Міністерство освіти і науки України

«Київський Національний університет імені Тараса Шевченка»

Факультет інформаційних технологій
Кафедра мережевих та інтернет технологій

ЗАТВЕРДЖУЮ
завідувач кафедри
мережевих та інтернет технологій
_____ Юрий КРАВЧЕНКО
«_____»_____2023 року

КВАЛІФІКАЦІЙНА РОБОТА
БАКАЛАВРА

галузі знань 17 «Електроніка та телекомунікації»
за спеціальністю 172 «Телекомунікації та радіотехніка»
освітньо-професійна програма «Мережеві та інтернет технології»

на тему:

РОЗРОБКА ТА ДОСЛІДЖЕННЯ МЕТОДІВ ОПТИМІЗАЦІЇ ВЕБ-ДОДАТКУ

Виконав: студент групи МІТ - 41

Олександр КОВАЛЬСЬКИЙ _____
(ім'я та ПРІЗВИЩЕ) (підпис)

Керівник: доцент кафедри мережевих та інтернет технологій

к.т.н., доцент Ольга ЛЕЩЕНКО _____
(науковий ступень, вчене звання, ім'я та ПРІЗВИЩЕ) (підпис)

Київ 2023

Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра мережевих та інтернет технологій

ЗАТВЕРДЖУЮ
завідувач кафедри
мережевих та інтернет
технологій
_____ Юрій КРАВЧЕНКО

«_____»_____2023 року

ЗАВДАННЯ
НА ДИПЛОМНУ РОБОТУ

Здобувачу вищої освіти Ковальський Олександр Вячеславович
(прізвище, ім'я, по батькові)

1. Тема роботи: Розробка та дослідження методів оптимізації веб-додатку

затверджена на засіданні кафедри МІТ «7» грудня 2022 р. протокол № 8

2. Термін здачі закінченої роботи «30» травня 2022 р

3. Вихідні дані до проекту (роботи)

Мова програмування – NodeJS, JavaScript, TypeScript

Технологія NextJS, React, HTML, CSS, Prisma (ORM)

4. Зміст пояснювальної записки (перелік питань, що їх потрібно розробити, обсяг – 35-40 стор.)

Вступ

1. Огляд методів оптимізації веб-додатку
    1.1 Сфера застосування методів оптимізації
    1.2 Визначення методів оптимізації
    1.3 Вимоги до продуктивності та користувацького досвіду
    1.4 Висновок
2. Аналіз методів оптимізації
    2.1 Інструменти аналізу
    2.2 Висновок
3. Розробка оптимізованого веб-додатку
    3.1 Проектування архітектури комерційного веб-додатку
    3.2 Розробка серверної частини
    3.3 Розробка клієнтської частини
    3.4 Тестування та оцінка ефективності оптимізованого веб-додатку
    3.5 Оцінка ефективності веб-додатку
    3.6 Висновок
4. Експериментальне дослідження
    4.1 Опис експериментального дослідження
    4.2 Збір та аналіз даних
    4.3 Переваги та недоліки використаних методів

4.4 Висновок

Висновок

5. Перелік графічного матеріалу (10-12 слайдів)

Дата видачі завдання

Керівник роботи доцент Ольга ЛЕЩЕНКО
(підпис) (посада, ім'я, ПРІЗВИЩЕ)

Завдання прийняв до виконання Олександр КОВАЛЬСЬКИЙ

КАЛЕНДАРНИЙ ПЛАН ВИКОНАННЯ РОБОТИ

Номер	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Підготовчий	02.05.2023	
2	Розділ 1	10.05.2023	
3	Розділ 2	15.05.2023	
4	Розділ 3	24.05.2023	
5	Розділ 4	25.05.2023	
6	Доповідь і слайди	28.0.5.2023	
7	Пояснювальна записка	30.05.2023	

Здобувач вищої освіти _____ Олександр КОВАЛЬСЬКИЙ
(підпис) (ім'я, ПРІЗВИЩЕ)

Керівник _____ Ольга ЛЕЩЕНКО
(підпис) (ім'я, ПРІЗВИЩЕ)

РЕФЕРАТ

Дипломна робота на тему "Розробка та дослідження методів оптимізації веб-додатку" присвячена вдосконаленню продуктивності та користувацького досвіду веб-додатків.

Пояснювальна записка до дипломного проекту “Розробка та дослідження методів оптимізації веб-додатку” містить 75 сторінок, 33 рисунків та 10 джерел.

Мета дослідження є підвищення ефективності розробки веб-додатку, за допомогою методів оптимізації яка спрямована на покращення швидкодії та реактивності веб-додатку.

Об’єкт дослідження — оптимізація веб-додатку, який створенний за допомогою мови програмування TypeScript, фреймворку NextJS, бази даних PostgreSQL, ORM (Prisma), бібліотеки для створення API (TRPC) та бібліотеки для авторизації та автентифікації Next-Auth.

Предмет дослідження — підходи до оптимізації веб-додатку.

Методи дослідження — методи оптимізації, за допомогою яких проводиться дослідження оптимізації, а саме Lighthouse та K6.

Практичне значення роботи полягає у тому, що результати дослідження методів оптимізації веб-додатку з Next.js можуть мати безпосереднє застосування в реальних проектах веб-розробки. Це дозволить покращити взаємодію з користувачами, забезпечити кращий досвід використання, знизити витрати на хостинг та забезпечити оптимальну продуктивність навіть при високому навантаженні.

В рамках дослідження будуть визначені ключові фактори, що впливають на продуктивність веб-додатків, такі як завантаження сторінок, операції з базою даних та робота з ресурсами.

Ключові слова:

ВЕБ-ДОДАТОК, ОПТИМІЗАЦІЯ, NEXTJS, TYPESCRIPT, JAVASCRIPT HTML, CSS.

ABSTRACT

The thesis on the topic "Development and research of web application optimization methods" is devoted to improving the performance and user experience of web applications.

The explanatory note to the diploma project "Development and research of web application optimization methods" contains 75 pages, 33 figures and 10 sources.

The **purpose of the study** is to increase the efficiency of web application development, using optimization methods aimed at improving the speed and reactivity of the web application.

The **object of research** is the optimization of a web application created using the TypeScript programming language, the NextJS framework, the PostgreSQL database, ORM (Prisma), the API creation library (TRPC) and the Next-Auth authorization and authentication library.

The **subject of research** is approaches to web application optimization.
Research methods — optimization methods used to conduct optimization research, namely Lighthouse and K6.

The **practical significance** of the work lies in the fact that the results of research on methods of optimizing a web application with Next.js can be directly applied to real web development projects. This will improve user interaction, provide a better user experience, reduce hosting costs and ensure optimal performance even under high load.

The study will identify key factors affecting web application performance, such as page loading, database operations, and resource handling.

Keywords:

WEB APP, OPTIMIZATION, NEXTJS, TYPESCRIPT, JAVASCRIPT HTML, CSS.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ДП — дипломний проект;

ДР — дипломна робота;

БД — база даних;

РБД — реляційні бази даних;

НРБД — нереляційні бази даних;

SSG — Server Side Generation;

SSR — Server Side Rendering;

TTFB — Time to First Byte;

FCP — First Contentful Paint;

FID — First Input Delay;

CLS — Cumulative Layout Shift;

TTI — Time to Interactive;

LCP — Largest Contentful Paint;

TBT — Total Blocking Time;

API - Application Programming Interface.

ВСТУП

У сучасному цифровому світі веб-додатки залишаються необхідними інструментами для бізнесу та споживачів. Швидкість, продуктивність та користувацький досвід веб-додатків стали ключовими факторами їх успіху. Оптимізація веб-додатків є важливою задачею для розробників із забезпечення швидкості надання послуг, ефективної роботи та задоволення користувачів.

Ця бакалаврська робота присвячена розробці та дослідженню методів оптимізації веб-додатку з наданням покращення його продуктивності та користувацького досвіду. Оптимізація веб-додатка включає різні аспекти, від архітектурних розробок до оптимізації запитів до бази даних та оптимізації сторінок завантаження.

Мета цієї роботи є дослідження методів оптимізації, розробка веб-додатку та застосування ефективних методів оптимізації, спрямованих на покращення швидкодії та реактивності веб-додатків.

Результати дослідження та розробки можуть бути використані для вдосконалення веб-додатків, забезпечення швидкого завантаження сторінок, ефективної роботи з базою даних та оптимального використання ресурсів.

Для досягнення поставленої мети вирішуються такі задачі:

1. Створення веб-додатку;

2. Застосування методів оптимізації;

3. Порівняння веб-додатку до і після застосування методів оптимізації за допомогою інструментів Lighthouse та K6.

В результаті роботи очікується отримати оптимізований веб-додаток, який буде відповідати високим стандартам продуктивності та користувацького досвіду. Результати досліджень та розробок можуть бути використані як практичний внесок у галузь оптимізації веб-додатків та служити основою для подальших досліджень у цій області.

Ця робота має важливе значення, оскільки продуктивність та користувацький досвід веб-додатків є вирішальними факторами для задоволення

потреб сучасного користувача. Розробка ефективних методів оптимізації веб-додатків сприятиме розвитку інтернет-простору та покращенню якості веб-сервісів для користувачів.

РОЗДІЛ 1. ОГЛЯД МЕТОДІВ ОПТИМІЗАЦІЇ ВЕБ-ДОДАТКУ

Огляд методів оптимізації веб-додатку є важливим етапом у процесі розробки та покращення продуктивності веб-додатків. Швидкодія та ефективність веб-додатку є ключовими факторами, які впливають на користувацький досвід. Огляд методів оптимізації надає змогу оцінити різні підходи, техніки та інструменти, які можуть допомогти забезпечити оптимальну продуктивність веб-додатку.

Будуть розглянуті різні аспекти оптимізації веб-додатків. Серед них - швидкість завантаження сторінок, оптимізація ресурсів, зменшення запитів до сервера, кешування, компресія даних і так далі. Дослідження цих аспектів дозволить виявити ефективні методи оптимізації та впровадити їх у веб-додаток.

Огляд методів оптимізації веб-додатку має на меті забезпечити кращу швидкодію, зменшення завантаження та поліпшення користувацького досвіду. Цей розділ допоможе усвідомити важливість оптимізації та підібрати підходи, які найкраще підходять для їхнього веб-додатку, розділ включає в себе як технічні аспекти, так і стратегічні рішення, що дозволяють досягти максимальної продуктивності та задоволення користувачів.

1.1. Сфера застосування методів оптимізації

Існує багато варіантів того, як оптимізувати веб-додаток, але щоб їх згрупувати та систематизувати, можна використовувати модель OSI (Open Systems Interconnection). Рівні моделі OSI зображені на рисунку 1.1.

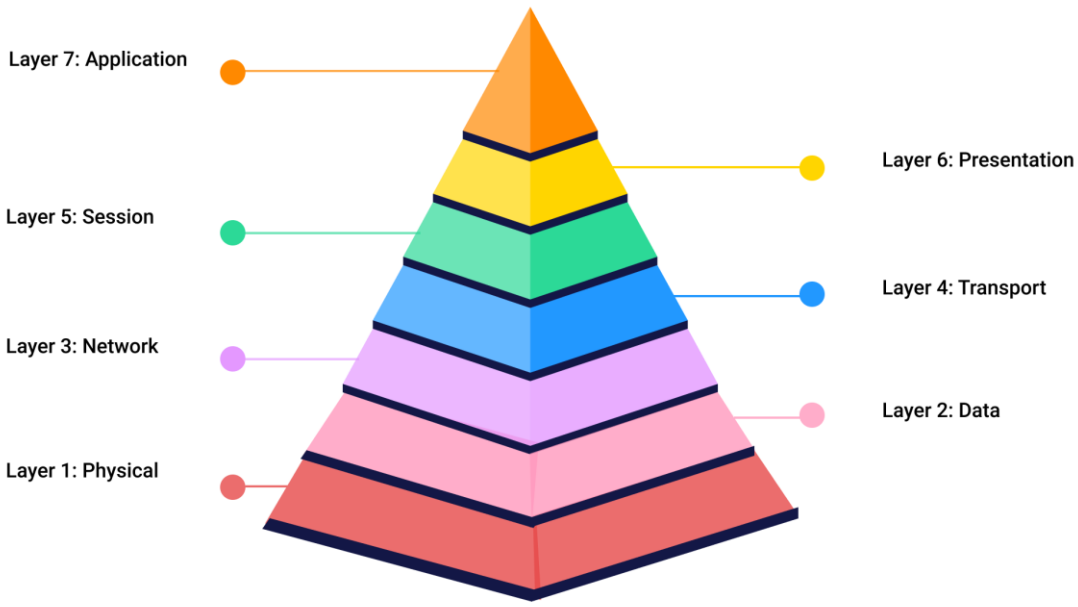


Рисунок 1.1 - Рівень моделі OSI

Ця модель визначає сім рівнів, кожен з яких має свої функції та відповідальності в процесі передачі даних у мережі:

1. Application Layer (Додатковий рівень): Додатковий рівень є найвищим рівнем OSI. Він надає інтерфейс для взаємодії користувача з мережею та додатками. Протоколи, такі як HTTP (Hypertext Transfer Protocol), SMTP (Simple Mail Transfer Protocol), FTP (File Transfer Protocol) та інші, працюють на цьому рівні.
2. Presentation Layer (Подання): Цей рівень забезпечує стандартизацію подання даних, що передаються в мережі. Він відповідає за перекодування, шифрування, стиск та інші процеси, що стосуються представлення даних.
3. Session Layer (Рівень сесії): Рівень сесії встановлює, керує та припиняє з'єднання між двома кінцевими вузлами. Він відповідає за управління сеансами зв'язку, синхронізацію та управління обміном даними.

4. Transport Layer (Транспортний рівень): Транспортний рівень забезпечує надійну передачу даних між кінцевими точками. Він контролює потік даних, поділяє їх на сегменти та забезпечує доставку в правильному порядку. Протоколи, такі як TCP (Transmission Control Protocol) і UDP (User Datagram Protocol), працюють у цьому рівні.
5. Network Layer (Мережевий рівень): Мережевий рівень відповідає за маршрутизацію пакетів даних через мережу. Він визначає адресацію, вибір шляху маршрутизації, керування навантаженням та інші функції, що стосуються передачі на рівні мережі. Протоколи, такі як IP (Internet Protocol), працюють у цьому рівні.
6. Data Link Layer (Рівень каналу передачі даних): Цей рівень забезпечує надійну передачу даних між сусідніми вузлами мережі. Він поділяє дані на кадри, здійснює контроль помилок, керування доступом до каналу та виконує інші функції, що гарантують безперервну передачу даних. На цьому рівні працюють протоколи Ethernet, Wi-Fi, PPP (Point-to-Point Protocol) та інші.
7. Physical Layer (Фізичний рівень): Цей рівень відповідає за передачу бітів даних через фізичні канали зв'язку, такі як дротові кабелі, оптоволокно або бездротові медіа. Він визначає характеристики фізичного з'єднання, наприклад рівень напруги, швидкість передачі даних та методи кодування.

Розглянувши модель OSI, можемо ідентифікувати, на якому рівні мережі можна застосовувати оптимізаційні методи для покращення продуктивності веб-додатку.

Application layer, який є верхнім рівнем моделі OSI і відіграє найбільш важливу роль у використанні методів оптимізації. Цей рівень включає програми та сервіси, які надають користувачам багатофункціональні можливості. Застосування методів оптимізації на рівні застосункового шару має великий потенціал для покращення продуктивності, ефективності та взаємодії з користувачами.

На Application layer можна виконати оптимізацію алгоритмів, що використовуються для обробки даних, забезпечити оптимальне використання ресурсів, зменшити завантаження мережі та прискорити швидкість відповіді запитів. Такі методи оптимізації можуть включати кешування, компресію даних, паралельну обробку, передачу даних за запитом, використання масштабованих архітектур та багато інших методів.

Застосування методів оптимізації на Application layer шару може суттєво поліпшити продуктивність та ефективність веб-додатків, дозволяючи їм працювати швидше, споживати менше ресурсів і забезпечувати кращий досвід користувачів. Такий підхід дозволяє зосередитися на ключових аспектах оптимізації, що мають прямий вплив на функціональність та продуктивність веб-додатків.

1.2 Визначення методів оптимізації

Визначення методів оптимізації є важливим етапом у процесі розробки та покращення продуктивності веб-додатків. На цьому етапі проводиться дослідження та аналіз різних підходів, технік та інструментів, що сприяють поліпшенню продуктивності додатку. Визначення методів оптимізації передбачає уважне вивчення особливостей самого додатку, його архітектури, використовуваних технологій та потреб користувачів. Це дозволяє виявити слабкі місця та потенційні області для оптимізації з метою досягнення більшої продуктивності, швидкодії та задоволення користувачів.

У процесі визначення методів оптимізації будуть використані різноманітні підходи, спрямовані на поліпшення продуктивності веб-додатку. Серед цих методів можуть бути включені:

1. Швидкість завантаження сторінок[2]:

Кешування сторінок і ресурсів;

Мінімізація файлів і ресурсів;

Компресія зображень;

Передавання даних за допомогою CDNs.

2. Оптимізація виконання клієнтського коду:

Мінімізація та компресія JavaScript і CSS;

Використання асинхронних запитів;

Лінива загрузка ресурсів;

Оптимізація роботи з DOM.

3. Оптимізація серверної частини[3]:

Кешування запитів;

Оптимізація бази даних;

Масштабування серверів;

Використання кешування результатів.

4. Оптимізація бази даних:

Використання індексів і оптимізованих запитів;

Нормалізація даних.

5. Оптимізація зображень та мультимедіа:

Компресія та оптимізація зображень;

Використання форматів зображень з низьким розміром;

Лінива загрузка зображень;

Оптимізація відео та аудіо контенту.

6. Використання кешування[4]:

Кешування на різних рівнях (клієнтському та серверному);

Використання HTTP-кешування.

Усі ці методи оптимізації веб-додатку взаємодоповнюються та допомагають зрозуміти різні аспекти та перспективи оптимізації. Вони служать основою для подальшого дослідження, впровадження та вдосконалення методів оптимізації веб-додатку з метою покращення його продуктивності та користувацького досвіду

1.3 Вимоги до продуктивності та користувацького досвіду

Користувацький досвід (User Experience, UX) - це спосіб, яким користувач сприймає та взаємодіє з продуктом, системою або додатком. Він охоплює всі аспекти взаємодії, від першого враження до кінцевого результату. Користувацький досвід визначається зручністю, задоволеністю та ефективністю взаємодії, які надаються користувачам під час використання продукту.

Швидкість загрузки сторінки, зокрема, відіграє велике значення у користувацькому досвіді. Швидке завантаження сторінки є важливим для задоволення користувача і забезпечення позитивного першого враження. Користувачі очікують, що сторінки будуть завантажуватися миттєво або з мінімальним затримкою. Довгий час завантаження може призвести до незадоволення, втрати зацікавленості та навіть відмови від використання продукту.

Швидкість завантаження сторінки також має вплив на конверсію, трафік та рангування у пошукових системах. Повільні сторінки можуть знизити конверсію, оскільки користувачі можуть покинути сайт перед тим, як вони зможуть скористатися його перевагами. Пошукові системи також враховують швидкість завантаження сторінок під час ранжування, тому швидкість може вплинути на видимість та відвідуваність вашого веб-додатку або сайту.

Для досягнення задоволення користувачів та високої продуктивності веб-додатку необхідно враховувати ряд вимог і рекомендацій. Ось деякі ключові аспекти, на які варто звернути увагу:

1. Швидкість завантаження: Користувачі очікують, що веб-додатку буде швидким і буде відповідати миттєво на їхні дії[7]. Забезпечення швидкого завантаження сторінок, швидкої відповіді на запити та ефективної роботи веб-додатку в цілому є важливими факторами продуктивності.
2. Адаптивний дизайн: Веб-додаток повинен бути зручним у використанні на різних пристроях та розмірах екрану. Адаптивний дизайн дозволяє користувачам з легкістю взаємодіяти з додатком на різних пристроях, включаючи комп'ютери, планшети та смартфони.

3. Простота навігації: Веб-додаток повинен мати зрозумілу та логічну структуру навігації, щоб користувачі могли швидко зорієнтуватися та знайти потрібну інформацію чи функціонал.
4. Відповідність стандартам безпеки: Забезпечення безпеки даних користувачів та захист веб-прикладання від потенційних загроз є критичним аспектом. Дотримання стандартів безпеки, таких як захищений протокол HTTPS, криптографічні заходи та механізми автентифікації, є обов'язковим.
5. Інтуїтивний і зручний інтерфейс: веб-додаток повинен мати інтуїтивно зрозумілий та зручний для використання інтерфейс. Чітко позначені елементи управління, доступність важливих функцій та зручність взаємодії з додатком сприяють покращенню користувацького досвіду.
6. Підтримка різних браузерів та платформ: веб-додаток має працювати на різних популярних браузерах (наприклад, Google Chrome, Mozilla Firefox, Safari, Microsoft Edge) та підтримувати різні операційні системи (Windows, macOS, Linux, Android, iOS).
7. Масштабованість: Веб-додаток повинен бути здатним ефективно працювати навіть при збільшенні обсягу даних або навантаженні на сервер[8]. Оптимізація продуктивності додатку та використання масштабованих архітектур допоможуть підтримувати продуктивність при зростанні використання.
8. Надійність і тестування: Веб-додаток повинен бути стабільним і надійним, мінімізуючи відмови, помилки та непередбачені ситуації. Регулярне тестування, в тому числі функціональне тестування, тестування продуктивності та тестування безпеки, допомагає виявити проблеми та покращити якість додатку.
9. Підтримка та оновлення: Забезпечення регулярної підтримки веб-додатку, виправлення помилок, удосконалення функціоналу та реагування на вимоги користувачів є важливими аспектами для підтримки задоволення користувачів та високої продуктивності.

10. Використання сучасних форматів даних для файлів[9]. Наприклад, для статичних файлів, таких як зображень це webp, для шрифтів woff або woff2, для динамічних файлів, таких як відео це webm і так далі.

Усе це в сукупності допомагає забезпечити оптимальну роботу веб-додатку, забезпечити задоволення користувачів та досягнути високої продуктивності. Важливо не лише знати різні методи оптимізації, але й вміти використовувати їх на практиці, здійснюючи аналіз, вибір та впровадження найефективніших з них. Тільки таким чином можна досягти успіху і забезпечити високу якість веб-додатку для задоволення потреб користувачів.

1.4 Висновок

В результаті огляду методів оптимізації веб-додатку було розглянуто підходи та інструменти, спрямовані на покращення продуктивності.

Отже, огляд методів оптимізації веб-додатку є необхідним етапом у процесі розробки, який допомагає виявити ефективні підходи та засоби для досягнення оптимальної продуктивності. Цей огляд надає важливу інформацію для прийняття обґрунтованих рішень та реалізації оптимізаційних методів, спрямованих на поліпшення веб-додатку.

РОЗДІЛ 2. АНАЛІЗ МЕТОДІВ ОПТИМІЗАЦІЇ ВЕБ-ДОДАТКУ

Аналіз методів оптимізації веб-додатку включає використання різноманітних інструментів для оцінки та аналізу ефективності різних підходів до оптимізації. Ці інструменти допомагають зрозуміти, які методи оптимізації найбільш вдало використовувати для досягнення покращення продуктивності та швидкодії веб-додатку.

Один з основних інструментів аналізу - це аналіз продуктивності додатку. Це включає збір та аналіз даних про завантаження сервера, відгуки користувачів, швидкість завантаження сторінок та інші метрики[1] продуктивності. Інформація, отримана з моніторингу, допомагає виявити проблемні місця та потенційні області оптимізації.

Завдяки використанню інструментів аналізу можна оцінити ефективність різних методів оптимізації та визначити, які з них є найбільш вдалими для веб-додатку. Цей аналіз допомагає знайти способи покращення продуктивності, зниження часу завантаження сторінок та забезпечення більш задоволеного користувачами досвіду використання веб-додатку.

2.1 Інструментальні засоби для аналізу продуктивності веб-додатків

Інструменти аналізу відіграють важливу роль у процесі оптимізації веб-додатків. Вони надають нам можливість отримати детальну інформацію про продуктивність та ефективність свого додатку, виявити проблемні місця та знайти шляхи їх вирішення.

Перш за все, інструменти аналізу дозволяють здійснити моніторинг продуктивності веб-додатку в реальному часі. Вони забезпечують збір та візуалізацію ключових метрик, таких як час відкликання сервера, час завантаження сторінок, використання пам'яті та інші. Це дозволяє виявити швидкісні проблеми, які можуть впливати на користувацький досвід.

Далі, інструменти аналізу надають можливість здійснювати детальний аудит веб-додатків з точки зору швидкості завантаження, розміру ресурсів, кешування та інших факторів. Вони допомагають ідентифікувати проблемні зони, які можуть бути оптимізовані, наприклад, зменшення розміру файлів, використання кешування для зменшення запитів до сервера тощо.

Крім того, інструменти аналізу надають можливість проводити навантажувальні тести, що дозволяють симулювати великі навантаження на веб-додаток та оцінювати його продуктивність та стабільність. Це дозволяє виявити проблеми масштабованості, недостатньої оптимізації чи нестійкості системи.

Популярні інструменти аналізу та оптимізації веб-додатків включають:

1. k6 - це високопродуктивний інструмент для навантажувального тестування та перевірки продуктивності. Він дозволяє розробникам та інженерам тестувати шкалування веб-додатків та мережевих служб, а також оцінювати їх продуктивність за різних навантажень. Переваги: Простий у використанні інструмент для навантажувального тестування та перевірки продуктивності. Підтримує написання скриптів на JavaScript та надає докладні звіти. Недоліки: Деякі розширені функції можуть бути доступні тільки у комерційній версії. На рисунку 2.1 приклад використання.

Рисунок 2.1 – Приклад використання k6

2. GTmetrix: Це онлайн-інструмент, який надає детальний звіт про продуктивність веб-сторінок. GTmetrix оцінює швидкість завантаження сторінки, розмір файлів, кількість запитів до сервера та інші метрики. Він також надає рекомендації щодо оптимізації для покращення продуктивності. Переваги: Надає докладний аналіз продуктивності, включаючи оцінку швидкості завантаження, оптимізацію зображень, кешування та інші поради. Має зручний інтерфейс користувача та підтримує багато місць для тестування. Недоліки: Деякі функції доступні тільки у платній версії. Звіти можуть бути трохи складними для новачків. На рисунку 2.2 приклад використання.

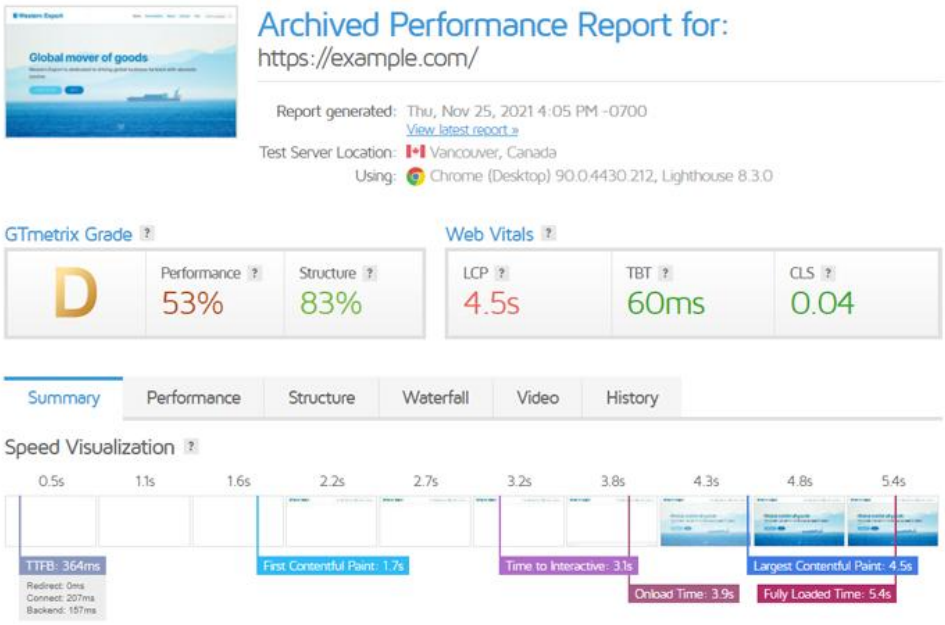


Рисунок 2.2 – Приклад використання GTmetrix

3. WebPageTest: Це інструмент, який дозволяє виконувати тестування швидкості завантаження веб-сторінок з різних місць світу. Він надає детальну інформацію про годину завантаження, розмір сторінки, запити до сервера та інші метрики. WebPageTest також дозволяє виконувати повторні тести для моніторингу продуктивності в годині. Переваги: Надає детальні звіти про продуктивність, включаючи швидкість завантаження, аналіз завантаження сторінки, водоспад запитів та інші метрики. Дозволяє вибирати місце тестування та різні конфігурації. Недоліки: Інтерфейс може здатися складним для новачків. Є обмежена кількість безкоштовних запитів. На рисунку 2.3 приклад використання.

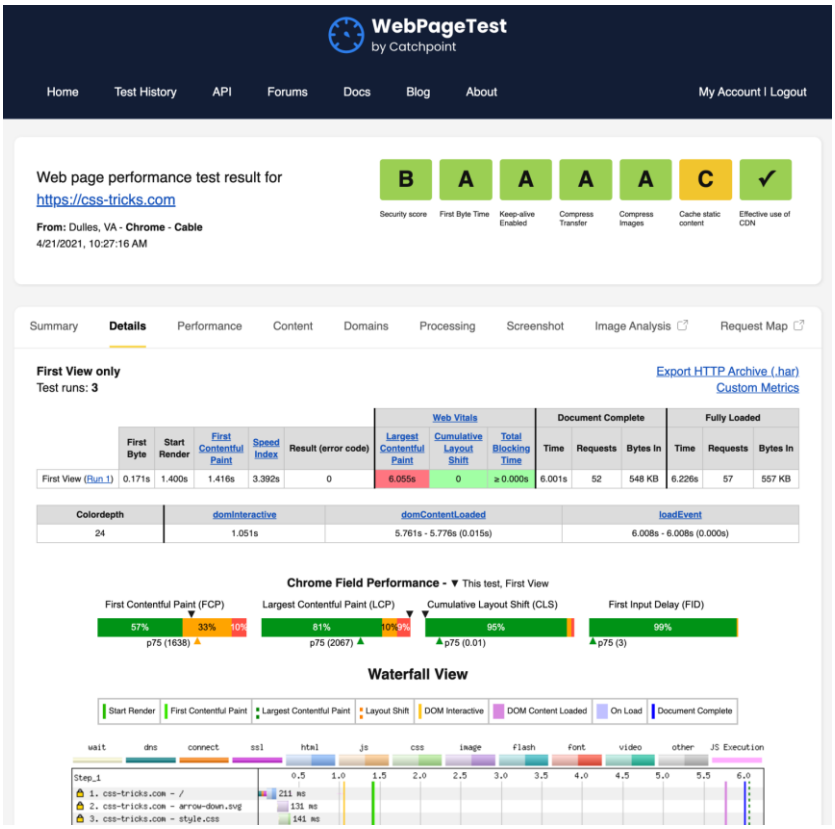


Рисунок 2.3 – Приклад використання WebPageTest

4. Pingdom: Це інструмент мониторингу продуктивності, який надає інформацію про годину завантаження сторінки, розмір файлів, кількість запитів та інші метрики. Він також має можливість мониторити продуктивність сайту на постійній основі та надсилати сповіщення про будь-які проблеми. Переваги: Простий у використанні інструмент, який надає інформацію про швидкість завантаження та доступність веб-сайту. Має зручний інтерфейс користувача та підтримує багато місць для тестування. Недоліки: Обмежені функціональні можливості порівняно з іншими інструментами. Деякі розширені функції доступні тільки у платній версії. На рисунку 2.4 приклад використання.

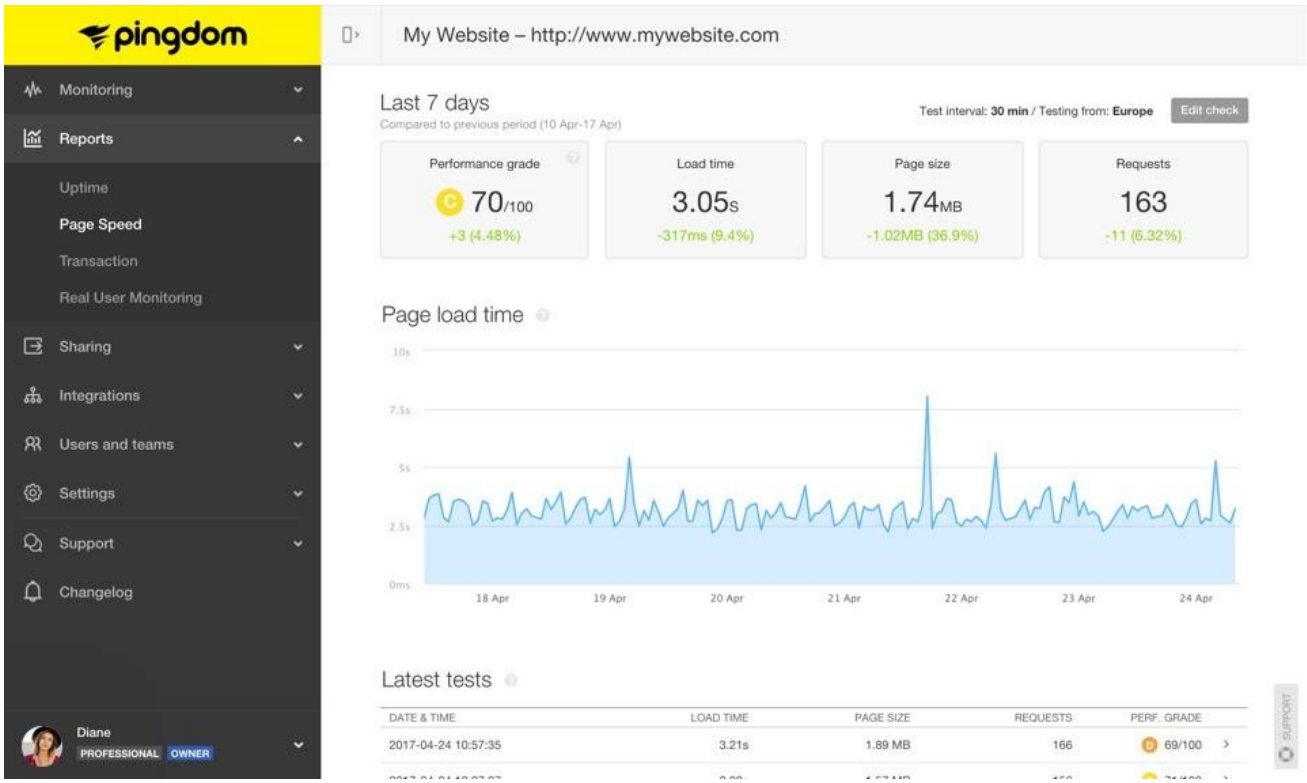


Рисунок 2.4 – Приклад використання Pingdom

5. YSlow: Це розширення для браузерів, що дає оцінку продуктивності веб-сторінок на основі рекомендацій Yahoo. YSlow аналізує різні аспекти, включаючи кешування, компресію файлів, використання CDN та інші, та надає рекомендації щодо покращення продуктивності. Переваги: Аналізує продуктивність веб-сторінки, пропонує поради щодо оптимізації, такі як стиснення ресурсів, кешування та інші покращення. Інтегрований з браузерним розширенням Firebug. Недоліки: Розвиток інструменту був заморожений, тому він може бути менш актуальним порівняно з більш новими інструментами. На рисунку 2.5 приклад використання.

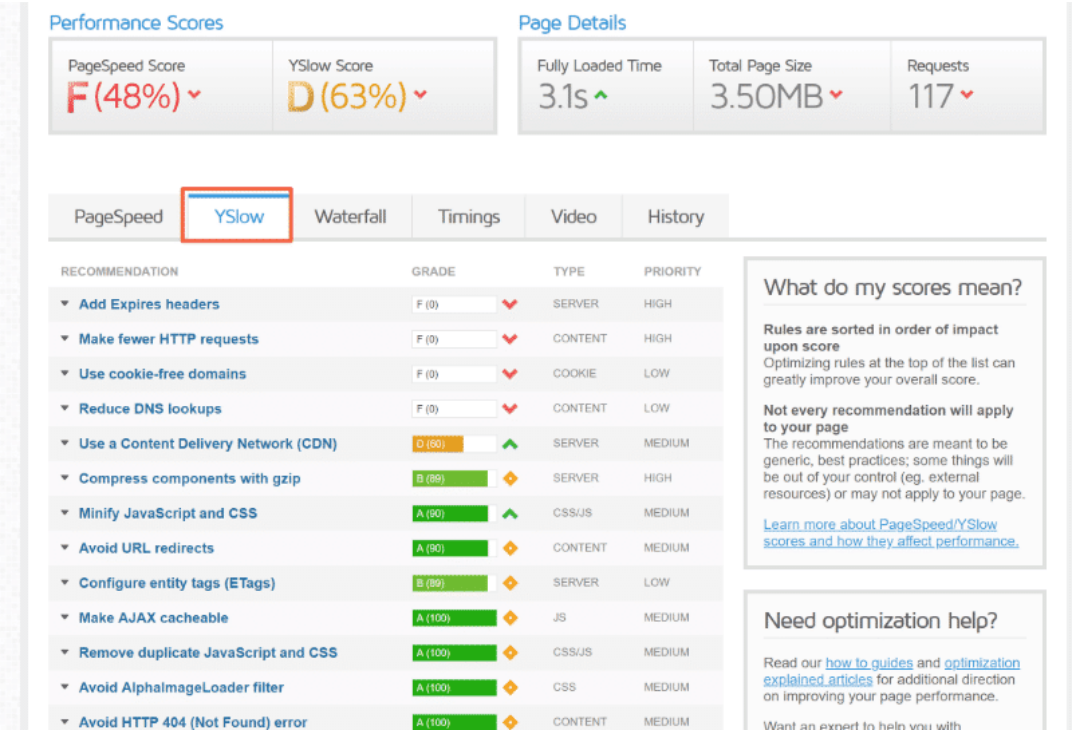


Рисунок 2.5 – Приклад використання YSlow

6. Apache JMeter: Це інструмент для тестування продуктивності та навантаження сервера. Він дозволяє виконувати симуляцію великого обсягу запитів до веб-додатку, що допомагає оцінити його продуктивність та виявити проблемні місця. Переваги: Потужний інструмент для тестування продуктивності та навантаження. Підтримує багато протоколів та може симулювати різні сценарії навантаження. Недоліки: Вимагає вивчення та досвіду для ефективного використання. Не надає візуального звіту, потрібен аналіз результатів. На рисунку 2.6 приклад використання.

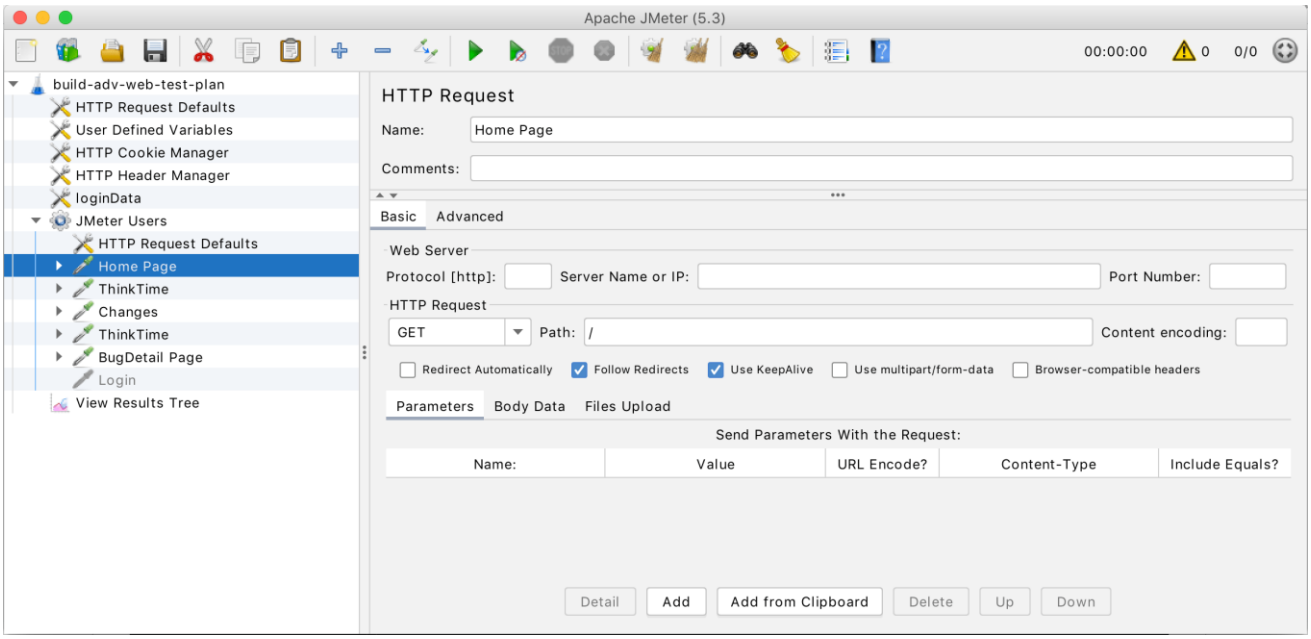


Рисунок 2.6 – Приклад використання Apache JMeter

Ці інструменти допомагають аналізувати та оптимізувати продуктивність своїх веб-додатків для забезпечення кращого користувацького досвіду та високої швидкодії завантаження сторінок.

2.2 Висновок

Завдяки інструментам аналізу є впевненість в тому, що веб-додаток працює з оптимальною продуктивністю та надає швидкий та зручний користувацький досвід. Вони допомагають виявити проблеми, зробити налагодження та зміни, необхідні для покращення продуктивності та ефективності веб-додатку.

Отже, враховуючи ці фактори, k6 може бути ефективним інструментом для тестування запитів у комерційному веб-додатку. Він надає зручність використання, гнучкість налаштування тестів, аналіз результатів та можливості розширення. Він допомагає ідентифікувати проблеми, аналізувати результати тестів та приймати обґрунтовані рішення щодо оптимізації додатку.

РОЗДІЛ 3. РОЗРОБКА ОПТИМІЗОВАННОГО ВЕБ-ДОДАТКУ

Розробка веб-додатка включає дві основні складові частини: розробку клієнтської та серверної частин. Кожна з цих складових має свої унікальні вимоги та завдання, які потрібні для успішної реалізації проекту.

Клієнтська частина веб-додатка відповідає взаємодії з користувачем та відображенню інформації на його пристроях. Для розробки клієнтської частини використовують технології, такі як HTML, CSS та мову програмування JavaScript, а також сучасні фреймворки та бібліотеки, які спрощують роботу зі створенням інтерактивного інтерфейсу.

Серверна частина веб-додатка відповідає за обробку запитів користувачів, збереження та отримання даних із баз даних, а також за логіку бізнес-процесів додатку.

При розробці оптимізованої веб-додатки буде використаний фреймворк Next.js, який є одним із популярних і потужних фреймворків для розробки веб-додаток на базі React. Одна з головних переваг Next.js виявляється в тому, що він працює як на клієнтській, так і на серверній частині, що його ідеальним вибором робить для оптимізації додатків[6].

Основні переваги використання фреймворка Next.js в оптимізації веб-додатків:

1. Відтворення на стороні сервера (SSR): Next.js дозволяє відмовитися від відтворення компонентів на стороні сервера перед тим, як вони будуть надіслані на клієнта. Це дозволяє швидко відображати вміст і покращує індексацію додатків пошуковими системами. Крім того, це зменшує час завантаження сторінок і покращує перший огляд користувача.
2. Автоматичне розбиття коду (Automatic Code Splitting): Next.js автоматично розбиває ваш код на невеликі фрагменти та динамічно їх завантажує за потреби. Це дозволяє зменшити початковий обсяг завантажуваного коду та прискорити сторінку завантаження.
3. Передзавантаження (попереднє завантаження): Next.js вміє передбачити, яку сторінку користувач може відвідати далі, і раніше завантажує необхідні

дані та ресурси. Це дозволяє забезпечити більш плавну навігацію між сторінками та пошкодити час очікування користувача.

4. Гнучкість і простота використання: Next.js пропонує простий API і широкі можливості для розширення функціональності. Він дозволяє легко налаштувати стан маршрутизації, роботу зі статичними файлами, збереження, підтримку стилів і багато іншого.
5. Швидкість розробки: Завдяки вбудованій підтримці гарячої заміни модулів (Hot Module Replacement), Next.js дозволяє нам швидко бачити результати своєї роботи без перезавантаження сервера або перебудови проекту. Це завершить прискорити розробку та покращити продуктивність команд.
6. Статичне рендерінг та генерація на стороні сервера (SSG): Next.js підтримує можливість статичного рендерінгу, коли сторінки генеруються на стороні сервера під час збірки проекту. Це дозволяє знизити навантаження на сервер та покращити швидкість завантаження сторінок. Крім того, Next.js може автоматично оновлювати статичні сторінки при зміні даних, що робить їх динамічними без необхідності виконувати запити до сервера під час перегляду.
7. Оптимізація зображень: Next.js має вбудовану підтримку оптимізації зображень. Він автоматично стискає та оптимізує зображення, що допомагає знизити їх розмір та покращити швидкість завантаження сторінок.

Серед тематик розробки був вибрана тематика комерційного веб-додатку. Вибір комерційного веб-додатку як об'єкта розробки та оптимізації має кілька обґрунтованих причин. Ось декілька з них:

1. Підвищення конверсій: Комерційні веб-додатки мають за мету перетворити відвідувачів на покупців. Швидкість завантаження сторінок і зручність використання можуть масово вплинути на конвертації. Оптимізація може збільшити конверсію через покращення користувацького досвіду та зменшення часу, необхідного для здійснення покупки.

2. Великі обсяги даних: Комерційні веб-додатки часто працюють і містять великий обсяг даних, таких як каталоги товарів, інформація про склад, історія покупок тощо. Ефективна робота з цими даними, їх швидкий доступ та обробка є промисловими аспектами комерційних добавок, які можна оптимізувати.
3. SEO і пошуковий рейтинг: Комерційні веб-додатки часто прагнуть досягти високого рейтингу в пошукових системах, щоб отримати більше органічного трафіку. Оптимізація сторінок для SEO, швидкість завантаження та інші фактори можуть вплинути на позиції в результатах пошуку.

Вибір комерційного веб-додатку дозволяє розкрити найбільш можливих методів оптимізації, після чого він вимагає уваги до швидкості, продуктивності, безпеки, користувальницького досвіду та інших факторів, які мають вирішальне значення. Це дає можливість впроваджувати та вдосконалювати різноманітні методи оптимізації за допомогою створення найефективніших та конкурентоспроможних комерційних веб-додатків.

Загалом, використання фреймворка Next.js у розробці комерційного веб-додатку дозволяє отримати швидкий, ефективний та оптимізований додаток, який забезпечує відмінний користувацький досвід та високу продуктивність.

3.1. Проектування архітектури комерційного веб-додатку

Next.js надає гнучкість та можливості для ефективного розподілу функціональності між клієнтською та серверною частинами додатку. Ось деякі ключові аспекти проектування архітектури з використанням Next.js:

1. Клієнтська архітектура: У клієнтській частині Next.js використовується реактивний підхід до розробки із застосуванням компонентів React. Компоненти відповідають за рендеринг і взаємодію з користувачем на клієнтській стороні. При проектуванні клієнтської архітектури ми маємо можливість використовувати розширення Next.js, таке як маршрутизація,

стилізація, форми та інші, щоб спростити розробку та покращити користувацький досвід.

2. Серверна архітектура: Next.js надає можливість виконувати серверний рендеринг, що дозволяє генерувати сторінки на сервері перед їх відправкою клієнтам. Цей підхід дозволяє відображати динамічний контент на сторінках і покращує SEO. Ми можемо використовувати серверний рендеринг для оптимізації сторінок завантаження, підтримки кешування та отримання більш швидкого перегляду для користувачів.
3. Взаємодія між клієнтом та сервером: Next.js надає зручний механізм для взаємодії між клієнтом і сервером. Ми можемо використовувати API-маршрути для створення власних серверних кінцевих точок та обробки запитів із клієнтської сторони. Це дозволяє передавати та отримувати дані між клієнтом і сервером разом із використанням Next.js.
4. Простота розгортання: Next.js надає зручні інструменти для розгортання веб-додатків. Ми можемо використовувати функціональне, таке як статичне експортування (static exporting), SSR (server-side rendering) або рендеринг на клієнті, для отримання готового до розгортання коду. Це спрощує процес розгортання та публікації додатку.

Усі ці аспекти дозволяють нам проектувати ефективну та оптимізовану клієнтську та серверну архітектуру з використанням фреймворка Next.js. Завдяки його функціональним можливостям, ми можемо побудувати потужні, швидкі та масштабовані веб-додатки з багатьма методами оптимізації для досягнення високої продуктивності та задоволення користувачів.

За вказаною посиланням у Додатку Б пункт 2 знаходиться репозиторій веб-додаток під назвою "shop" на платформі GitHub та викладений на домен вказанному у Додатку Б пункт 3.

3.2.1. Розробка серверної частини

Серверна частина була розроблена з використанням середовища виконання коду Node.js, мови програмування TypeScript, ORM (Object-Relational Mapping) Prisma, бібліотеки TRPC, бібліотеки next-auth, бази даних PostgreSQL, а також хостинг-провайдера Vercel для розгортання та керування додатком і доменом, також окремо слід пояснити про автентифікацію і авторизацію.

JWT (JSON Web Token)[4] є відкритим стандартом (RFC 7519), який використовується для аутентифікації і авторизації в розподілених системах. Алгоритм авторизації за допомогою JWT токена включає такі кроки:

1. Користувач аутентифікується на сервері, з використанням ідентифікатора користувача і пароля.
2. Після успішної аутентифікації сервер генерує JWT токен.
3. JWT токен складається з трьох основних частин: заголовку (header), корисного навантаження (payload) і підпису (signature). Заголовок містить алгоритм шифрування і тип токена, наприклад: {"alg": "HS256", "typ": "JWT"}. Корисне навантаження містить інформацію про користувача, наприклад: {"user_id": "123", "username": "john_doe"}. Підпис генерується з заголовка, корисного навантаження і секретного ключа сервера.
4. Сервер підписує JWT токен, використовуючи секретний ключ, і повертає його користувачеві.
5. Користувач отримує JWT токен і зберігає його, наприклад, в локальному сховищі (наприклад, у веб-браузері) або передає його з кожним запитом до сервера у заголовку Authorization.
6. При кожному запиті до захищеного ресурсу користувач включає JWT токен у заголовок Authorization з префіксом Bearer, наприклад: Authorization: Bearer <JWT token>.
7. Сервер отримує запит і перевіряє JWT токен. Сервер розбирає JWT токен на заголовок, корисне навантаження і підпис. Після цього сервер перевіряє, чи підпис збігається зі секретним ключем. Якщо підпис вірний і токен не прострочений або підозрілий, сервер продовжує обробку запиту.

8. Якщо JWT токен валідний, сервер вважає користувача авторизованим і надає доступ до захищених ресурсів або виконує запитані дії від імені користувача.

Для публічних маршрутів формується SSR сторінка. Для усіх маршрутів маршрутів веб-додатку не існує SSG, так як кожна сторінка має відображати релевантні дані з бази даних.

Для розміщення веб-додатку на домені та його хостинг був використаний сервіс Vercel. Хостинг - це послуга, яка дозволяє розмістити веб-сайт або додаток у мережі Інтернет. Хостинг-провайдер надає серверні ресурси, необхідні для зберігання файлів та виконання програмного забезпечення вашого веб-додатку.

Vercel - це хмарна платформа для розгортання та хостингу веб-додатків, яка є популярним вибором для розробки за допомогою Next.js. Ось деякі причини, чому Vercel підходить до розробки з використанням Next.js:

1. Легкість використання: Vercel надає простий та інтуїтивно зрозумілий інтерфейс для розгортання та керування веб-додатками. Ви можете легко налаштувати свій проект Next.js та розгорнути його з декількома клацаннями. Це особливо корисно для команд розробників, які швидко хочуть відобразити свої зміни виробництво.
2. Нативна підтримка Next.js: Vercel підтримує Next.js. Він надає оптимізований сервер рендеринг та автоматичну підтримку статичного рендерингу. Це дозволяє забезпечити високу продуктивність та ефективну доставку контенту користувачам.
3. Висока продуктивність та масштабованість: Vercel працює на глобальній мережі доставки контенту (CDN), що дозволяє швидку доставку веб-додатку користувачам з будь-якої точки світу. Вона також автоматично масштабується для виконання зростаючих навантажень, що робить її ідеальним вибором для високонавантажених додатків.
4. Інтеграція з Git та CI/CD: Vercel глибоко інтегрований з Git та забезпечує безпечні та автоматичні процеси розгортання через CI/CD (Continuous

Integration/Continuous Deployment). Ви можете легко налаштувати автоматичне розгортання з кожним зміненням коду в вашому репозиторії Git, що спрощує роботу в команді та забезпечує безпечне впровадження змін.

В цілому, Vercel є потужною платформою для розгортання та хостингу веб-додатків Next.js, що забезпечує легкість використання, високу продуктивність та масштабованість, а також інтеграцію з Git та CI/CD. Вона допомагає ефективно виробляти та доставляти свої додатки на виробництво.

3.2.2. Розробка бази даних

Реляційні бази даних (РБД) та нереляційні бази даних (НРБД) є двома основними типами систем управління базами даних (СУБД) з різними підходами до зберігання та організації даних. У випадку створення високонавантаженого комерційного веб-додатку, реляційна база даних Postgres більше всього підходить, і ось чому:

Надійність та цілісність даних: Postgres є добре відомою своєю надійністю та здатністю до забезпечення цілісності даних. Він підтримує ACID-властивості (Atomicity, Consistency, Isolation, Durability), що забезпечують надійність операцій та відновлення після відмови.

Підтримка складних запитів: Postgres надає потужний та ефективний мову запитів SQL, яка дозволяє виконувати складні запити до бази даних. Вона підтримує групування, сортування, об'єднання таблиць та інші операції, що дозволяють ефективно вибирати та маніпулювати даними.

Масштабованість: Postgres має добре розроблену архітектуру, що дозволяє масштабувати базу даних як вертикально (шляхом збільшення обсягу обробки на одному сервері), так і горизонтально (шляхом розподілення бази даних на кілька серверів). Це дозволяє високонавантаженим комерційним веб-додаткам ефективно масштабуватись з ростом обсягу даних та навантаження.

Розширені можливості: Postgres має широкий набір додаткових функцій та розширень, які дозволяють розширити його функціональність для конкретних потреб. Це включає в себе геопросторові розширення, розширення для роботи з JSON та інші типи даних, можливості повнотекстового пошуку та багато іншого.

Активна спільнота та підтримка: Postgres має велику та активну спільноту користувачів та розробників, яка надає постійну підтримку, оновлення та виправлення помилок. Це забезпечує стабільність, безпеку та покращення продуктивності бази даних.

Враховуючи ці фактори, Postgres є прекрасним вибором для створення високонавантаженого комерційного веб-додатку. Він надійний, масштабований, має потужну мову запитів та широкий набір функціональних можливостей, що дозволяють ефективно управляти даними та забезпечувати високу продуктивність системи.

Для комерційного веб-додатку використання PostgreSQL має такі переваги:

1. Надійність та стабільність: PostgreSQL відомий своєю високою стабільністю та надійністю. Це особливо важливо для комерційних додатків, які мають обробляти значні обсяги даних та виконувати складні операції. PostgreSQL забезпечує захист від втрати даних, надійне відновлення після збоїв та механізми реплікації для забезпечення високої доступності.
2. Складні запити та аналітика: Комерційні веб-додатки часто потребують складних запитів та аналітичних операцій з даними. PostgreSQL має розширену функціональність SQL, що дозволяє виконувати різноманітні запити, включаючи групування, сортування, фільтрацію, об'єднання даних та агрегацію. Він також підтримує аналітичні функції, які дозволяють розраховувати статистику, тенденції та інші показники.
3. Розширюваність та гнучкість: PostgreSQL дозволяє розширити функціональність бази даних за допомогою користувацьких типів даних, функцій та операторів. Це дозволяє створювати власні розширення та

оптимізувати роботу з даними під конкретні потреби комерційного додатку. PostgreSQL також підтримує зовнішні розширення та інтеграцію з різноманітними інструментами, що дозволяє розширити можливості додатку.

4. Підтримка геоданих та пов'язаних додатків: Для комерційних веб-додатків, пов'язаних з геоданими, PostgreSQL має вбудовану підтримку геопросторових запитів та індексів. Це дозволяє зберігати, опрацьовувати та аналізувати географічні дані, такі як місцезнаходження користувачів, об'єкти на мапі, маршрути та інше.

PostgreSQL має багато переваг для комерційних веб-додатків, також є деякі мінуси, які можуть бути враховані, а саме:

1. Вимоги до ресурсів: PostgreSQL може вимагати більшої кількості ресурсів (пам'яті, процесорного часу) порівняно з деякими іншими СКБД. Це може бути особливо важливо для веб-додатків з великими обсягами даних або високим навантаженням. Вимоги до ресурсів можуть впливати на масштабованість та продуктивність додатку.
2. Складність налаштування: PostgreSQL має багато налаштувальних параметрів, що можуть бути складними для розуміння та налаштування. Оптимальна конфігурація бази даних може вимагати додаткових знань та досвіду. Неправильні налаштування можуть призвести до погіршення продуктивності або нестабільності додатку.
3. Обмеження масштабування: У порівнянні з деякими розподіленими базами даних, PostgreSQL має деякі обмеження масштабування, особливо горизонтального масштабування (широке масштабування за допомогою розподілення даних на кілька серверів). Це може бути важливим фактором для веб-додатків з високим трафіком або потребою у масштабуванні.

Незважаючи на ці компроміси, використання PostgreSQL для комерційного веб-додатку є виправданим.

Детальна інформація про базу даних зображена на рисунку 3.1, а схема бази даних зображена на рисунку 3.2

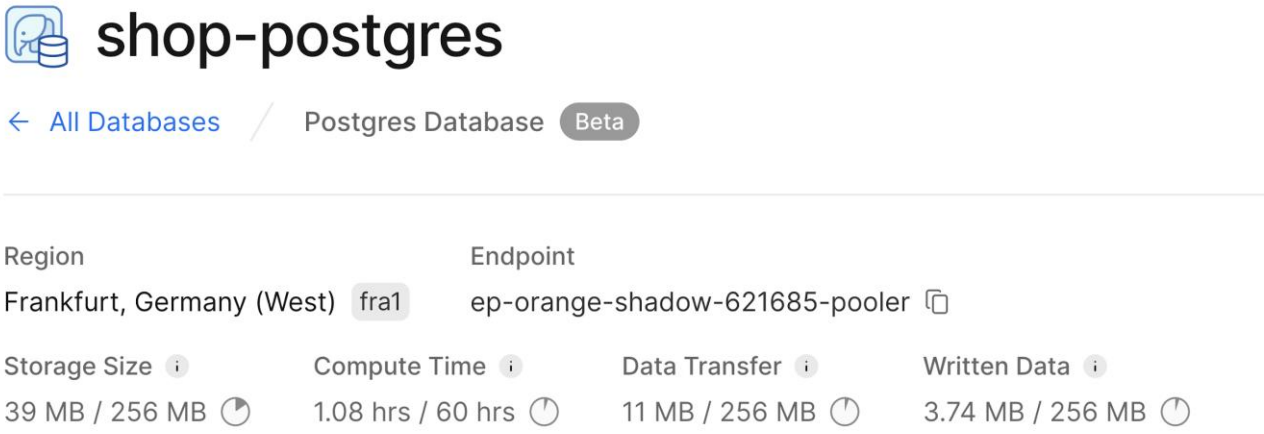


Рисунок 3.1 – Інформація про базу даних PostgreSQL

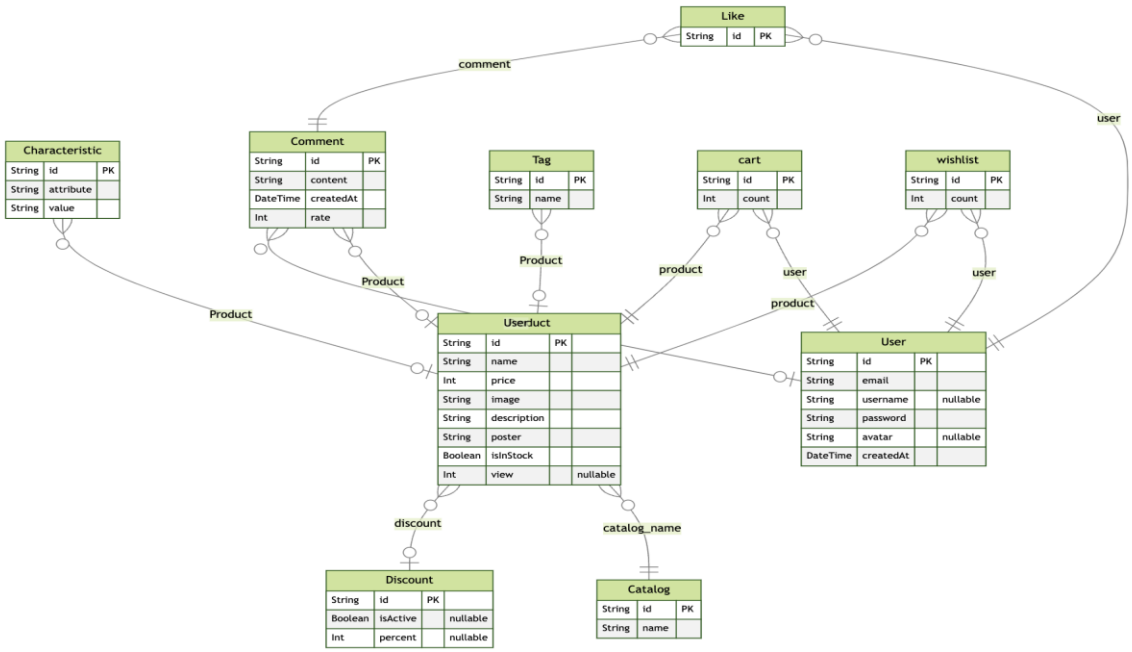


Рисунок 3.2 – Схема бази даних комерційного веб-застосунку

У базі даних оптимізованого веб-додатку можуть бути окремі таблиці для різних сутностей і функцій. Давайте розглянемо кожну таблицю окремо:

1. Таблиця "Discount": ця таблиця містить дані про знижки, які застосовуються до продуктів. Кожен запис в таблиці містить поля, такі як ідентифікатор знижки, назва, опис, відсоток знижки або конкретна цінова

пропозиція. Це дозволить ефективно управляти знижками і використовувати їх у веб-додатку.

2. Таблиця "Catalog": у цій таблиці збережені дані про категорії і розділи продуктів. Кожен запис містить поля, такі як ідентифікатор категорії, назва, опис та інші відповідні атрибути. Це дозволить організувати продукти у веб-додатку за категоріями або розділами, спрощуючи навігацію користувачів.
3. Таблиця "User": у цій таблиці збережені дані про користувачів, які реєструються у веб-додатку. Кожен запис містить поля, такі як ідентифікатор користувача, ім'я, електронна пошта, пароль, адреса доставки та інші відповідні атрибути. Це дозволяє зберігати інформацію про користувачів і використовувати її для авторизації, доставки та інших функцій.
4. Таблиця "Product": у цій таблиці збережені дані про продукти, які пропонуються у веб-додатку. Кожен запис містить поля, такі як ідентифікатор продукту, назва, опис, ціна, зображення, зовнішні ключ до коментарів.
5. Таблиця "Cart": ця таблиця може включати дані про кошики користувачів, в яких вони зберігають продукти перед покупкою. Кожен запис містить поля, такі як ідентифікатор користувача, ідентифікатор продукту, кількість одиниць товару та інші відповідні атрибути. Це дозволить відстежувати вибрані продукти користувачів і обробляти їхні замовлення.
6. Таблиця "Wishlist": у цій таблиці збережені дані про списки бажань користувачів, де вони зберігають продукти, які їм подобаються, але не додані до кошика. Кожен запис містить поля, такі як ідентифікатор користувача, ідентифікатор продукту та інші відповідні атрибути. Це дозволить користувачам зберігати вибрані продукти для майбутньої покупки.
7. Таблиця "Like": ця таблиця містить дані про лайки, які користувачі надають конкретним продуктам або коментарям. Кожен запис містить

поля, такі як ідентифікатор користувача, ідентифікатор об'єкта (продукту або коментаря) та інші відповідні атрибути. Це дозволить відстежувати популярність продуктів та коментарів, а також розраховувати рейтинг користувачів.

8. Таблиця "Characteristic": у цій таблиці збережені характеристики продуктів, такі як розмір, кольори, матеріали тощо. Кожен запис містить поля, такі як ідентифікатор продукту, назва характеристики, значення та інші відповідні атрибути. Це дозволить зберігати детальну інформацію про характеристики продуктів та допоможе користувачам здійснювати точні пошуки і фільтрування.

Ці таблиці у базі даних веб-додатку допоможуть зберігати та організовувати дані про знижки, каталоги продуктів, користувачів, продукти, кошики, списки бажань, лайки та характеристики. Взаємозв'язки між цими таблицями дозволять вам ефективно управляти функціями додатку і забезпечити зручний досвід користувачів.

Зв’язки:

1. Один до одного: немає;

2. Один до багатьох: Продукт до коментарів, характеристик, тегів; Категорії до продукту; Користувач до коментарів; Скидки до продукту;

3. Багато до багатьох: Коментарі до користувача (це таблиця лайків), Користувач і коментар (це таблиця коментарів), продукту до користувача (це таблиця кошику та списку побажань)

3.2.3 ORM

Також в веб-додатку використовується ORM, використання ORM забезпечує багато переваг. Воно спрощує процес розробки, зменшує кількість написаного коду та ризик помилок. ORM автоматично генерує SQL-запити на основі об'єктів із бази даних, що зменшує необхідність вручну писати складні запити. Крім того, ORM дозволяє здійснювати мапінг об'єктів на таблиці бази

даних, що полегшує роботу зі структурованою інформацією та забезпечує більш простий доступ до даних.

ORM (Object-Relational Mapping) - це підхід до взаємодії з базами даних, що дозволяє розробникам працювати з базами даних, використовуючи об'єктно-орієнтовані концепції. Prisma - це одна з популярних ORM-бібліотек, яка надає розробникам зручні інструменти для взаємодії з базами даних.

Ось деякі плюси використання Prisma ORM у веб-додатках:

1. Простота використання: Prisma надає простий та інтуїтивно зрозумілий API для взаємодії з базами даних. Вона автоматично генерує типизовані моделі даних на основі схеми бази даних, що спрощує роботу з даними та запитами до бази даних.
2. Висока продуктивність: Prisma пропонує оптимізовані запити до баз даних, які дозволяють ефективно отримувати, оновлювати та видаляти дані. Вона використовує механізми кешування та попередньої загрузки, що покращує продуктивність додатка та зменшує навантаження на базу даних.
3. Міграції бази даних: Prisma дозволяє легко виконувати міграції бази даних. Вона автоматично стежить за змінами в схемі бази даних та надає інструменти для генерації та застосування міграційних скриптів. Це дозволяє зручно керувати структурою бази даних під час розвитку додатку.
4. Безпека даних: Prisma надає механізми для захисту даних від SQL-ін'єкцій та інших вразливостей. Вона автоматично використовує параметризовані запити, що допомагає запобігти потенційним атакам на базу даних.

У високонавантажених веб-додатках Prisma використовується так як:

1. Ефективність: Завдяки оптимізованим запитам та кешуванню, Prisma допомагає знизити навантаження на базу даних та покращити продуктивність додатка. Вона дозволяє ефективно працювати з великими обсягами даних та високим навантаженням.
2. Масштабованість: Prisma дозволяє розподіляти базу даних та запити на кілька серверів, що сприяє горизонтальній масштабованості додатка. Це

особливо важливо в високонавантажених сценаріях, коли необхідно обробляти багато запитів одночасно.

3. Розробнича продуктивність: Prisma спрощує розробку та обслуговування бази даних. Вона надає розробникам інтуїтивний інтерфейс та інструменти для швидкого розгортання та зміни схеми бази даних. Це допомагає прискорити процес розробки та забезпечити більшу продуктивність команди розробників.
4. Типізація та статичний аналіз: Prisma генерує типизовані моделі даних, що допомагає виявляти помилки на етапі компіляції та полегшує роботу з даними. Це особливо корисно в великих проектах, де важлива надійність та безпека даних.

Тому, Prisma є виправданим інструментом, який допомагає забезпечити ефективну взаємодію з базою даних у веб-додатках, особливо в високонавантажених сценаріях. Вона спрощує розробку, покращує продуктивність та допомагає забезпечити безпеку та масштабованість додатка. Описання схеми для створення таблиць у базі даних описані у додатку А. Після створення таблиці, за допомогою бібліотеки faker.js ми можемо її заповнити даними.

Сервер та база даних знаходиться в одному регіоні країни Германії, а саме у Франкфурті. Це є вигідним для споживачів з Європи, включаючи Україну, тому що:

1. Близькість до місця розташування користувачів: Розміщення хостингу у Германії забезпечує фізичну близькість сервера до користувачів в Європі та Україні. Це може позитивно вплинути на швидкість завантаження сторінок та відповідей з сервера, оскільки час передачі даних скорочується.
2. Надійність та швидкість мережі: Германія має добре розвинену інфраструктуру мережі, що забезпечує стабільну та швидку з'єднаність з інтернетом. Це дозволяє забезпечити надійність та продуктивність веб-

додатку для користувачів, особливо тих, що знаходяться у Європі та Україні.

3. Мовні та культурні спільноти: Розміщення веб-додатку у Германії може сприяти легшій спілкуванню з мовними та культурними спільнотами Європи та України. Це може бути корисним для підтримки, комунікації з користувачами та взаєморозуміння.

Загалом, розміщення хостингу та бази даних PostgreSQL у Германії може мати позитивний вплив на продуктивність, надійність та виконання вимог щодо захисту даних для користувачів з Європи та України.

3.2.4 Організація запитів веб-додатків

Підходи до організації запитів веб-додатків включають різні методики та архітектурні стилі, які визначають спосіб взаємодії між клієнтами та серверами, наприклад REST або RPC.

REST (Representational State Transfer) та RPC (Remote Procedure Call) є двома розповсюдженими підходами до організації запитів у веб-додатках. Обидва підходи дозволяють клієнтам взаємодіяти з сервером, але вони мають різні концепції та використовуються для різних цілей. REST є архітектурним стилем, який базується на принципах Інтернету і використовує HTTP протокол для передачі даних. REST розглядає ресурси (наприклад, об'єкти бази даних) як основну абстракцію та надає їм уніфікований набір операцій, таких як отримання (GET), створення (POST), оновлення (PUT/PATCH) та видалення (DELETE). REST запити виконуються за допомогою URL-шляхів та HTTP методів, а дані передаються у форматі, який зазвичай є JSON або XML. REST підходить для створення публічних API та веб-сервісів, які мають простий та масштабований інтерфейс.

RPC (віддалений виклик процедур) є парадигмою взаємодії між комп'ютерними системами, де одна система може викликати функції або методи на віддаленому сервері. У випадку RPC, клієнтська програма викликає метод на

сервері та отримує результат. RPC може використовувати різні протоколи для передачі даних, такі як SOAP, XML-RPC або JSON-RPC. Він акцентується на викликах методів та приховує деталі комунікації між клієнтом та сервером. RPC підходить для використання в розподілених системах та великих мережевих додатках.

У веб-додатку використовується tRPC, це є одним із підходів до організації запитів у веб-додатках. tRPC (tagged RPC) є високопродуктивним, типобезпечним та масштабованим рішенням для веб-розробки. Він поєднує переваги REST та RPC, дозволяючи використовувати типи даних та компіляцію на етапі розробки. Основна ідея tRPC полягає в тому, що описується API за допомогою типів даних, а потім генерувати клієнтський та серверний код на основі цих описів. Це дозволяє знизити кількість ручної роботи та уникнути потенційних помилок, пов'язаних з ручним введенням даних. tRPC використовує JSON як основний формат для передачі даних та HTTP/2 як протокол для комунікації між клієнтом та сервером. Він підтримує типи даних, валідацію, розгортання API та багато інших корисних функцій. Один із головних плюсів tRPC полягає в його ефективності та швидкості, оскільки він використовує HTTP/2 та може використовувати протоколи мультиплексування, які дозволяють обробляти багато запитів одночасно. Загалом, tRPC є потужним інструментом для розробки веб-додатків, який поєднує переваги REST та RPC, надаючи типобезпечну комунікацію між клієнтом і сервером та полегшуючи розробку та підтримку API.

3.3 Розробка клієнтської частини веб-додатку

Next.js дозволяє не тільки розробляти серверну частину додатка, але і створювати клієнтську частину на основі React. Основна перевага полягає в тому, що Next.js поєднує можливості серверного рендерингу та статичного генерації з підтримкою React, що дозволяє створювати багатосторінкові додатки з швидким завантаженням сторінок та взаємодією на стороні клієнта.

Клієнтська частина веб-додатку складається з декількох сторінок, які взаємодіють з користувачем і надають різні функціональні можливості. Ось короткий опис кожної з цих сторінок:

1. Головна сторінка: Це сторінка, на яку користувач потрапляє при першому завантаженні веб-додатку. Головна сторінка зазвичай містить загальну інформацію про ваш додаток, акції, промоції або рекомендації. Інтерфейс сторінки зображений на рисунку 3.3

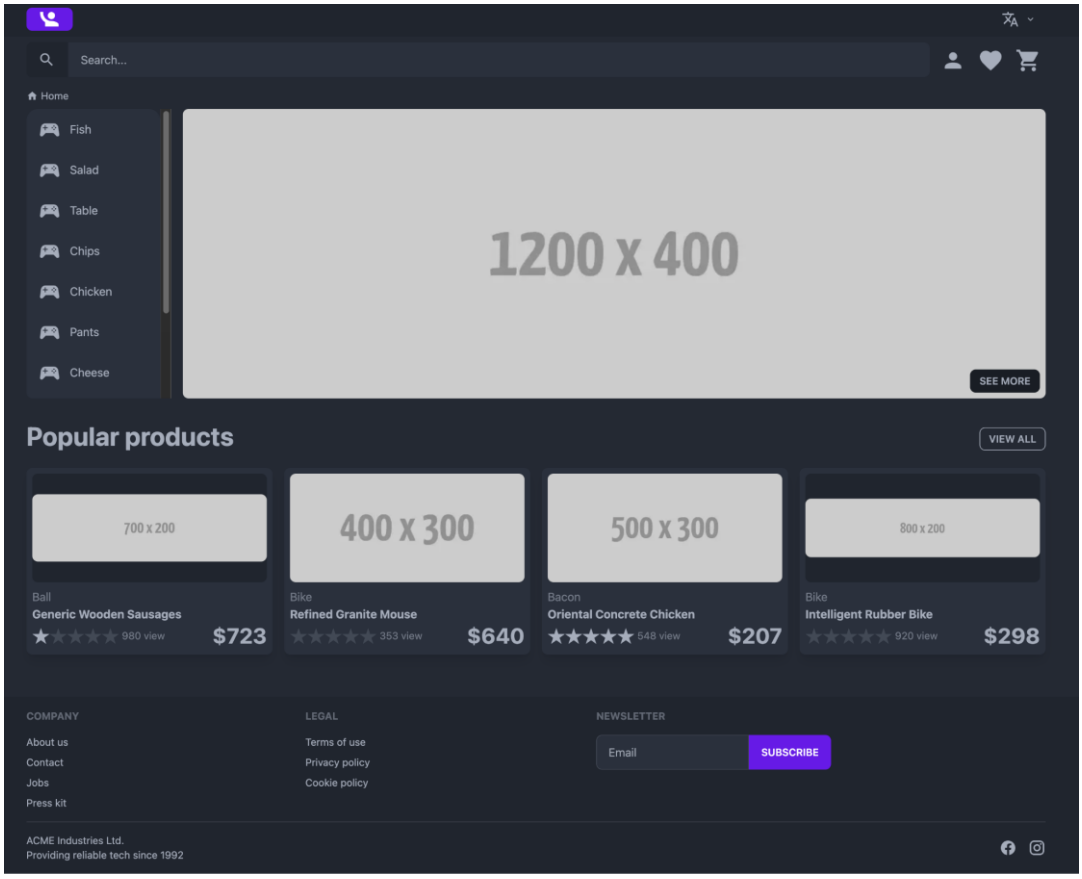


Рисунок 3.3 - Головна сторінка

2. Каталог: Сторінка каталогу містить список продуктів, які доступні для перегляду або покупки. Користувач може прокручувати список, фільтрувати продукти за категоріями. Каталог зазвичай відображає зображення, назву, ціну та короткий опис кожного продукту. Інтерфейс сторінки зображений на рисунку 3.4

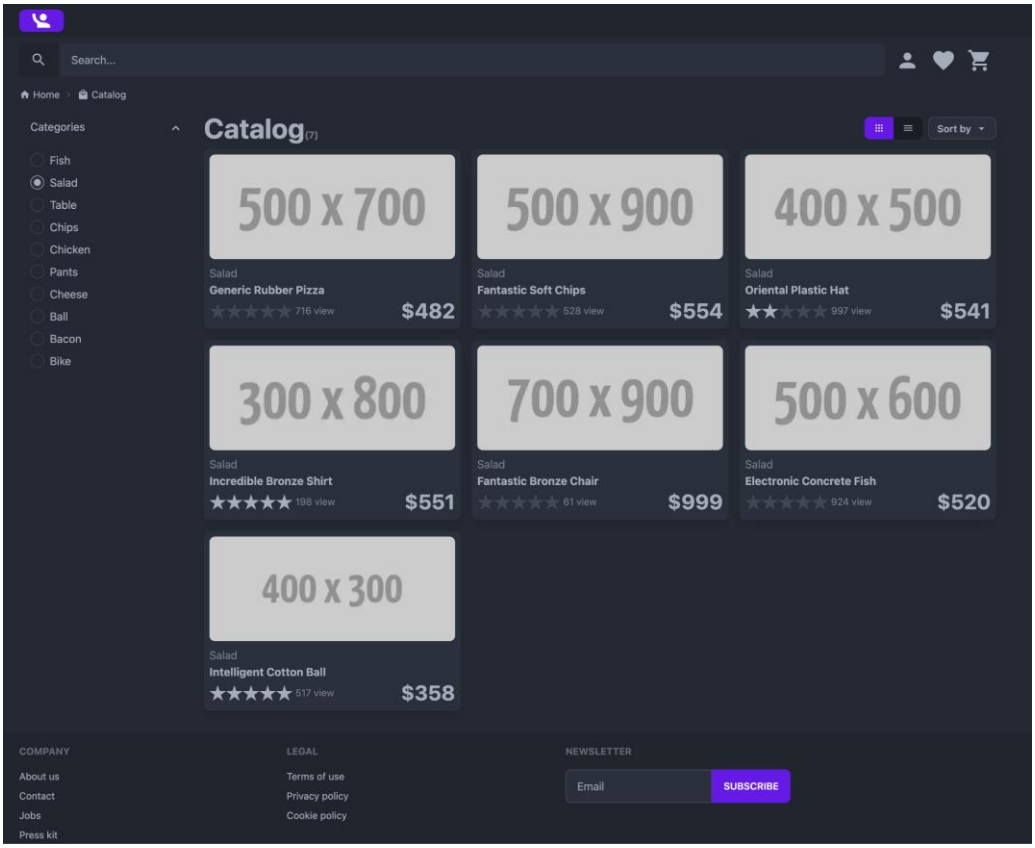


Рисунок 3.4 - Сторінка каталогу

3. Сторінка продукту: Кожний продукт має свою власну сторінку, де відображається більш детальна інформація про продукт. На цій сторінці можуть бути вказані детальний опис, технічні характеристики, відгуки користувачів, фотографії продукту. Користувач може додати продукт до свого кошика або списку бажань зі сторінки продукту. Інтерфейс сторінки зображений на рисунку 3.5

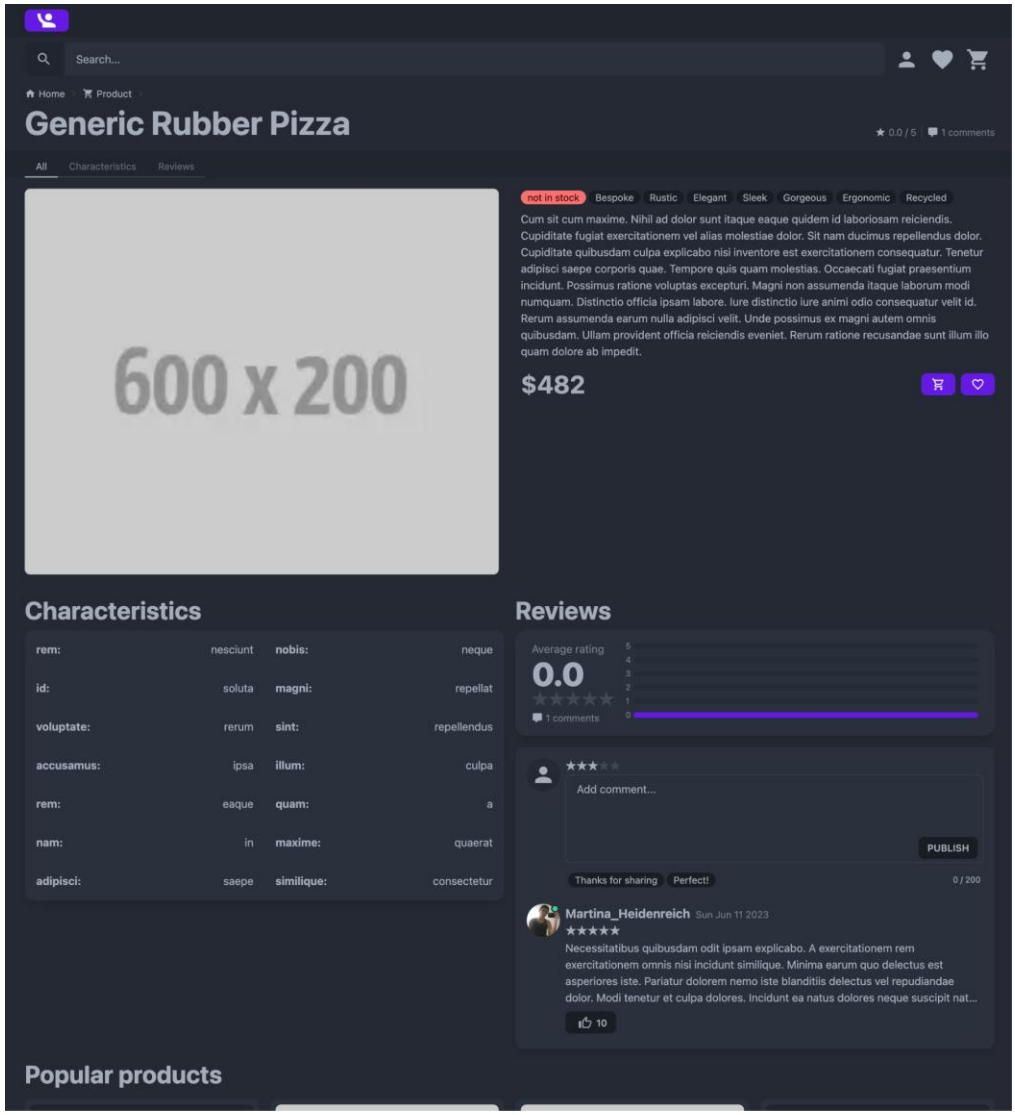


Рисунок 3.5 - Сторінка продукту

4. Кошик: Сторінка кошика відображає список продуктів, які користувач додав до свого кошика. Користувач може переглядати та редагувати кількість продуктів, видаляти їх з кошика або продовжувати оформлення замовлення. Кошик також може показувати підсумкову ціну та вартість доставки. Інтерфейс сторінки зображений на рисунку 3.6

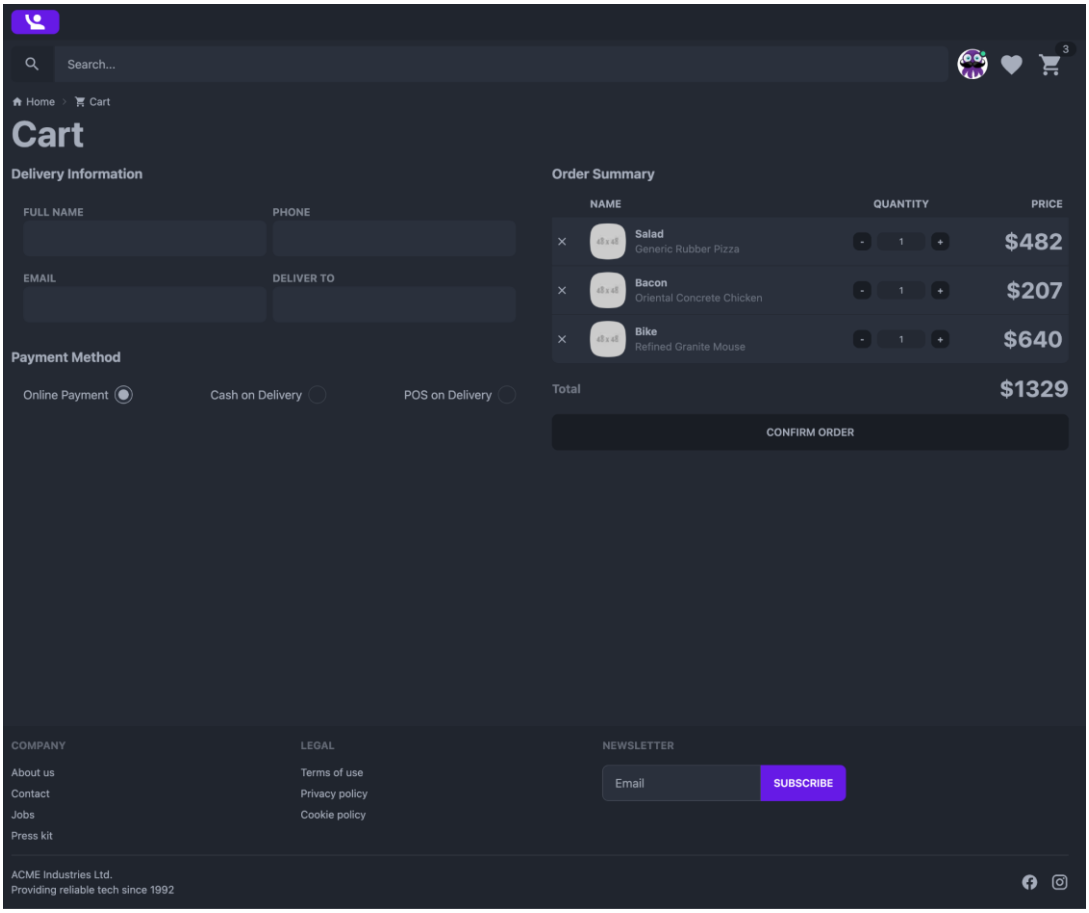


Рисунок 3.6 - Сторінка кошику (для авторизованих користувачів)

5. Список бажань: Ця сторінка містить список продуктів, які користувач додав до свого списку бажань. Вона дозволяє користувачеві зберігати продукти для подальшого перегляду або покупки. Користувач може видаляти продукти зі списку бажань. Інтерфейс сторінки зображений на рисунку 3.7

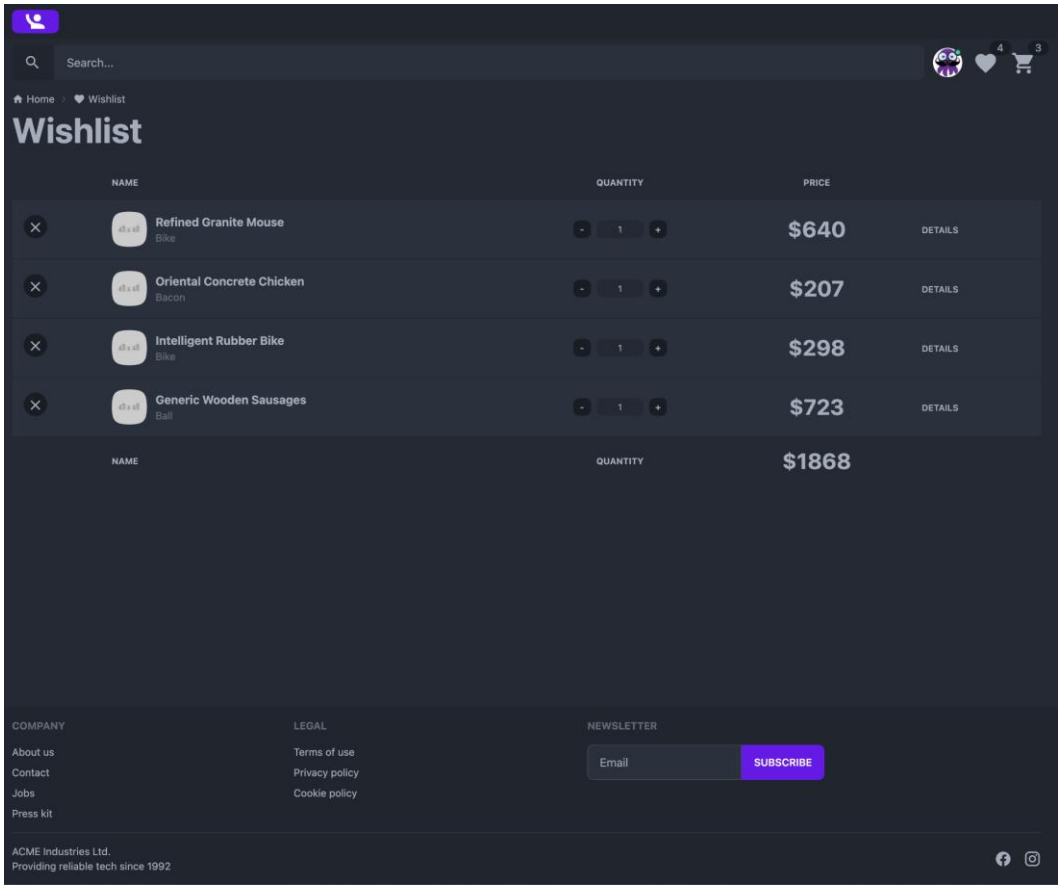


Рисунок 3.7 - Сторінка каталогу

Кожна з цих сторінок взаємодіє з сервером для отримання необхідних даних, таких як інформація про продукти, деталі замовлення або стан кошика. Вони також можуть використовувати різні функції, такі як, фільтрація або сортування, щоб полегшити навігацію та вибір продуктів для користувача.

3.4 Тестування та оцінка ефективності оптимізованого веб-додатку

Тестування веб-додатку, заснованого на Next.js, включає перевірку функціональних можливостей, що надає цей фреймворк. Основні аспекти тестування використаних можливостей в Next.js включають маршрутизацію, серверний рендерінг та статичне генерування. Нижче наведено опис кожного з цих аспектів тестування.

Маршрутизація: В Next.js маршрутизація здійснюється за допомогою файлів з префіксом "pages". Кожен файл в цій папці стає окремою сторінкою додатку. Під час тестування слід перевірити, чи правильно визначені маршрути та чи працює навігація між сторінками. У Next.js, папка "api" в "pages" використовується для створення API-шляхів або серверного роутингу. Це означає, що можно створювати власні API-маршрути відповідно до вашого додатку без необхідності налаштування окремого сервера. Файлова структура проекту зображена на рисунку 3.8

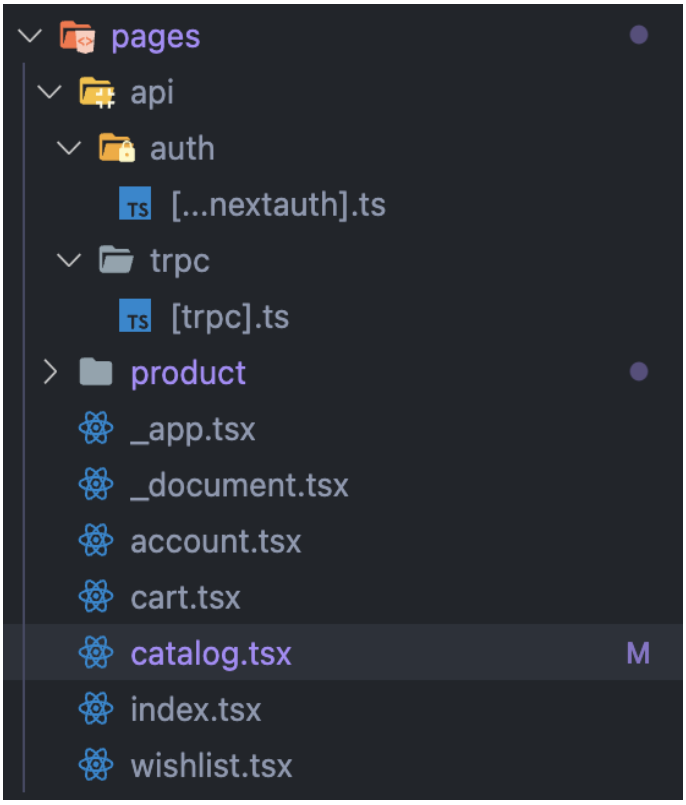


Рисунок 3.8 – Файлова структура проекту

У Next.js, файли з патерном [...nextauth].ts та [trpc].ts відносяться до певних бібліотек та шаблонів, які використовуються для створення функціональних частин веб-додатку:

[...nextauth].ts: файл з патерном [...nextauth].ts використовується разом з бібліотекою NextAuth.js. Цей файл використовується для налаштування та реалізації серверного API, яке обробляє аутентифікацію та авторизацію в додатку. В цьому файлі є налаштування провайдерів аутентифікації,

конфігурація різних опцій авторизації та визначаєте різних маршрутів для обробки різних етапів аутентифікації.

[trpc].ts: файл з патерном [trpc].ts використовується разом з бібліотекою TRPC (TypeScript RPC). Ця бібліотека дозволяє створювати типізовані API на основі RPC (Remote Procedure Call). У файлі [trpc].ts визначається TRPC маршрути, які відповідають за обробку різних запитів та відповідей сервера.

Ці файли є спеціальними шаблонами та конвенціями, які дозволяють побудувати функціональну аутентифікацію та API-логіку в Next.js додатку з використанням популярних бібліотек NextAuth.js та TRPC. Можна використовувати ці файли для налаштування та розширення функціональності вашого додатку з аутентифікацією та API.

Серверний рендерінг: Next.js підтримує серверний рендерінг, що дозволяє відрендерювати сторінку на сервері перед її відправкою до клієнта.

Статичне генерування: Next.js також підтримує статичне генерування, що дозволяє створювати статичні файли для покращення продуктивності та швидкодії додатку.

Тестування та оцінка ефективності оптимізованого веб-додатку є важливим етапом розробки, який дозволяє визначити швидкодію та якість роботи додатку перед його впровадженням. Одним із інструментів, який може бути використаний для цього, є Lighthouse.

3.5. Оцінка ефективності веб-застосунку

Lighthouse - це відкритий інструмент для аналізу продуктивності та якості веб-додатків. Він розроблений командою Chrome DevTools і надає засоби для оцінки швидкодії, доступності, SEO-оптимізації та інших аспектів веб-додатку. Основна функція Lighthouse - автоматичне проведення аудиту веб-сторінки з використанням набору правил та рекомендацій.

Audit	Weight
First Contentful Paint	10%
Speed Index	10%
Largest Contentful Paint	25%
Total Blocking Time	30%
Cumulative Layout Shift	25%

Рисунок 3.9 — Показник та його вага у розрахунку ефективності

Оцінка ефективності у Lighthouse є середньозваженим показником показників, більш зважені показники мають більший вплив на загальний показник ефективності. Оцінки показників не відображаються у звіті, але обчислюються за формулою яка зображена на рисунку 3.9.

У контексті оптимізованого веб-додатку, Lighthouse був використаний для оцінки різних аспектів додатку, що впливають на продуктивність та користувацький досвід.

Основні метрики продуктивності сторінки, які можна виміряти, включають:

1. First Contentful Paint (FCP): Це час, який потрібен для відображення першого контенту на сторінці. Це може бути текст, зображення чи інші елементи. На рисунку 3.10 показано, як значення показника інтерпретується для завантаження.

Рисунок 3.10 - Інтерпретування показника FCP до часу

2. Cumulative Layout Shift (CLS): Це метрика, яка вимірює, наскільки нестабільно розташовані елементи на сторінці під час завантаження. Вона оцінює, наскільки сильно змінюється макет сторінки під час завантаження, що може призвести до неприємного досвіду користувача, наприклад, коли він намагається натиснути на елемент, але він раптово зсувається. На рисунку 3.11 показано, як значення показника інтерпретується для завантаження.

Рисунок 3.11 - Інтерпретування показника FCP до часу

3. Largest Contentful Paint (LCP): Це час, який потрібна для відображення найбільшого елемента контенту на сторінці. Це може бути велике зображення, відео або інший важливий елемент, який привертає увагу користувача. На рисунку 3.12 показано, як значення показника інтерпретується для завантаження.

Рисунок 3.12 - Інтерпретування показника LCP до часу

4. Total Blocking Time (TBT): Це час, коли сторінка блокується та недоступна для взаємодії з користувачем через виконання JavaScript-коду. На рисунку 3.13 показано, як значення показника інтерпретується для завантаження.

Рисунок 3.13 – Інтерпретування показника TBT до часу

5. Speed Index: Це час, яка показує, як швидко контент відображається візуально під час завантаження сторінки. Спочатку Lighthouse знімає відео завантаження сторінки у браузері та обчислює візуальний перехід між кадрами, потім використовує модуль Speedline Node.js для розрахунку індексу швидкості. На рисунку 3.14 показано, як значення показника інтерпретується для завантаження.

Speed Index (in seconds)	Color-coding
0–3.4	Green (fast)
3.4–5.8	Orange (moderate)
Over 5.8	Red (slow)

Рисунок 3.14 – Інтерпретування показника Speed Index до часу

3.6 Висновок

Врахування цих метрик оптимізації допомагає здійснити об'єктивну оцінку ефективності веб-додатку. Порівняння цих метрик до заданих цілей та стандартів допоможе виявити проблеми та здійснити оптимізацію для поліпшення продуктивності та користувацького досвіду.

РОЗДІЛ 4. ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕНЬ

4.1 Опис експериментального дослідження

Експериментально дослідження оптимізації веб-додатка може включати кілька етапів, а основний акцент буде зроблено на використанні інструменту Lighthouse для локального тестування та проведення тестів на різних регіонах для хостингу на домені, де знаходиться наш веб-додаток, а також стрес-тесту за допомогою інструменті K6. Результати тестування в різних регіонах та тільки на головній сторінки знаходяться на відповідно рисунку 4.1 та рисунку 4.2.

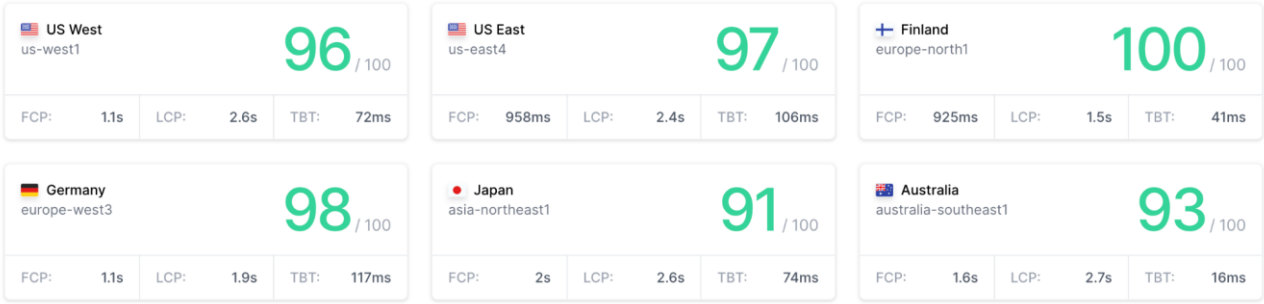


Рисунок 4.1 – Результати тестування Lighthouse у різних регіонах головної сторінки [3]

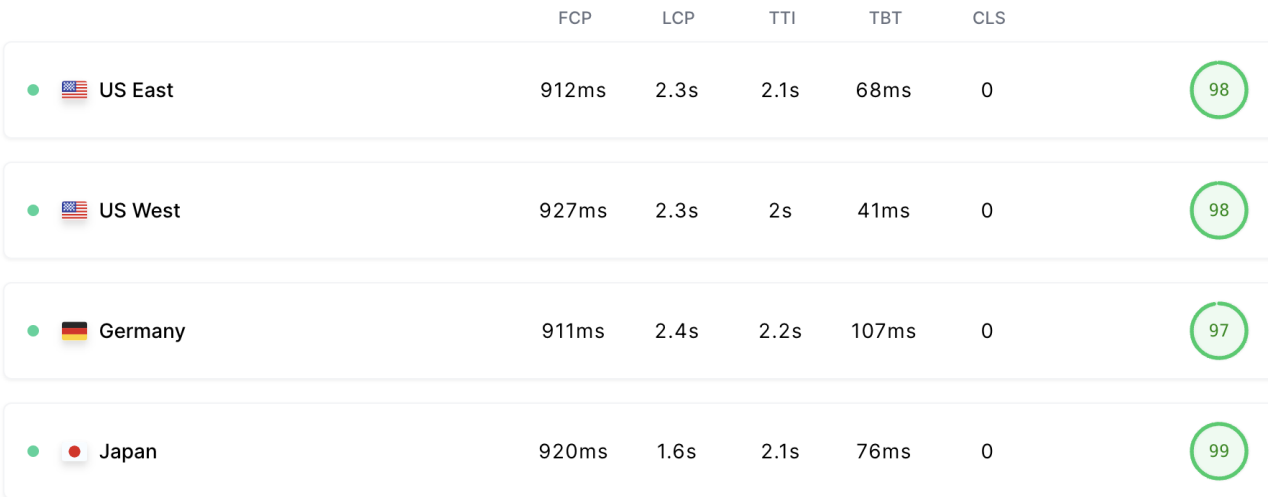


Рисунок 4.2 – Результати тестування Lighthouse у різних регіонах головної сторінки [3]

Тестування продуктивності хостингу для різних регіонів є дуже важливим аспектом при розгляді глобального впливу вашого веб-додатка на користувачів з усього світу:

1. Глобальна доступність: Користувачі з різних країн та регіонів світу мають різний доступ до інтернету та різний рівень з'єднання. Тестування продуктивності для різних регіонів допомагає виявити можливі проблеми з швидкістю завантаження, які можуть виникати для користувачів з віддалених регіонів з менш стабільним з'єднанням.
2. Вплив географічного розташування: Сервери хостингу зазвичай мають свої місця розташування в різних регіонах. Тестування продуктивності з різних регіонів допомагає виявити, як далекі регіони можуть впливати на час відгуку сервера та швидкість завантаження сторінок. Це дозволяє прийняти рішення щодо вибору оптимального хостингу та налаштувань, щоб забезпечити швидку та надійну роботу вашого додатка для користувачів з усього світу.
3. Вдосконалення користувацького досвіду: Швидкість завантаження веб-сторінок має велике значення для користувачів. Дослідження продуктивності в різних регіонах допомагає виявити можливі затримки та оптимізувати ваш веб-додаток для покращення користувацького досвіду. Це дозволить забезпечити швидке завантаження сторінок та зменшити ймовірність втрати користувачів через довгий час очікування.

Тестування продуктивності для різних регіонів є важливою складовою процесу оптимізації веб-додатка. Воно допомагає забезпечити глобальну доступність, покращити користувацький досвід та визначити оптимальні налаштування хостингу для вашого додатка.

Для стрес-тесту був обраний інструмент K6 – це дуже потужний інструмент для проведення стрес-тестів та навантаження веб-додатків і API. Він розроблений для того, щоб допомагати розробникам та інженерам випробувати швидкодію та стабільність своїх систем в умовах великого навантаження.

Однією з переваг K6 є його простота використання. Він має простий синтаксис, що дозволяє швидко створювати та налаштовувати тести. K6 написаний на мові програмування Go, що робить його швидким та ефективним. Він також має вбудовану підтримку для JavaScript, що дозволяє використовувати власний код для створення складних сценаріїв тестування.

K6 має розширені можливості для симуляції реальних навантажень. Можна налаштувати велику кількість віртуальних користувачів, які будуть виконувати різні дії на вашому веб-додатку або API. Це дозволяє вам оцінити швидкодію та стабільність системи під час інтенсивного навантаження.

K6 надає багато корисних статистичних даних та метрик під час тестування, що допомагає аналізувати продуктивність системи. Ви можете переглядати кількість запитів, середні часи відповіді, пропускну здатність та багато іншого. Це дозволяє виявити слабкі місця та вирішити проблеми з продуктивністю.

Крім того, K6 має гнучкість і розширюваність завдяки своїй архітектурі. Ви можете використовувати багато різних бібліотек та плагінів, щоб розширити його функціональність. Крім того, ви можете легко інтегрувати K6 з іншими інструментами тестування або засобами автоматизації розробки.

Загалом, K6 є потужним інструментом для стрес-тестування, який допомагає забезпечити стабільність і проду ктивність веб-додатків і API. Його простий синтаксис, розширюваність та багатий набір функцій роблять його відмінним вибором для розробників та інженерів, які шукають ефективний спосіб випробування своїх систем. Код скрипу за допомогою формується запит зображений на рисунку 4.3, а на рисунку 4.4 зображена конфігурація скрипка з коментарями.

```
export default function () {
  const url = "https://shop-bachelor-thesis.vercel.app/";
  http.get(url);
  sleep(1);
}
```

Рисунок 4.3 – Код скрипту для тестування запитів

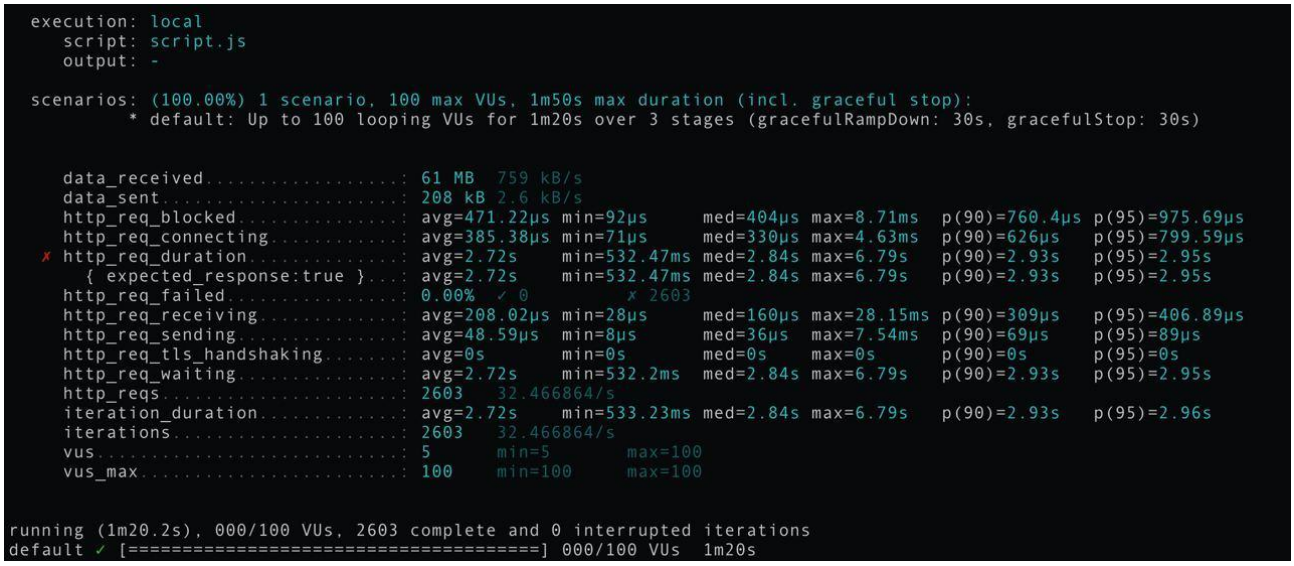
```
export const options = {
  stages: [
    { duration: "10s", target: 100 }, // Навантаження 100 віртуальних користувачів протягом 10 секунд
    { duration: "1m", target: 100 }, // Утримання навантаження 100 віртуальних користувачів протягом наступну хвилину
    { duration: "10s", target: 0 }, // Зниження навантаження до 0 протягом 10 секунд
  ],
  thresholds: {
    http_req_duration: ["p(95)<500"], // Установлення порогу, що 95% запитів мають тривалість менше 500 мс
  },
};
```

Рисунок 4.4 – Конфігурація скрипту для тестування запитів

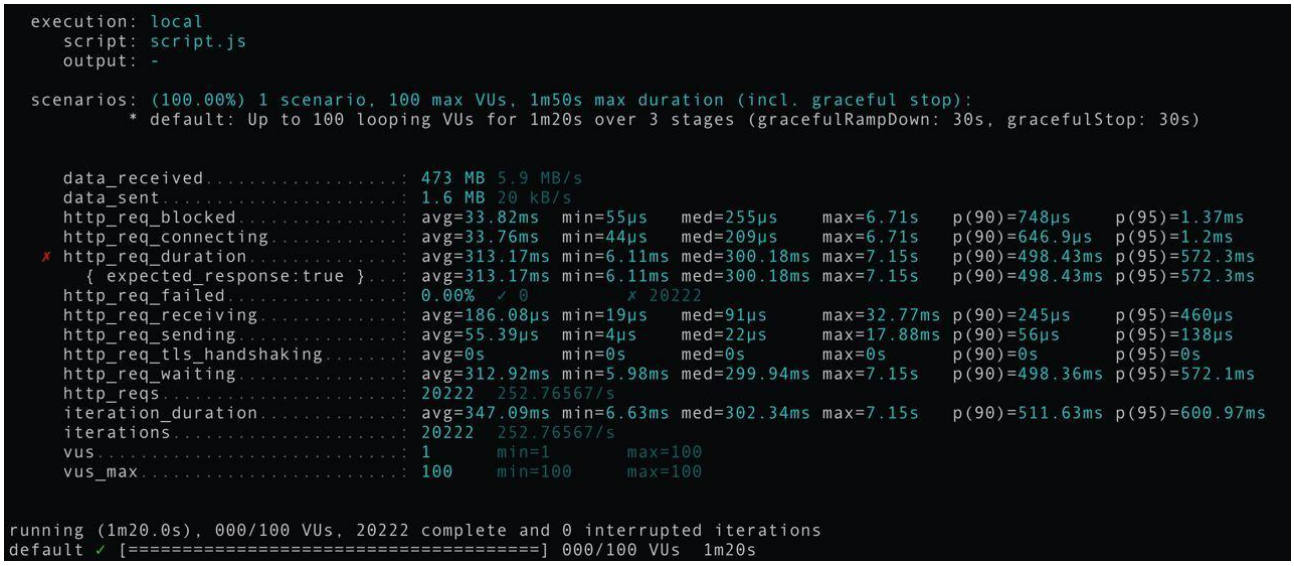
Використовується модуль http з K6 для виконання GET-запиту до URL-адреси веб-додатку.

options об'єкт містить конфігураційні параметри K6. У цьому прикладі ми визначаємо різні етапи (stages) для навантаження, збільшуючи його до 100 віртуальних користувачів на протязі 10 секунд, підтримуючи його на цьому рівні протягом наступну хвилину, а потім знижуючи до 0 протягом ще 10 секунд.

thresholds об'єкт дозволяє встановити порогові значення для метрик. У цьому прикладі ми використовуємо http_req_duration (тривалість запитів) і встановлюємо поріг, що 95% запитів мають тривалість менше 500 мс. Результати k6 веб-додатку до та після застосування методів оптимізації відповідно на рисунках 4.5 та рисунку 4.6.



4.5 – Вивід інформації інструменту K6 до запитів головної сторінки до оптимізації



4.6 – Вивід інформації інструменту K6 до запитів головної сторінки після оптимізації

Де:

1. http_reqs: Кількість виконаних HTTP-запитів. Це включає всі успішні та неуспішні запити.
2. http_req_duration: Тривалість кожного HTTP-запиту в мілісекундах. Ця метрика надає огляд про продуктивність запитів і може допомогти виявити повільні запити або проблеми з продуктивністю.

3. http_req_waiting: Час очікування перед початком відправлення запиту на сервер. Ця метрика включає час, який пройшов від моменту, коли запит було підготовлено до відправлення, до моменту, коли він було фактично відправлено.
4. http_req_connecting: Час, який займає встановлення з'єднання з сервером. Ця метрика включає час, який пройшов від моменту, коли було встановлено з'єднання, до моменту, коли запит було фактично відправлено.
5. http_req_receiving: Час отримання відповіді від сервера. Ця метрика включає час, який пройшов від моменту, коли відправлений запит до сервера, до моменту отримання повної відповіді.
6. http_req_failed: Кількість неуспішних HTTP-запитів. Це включає запити, які завершилися помилкою або з невдалою відповіддю.
7. http_reqs/sec: Кількість виконаних HTTP-запитів на секунду. Ця метрика показує пропускну здатність системи і може допомогти визначити, скільки запитів обробляється на одиницю часу.

Після проведення двох стрес-тестів до вашого Next.js веб-додатка - до оптимізації і після оптимізації, можна зробити такі висновки, а саме покращення продуктивності, зменшення завантаження часу, зменшення ресурсоємності та краща масштабованість.

4.2 Збір та аналіз даних

Vercel надає різноманітні інструменти для аналізу та збору інформації про ваші додатки, включаючи інструмент "vercel analyze". "vercel analyze" - це командний інтерфейс, наданий Vercel, який допомагає аналізувати продуктивність вашого додатка Next.js.

При виконанні команди "vercel analyze" Vercel збирає інформацію про завантаження, продуктивність та використання ресурсів вашого додатка. Він створює звіт, який надає дані про час завантаження сторінок, розмір файлів,

використання пам'яті, мережеві запити та інші метрики продуктивності. На Vercel можна переглянути аналіз запитів головної сторінки та більш детальний аналіз головної сторінки знаходяться на рисунках відповідно 4.7 та 4.8.

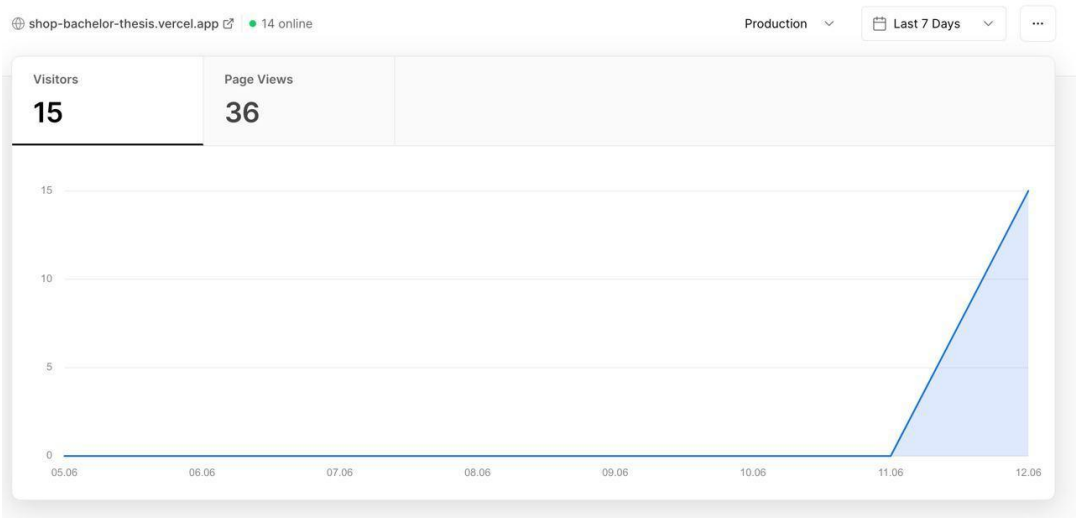


Рисунок 4.7 – Аналіз запитів головної сторінки за останні 7 днів

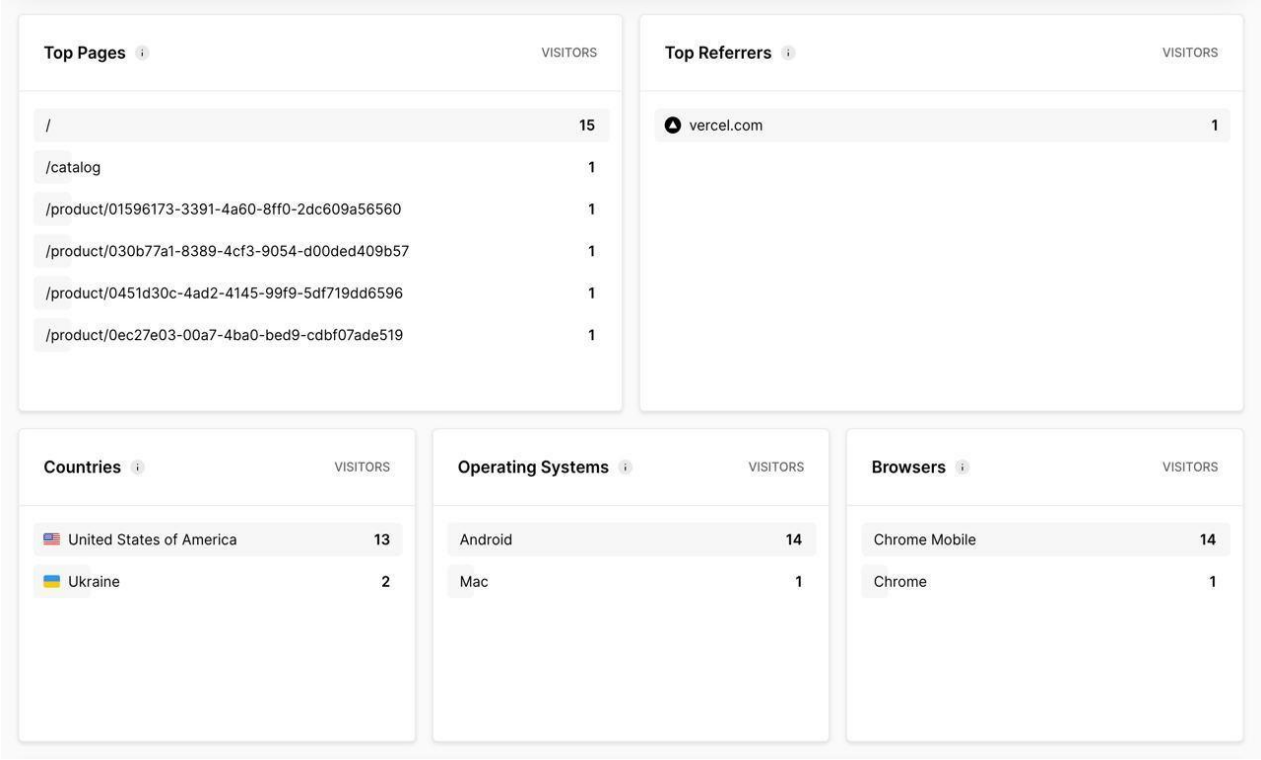


Рисунок 4.8 – Докладний аналіз головної сторінки

Аналіз отриманих даних допомагає виявити проблемні місця у додатку та оптимізувати його для швидкого завантаження та кращої продуктивності.

Наприклад, можна оптимізувати розмір файлів, покращити кешування, усунути проблеми з рендерингом та інші.

4.3. Переваги та недоліки використаних методів

Next.js та Vercel надають кілька оптимізаційних можливостей для поліпшення швидкодії та продуктивності веб-додатків, включаючи використання компонента Image та Content Delivery Network (CDN). Ось деякі з цих можливостей:

1. Компонент Image: Next.js постачає вбудований компонент <Image/>, який автоматично оптимізує та стискає зображення для досягнення кращої продуктивності. Цей компонент автоматично генерує різні розміри зображень для різних пристроїв та екранів, завантажуючи лише необхідний обсяг даних. Він також використовує ліниву загрузку, щоб завантажувати зображення тільки тоді, коли вони стають видимими на екрані.
2. Автоматична оптимізація зображень: Next.js та Vercel мають вбудовані інструменти для оптимізації зображень. При кожному розгортанні, ці платформи автоматично стискають та оптимізують зображення, щоб зменшити їх розмір без втрати якості. Це допомагає зменшити час завантаження сторінок та знижує використання пропускної здатності мережі.
3. Використання CDN: Vercel надає глобальну мережу доставки контенту (CDN), яка автоматично кешує ваші статичні ресурси, такі як HTML, CSS, зображення і JavaScript, на серверах розташованих ближче до користувачів. Це дозволяє знизити час завантаження контенту для віддалених користувачів та зменшити навантаження на ваш сервер.
4. Інкрементальна рендерінг: Next.js має підтримку інкрементального рендерінгу, що означає, що сторінки можуть бути рендернуті частинами при запиті користувача. За допомогою цього підходу, користувачі

отримують швидкий перший відгук від додатку, тоді як решта сторінки завантажується. Це поліпшує загальний час завантаження та взаємодії з додатком.

5. Prefetching та попередні рендери (prerendering): Next.js дозволяє завантажувати (prefetch) або попередньо рендерити (prerender) сторінки, які користувачі, ймовірно, відвідають. Це дозволяє зменшити затримку при переході між сторінками та покращити загальний досвід використання додатку.
6. Кешування: Використання кешування може суттєво зменшити час завантаження сторінок та ресурсів. За допомогою правильно налаштованого кешування, сторінки або частини сторінок можуть бути збережені в кеші на боці клієнта або сервера. Це дозволяє повторно використовувати вже завантажений контент та зменшити потребу в повторних запитах до сервера.
7. Мінімізація та стиснення ресурсів: Зменшення розміру файлів, таких як HTML, CSS та JavaScript, може покращити швидкодію завантаження. Методи, такі як мініфікація, стиснення та злиття файлів, можуть допомогти знизити їх розмір. Також варто використовувати компресію з втратами (наприклад, gzip) для передачі стиснутих версій файлів до браузера.
8. Відкладене завантаження ресурсів: Використання атрибутів, таких як async або defer для скриптів та async для стилів, дозволяє відкласти завантаження ресурсів, що не є критичними для першого відображення сторінки. Це дозволяє браузеру швидше відобразити сторінку, не очікуючи повного завантаження всіх ресурсів.
9. Оптимізація запитів до сервера: Зменшення кількості та розміру запитів до сервера може покращити час завантаження сторінок. Наприклад, можна об'єднати декілька запитів в один, використовувати кешування на сервері, асинхронно завантажувати дані та використовувати HTTP-кешування.

На рисунку 4.9 зображено результати Lighthouse для веб-додатку без додавання методів оптимізацій, а на рисунку 4.10 зображено значення цих метрик.

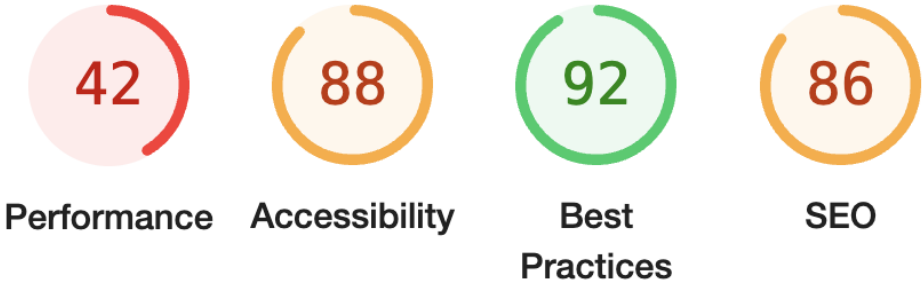


Рисунок 4.9 – Результати базового образу веб-додатку

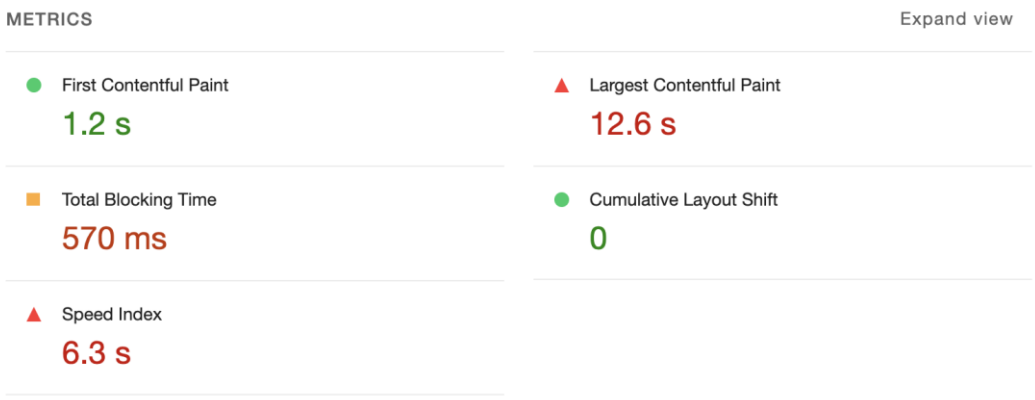


Рисунок 4.10 - Значення метрик базового образу веб-додатку

На рисунку 4.11 зображено результати Lighthouse для веб-додатку після додавання методів оптимізацій, а на рисунку 4.12 зображено значення цих метрик.

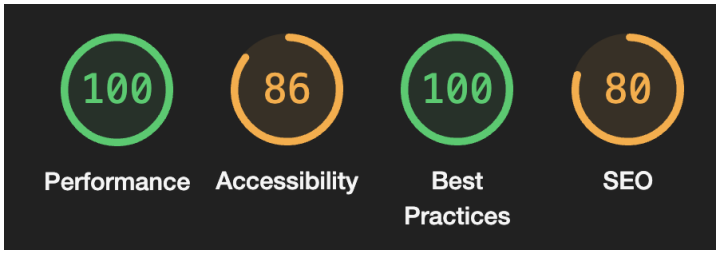


Рисунок 4.11 - Результати базового образу веб-додатку після додавання елементів оптимізації

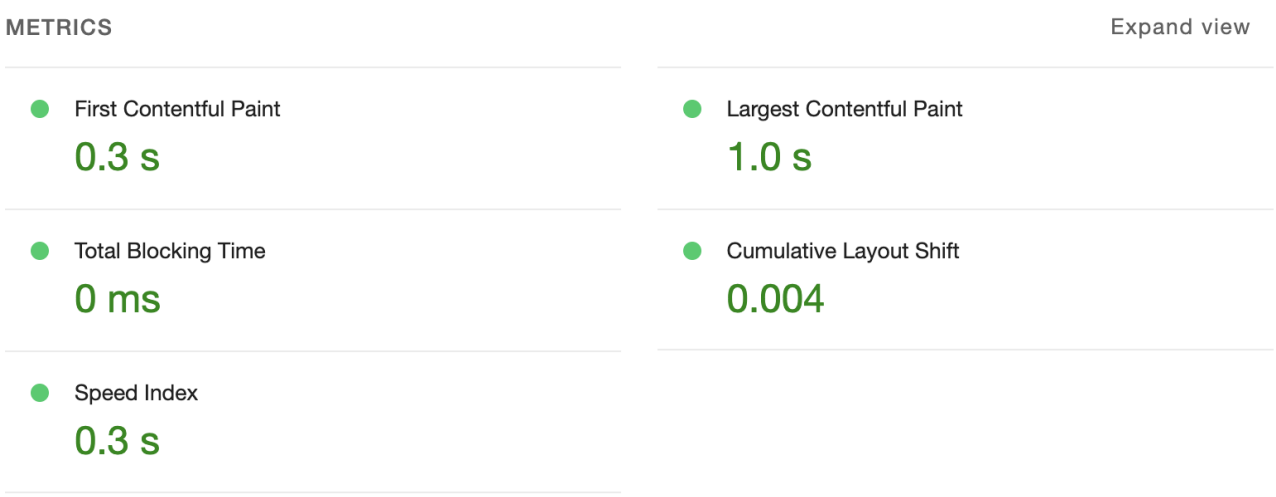


Рисунок 4.12 - Значення метрик базового образу веб-додатку після додавання методів оптимізації

Після аналізу базового веб-додатку та додатку з впровадженими рекомендаціями за допомогою Lighthouse, можна порівняти отримані метрики та зрозуміти, наскільки оптимізаційні методи покращили продуктивність, доступність, дотримання найкращих практик та SEO веб-додатку. Результати порівняння допоможуть виявити сильні та слабкі сторони додатку і спрямувати зусилля на подальшу оптимізацію та покращення продуктивності.

4.2 Висновки

Після впровадження оптимізаційних методів і проведення оцінки ефективності за допомогою інструменту Lighthouse, можна зробити кілька висновків щодо покращення продуктивності та швидкодії веб-додатку:

1. Зменшення часу завантаження: Оптимізаційні методи сприяли зменшенню часу завантаження сторінок. Швидке рендеринг, асинхронне завантаження даних та використання CDN допомогли покращити швидкість завантаження, що забезпечує кращий користувацький досвід.
2. Покращена продуктивність: Завдяки використанню оптимізаційних методів, веб-додаток став більш продуктивним. Зменшення завантаження

сервера, оптимізація роботи з базою даних та ефективна доставка статичних ресурсів допомогли забезпечити стабільну та швидку роботу додатку.

3. Покращена доступність: Застосування найкращих практик доступності та оптимізоване використання компонента Image допомогли поліпшити доступність веб-додатку для користувачів з обмеженими можливостями. Це робить додаток більш доступним та інклюзивним.
4. Покращена SEO: Застосування оптимізаційних методів допомогло покращити пошукову оптимізацію веб-додатку. Налаштування правильних метатегів, оптимізоване використання URL-адрес та швидкодія сторінок сприяють покращенню видимості додатку у пошукових системах.

В цілому, використання оптимізаційних методів дає позитивні результати щодо продуктивності, швидкодії, доступності та SEO веб-додатку.

ВИСНОВОК

В рамках бакалаврської роботи успішно створено оптимізований веб-додаток за допомогою Next.js, TypeScript, TRPC, Prisma, Postgres і Vercel.

Проаналізувавши сучасні інструменти з відкритим вихідним кодом для покращення якості веб-додатків я обрав Lighthouse та K6.

При використанні інструменту K6 були отримані наступні результати:

1. Кількість HTTP-запитів (http_reqs) збільшилась на 665%;
2. Тривалість HTTP-запитів (http_req_duration) зменшилась на 505%.
3. Очікування HTTP-запитів (http_req_waiting) зменшилось на 627%.
4. Кількість HTTP-запитів за секунду (http_reqs/sec) збільшилась на 687%.

При виконанні інструменту Lighthouse були отримані наступні результати:

1. First Contentful Paint (FCP) зменшилось на 300%, з 1.2 секунд до 0.3 секунди.
2. Largest Contentful Paint (LCP) зменшилось на 1260%, з 12.6 секунд до 1.0 секунди.
3. Total Blocking Time (TBT) зменшилось з 570 мілісекунд до 0 мілісекунд.
4. Cumulative Layout Shift (CLS) змінився незначно, з 0 секунд до 0.004 секунди.
5. Speed Index зменшилось на 2300%, з 6.3 секунд до 0.3 секунди.

Тому можна зробити висновок, що обрані інструменти, такі як Lighthouse та K6, дозволяють підвищити продуктивність і швидкість роботи веб-додатка, а також виявити його слабкі місця та проблеми з продуктивністю.

ПЕРЕЛІК ДЖЕРЕЛ

1. Інтернет-ресурс https://web.dev/vitals/
2. [O'Reilly] Osmani, A. (2018). Practical Performance Profiling: Improving the Efficiency of JavaScript Applications. O'Reilly Media.
3. [Manning] Souders, S. (2009). High Performance Web Sites: Essential Knowledge for Front-End Engineers. Manning Publications.
4. [O'Reilly] Grigorik, I. (2013). High Performance Browser Networking: What Every Web Developer Should Know About Networking and Web Performance. O'Reilly Media.
5. JWT token docs https://jwt.io/introduction
6. Документація NextJS https://nextjs.org/docs
7. [O'Reilly] Kleppmann, M. (2017). Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems (1st Edition). O'Reilly Media.
8. [Apress] Barker, R. (2014). High Performance Responsive Design: Building Faster Sites Across Devices. Apress.
9. [O'Reilly] Zimarev, V. (2020). Web Scalability for Startup Engineers. O'Reilly Media.

ДОДАТКИ

Додаток А

```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider          = "postgresql"
  url               = env("POSTGRES_PRISMA_URL")
  directUrl         = env("POSTGRES_URL_NON_POOLING")
  shadowDatabaseUrl = env("POSTGRES_URL_NON_POOLING")
}

model User {
  id        String     @id @default(cuid())
  email     String     @unique
  username  String?    @unique
  password  String
  avatar    String?
  comments  Comment[]
  createdAt DateTime   @default(now())
  Cart      Cart[]
  Wishlist  Wishlist[]
  Like      Like[]
}

model Product {
  id            String          @id @default(cuid())
```

```
  name          String
  price         Int
  image         String
  description   String
  poster        String
  commented     Comment[]
  catalog_name  Catalog        @relation(fields: [catalogId], references: [id])
  discount      Discount?      @relation(fields: [discountId], references: [id])
  tags_name     Tag[]
  isInStock     Boolean        @default(true)
  view          Int?           @default(0)
  Characteristic Characteristic[]
  catalogId     String
  discountId    String?        @unique
  Cart          Cart[]
  Wishlist      Wishlist[]
}

model Characteristic {
  id        String   @id @default(cuid())
  attribute String
  value     String
  Product   Product? @relation(fields: [productId], references: [id])
  productId String
}

model Comment {
  id        String   @id @default(cuid())
  content   String
  createdAt DateTime @default(now())
```

```
  rate      Int
  Product   Product? @relation(fields: [productId], references: [id])
  productId String?
  User      User?    @relation(fields: [userId], references: [id])
  userId    String?
  Like      Like[]
  @@index([productId])
}

model Like {
  id        String  @id @default(cuid())
  user      User    @relation(fields: [userId], references: [id])
  comment   Comment @relation(fields: [commentId], references: [id])
  commentId String
  userId    String
}

model Tag {
  id        String   @id @default(cuid())
  name      String
  Product   Product? @relation(fields: [productId], references: [id])
  productId String?
}

model Catalog {
  id      String    @id @default(cuid())
  name    String    @unique
  Product Product[]
}
```

```
model Discount {
  id       String    @id @default(cuid())
  isActive Boolean?  @default(true)
  percent  Int?
  Product  Product[]
}

model Cart {
  id       String  @id @default(cuid())
  user     User    @relation(fields: [userId], references: [id])
  product  Product @relation(fields: [productId], references: [id])
  userId    String
  productId String
  count     Int     @default(1)

  @@map("cart")
}

model Wishlist {
  id       String  @id @default(cuid())
  user     User    @relation(fields: [userId], references: [id])
  product  Product @relation(fields: [productId], references: [id])
  userId    String
  productId String
  count     Int     @default(1)

  @@map("wishlist")
}
```

Додаток Б

1. Результати тестування для головної сторинки https://lighthouse-metrics.com/lighthouse/checks/be00382f-301e-47e4-84c8-d0142b4ff870
2. Результати тестування випадкового продукту https://lighthouse-metrics.com/app/asdfasdfasdf/measure/94f3aa8e-e300-48c4-a8db-0d8502909ff1
3. Репозиторій веб-додатоку https://github.com/modernxpunk/shop
4. Домен веб-додатку https://shop-bachelor-thesis.vercel.app/