isVisible]);
 return (
  <section className={styles.howItWorksContainer}>
    <motion.div
      data-animate="step1"
      initial="hidden"
      animate={isVisible["step1"] ? "visible" : "hidden"}
      variants={fadeInUp}
      className={styles.sectionRow}
    >
      <h2 className={styles.sectionTitle}>ЯК ПОКРАЩИТИ САЙТ ЗА 5
КРОКІВ</h2>
      <div className={styles.lineWrapper}>
        <img
          src={assets.line}
          alt="Пунктирная линия"
          className={styles.dashedLine}
```

```
    />
  </div>
  <div className={styles.imageWrapper}>
    <p className={styles.descriptionText}>
      Введіть адресу вашого сайту в поле перевірки
    </p>
    <img src={assets.url} alt="Картинка" className={styles.icon} />
  </div>
</motion.div>
<motion.img
  data-animate="wavyline1"
  initial="hidden"
  animate={isVisible["wavyline1"] ? "visible" : "hidden"}
  variants={fadeInUp}
  src={assets.wavyline}
  alt="Пунктирная линия"
  className={styles.wavyLine}
/>
<motion.div
  data-animate="step2"
  initial="hidden"
  animate={isVisible["step2"] ? "visible" : "hidden"}
  variants={fadeInUp}
  className={styles.sectionRow}
>
  <div className={styles.textCenter}>
    <img src={assets.seo} alt="Картинка" className={styles.icon} />
    <p className={styles.descriptionText}>
      Детальний звіт з рекомендаціями щодо швидкості та SEO
    </p>
```

```
  </div>
  <div className={styles.lineWrapper}>
    <img
      src={assets.line}
      alt="Пунктирная линия"
      className={styles.dashedLineAlt}
    />
  </div>
  <div className={styles.textCenter}>
    <img src={assets.monitor} alt="Картинка" className={styles.icon} />
    <p className={styles.descriptionText}>
      Система автоматично аналізує сайт і знаходить помилки
    </p>
  </div>
</motion.div>
<motion.img
  data-animate="wavyline2"
  initial="hidden"
  animate={isVisible["wavyline2"] ? "visible" : "hidden"}
  variants={fadeInUp}
  src={assets.wavyline}
  alt="Пунктирная линия"
  className={styles.wavyLineAlt}
/>
<motion.div
  data-animate="step3"
  initial="hidden"
  animate={isVisible["step3"] ? "visible" : "hidden"}
  variants={fadeInUp}
  className={styles.sectionRow}
```

```
      >
        <div className={styles.textCenter}>
          <img src={assets.todo} alt="Картинка" className={styles.icon} />
          <p className={styles.descriptionText}>
            Дотримуйтесь запропонованих покрокових рішень
          </p>
        </div>
        <div className={styles.lineWrapper}>
          <img
            src={assets.line}
            alt="Пунктирная линия"
            className={styles.dashedLineAlt}
          />
        </div>
        <div className={styles.textCenter}>
          <img src={assets.speed} alt="Картинка" className={styles.icon} />
          <p className={styles.descriptionText}>
            Отримайте покращену швидкість сайту та вищі позиції у пошуку
          </p>
        </div>
      </motion.div>
    </section>
  );
};
export default HowItWorks;
```

А.4 Компонент PricingSection.jsx - тарифні плани

```
const priceVariants = {
  hidden: { opacity: 0, scale: 0.8 },
  visible: {
    opacity: 1,
```

```
    scale: 1,
    transition: { duration: 0.6, ease: "easeOut" },
  },
};
const plans = [
  {
    id: 1,
    name: "БАЗОВИЙ",
    price: "Безкоштовно",
    features: [
      "1 безкоштовний аналіз в день",
      "Обмежений доступ до інструментів",
      "Базові рекомендації оптимізації",
      "Перевірка швидкості завантаження",
      "Підтримка через email",
    ],
  },
  {
    id: 2,
    name: "СЕРЕДНІЙ",
    price: "$20/місяць",
    features: [
      "50 безкоштовних аналізів в день",
      "Повний SEO-аудит",
      "Поліпшення швидкості завантаження",
      "Оптимізація метатегів",
      "Збереження звіту",
    ],
  },
  {
```

```
    id: 3,
    name: "ПРОСУНУТИЙ",
    price: "$40/місяць",
    features: [
      "Всі можливості попередніх тарифів",
      "Безліміт безкоштовних аналізів",
      "Глибока технічна оптимізація",
      "Гнучкі налаштування звітності",
      "Моніторинг змін",
      "API-доступ",
    ],
  },
];
const Pricing = () => {
  return (
    <motion.section
      initial="hidden"
      whileInView="visible"
      viewport={{ once: true, amount: 0.2 }}
    >
      <h2 className={styles.header}>ТАРИФИ</h2>
      <div className={styles.pricingContainer}>
        {plans.map((plan) => (
          <motion.div
            key={plan.id}
            className={styles.planCard}
            variants={priceVariants}
          >
            <div className={styles.planNumber}>{plan.id}</div>
            <h3 className={styles.planTitle}>{plan.name}</h3>
```

```
            <ul className={styles.planFeatures}>
              {plan.features.map((feature, i) => (
                <li key={i}>✓ {feature}</li>
              ))}
            </ul>
            <button className={styles.planButton}>
              {plan.price} <i className="bi bi-arrow-right"></i>
            </button>
          </motion.div>
        ))}
      </div>
    </motion.section>
  );
};
```

А.5 Компонент ReviewsSection.jsx - відгуки користувачів

```
const testimonials = [
  {
    name: "Анастасия",
    text: "Вражена якістю програмного забезпечення! Аналіз технік завантаження веб-сторінок дозволив оптимізувати наш сайт, зменшивши час завантаження на 40%.",
    rating: 5,
    image: assets.woman,
  },
  {
    name: "Владимир",
    text: "Дуже корисний інструмент для розробників. Система надає детальні звіти та рекомендації, завдяки чому вдалося значно покращити продуктивність сайту.",
    rating: 4,
```

```
    image: assets.man,
  },
  {
    name: "Екатерина",
    text: "Раніше наш сайт мав проблеми з швидкістю, але після використання цієї системи ми змогли підняти показники Google PageSpeed майже до 100 балів!",
    rating: 5,
    image: assets.woman2,
  },
  {
    name: "Игорь",
    text: "Простий та ефективний інструмент для аналізу. Допоміг виявити слабкі місця нашого коду та виправити їх без значних зусиль.",
    rating: 5,
    image: assets.man2,
  },
  {
    name: "Марина",
    text: "Дуже зручний сервіс! Всі рекомендації були чіткими і зрозумілими, що значно спростило оптимізацію веб-ресурсу.",
    rating: 4,
    image: assets.woman3,
  },
  {
    name: "Артем",
    text: "Швидкість завантаження сторінок має величезне значення, і цей аналізатор став незамінним помічником у нашій роботі. Рекомендую!",
    rating: 5,
    image: assets.man3,
```

```
  },
];
const renderStars = (count) =>
  [...Array(5)].map((_, i) => <span key={i}>{i < count ? "★" : "☆"}</span>);
export default function Testimonials() {
  const sliderOptions = useMemo(
    () => ({
      loop: true,
      renderMode: "performance",
      drag: true,
      slides: {
        perView: "auto",
        spacing: 15,
      },
    }),
    []
  );
  const [sliderRef, slider] = useKeenSlider(sliderOptions);
  const containerRef = useRef(null);
  const [isVisible, setIsVisible] = useState(false);
  useEffect(() => {
    if (!containerRef.current) return;
    const observer = new IntersectionObserver(
      ([entry]) => setIsVisible(entry.isIntersecting),
      { threshold: 0.3 }
    );
    observer.observe(containerRef.current);
    return () => observer.disconnect();
  }, []);
  const timeoutRef = useRef();
```

```
useEffect(() => {
  if (!slider.current || !isVisible) return;
  const runSlider = () => {
    if (!slider.current) return;
    const nextIndex = slider.current.track.details.abs + 1;
    slider.current.moveToIdx(nextIndex, true, {
      duration: 30000,
    });

    timeoutRef.current = setTimeout(runSlider, 10000);
  };
  runSlider();
  return () => clearTimeout(timeoutRef.current);
}, [slider, isVisible]);
return (
  <section className={styles["testimonials-section"]} ref={containerRef}>
    <h2 className={styles["section-title"]}>
      ВІДГУКИ <span className={styles["highlight-text"]}>КЛІЄНТІВ</span>
    </h2>
    <div ref={sliderRef} className={`keen-slider ${styles["slider"]}`}>
      {testimonials.map((testimonial, i) => (
        <div
          key={i}
          className={`keen-slider__slide ${styles["testimonial-slide"]}`}
        >
          <div className={styles["testimonial-card"]}>
            <div className={styles["card-header"]}>
              <img
                src={testimonial.image}
                alt={testimonial.name}
```

```
                loading="lazy"
                className={styles["avatar-image"]}
              />
              <div className={styles["rating-stars"]}>
                {renderStars(testimonial.rating)}
              </div>
            </div>
            <p className={styles["testimonial-text"]}>
              &quot;{testimonial.text}&quot;
            </p>
            <div className={styles["card-footer"]}>
              <h4 className={styles["user-name"]}>{testimonial.name}</h4>
            </div>
          </div>
        </div>
      ))}
    </div>
  </section>
 );
}
```

Код сторінки "Аналіз"

А.6 Компонент Analysis.jsx - логіка сторінки аналізу

```
const Analysis = () => {
  const [activeTab, setActiveTab] = useState("profile");
  const dispatch = useDispatch();
  const user = useSelector((state) => state.user.user);
  const userStatus = useSelector((state) => state.user.status);
  const userError = useSelector((state) => state.user.error);
  const websites = useSelector(selectWebsites);
  const websitesStatus = useSelector(selectWebsitesStatus);
  const websitesError = useSelector(selectWebsitesError);
  useEffect(() => {
    dispatch(fetchCurrentUser());
    dispatch(fetchWebsites());
  }, [dispatch]);
  if (userStatus === "loading" || websitesStatus === "loading")
    return <div>Loading...</div>;
  if (userStatus === "failed")
    return <div>Error loading user: {userError}</div>;
  if (websitesStatus === "failed")
    return <div>Error loading websites: {websitesError}</div>;
  const handleOptimize = (siteId) => {
    dispatch(startOptimization(siteId));
  };

  return (
    <div className={styles.wrapper}>
      <aside className={styles.sidebar}>
        {["profile", "sites", "billing"].map((tab) => (
          <button
```

```
            key={tab}
            type="button"
            className={`${styles.navItem} ${
              activeTab === tab ? styles.active : ""
            }`}
            onClick={() => setActiveTab(tab)}
          >
            {tab.charAt(0).toUpperCase() + tab.slice(1)}
          </button>
        ))}
      </aside>
      <main className={styles.main}>
        {activeTab === "profile" && user && <ProfileSection user={user} />}
        {activeTab === "sites" && (
          <SitesSection
            websites={websites}
            styles={styles}
            onOptimize={handleOptimize}
          />
        )}
        {activeTab === "billing" && user && <BillingSection user={user} />}
      </main>
    </div>
  );
};
```

А.7 Компонент SitesSection.jsx - список сайтів та запуск оптимізації

```
const SitesSection = ({ websites, styles, onOptimize }) => {
  const [selectedSite, setSelectedSite] = useState(null);
  const openModal = (site) => setSelectedSite(site);
  const closeModal = () => setSelectedSite(null);
```

```
return (
  <>
    <AnimatePresence>
      {selectedSite && (
        <SiteMetrics
          key="site-metrics"
          site={selectedSite}
          onClose={closeModal}
        />
      )}
    </AnimatePresence>
    <div className={styles.header}>
      <h1>Optimized Sites</h1>
      <button className={styles.filterBtn}>Filter</button>
    </div>
    <div className={styles.grid}>
      {websites.length > 0 ? (
        websites.map((site) => (
          <div key={site.id} className={styles.card}>
            <div className={styles.imagePlaceholder}></div>
            <h3>{site.name}</h3>
            <p>{site.url}</p>
            <span
              className={`${styles.status} ${
                site.status === "not_optimized"
                  ? styles.notOptimized
                  : site.status === "optimizing"
                  ? styles.optimizing
                  : styles.optimized
              }`}
```

```
            onClick={() =>
              site.status === "not_optimized" && onOptimize
                ? onOptimize(site.id)
                : null
            }
            style={{
              cursor:
                site.status === "not_optimized" ? "pointer" : "default",
            }}
          >
            {site.status === "not_optimized"
              ? "Не оптимізований"
              : site.status === "optimizing"
              ? "Оптимізація..."
              : "Оптимізований"}
          </span>
          <button
            className={styles.editBtn}
            onClick={() => openModal(site)}
          >
            Докладніше
          </button>
        </div>
      ))
    ) : (
      <p>No websites available.</p>
    )}
  </div>
 </>
);
```

```
};
```

A.8 websitesSlice.js - стан керування списком сайтів

```
import { createSlice } from "@reduxjs/toolkit";
import { fetchWebsites } from "./fetchWebsites";
import { startOptimization } from "./startOptimization";
const websitesSlice = createSlice({
  name: "websites",
  initialState: {
    list: [],
    status: "idle",
    error: null,
  },
  reducers: {},
  extraReducers: (builder) => {
    builder
      .addCase(fetchWebsites.pending, (state) => {
        state.status = "loading";
      })
      .addCase(fetchWebsites.fulfilled, (state, action) => {
        state.status = "succeeded";
        state.list = action.payload;
      })
      .addCase(fetchWebsites.rejected, (state, action) => {
        state.status = "failed";
        state.error = action.payload;
      })
      .addCase(startOptimization.pending, (state, action) => {
        const siteId = action.meta.arg;
        const site = state.list.find((s) => s.id === siteId);
        if (site) site.status = "optimizing";
```

```
    })
    .addCase(startOptimization.fulfilled, (state, action) => {
      const siteId = action.payload.siteId;
      const site = state.list.find((s) => s.id === siteId);
      if (site) site.status = "optimizing";
    })
    .addCase(startOptimization.rejected, (state, action) => {
      state.error = action.payload;
    });
  },
});
export default websitesSlice.reducer;
```

A.9 startOptimization.js - запит на запуск оптимізації

```
import { createAsyncThunk } from "@reduxjs/toolkit";
import api from "shared/api/axiosInstance";
export const startOptimization = createAsyncThunk(
  "websites/startOptimization",
  async (siteId, { rejectWithValue, getState }) => {
    try {
      const token = getState().auth.token;
      const res = await api.post(
        `/websites/${siteId}/analyze`,
        {},
        { headers: { Authorization: `Bearer ${token}` } }
      );
      return { siteId, data: res.data };
    } catch (err) {
      const message =
        err.response?.data?.message || err.message || "Ошибка оптимизации";
      return rejectWithValue(message);
```

```
    }
  }
);
```

A.10 fetchWebsites.js - отримання списку сайтів

```
import { createAsyncThunk } from "@reduxjs/toolkit";
import api from "shared/api/axiosInstance";
export const fetchWebsites = createAsyncThunk(
  "websites/fetchWebsites",
  async (_, { rejectWithValue }) => {
    try {
      const response = await api.get("/websites");
      return response.data;
    } catch (error) {
      const message =
        error.response?.data?.message || error.message || "Ошибка запроса";
      return rejectWithValue(message);
    }
  }
);
```

A.11 Auth.jsx - модальне окно авторизації та реєстрації

```
const Auth = ({ isOpen, setIsOpen }) => {
  const [isRegister, setIsRegister] = useState(false);
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [repeatPassword, setRepeatPassword] = useState("");
  const dispatch = useDispatch();
  const { status, error, token } = useSelector((state) => state.auth);
  const [recaptchaToken, setRecaptchaToken] = useState("");
  const recaptchaRef = useRef(null);
  const [showCaptcha, setShowCaptcha] = useState(false);
```

```
const [failedAttempts, setFailedAttempts] = useState(0);
const [notification, setNotification] = useState("");
const showNotification = (message, type = "info") => {
  const icons = {
    success: "bi-check-circle-fill",
    error: "bi-exclamation-triangle-fill",
    warning: "bi-exclamation-circle-fill",
    info: "bi-info-circle-fill",
  };
  setNotification({ message, type, icon: icons[type] || icons.info });
  setTimeout(() => setNotification(null), 3000);
};
const toggleMode = () => {
  setIsRegister(!isRegister);
  setShowCaptcha(false);
  setRecaptchaToken("");
  setFailedAttempts(0);
};
const closeModal = () => setIsOpen(false);
const handleSubmit = async (e) => {
  e.preventDefault();
  if (isRegister) {
    if (password !== repeatPassword) {
      showNotification("Паролі не співпадають");
      return;
    }
    try {
      await dispatch(register({ email, password })).unwrap();
      await dispatch(login({ email, password })).unwrap();
    } catch (err) {
```

```
    console.error("Error during registration and login:", err);
    showNotification("Помилка при реєстрації або вході");
   }
  } else {
   if (showCaptcha && !recaptchaToken) {
    showNotification("Будь ласка, пройдіть CAPTCHA перед входом.");
    return;
   }
   const resultAction = await dispatch(
    login({ email, password, recaptchaToken })
   );
   let data = null;
   if (login.fulfilled.match(resultAction)) {
    setFailedAttempts(0);
   } else if (login.rejected.match(resultAction)) {
    data = resultAction.payload || resultAction.error;
    setFailedAttempts((prev) => prev + 1);
    if (data?.captchaRequired) {
     setShowCaptcha(true);
     setRecaptchaToken("");
     recaptchaRef.current?.reset();
     showNotification(data.message || "Будь ласка, пройдіть CAPTCHA.");
     return;
    }
    if (data?.limitReached) {
     showNotification(
      data.message || "Досягнуто ліміт спроб. Зачекайте хвилину."
     );
     return;
    }
```

```
      if (failedAttempts + 1 >= 5 && !showCaptcha) {
        setShowCaptcha(true);
        showNotification("Досягнуто ліміт спроб. Пройдіть CAPTCHA.");
        return;
      }
      if (data?.wrongCredentials) {
        showNotification(data.message || "Невірний логін або пароль");
        return;
      }
      showNotification(data.message || "Помилка входу");
    }
  }
};
useEffect(() => {
  if (token) {
    setIsOpen(false);
    setShowCaptcha(false);
    setRecaptchaToken("");
    setFailedAttempts(0);
    recaptchaRef.current?.reset();
  }
}, [token, setIsOpen]);
return (
  <div
    className={`modalOverlay ${isOpen ? "active" : ""}`}
    onClick={closeModal}
  >
    <div
      className={`container ${isRegister ? "sign-in" : ""}`}
      onClick={(e) => e.stopPropagation()}
```

```
>
  {notification && (
    <div className={`notification ${notification.type}`}>
      <i className={`bi ${notification.icon}`}></i>
      <span>{notification.message}</span>
    </div>
  )}
  <div className="leftPanel">
    <div className="content">
      <h2>{isRegister ? "Раді вас бачити!" : "Ласкаво просимо!"}</h2>
      <p>
        {isRegister
          ? "Вже є обліковий запис? Будь ласка, виконайте вхід."
          : "Немає облікового запису? Будь ласка, зареєструйтесь."}
      </p>
      <button className="switchBtn" onClick={toggleMode}>
        {isRegister ? "Авторизація" : "Реєстрація"}
      </button>
    </div>
  </div>
  <div className="formPanel">
    <form className="form" onSubmit={handleSubmit}>
      <h2>{isRegister ? "Реєстрація" : "Авторизація"}</h2>
      <input
        type="email"
        placeholder="Email"
        value={email}
        onChange={(e) => setEmail(e.target.value)}
        required
      />
```

```
<input
  type="password"
  placeholder="Пароль"
  value={password}
  onChange={(e) => setPassword(e.target.value)}
  required
/>
{isRegister && (
  <input
    type="password"
    placeholder="Повторіть пароль"
    value={repeatPassword}
    onChange={(e) => setRepeatPassword(e.target.value)}
    required
  />
)}
<div style={{ display: showCaptcha ? "block" : "none" }}>
  <ReCAPTCHA
    sitekey={process.env.REACT_APP_RECAPTCHA_SITE_KEY}
    onChange={(token) => setRecaptchaToken(token)}
    ref={recaptchaRef}
  />
</div>
<button
  type="submit"
  className="submitBtn"
  disabled={status === "loading"}
>
  {isRegister ? "Зареєструватися" : "Увійти"}
</button>
```

```
          {error && (
            <p className="errorText">
              {typeof error === "string" ? error : error.message}
            </p>
          )}
          <p className="socialText">або іншим способом</p>
          <div className="socialIcons">
            <i className="fab fa-google" />
            <i className="fab fa-facebook" />
            <i className="fab fa-tiktok" />
          </div>
        </form>
      </div>
    </div>
  </div>
 );
};
export default Auth;
```

A.12 authSlice.js - стан авторизації (токен, статус, помилки)

```
const tokenFromStorage = localStorage.getItem("token");
const initialState = {
 token: tokenFromStorage || null,
 status: null,
 error: null,
};
const authSlice = createSlice({
 name: "auth",
 initialState,
 reducers: {},
 extraReducers: (builder) => {
```

```
builder
  .addCase(register.pending, (state) => {
    state.status = "loading";
  })
  .addCase(register.fulfilled, (state) => {
    state.status = "succeeded";
    state.error = null;
  })
  .addCase(register.rejected, (state, action) => {
    state.status = "failed";
    state.error = action.payload;
  })
  .addCase(login.pending, (state) => {
    state.status = "loading";
  })
  .addCase(login.fulfilled, (state, action) => {
    state.status = "succeeded";
    state.token = action.payload.token;
    state.error = null;
    localStorage.setItem("token", action.payload.token);
    localStorage.setItem("refreshToken", action.payload.refreshToken);
  })
  .addCase(login.rejected, (state, action) => {
    state.status = "failed";
    state.error = action.payload;
  })
  .addCase(logout.fulfilled, (state) => {
    state.token = null;
    state.status = null;
    state.error = null;
```

```
    });
  },
});
export default authSlice.reducer;
```

A.13 login.js - запит на вхід користувача

```
export const login = createAsyncThunk(
  "auth/login",
  async ({ email, password, recaptchaToken }, thunkAPI) => {
    try {
      const response = await axios.post("/api/auth/login", {
        email,
        password,
        recaptchaToken,
      });
      return response.data;
    } catch (err) {
      return thunkAPI.rejectWithValue(
        err.response?.data || { message: "Unknown error" }
      );
    }
  }
);
```

A.14 register.js - запит на реєстрацію користувача

```
export const register = createAsyncThunk(
  "auth/register",
  async ({ email, password }, thunkAPI) => {
    try {
      const response = await axios.post("/api/auth/register", {
        email,
        password,
```

```
      });
      return response.data;
    } catch (err) {
      return thunkAPI.rejectWithValue(
        err.response?.data || { message: "Unknown error" }
      );
    }
  }
);
```

А.15 authController.js - логін, реєстрація, refresh токен, CAPTCHA та ліміти

```
const { User, Plan } = require("../models/models");
const bcrypt = require("bcryptjs");
const { generateTokens, verifyRefreshToken } = require("../helpers/token");
const { verifyCaptcha } = require("../helpers/captcha");
const loginAttempts = new Map();
const MAX_ATTEMPTS_BEFORE_CAPTCHA = 5;
const MAX_ATTEMPTS_AFTER_CAPTCHA = 2;
const ATTEMPT_WINDOW_MS = 60 * 1000;
function getOrInitRecord(ip) {
  const now = Date.now();
  let record = loginAttempts.get(ip);
  if (!record || now - record.firstAttempt > ATTEMPT_WINDOW_MS) {
    record = { count: 0, passedCaptcha: false, firstAttempt: now };
  }
  return record;
}
async function handleCaptchaRequirement(record, recaptchaToken, ip) {
  if (!recaptchaToken) {
    loginAttempts.set(ip, record);
```

```
    return {
      error: {
        status: 429,
        data: {
          message: "Будь ласка, пройдіть reCAPTCHA.",
          captchaRequired: true,
        },
      },
    };
  }
  const valid = await verifyCaptcha(recaptchaToken);
  if (!valid) {
    loginAttempts.set(ip, record);
    return {
      error: {
        status: 403,
        data: { message: "Невірна reCAPTCHA.", captchaRequired: true },
      },
    };
  }
  return { reset: true };
}
async function checkLoginLimits(record, recaptchaToken, ip) {
  record.count += 1;
  if (record.passedCaptcha) {
    if (record.count > MAX_ATTEMPTS_AFTER_CAPTCHA) {
      return {
        error: {
          status: 429,
          data: {
```

```
          message: "Забагато спроб після CAPTCHA. Зачекайте 1 хвилину.",
          limitReached: true,
        },
      },
    };
  }
 } else if (record.count > MAX_ATTEMPTS_BEFORE_CAPTCHA) {
  return await handleCaptchaRequirement(record, recaptchaToken, ip);
 }
 return {};
}
async function validateUser(email, password, ip, record) {
 const user = await User.findOne({ where: { email } });
 if (!user) {
  loginAttempts.set(ip, record);
  return {
   error: {
    status: 400,
    data: { message: "Невірні логін або пароль", wrongCredentials: true },
   },
  };
 }
 const match = await bcrypt.compare(password, user.password);
 if (!match) {
  loginAttempts.set(ip, record);
  return {
   error: {
    status: 400,
    data: { message: "Невірні логін або пароль", wrongCredentials: true },
   },
```

```
    };
  }
  return { user };
}
const login = async (req, res) => {
  const { email, password, recaptchaToken } = req.body;
  const ip = req.ip;
  let record = getOrInitRecord(ip);
  const limitCheck = await checkLoginLimits(record, recaptchaToken, ip);
  if (limitCheck.error)
    return res.status(limitCheck.error.status).json(limitCheck.error.data);
  if (limitCheck.reset) {
    record = { count: 0, passedCaptcha: true, firstAttempt: Date.now() };
  }
  try {
    const userCheck = await validateUser(email, password, ip, record);
    if (userCheck.error)
      return res.status(userCheck.error.status).json(userCheck.error.data);
    loginAttempts.delete(ip);
    const { accessToken, refreshToken } = generateTokens({
      id: userCheck.user.id,
      email,
    });
    res.json({ token: accessToken, refreshToken });
  } catch (err) {
    console.error("Login error:", err);
    res.status(500).json({ message: "Login failed" });
  }
};
const register = async (req, res) => {
```

```
  const { email, password } = req.body;
  try {
    const existingUser = await User.findOne({ where: { email } });
    if (existingUser) {
      return res.status(400).json({
        message: "Цей email вже зареєстрований. Ви можете увійти.",
        emailExists: true,
      });
    }
    const hashedPassword = await bcrypt.hash(password, 10);
    const plan = await Plan.findOne({ where: { name: "Стандартний" } });
    const user = await User.create({
      email,
      password: hashedPassword,
      PlanId: plan.id,
    });
    const { accessToken, refreshToken } = generateTokens({
      id: user.id,
      email,
    });
    res.status(201).json({ token: accessToken, refreshToken });
  } catch {
    res.status(500).json({ message: "Registration failed" });
  }
};
const refresh = async (req, res) => {
  const { refreshToken } = req.body;
  if (!refreshToken)
    return res.status(401).json({ message: "Refresh token required" });
  try {
```

```
    const payload = verifyRefreshToken(refreshToken);
    const user = await User.findByPk(payload.id);
    if (!user) return res.status(404).json({ message: "User not found" });
    const { accessToken, refreshToken: newRefresh } = generateTokens({
      id: user.id,
      email: user.email,
    });
    res.json({ token: accessToken, refreshToken: newRefresh });
  } catch {
    res.status(403).json({ message: "Invalid or expired refresh token" });
  }
};
module.exports = { register, login, refresh };
```

A.16 profileController.js - отримання профілю користувача

```
const { User, Plan, Website } = require("../models/models");
const getProfile = async (req, res) => {
  try {
    const user = await User.findByPk(req.user.id, {
      include: [
        {
          model: Plan,
          attributes: ["name"],
        },
        {
          model: Website,
          attributes: ["id", "name", "url", "status"],
        },
      ],
    });
    if (!user) {
```

```
      return res.status(404).json({ message: "User not found" });
    }
    const websites = user.Websites.map((site) => ({
      title: site.name || site.url,
      description: site.url,
      status: site.status,
    }));
    res.json({
      user: {
        email: user.email,
        name: user.name || "No name",
        plan: user.Plan?.name || "Free",
        activity: [
          { action: "Logged in", time: "just now" },
          { action: "Registered", time: "yesterday" },
        ],
        stats: {
          totalSites: user.Websites.length,
          avgLoadTime: 4.7,
          optimization: 86,
        },
      },
      websites,
    });
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: "Failed to fetch profile" });
  }
};
module.exports = { getProfile };
```

A.17 websiteController.js - додавання сайту, метрики, аналіз, SEO, PageSpeed

```
const { Website, WebsiteMetric } = require("../models/models");
const axios = require("axios");
const cheerio = require("cheerio");
const addWebsite = async (req, res) => {
  const { name, url } = req.body;
  try {
    const newWebsite = await Website.create({
      name,
      url,
      status: "not_optimized",
      UserId: req.user.id,
    });
    res.status(201).json(newWebsite);
  } catch (error) {
    console.error("Error creating site:", error);
    res.status(500).json({ message: "Failed to create site" });
  }
};
async function getUserWebsitesWithMetrics(req, res) {
  try {
    const websites = await Website.findAll({
      where: { UserId: req.user.id },
      include: [
        {
          model: WebsiteMetric,
          required: false,
        },
      ],
```

```
      order: [[WebsiteMetric, "createdAt", "DESC"]],
    });
    console.log("📊 Websites with metrics:", JSON.stringify(websites, null, 2));
    const result = websites.map((site) => {
      const siteJSON = site.toJSON();
      return {
        ...siteJSON,
        latestMetric: siteJSON.WebsiteMetrics?.[0] || null,
      };
    });
    res.json(result);
  } catch (err) {
    console.error("Ошибка при получении сайтов и метрик:", err);
    res.status(500).json({ error: "Ошибка при получении данных" });
  }
}
const clamp = (v, a, b) => Math.max(a, Math.min(b, v));
const normalizeLinear = (value, best, worst) => {
  if (value == null || isNaN(value)) return null;
  if (value <= best) return 1;
  if (value >= worst) return 0;
  return 1 - (value - best) / (worst - best);
};
function calcSeoScore(m) {
  const totalImages = (m.images_with_alt || 0) + (m.images_without_alt || 0);
  const imageAltScore =
    totalImages === 0 ? 1 : (m.images_with_alt || 0) / totalImages;
  const titleLen = m.title_length || 0;
  let titleScore;
  if (!titleLen) titleScore = 0;
```

```
else if (titleLen >= 10 && titleLen <= 60) titleScore = 1;
else if (titleLen < 10) titleScore = titleLen / 10;
else titleScore = Math.max(0, 1 - (titleLen - 60) / 40);

const h1 = m.h1_count || 0;
let h1Score;
if (h1 === 0) h1Score = 0;
else if (h1 === 1) h1Score = 1;
else h1Score = Math.max(0.5, 1 - (h1 - 1) * 0.1);
const metaDescScore = m.meta_description_present ? 1 : 0;
const canonicalScore = m.canonical_link ? 1 : 0;
const sitemapScore = m.sitemap_present ? 1 : 0;
const robotsScore = m.robots_txt_present ? 1 : 0;
const httpsScore = m.uses_https ? 1 : 0;
const viewportScore = m.viewport_tag_present ? 1 : 0;
const mobileFriendlyScore = m.mobile_friendly ? 1 : 0;
const securityHeadersScore = m.security_headers_present ? 1 : 0;
const weights = {
  title: 0.16,
  meta: 0.15,
  h1: 0.12,
  images_alt: 0.12,
  canonical: 0.07,
  sitemap: 0.06,
  robots: 0.05,
  https: 0.1,
  viewport: 0.07,
  mobile_friendly: 0.06,
  security_headers: 0.04,
};
```

```
  const factors = {
    title: titleScore,
    meta: metaDescScore,
    h1: h1Score,
    images_alt: imageAltScore,
    canonical: canonicalScore,
    sitemap: sitemapScore,
    robots: robotsScore,
    https: httpsScore,
    viewport: viewportScore,
    mobile_friendly: mobileFriendlyScore,
    security_headers: securityHeadersScore,
  };
  let weightedSum = 0;
  let weightSum = 0;
  Object.keys(weights).forEach((k) => {
    const w = weights[k];
    const val = factors[k];
    if (val == null) return;
    weightedSum += w * clamp(val, 0, 1);
    weightSum += w;
  });
  const final = weightSum > 0 ? (weightedSum / weightSum) * 100 : 0;
  return Math.round(final);
}
function calcOptimizationScore(m) {
  const isPos = (v) => typeof v === "number" && v > 0;
  const pageSizeScore =
    typeof m.page_size_kb === "number" && m.page_size_kb > 0
      ? normalizeLinear(m.page_size_kb, 200, 3000)
```

```
    : null;
const requestsScore =
  typeof m.number_of_http_requests === "number"
    ? normalizeLinear(m.number_of_http_requests, 20, 300)
    : null;
const fcpScore = isPos(m.first_contentful_paint_ms)
  ? normalizeLinear(m.first_contentful_paint_ms, 0, 4000)
  : null;
const lcpScore = isPos(m.largest_contentful_paint_ms)
  ? normalizeLinear(m.largest_contentful_paint_ms, 0, 4000)
  : null;
const tbtScore = isPos(m.total_blocking_time_ms)
  ? normalizeLinear(m.total_blocking_time_ms, 0, 600)
  : null;
const speedIndexScore = isPos(m.load_time_ms)
  ? normalizeLinear(m.load_time_ms, 0, 5000)
  : null;
const thirdPartyScore =
  typeof m.third_party_scripts_count === "number"
    ? normalizeLinear(m.third_party_scripts_count, 0, 50)
    : null;
const cssMinified = m.css_minified ? 1 : 0;
const jsMinified = m.js_minified ? 1 : 0;
const cached = m.static_assets_cached ? 1 : 0;
const modernImg = m.modern_image_formats_used ? 1 : 0;
const lazy = m.lazy_loading_used ? 1 : 0;
const criticalCss = m.critical_css_inlined ? 1 : 0;
const weights = {
  pageSize: 0.18,
  requests: 0.14,
```

```
  fcp: 0.12,
  lcp: 0.12,
  tbt: 0.1,
  speedIndex: 0.1,
  thirdParty: 0.05,
  cssMinified: 0.03,
  jsMinified: 0.03,
  cached: 0.05,
  modernImg: 0.03,
  lazy: 0.03,
  criticalCss: 0.02,
};
const factors = {
  pageSize: pageSizeScore,
  requests: requestsScore,
  fcp: fcpScore,
  lcp: lcpScore,
  tbt: tbtScore,
  speedIndex: speedIndexScore,
  thirdParty: thirdPartyScore,
  cssMinified,
  jsMinified,
  cached,
  modernImg,
  lazy,
  criticalCss,
};
let weightedSum = 0;
let weightSum = 0;
Object.keys(weights).forEach((k) => {
```

```
    const w = weights[k];
    const val = factors[k];
    if (val == null) return;
    weightedSum += w * clamp(val, 0, 1);
    weightSum += w;
  });
  const final = weightSum > 0 ? (weightedSum / weightSum) * 100 : 0;
  return Math.round(final);
}
const analyzeWebsite = async (req, res) => {
  try {
    const { websiteId } = req.params;
    const website = await Website.findByPk(websiteId, {
      include: [WebsiteMetric],
    });

    if (!website) return res.status(404).json({ message: "Website not found" });
    const apiKey = process.env.PAGESPEED_API_KEY;
    let lhr = {};
    try {
      const { data } = await axios.get(
        "https://www.googleapis.com/pagespeedonline/v5/runPagespeed",
        { params: { url: website.url, key: apiKey, strategy: "mobile" } }
      );
      lhr = data.lighthouseResult || {};
    } catch (e) {
      console.warn(
        "⚠ PageSpeed API error (will still compute local metrics):",
        e.response?.data || e.message
      );
```

```
  lhr = {};
}
const audits = lhr.audits || {};
const categories = lhr.categories || {};
let html = "";
let headers = {};
let httpStatus = 0;
try {
  const response = await axios.get(website.url, { timeout: 10000 });
  html = response.data;
  headers = response.headers;
  httpStatus = response.status;
} catch (e) {
  console.warn("⚠ Ошибка загрузки сайта:", e.message);
  httpStatus = e.response?.status || 0;
}
const $ = cheerio.load(html || "");
const h1Count = $("h1").length;
const titleLength = ($("title").text() || "").length;
const metaDescriptionPresent = $("meta[name='description']").length > 0;
const canonicalLink = $("link[rel='canonical']").attr("href") || null;
const images = $("img");
const imagesWithAlt = images.filter((i, el) => $(el).attr("alt")).length;
const imagesWithoutAlt = images.length - imagesWithAlt;
const internalLinks = $("a[href^='/']").length;
const externalLinks = $("a[href^='http']").length;
const modernImageFormatsUsed =
  $("img[src$='.webp'], img[src$='.avif']").length > 0;
const lazyLoadingUsed = $("img[loading='lazy']").length > 0;
let sitemapPresent = false;
```

```
let robotsTxtPresent = false;
try {
  await axios.get(new URL("/sitemap.xml", website.url).href);
  sitemapPresent = true;
} catch {}
try {
  await axios.get(new URL("/robots.txt", website.url).href);
  robotsTxtPresent = true;
} catch {}
const securityHeadersPresent = !!(
  headers["content-security-policy"] ||
  headers["x-frame-options"] ||
  headers["x-content-type-options"]
);
const htmlSizeKb = html ? Buffer.byteLength(html, "utf8") / 1024 : 0;
const pageSizeKb = headers["content-length"]
  ? parseInt(headers["content-length"]) / 1024
  : htmlSizeKb;
const speedIndex = Math.max(
  0,
  Math.round(audits["speed-index"]?.numericValue || 0)
);
const firstContentfulPaint = Math.max(
  0,
  Math.round(audits["first-contentful-paint"]?.numericValue || 0)
);
const largestContentfulPaint = Math.max(
  0,
  Math.round(audits["largest-contentful-paint"]?.numericValue || 0)
);
```

```
const totalBlockingTime = Math.max(
  0,
  Math.round(audits["total-blocking-time"]?.numericValue || 0)
);
const clsRaw = audits["cumulative-layout-shift"]?.numericValue ?? 0;
const cumulativeLayoutShift = Math.round(clsRaw * 1000);
const metrics = {
  load_time_ms: speedIndex,
  first_contentful_paint_ms: firstContentfulPaint,
  largest_contentful_paint_ms: largestContentfulPaint,
  total_blocking_time_ms: totalBlockingTime,
  cumulative_layout_shift: cumulativeLayoutShift,
  accessibility_score: Math.max(
    0,
    Math.round((categories.accessibility?.score || 0) * 100)
  ),
  seo_score: 0,
  optimization_score: 0,
  page_size_kb: Math.max(0, Math.round(pageSizeKb || 0)),
  html_size_kb: Math.max(0, Math.round(htmlSizeKb || 0)),
  number_of_http_requests: Math.max(0, $("*").length || 0),
  third_party_scripts_count: Math.max(
    0,
    $("script[src*='//']").length || 0
  ),
  css_minified: $("link[rel='stylesheet'][href*='.min.css']").length > 0,
  js_minified: $("script[src*='.min.js']").length > 0,
  static_assets_cached: headers["cache-control"]
    ? headers["cache-control"].includes("max-age")
    : false,
```

```
  critical_css_inlined: ($("style").text() || "").length > 0,
  internal_links: Math.max(0, internalLinks),
  external_links: Math.max(0, externalLinks),
  images_with_alt: Math.max(0, imagesWithAlt),
  images_without_alt: Math.max(0, imagesWithoutAlt),
  modern_image_formats_used: modernImageFormatsUsed,
  lazy_loading_used: lazyLoadingUsed,
  h1_count: Math.max(0, h1Count),
  title_length: Math.max(0, titleLength),
  meta_description_present: metaDescriptionPresent,
  canonical_link: !!canonicalLink,
  sitemap_present: sitemapPresent,
  robots_txt_present: robotsTxtPresent,
  uses_https: website.url.startsWith("https"),
  http2_or_higher: (headers["via"] || "").includes("h2") || false,
  security_headers_present: securityHeadersPresent,
  viewport_tag_present: $("meta[name='viewport']").length > 0,
  mobile_friendly: audits["viewport"]?.score === 1,
  http_statuses: httpStatus,
};
try {
  metrics.seo_score = calcSeoScore(metrics);
  metrics.optimization_score = calcOptimizationScore({
    ...metrics,
    first_contentful_paint_ms: metrics.first_contentful_paint_ms,
    largest_contentful_paint_ms: metrics.largest_contentful_paint_ms,
    total_blocking_time_ms: metrics.total_blocking_time_ms,
    load_time_ms: metrics.load_time_ms,
  });
} catch (e) {
```

```
      console.error("Error computing derived scores:", e);
    }
    const savedMetric = await WebsiteMetric.create({
      WebsiteId: website.id,
      ...metrics,
    });
    await website.update({ status: "optimized", lastCheckedAt: new Date() });
    res.json({ website, latestMetric: savedMetric });
  } catch (error) {
    console.error(error.response?.data || error.message);
    res.status(500).json({ message: "Failed to analyze site" });
  }
};
module.exports = {
  addWebsite,
  getUserWebsitesWithMetrics,
  analyzeWebsite,
};
```

Код backend частини

A.18 models.js - Sequelize моделі User, Plan, Website, WebsiteMetric

```
const sequelize = require("./database");
const { DataTypes } = require("sequelize");
const User = sequelize.define(
  "User",
  {
    email: {
      type: DataTypes.STRING,
      unique: true,
      allowNull: false,
      validate: {
        isEmail: true,
      },
    },
    password: {
      type: DataTypes.STRING,
      allowNull: false,
    },
  },
  { timestamps: true }
);
const Plan = sequelize.define(
  "Plan",
  {
    name: {
      type: DataTypes.STRING,
      allowNull: false,
```

```
      unique: true,
    },
    maxWebsites: {
      type: DataTypes.INTEGER,
      defaultValue: 1,
    },
    features: {
      type: DataTypes.JSON,
      allowNull: true,
    },
  },
  { timestamps: true }
);
const Website = sequelize.define(
  "Website",
  {
    url: {
      type: DataTypes.STRING,
      allowNull: false,
      validate: {
        isUrl: true,
      },
    },
    name: {
      type: DataTypes.STRING,
      allowNull: true,
    },
    status: {
      type: DataTypes.ENUM("not_optimized", "optimizing", "optimized"),
      defaultValue: "not_optimized",
```

```
      allowNull: false,
    },
    lastCheckedAt: {
      type: DataTypes.DATE,
      allowNull: true,
    },
  },
  { timestamps: true }
);
const WebsiteMetric = sequelize.define(
  "WebsiteMetric",
  {
    load_time_ms: DataTypes.INTEGER,
    page_size_kb: DataTypes.INTEGER,
    html_size_kb: DataTypes.INTEGER,
    first_contentful_paint_ms: DataTypes.INTEGER,
    largest_contentful_paint_ms: DataTypes.INTEGER,
    total_blocking_time_ms: DataTypes.INTEGER,
    cumulative_layout_shift: DataTypes.DECIMAL(4, 3),
    number_of_http_requests: DataTypes.INTEGER,
    third_party_scripts_count: DataTypes.INTEGER,
    css_minified: DataTypes.BOOLEAN,
    js_minified: DataTypes.BOOLEAN,
    static_assets_cached: DataTypes.BOOLEAN,
    critical_css_inlined: DataTypes.BOOLEAN,
    internal_links: DataTypes.INTEGER,
    external_links: DataTypes.INTEGER,
    images_with_alt: DataTypes.INTEGER,
    images_without_alt: DataTypes.INTEGER,
    modern_image_formats_used: DataTypes.BOOLEAN,
```

```
    lazy_loading_used: DataTypes.BOOLEAN,
    h1_count: DataTypes.INTEGER,
    title_length: DataTypes.INTEGER,
    meta_description_present: DataTypes.BOOLEAN,
    canonical_link: DataTypes.BOOLEAN,
    sitemap_present: DataTypes.BOOLEAN,
    robots_txt_present: DataTypes.BOOLEAN,
    uses_https: DataTypes.BOOLEAN,
    http2_or_higher: DataTypes.BOOLEAN,
    security_headers_present: DataTypes.BOOLEAN,
    viewport_tag_present: DataTypes.BOOLEAN,
    mobile_friendly: DataTypes.BOOLEAN,
    accessibility_score: DataTypes.DECIMAL(5, 2),
    seo_score: DataTypes.DECIMAL(5, 2),
    optimization_score: {
      type: DataTypes.DECIMAL(5, 2),
      allowNull: true,
    },
    http_statuses: {
      type: DataTypes.JSONB,
      allowNull: true,
    },
  },
  { timestamps: true }
);
User.hasMany(Website, { onDelete: "CASCADE" });
Website.belongsTo(User);
Website.hasMany(WebsiteMetric, { onDelete: "CASCADE" });
WebsiteMetric.belongsTo(Website);
Plan.hasMany(User);
```

```
User.belongsTo(Plan);
module.exports = {
  sequelize,
  User,
  Website,
  WebsiteMetric,
  Plan,
};
```