Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження моделей та технологій завантаження сторінок у web-застосунках з метою визначення оптимізованих підходів рендерингу
(тема)

Виконав:
студент (ка) 2 курсу, групи ІПЗм-22-1

Васильєв Д.О.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)

Тип програми освітньо-наукова

Керівник доц. Афанасьєва І.В.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ З.В.Дудар
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти другий (магістерський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми освітньо-наукова програма
Освітня програма Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___»_____20__ р.

ЗАВДАННЯ
НА КВАЛІФАКАЦІЙНУ РОБОТУ

Студентові Васильєву Денису Олександровичу

1. Тема роботи *«Дослідження моделей та технологій завантаження сторінок у web-застосунках з метою визначення оптимізованих підходів рендерингу»*
затверджена наказом університету від 29 березня 2024 р. № 250Ст
2. Термін подання студентом роботи до екзаменаційної комісії 25 червня 2024 р.
3. Вихідні дані до роботи *технології розробки веб-застосунків, методи веб-інтеграції, основи серверного рендерингу в React та Next.js.*
4. Перелік питань, що потрібно опрацювати у роботі *вступ, аналіз моделей завантаження сторінок у веб-застосунках, аналіз сучасних технік оптимізації завантаження сторінок, вибір методу рендерингу під сценарій використання, розробка із використанням SSR, висновки.*

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі та постановка задачі	20.01.24 – 14.02.24	*виконано*
2	Аналіз та вибір API для дослідження	15.02.24 – 24.02.24	*виконано*
3	Аналіз та моделювання предметної області	17.02.24 – 28.02.24	*виконано*
4	Планування експериментів	25.02.24 – 28.02.24	*виконано*
5	Програмна реалізація кожного з обраних для дослідження API	25.02.24 – 01.04.24	*виконано*
6	Експериментальні дослідження	02.04.24 – 20.04.24	*виконано*
7	Аналіз результатів експериментальних досліджень та розробка рекомендацій	20.04.24 – 23.04.24	*виконано*
8	Написання та оформлення тез доповіді	17.04.24 – 23.04.24	*виконано*
9	Підготовка пояснювальної записки	01.04.24 – 26.05.24	*виконано*
10	Підготовка презентації та доповіді	26.05.24 – 02.06.24	*виконано*
11	Перевірка на плагіат та нормоконтроль	21.06.24	*виконано*
12	Рецензування	21.06.24	*виконано*
13	Занесення диплома в електронний архів	22.06.2024	*виконано*
14	Попередній захист	22.06.2024	*виконано*
15	Допуск до захисту у зав. кафедри	24.06.2024	*виконано*

Дата видачі завдання 20 січня 2024 р.

Студент _____ Васильєв Д.О.
(підпис)

Керівник кваліфікаційної роботи _____ доц. Афанасьєва І.В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Робота містить: 59 с., 7 рис., 14 джер., 5 дод.

ВЕБ-ЗАСТОСУНКИ, КОМПОНЕНТИ, ШВИДКІСТЬ РЕНДЕРІНГУ, RENDERING, SERVER-SIDE.

Об'єктом дослідження є розробка та оптимізація веб-застосунків з використанням Server-side rendering на базі технологій React та Next.js.

Метою роботи є визначення оптимізованих підходів рендерингу та створення ефективних компонентів та програмного забезпечення для побудови веб-додатків будь-якої складності з використанням Server-side rendering та визначення швидкості завантаження сторінки в залежності від метода рендерінгу.

У процесі розробки використовуються сучасні методи розробки, бібліотеки та мова програмування JavaScript.

В результаті дослідження виявлено найбільш швидкий метод рендерінгу під конкретні задачі бізнесу.

WEB APPLICATIONS, COMPONENTS, RENDERING SPEED, RENDERING, SERVER-SIDE.

The subject of the research is the development and optimization of web applications using Server-side Rendering based on React and Next.js technologies.

The aim of the work is to identify optimized rendering approaches and create effective components and software for building web applications of any complexity using Server-side rendering and determining the page load speed depending on the rendering method.

The development process uses modern development methods, libraries, and the JavaScript programming language.

As a result of the research, the fastest rendering method for specific business tasks was identified.

Я, Васильєв Денис Олександрович, студент гр. ІПЗм-22-1, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя робота на тему «Дослідження моделей та технологій завантаження сторінок у web-застосунках з метою визначення оптимізованих підходів рендерингу», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElArKhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений(на) з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ВСТУП

У наш час веб-розробки, який характеризується стрімким технологічним розвитком, особливо важливим є вивчення та оптимізація методів завантаження сторінок у веб-застосунках. Забезпечення ефективності та швидкодії веб-додатків стає критичною вимогою в умовах зростаючої конкуренції та високих очікувань користувачів.

Об'єктом даного дослідження є аналіз, дослідження та оптимізація моделей та технологій завантаження сторінок у веб-застосунках з метою визначення оптимізованих підходів рендерингу. Зосереджуючись на забезпеченні максимальної продуктивності та якості користувацького досвіду, це дослідження висвітлює ключові аспекти та виклики, що виникають при розробці веб-застосунків.

Розглядаючи динаміку веб-розробки, відзначається, що вибір відповідних стратегій рендерингу стає значущим фактором для досягнення успіху у веб-просторі. Різні методи, такі як Client-side Rendering (CSR), Server-side Rendering (SSR) та Static Site Generation (SSG), мають свої переваги та обмеження, тому важливо визначити найбільш оптимальні підходи для конкретного випадку.

Однією з основних мет цього дослідження є аналіз переваг та недоліків різних підходів до рендерингу сторінок. Зокрема, вивчення ефективності CSR, де частину рендерингу виконує браузер користувача, порівнюється з принципами SSR, де сервер генерує HTML на стороні сервера, та SSG, яке дозволяє створювати статичні файли під час збірки додатку [1].

Розглядаючи сучасні технології у веб-розробці, особливу увагу приділено фреймворкам React та Next.js. Використання Server-side Rendering на базі цих інструментів визначається як ключовий аспект для забезпечення продуктивності та оптимізації веб-додатків.

Однією з важливих задач дослідження є проектування та реалізація прикладних веб-застосунків для практичного випробування та валідації розглянутих концепцій. Це передбачає розробку алгоритмів, які оптимально

використовують потужності SSR та SSG для забезпечення ефективного рендерингу сторінок.

Окрім того, дослідження включає аналіз результатів оптимізації та вивчення продуктивності розроблених додатків. Метою є визначення найбільш оптимальних підходів до рендерингу сторінок у веб-застосунках на основі отриманих знань.

Очікується, що результати дослідження нададуть глибоке розуміння процесів розробки веб-застосунків з використанням SSR та SSG. Важливою частиною висновків буде обговорення перспектив розвитку даної області та можливостей впровадження отриманих результатів у реальних веб-проектах.

Останні роки свідчать про значне зростання інтересу до використання SSR та SSG у веб-розробці, і важливо акцентувати їх використання у контексті створення сучасних та конкурентоздатних веб-додатків. Усвідомлення та реалізація оптимальних стратегій рендерингу є ключем до успішної розробки продуктивних та високопродуктивних веб-застосунків.

1 АНАЛІЗ МОДЕЛЕЙ ЗАВАНТАЖЕННЯ СТОРІНОК У ВЕБ-ЗАСТОСУНКАХ

1.1 Server-side Rendering(SSR)

Server-side Rendering (SSR) – це підхід до рендерингу веб-застосунків, де HTML-код формується на сервері та вже готовий відправляється на клієнтський браузер. Цей метод стає об'єктом уваги через свою здатність до оптимізації часу завантаження та SEO-придатності [2].

Однією з ключових переваг SSR є можливість отримання вже повноцінно сформованої сторінки на етапі її першого завантаження. Це призводить до зменшення часу, необхідного для користувача на очікування відображення контенту, оскільки відбувається зменшення обсягу робіт, які виконує клієнтський браузер. Передача вже згенерованого HTML також може сприяти зниженню обсягу передаваних даних, що важливо для оптимізації загального часу завантаження при слабкому Інтернет-з'єднанні.

Проте, SSR має свої недоліки. Один із них - збільшення навантаження на сервер у порівнянні з іншими методами рендерингу. Кожен запит на сторінку вимагає переробки на сервері, що може призвести до зниження продуктивності при великій кількості запитів. Додатково, SSR може бути складніше в управлінні та розгортанні порівняно з іншими стратегіями, такими як Client-side Rendering (CSR) або Static Site Generation (SSG).

Однак, враховуючи ситуаційну придатність, SSR залишається важливим елементом в арсеналі розробника веб-застосунків. Він забезпечує баланс між ефективністю SEO, продуктивністю та користувацькою взаємодією, дозволяючи розробникам обирати оптимальний метод рендерингу відповідно до конкретних вимог і особливостей їхнього проекту [3].

Таким чином, Server-side Rendering продовжує залишатися важливою стратегією рендерингу у веб-розробці, пропонуючи компроміс між завантаженням на сервер та продуктивністю, а також забезпечуючи переваги в плані SEO та часу відповіді, які роблять його популярним серед розробників.

Реакт (React) та Next.js – це дві потужні технології, що визначають реалізацію веб-застосунків, зокрема у контексті рендерингу сторінок [4]. Роль та вплив цих інструментів в розробці веб-застосунків визначають їхню популярність та використання у великій кількості сучасних проектів.

React - це бібліотека для створення інтерфейсів, яка дозволяє розробникам легко створювати перевідно-орієнтовані та динамічні веб-застосунки [5]. Однією з ключових особливостей React є використання віртуального DOM (Document Object Model), що дозволяє оптимізувати процес оновлення сторінок та забезпечує більш ефективне використання ресурсів.

Вплив React на розробку веб-застосунків проявляється у зручному компонентному підході. Розбиття інтерфейсу на компоненти дозволяє створювати частини додатків, які можна легко перевикористовувати та модифікувати. Це полегшує процес розробки та обслуговування, зменшуючи ймовірність помилок та сприяючи швидшій ітерації.

Next.js, з іншого боку, є фреймворком для реактивної розробки веб-застосунків. Він додає до React багато корисних можливостей, зокрема Server-side Rendering (SSR) та Static Site Generation (SSG). Це робить його потужним інструментом для оптимізації завантаження сторінок та покращення продуктивності веб-додатків.

Однією з ключових ролей Next.js є його вбудована підтримка SSR, яка дозволяє генерувати сторінки на сервері, зменшуючи час завантаження та поліпшуючи SEO-показники [6]. Також важливою є можливість SSG, що дозволяє попередньо генерувати статичні сторінки і використовувати їх для зниження навантаження на сервер у деяких сценаріях використання.

У взаємодії React та Next.js виявляється синергетичний ефект. Реакт дозволяє зручно створювати інтерфейс, а Next.js доповнює його можливостями оптимізації завантаження сторінок. Це робить їх ідеальними інструментами для веб-розробників, які прагнуть забезпечити швидку та ефективну роботу своїх додатків.

Загалом, роль та вплив React та Next.js у веб-розробці полягають в створенні потужних, ефективних та швидких веб-застосунків. Використання їхніх

можливостей сприяє вдосконаленню розробки, зменшенню часу на створення нового функціоналу та покращенню користувацького досвіду.

Рендеринг сторінок веб-застосунків – це ключовий етап розробки, оскільки від нього залежить не лише користувацький досвід, але й ефективність пошукової оптимізації (SEO) [7]. Розглянемо вплив різних стратегій рендерингу на обидва аспекти.

SSR є стратегією, при якій сервер генерує HTML-контент і відправляє його користувачеві. Це сприяє швидкому відображенню контенту при завантаженні сторінки. Для користувача це означає швидший перехід між сторінками та загальний високий рівень відзивчивості.

Щодо SEO, SSR є вигідним, оскільки пошукові системи можуть легко індексувати зміст на сервері та отримувати повністю готові сторінки. Це дозволяє поліпшити індексацію та рейтинг сторінок в пошукових системах.

CSR, навпаки, вимагає від клієнта (браузера) виконати роботу з генерацією HTML-контенту. Це може трошки затримати відображення сторінки, особливо при першому завантаженні. Однак цей підхід дозволяє створювати динамічний та високоінтерактивний контент.

Однією з головних переваг CSR є зменшення навантаження на сервер, оскільки він лише надсилає клієнту статичні файли, а рендеринг відбувається на боці клієнта. Проте це може негативно позначитися на SEO, оскільки пошукові системи мають обмежений доступ до динамічно генерованого контенту.

SSG – це компроміс між SSR та CSR. Він передбачає генерацію статичних сторінок під час збірки додатку або напередодні на сервері. Такий підхід дозволяє використовувати переваги обох стратегій, забезпечуючи швидкість SSR та гнучкість CSR.

Щодо користувацького досвіду, SSG забезпечує швидке відображення сторінок, подібне до SSR. Його вплив на SEO також позитивний, оскільки пошукові системи можуть легко індексувати статичний контент.

Вибір стратегії рендерингу залежить від конкретних вимог проекту. SSR підходить для проектів, де SEO важливий, і де необхідно надавати користувачам

максимально відзивчивий досвід. CSR використовується там, де важлива динаміка та висока інтерактивність. SSG є універсальним рішенням, яке дозволяє забезпечити ефективний користувацький досвід і високий рейтинг у пошукових

1.2 Оптимальне використання переваг SSR та CSR

В сучасному світі веб-розробки розробники зіштовхуються з вибором стратегії рендерингу, яка оптимально відповідає вимогам їхніх проектів. Server-side Rendering (SSR) та Client-side Rendering (CSR) є двома ключовими підходами, кожен з яких має свої переваги та особливості. Оптимальне використання цих стратегій може визначити швидкодію, ефективність та користувацький досвід веб-додатку.

SSR є підходом, при якому весь контент генерується на сервері та надсилається клієнту у вигляді готового HTML-коду. Це дозволяє отримувати повноцінні сторінки з сервера, що поліпшує швидкодію відображення контенту. Однак для оптимального використання SSR важливо враховувати кілька ключових моментів.

Оптимізація сервера: Швидкодія SSR визначається не лише якістю коду клієнта, але й швидкістю роботи сервера. Важливо використовувати ефективні серверні ресурси та мінімізувати час обробки запитів.

Контроль над кешуванням: Зберігання певних даних в кеші може суттєво покращити відгук веб-додатку. Оптимальне використання SSR передбачає розумне кешування для зменшення завантаження сервера та швидкодії відображення.

Підтримка SEO: Однією з головних переваг SSR є сприяння SEO. Для максимального використання цієї переваги важливо правильно налаштувати мета-теги та структуру сторінок.

CSR, навпаки, передбачає, що генерація контенту відбувається на боці клієнта за допомогою JavaScript. Це забезпечує високий рівень інтерактивності, але для оптимального використання CSR слід враховувати деякі ключові моменти.

Оптимізація клієнтського коду: Оскільки весь процес генерації відбувається в браузері, важливо оптимізувати клієнтський код для мінімізації часу

завантаження та виконання.

Розподіл функцій: Певні функції можуть бути виконані на сервері, а інші - на клієнті. Оптимальне використання CSR передбачає розподіл завдань так, щоб максимально використовувати переваги обох підходів.

Керування станом: Ефективне управління станом додатку на клієнтському боці важливо для уникнення зайвого завантаження та оптимізації продуктивності.

Тож, можна сказати, що оптимальне використання переваг SSR та CSR передбачає глибоке розуміння їхніх можливостей та відповідне впровадження в конкретних умовах проекту. Нерідко комбінація обох стратегій використовується для досягнення балансу між продуктивністю та інтерактивністю веб-додатку. Розробники повинні уважно вивчати потреби свого проекту та вибирати стратегію рендерингу, яка найкраще відповідає конкретним завданням та очікуванням користувачів.

1.3 Постановка задачі

В сучасному світі веб-технологій швидкість та ефективність завантаження веб-сторінок є критичними факторами успіху веб-застосунків. Зі збільшенням складності веб-додатків зростає необхідність вибору оптимальних методів рендерингу для забезпечення максимально швидкої та плавної роботи користувацького інтерфейсу. Основними підходами до рендерингу є клієнтський рендеринг, серверний рендеринг та статичний рендеринг, кожен з яких має свої переваги та недоліки.

Метою даного дослідження є проведення глибокого аналізу різних методів рендерингу веб-сторінок та визначення оптимальних підходів для різних типів веб-застосунків. Також дослідження спрямоване на аналіз сучасних технологій, таких як React та Next.js, для виявлення їх можливостей та обмежень у контексті рендерингу. Основні задачі дослідження наведені нижче.

Аналіз методів рендерингу:

– провести огляд основних методів рендерингу: клієнтський рендеринг (Client-Side Rendering, CSR), серверний рендеринг (Server-Side Rendering,

SSR) та статичний рендеринг (Static Site Generation, SSG);

- визначити переваги та недоліки кожного з методів рендерингу;
- оцінити вплив кожного методу на швидкість завантаження сторінок, споживання ресурсів та досвід користувача.

Визначення оптимальних методів рендерингу в залежності від задач:

- провести класифікацію типових веб-застосунків за їх функціональними вимогами та характером взаємодії з користувачем;
- визначити, які методи рендерингу найбільш підходять для кожного типу веб-застосунків.

Аналіз сучасних технологій для створення веб-застосунків:

- дослідити можливості та особливості технологій React та Next.js у контексті різних методів рендерингу;
- оцінити переваги та обмеження використання React та Next.js для клієнтського, серверного та статичного рендерингу.

Експериментальне дослідження методів рендерингу:

- провести серію експериментів для порівняння швидкості та ефективності різних методів рендерингу;
- використати реальні веб-застосунки або прототипи для проведення тестування;
- визначити, який метод рендерингу є найшвидшим та найефективнішим в залежності від конкретних умов та задач.

Результати дослідження дозволять зрозуміти переваги та недоліки кожного з методів рендерингу, визначити оптимальний метод для різних типів веб-застосунків, надати рекомендації щодо вибору методів рендерингу при розробці веб-застосунків, а також виявити можливості та обмеження використання технологій React та Next.js для рендерингу. Це дослідження надасть детальний огляд сучасних методів рендерингу та технологій для створення веб-застосунків, що допоможе розробникам та архітекторам обирати оптимальні підходи для покращення продуктивності та якості веб-додатків.

2 АНАЛІЗ СУЧАСНИХ ТЕХНІК ОПТИМІЗАЦІЇ ЗАВАНТАЖЕННЯ СТОРІНОК

2.1 Сучасні техніки оптимізації

Сучасний веб-ландшафт вимагає від розробників та веб-мастерів удосконалення та оптимізації завантаження сторінок для забезпечення найвищого рівня користувацького досвіду. Справедливо вважається, що швидкодія веб-застосунків безпосередньо впливає на задоволення користувачів та їхню лояльність. Тому сучасні техніки оптимізації завантаження сторінок стають необхідністю у веб-розробці.

Однією з ключових технік оптимізації є використання адаптивних зображень. Це дозволяє завантажувати зображення відповідно до характеристик пристрою користувача, таких як розмір екрану та роздільна здатність. Використання тега <picture> разом із атрибутом srcset дозволяє браузеру автоматично вибирати найкращий варіант для кожного пристрою, забезпечуючи оптимальну якість при мінімальному використанні ресурсів.

Окрім використання адаптивних зображень, важливо оптимізувати їхні розміри та формати. Використання стиснених форматів, таких як WebP, може суттєво зменшити розмір файлів, що призведе до швидшого завантаження сторінок. Також важливо використовувати інструменти автоматичної оптимізації, які забезпечать найкращу можливу якість при найменшому розмірі файлів.

Техніка "запізнене завантаження ресурсів" дозволяє призначити пріоритети завантаження для різних ресурсів на сторінці. Це дозволяє важливим елементам завантажуватися першими, а менш важливим - пізніше чи асинхронно. Використання атрибутів defer та async для завантаження скриптів також є важливою частиною цієї техніки.

Використання кешування є ефективним методом оптимізації завантаження сторінок. Кешування на рівні клієнта дозволяє зберігати ресурси локально, уникати повторного завантаження при подальших відвідуваннях. Кешування на рівні сервера дозволяє зберігати вже оброблені ресурси та надсилати їх безпосередньо

при запитах, що дозволяє зменшити час відповіді сервера.

Content Delivery Network (CDN) є ключовою технікою для прискорення завантаження сторінок. CDN забезпечує розподілення контенту на сервера по всьому світу, надаючи користувачам можливість завантаження ресурсів з найближчого до них сервера [8]. Це особливо важливо для глобальних проектів з великою аудиторією.

Мінімізація коду та компресія ресурсів (CSS, JavaScript) є стандартними практиками для зменшення їхнього обсягу. Використання спеціалізованих інструментів, таких як UglifyJS чи Terser для JavaScript, та CSSNano для CSS, може значно зменшити обсяг файлів, що передаються користувачам [9].

Асинхронне завантаження скриптів дозволяє продовжити завантаження сторінки, навіть якщо скрипти ще не повністю завантажилися. Це дозволяє поліпшити загальний час завантаження та інтерактивність сторінок. Атрибут async чи defer може бути використаний для контролю над асинхронним завантаженням скриптів.

Використання інструментів для вимірювання продуктивності, таких як Google Lighthouse, PageSpeed Insights, WebPageTest, дозволяє виявляти та вирішувати проблеми, що впливають на швидкодію сторінок. Ці інструменти надають детальну інформацію про різні аспекти продуктивності, такі як час першого відображення (FCP), час наявності перших пікселів (FP), час загального завантаження (TTI) та інші.

Завдяки впровадженню сучасних технік оптимізації, розробники можуть досягти не тільки прискорення завантаження сторінок, але й забезпечити зручний та ефективний досвід для користувачів у веб-середовищі. Ці техніки стають невід'ємною частиною стратегії веб-розробників для вдосконалення продуктивності та забезпечення високоякісного веб-досвіду.

2.2 Використання Lazy Loading для оптимізації завантаження ресурсів

Використання техніки Lazy Loading є важливим аспектом оптимізації завантаження ресурсів на веб-сайтах. Ця стратегія спрямована на покращення

швидкодії та продуктивності сторінок, особливо коли маємо справу з великою кількістю мультимедійного контенту, такого як зображення чи відео. Розглянемо детальніше, що таке Lazy Loading та як вона впливає на загальний досвід користувача та продуктивність веб-додатків.

Lazy Loading - це техніка, яка дозволяє відкладати завантаження певних ресурсів (зображень, скриптів, стилів тощо) до моменту, коли вони стають необхідними для відображення на сторінці. Основна перевага полягає в тому, що лише видима частина сторінки завантажується спочатку, покращуючи час завантаження та зменшуючи кількість передачі даних.

Швидша швидкість завантаження сторінок:

Завантаження тільки тих ресурсів, які стають видимими для користувача, дозволяє покращити час завантаження сторінок. Це особливо важливо для сторінок з великою кількістю контенту, таких як блоги, електронні магазини чи новинні сайти.

Lazy Loading дозволяє оптимізувати використання мережевих ресурсів, оскільки необхідно завантажувати лише ті ресурси, які реально потрібні. Це зменшує обсяг передачі даних та забезпечує ефективне використання інтернет-з'єднання користувача.

Завантаження всіх ресурсів одразу може суттєво вплинути на продуктивність сторінки, особливо на пристроях з обмеженими ресурсами. Lazy Loading дозволяє спростити завантаження, зменшуючи кількість одночасно завантажуваних ресурсів.

На мобільних пристроях з обмеженим об'ємом даних та швидкістю Інтернету Lazy Loading стає особливо важливою. Вона забезпечує зручний та швидкий доступ до контенту, зменшуючи споживання даних.

Однією з найпоширеніших областей застосування Lazy Loading є робота з зображеннями. Використання атрибуту loading="lazy" у тегах <img> дозволяє браузеру відкладати завантаження зображень, поки вони не стануть видимими в зоні видимості користувача.

Для скриптів та стилів використання атрибутів async чи defer може

допомогти у впорядкуванні завантаження цих ресурсів. Вони дозволяють визначити, коли і як саме вони будуть завантажені та виконані.

Додавання Lazy Loading до відео та інших ресурсів може бути реалізовано за допомогою JavaScript-бібліотек чи вбудованих атрибутів, таких як loading="lazy" для тега <video>.

Тож, Lazy Loading стає необхідною стратегією для сучасних веб-розробників, спрямованою на забезпечення найвищого рівня продуктивності та ефективності завантаження ресурсів на веб-сайтах. Ця техніка допомагає покращити взаємодію з користувачем, зменшити споживання даних та оптимізувати роботу сторінок, забезпечуючи приємний та ефективний веб-досвід для всіх користувачів.

2.3 Роль CDN (Content Delivery Network) у швидкодії завантаження сторінок

Content Delivery Network (CDN) в сучасному інтернет-середовищі відіграє ключову роль у покращенні швидкості завантаження веб-сторінок і взагалі оптимізації роботи веб-застосунків. Розглянемо роль CDN в контексті теми "Дослідження моделей та технологій завантаження сторінок у web-застосунках з метою визначення оптимізованих підходів рендерингу".

Content Delivery Network (CDN) представляє собою глобальну мережу серверів, розташованих у різних частинах світу. Ці сервери призначені для зберігання і доставки контенту до кінцевого користувача. Використання CDN дозволяє оптимізувати передачу даних, поліпшити безпеку та зменшити час завантаження веб-сторінок.

CDN працює на основі принципу розподіленого зберігання ресурсів. Замість того, щоб сервер в одному місці намагався відповісти на запити від користувачів з усього світу, CDN розподіляє вміст на сервери, які знаходяться ближче до кінцевого користувача. Це дозволяє скоротити відстань, яку повинні подолати дані, і в результаті прискорює завантаження сторінок.

Однією з ключових переваг CDN є зменшення латентності. Латентність - це затримка часу між відправленням запиту та отриманням відповіді. Без CDN запити можуть подорожувати значні відстані, що може призводити до затримок.

Розташування серверів CDN в різних регіонах дозволяє максимально зблизити контент до кінцевих користувачів, зменшуючи латентність і поліпшуючи загальний досвід використання веб-застосунків.

CDN використовує механізми кешування для зберігання копій ресурсів на своїх серверах. Це дозволяє вам отримувати доступ до ресурсів навіть у випадку проблем з основним сервером. Кешування забезпечує відмовостійкість, що є важливим аспектом забезпечення доступності веб-застосунків.

CDN автоматично оптимізує завантаження ресурсів, використовуючи різні техніки, такі як компресія, мінімізація та об'єднання файлів. Це дозволяє зменшити обсяг передачі даних і прискорює завантаження сторінок, особливо на мобільних пристроях або в умовах обмеженого Інтернет-з'єднання.

CDN може виконувати функції захисту від DDoS-атак та інших загроз безпеки. Великі мережі CDN мають ресурси для фільтрації та виявлення зловживань, що забезпечує безпеку веб-застосунків.

CDN надає інструменти для аналізу та моніторингу роботи веб-застосунків. За допомогою звітів можна відстежувати швидкість завантаження, обсяг трафіку, використання ресурсів і багато іншого. Це дозволяє вам налаштовувати роботу веб-застосунків для досягнення максимальної ефективності.

У світі швидко розвиваючихся технологій CDN стає невід'ємною складовою дослідження моделей та технологій завантаження сторінок у веб-застосунках. Вона допомагає оптимізувати завантаження, зменшує латентність, підвищує безпеку і є важливим інструментом для підвищення продуктивності веб-застосунків в умовах сучасного інтернет-середовища.

2.4 Методи оптимізації великих зображень для прискорення завантаження

Оптимізація великих зображень - ключовий аспект завантаження веб-сторінок, який має значущий вплив на їхню швидкість та продуктивність. У контексті теми "Дослідження моделей та технологій завантаження сторінок у web-застосунках з метою визначення оптимізованих підходів рендерингу" розглянемо різні методи оптимізації великих зображень, спрямовані на прискорення

завантаження веб-застосунків.

Одним із основних методів оптимізації великих зображень є їхнє стиснення або компресія. Цей процес зменшує розмір файлів, не суттєво втрачаючи якість візуального відображення. Існують два основних типи компресії: безвтратна, яка використовується для збереження всієї інформації без втрат, та втратна, яка призводить до деякого погіршення якості.

Метод адаптивного завантаження полягає в тому, щоб завантажувати зображення, яке оптимально підходить для конкретного витягуваного обсягу екрану та пристрою. За допомогою медіа-запитів та тегу <picture> можна вибирати різні варіанти зображень залежно від розміру екрану та роздільної здатності, тим самим зменшуючи обсяг передаваних даних.

Використання ефективних форматів зображень, таких як WebP, є ще одним способом оптимізації. WebP надає високу якість при низькому розмірі файлу, що дозволяє прискорити завантаження сторінок. Крім того, зазначення інших форматів, таких як JPEG 2000 чи AVIF, також може сприяти ефективній оптимізації.

Застосування відкладеного завантаження (Lazy Loading) зображень передбачає завантаження лише тих зображень, які потрібні на поточній сторінці, а не всіх разом при завантаженні сторінки. Це дозволяє прискорити ініціальне завантаження та зменшити обсяг передаваних даних, оскільки лише видимі зображення завантажуються відразу.

Важливим аспектом оптимізації великих зображень є вилучення зайвого або непотрібного коду. Деякі редактори зображень автоматично додають додаткову інформацію або метадані, які можуть бути зайвими. Це також може включати вилучення метаданих з самого HTML-коду, таких як атрибути width та height, які можуть бути автоматично визначені.

Для ефективної оптимізації можна використовувати різноманітні інструменти та сервіси, які спеціалізуються на обробці та оптимізації зображень. Такі інструменти автоматично виконують різні операції, включаючи стиснення та конвертацію форматів.

Загалом, величезний обсяг інформації та зображень на веб-сторінці може впливати на її швидкість завантаження. Використання вищезгаданих методів допомагає ефективно оптимізувати завантаження великих зображень, забезпечуючи при цьому кращий досвід для користувачів та підвищуючи продуктивність веб-застосунків.

2.5 Використання кешування на клієнтському та серверному рівнях для покращення швидкодії

Оптимізація завантаження сторінок є ключовим завданням веб-розробників, і одним із ефективних методів досягнення цієї мети є використання кешування. Кешування може бути реалізоване на різних рівнях - як на клієнтському, так і на серверному. У контексті теми "Дослідження моделей та технологій завантаження сторінок у web-застосунках з метою визначення оптимізованих підходів рендерингу" розглянемо різновиди та переваги кешування на обох рівнях.

Локальне кешування відбувається безпосередньо на пристрої користувача. Використовуючи локальне сховище, таке як Local Storage чи Cache API, веб-застосунок може зберігати ресурси, такі як стилі, скрипти чи зображення. Кешування на клієнтському рівні дозволяє значно зменшити час завантаження сторінки при повторних відвідуваннях.

Service Workers - це скрипти, які працюють в фоновому режимі та можуть керувати подіями, такими як запити на сервер та відповіді. Вони дозволяють встановлювати стратегії кешування для різних типів ресурсів, наприклад, можна налаштувати кешування зображень або сторінок.

Користувачі мають контроль над тим, як і коли вони хочуть очищати кеш. Зазвичай браузери надають можливість вручну видаляти кеш для окремих ресурсів або цілком очищати його. Це може бути корисно в ситуаціях, коли виникають проблеми з оновленням сторінок.

Content Delivery Network (CDN) використовується для розподілення контенту на сервера, що розташовані у різних регіонах. Крім цього, CDN може кешувати ресурси і надавати їх з ближчого сервера, що суттєво зменшує час завантаження

для користувачів з різних частин світу.

Серверне кешування забезпечує збереження готових відображень веб-сторінок на сервері. Це дозволяє відправляти попередньо згенеровані сторінки користувачам, замість того, щоб генерувати їх знову для кожного запиту. Такий підхід ефективно зменшує навантаження на сервер та прискорює обробку запитів.

Кешування на клієнтському та серверному рівнях грає важливу роль у покращенні швидкодії веб-застосунків. Використання локального кешування та Service Workers забезпечує ефективне управління ресурсами на стороні користувача. У той час як CDN та серверне кешування роблять важливий внесок у прискорення завантаження сторінок для користувачів з усього світу. У комплексі ці стратегії кешування стають потужним інструментом для досягнення оптимальної швидкодії веб-застосунків та поліпшення користувацького досвіду.

2.6 Переваги та недоліки різних моделей завантаження сторінок

Сучасний ландшафт веб-розробки вимагає уваги до ефективності завантаження сторінок веб-застосунків. Оптимізація цього процесу включає в себе ряд технік, які впливають на загальну продуктивність та користувацький досвід. Однією з ключових стратегій є використання Lazy Loading.

Lazy Loading — це техніка, що дозволяє відкладати завантаження певних ресурсів (зображення, скрипти, стилі) до моменту їх реальної потреби. Це знижує час завантаження сторінки та обсяг передаваних даних, що особливо актуально в умовах обмеженого інтернет-з'єднання.

Content Delivery Network (CDN) відіграє важливу роль у прискоренні завантаження веб-застосунків. Це глобальна мережа серверів, розташованих в різних регіонах світу, яка забезпечує швидкий доступ до контенту для кінцевого користувача. Використання CDN дозволяє зменшити час завантаження сторінок та покращити їхню доступність для користувачів у різних частинах світу.

Великі зображення можуть суттєво уповільнювати час завантаження веб-сторінок. Оптимізація зображень включає в себе стиснення, використання форматів з втратами та визначення оптимального розміру для відображення на

конкретних екранах. Це допомагає забезпечити швидке завантаження сторінок та знизити споживання трафіку.

Кешування грає ключову роль у поліпшенні швидкодії завантаження сторінок. Кешування на клієнтському рівні дозволяє зберігати локальні копії ресурсів, тим самим зменшуючи час завантаження при повторних відвідуваннях. Кешування на сервері дозволяє зменшити обчислювальне навантаження та прискорити формування вмісту для користувачів.

Обираючи модель завантаження сторінок, розробники стикаються з рядом переваг та недоліків. SSR забезпечує швидке відображення на першому перегляді, тоді як CSR дозволяє покращити взаємодію під час подальшого використання. Застосування гібридних підходів може бути вигідним в різних сценаріях, але вимагає додаткового управління та реалізації.

Використання SSR для перших переглядів і CSR для подальших взаємодій дозволяє оптимізувати час завантаження та SEO-показники. Однак складніше використання може виникнути при імплементації гібридного рендерингу, коли потрібно управляти різними типами рендерингу в залежності від умов використання.

Вибір моделі завантаження сторінок веб-застосунків визначає їхню ефективність та користувацький досвід. Ретельний аналіз та розуміння переваг та недоліків різних технік оптимізації дозволяє розробникам обирати оптимальні стратегії для конкретних випадків використання.

2.7 SSG та SSR

Статична генерація сторінок (SSG) та серверний рендерінг (SSR) є двома популярними методами для рендерингу веб-сторінок, кожен з яких має свої особливості та сфери застосування, що робить їх важливими інструментами в сучасній веб-розробці. SSG передбачає попереднє генерування сторінок на етапі збірки проекту. Це означає, що всі сторінки веб-сайту генеруються як статичні HTML-файли перед тим, як користувач навіть зробить запит. Коли користувач запитує певну сторінку, веб-сервер просто відправляє вже готовий HTML-файл, що

дозволяє швидко завантажувати сторінку без додаткової обробки на сервері або на клієнті. Цей метод ідеально підходить для сайтів, де контент не змінюється часто, таких як блоги або інформаційні портали.

З іншого боку, SSR забезпечує динамічний рендерінг сторінок, що відбувається на сервері в момент, коли користувач робить запит. Це означає, що сервер обробляє запит, генерує відповідний HTML на основі останніх даних і відправляє його клієнту [10]. Такий підхід забезпечує актуальність контенту, адже всі дані на сторінці є найсвіжішими на момент відвідування. SSR є більш придатним для динамічних застосунків, де контент регулярно оновлюється, як наприклад, в електронній комерції або інтерактивних сервісах, де вміст має бути налаштований під конкретного користувача або відповідати його взаємодії в реальному часі.

Обидва методи, SSG та SSR, використовують сучасні веб-технології для оптимізації продуктивності та користувацького досвіду. Вибір між SSG та SSR залежить від специфічних потреб проекту та його контенту. Наприклад, використання SSG може значно знизити навантаження на сервер та зменшити час відповіді на запити користувачів, що важливо для сайтів з високим трафіком. В той же час, SSR забезпечує більшу гнучкість у забезпеченні персоналізованого контенту, але може вимагати більших обчислювальних ресурсів та оптимізації для забезпечення швидкого завантаження сторінок.

3 ВИБІР МЕТОДУ РЕНДЕРИНГУ ПІД СЦЕНАРІЙ ВИКОРИСТАННЯ

Вибір методу рендерінга для веб-додатків є важливим етапом розробки, оскільки правильний підхід може значно покращити продуктивність, швидкість завантаження та загальний користувацький досвід. Для статичного контенту, де зміни відбуваються рідко і вміст залишається постійним протягом тривалого часу, оптимальним вибором є Static Site Generation (SSG). Цей метод дозволяє попередньо генерувати статичні сторінки на етапі збору проекту, що призводить до значного зменшення часу завантаження. Використання SSG забезпечує високу продуктивність, оскільки готові HTML-файли можна безпосередньо доставляти клієнту без необхідності додаткової обробки на сервері. Це також поліпшує безпеку, оскільки зменшується кількість динамічних взаємодій, які можуть стати мішенню для атак.

Для контенту, який частково або повністю динамічний, більш підходящим варіантом є Server-Side Rendering (SSR). Використання SSR дозволяє генерувати сторінки на сервері для кожного запиту, забезпечуючи оновлення контенту в реальному часі. Це особливо важливо для веб-додатків, де вміст часто змінюється або залежить від дій користувача. З SSR користувач отримує актуальну інформацію з кожним запитом, що покращує взаємодію та задовольняє потреби в оновлюваному контенті. Крім того, цей метод покращує SEO, оскільки пошукові системи можуть легше індексувати готові сторінки, що позитивно впливає на видимість сайту в пошукових результатах.

Коли мова йде про швидкість завантаження та взаємодію користувача, варто розглянути Client-Side Rendering (CSR). Цей метод передбачає завантаження базової HTML-структури та подальше використання JavaScript для динамічного відображення вмісту на стороні клієнта. CSR дозволяє створювати високовідгукливі веб-додатки, що забезпечують плавну взаємодію без перезавантаження сторінок. Це особливо ефективно для додатків з великою кількістю інтерактивних елементів, де швидкість реакції на дії користувача є критично важливою. Однак, варто враховувати, що CSR може мати негативний

вплив на SEO, оскільки початковий вміст може бути недоступним для пошукових роботів до виконання JavaScript.

Таким чином, вибір методу рендерінга для веб-додатків залежить від специфіки вмісту та вимог до продуктивності і SEO. SSG є відмінним вибором для статичного контенту з високою продуктивністю та швидким завантаженням. SSR підходить для динамічного контенту, забезпечуючи актуальність і оновлюваність, а також покращуючи SEO. CSR забезпечує швидкість завантаження та високу відгукливість для інтерактивних додатків, хоча може потребувати додаткових заходів для оптимізації SEO. Кожен з цих методів має свої переваги та недоліки, і вибір між ними залежить від конкретних потреб вашого проекту.

Важливість SEO для веб-додатків важко переоцінити, адже саме пошукова оптимізація є ключовим фактором для забезпечення видимості сайту в інтернеті. Веб-сайти, що не мають належної оптимізації, ризикують втратити значну кількість потенційних користувачів, оскільки пошукові системи не зможуть ефективно індексувати їхній контент. У цьому контексті методи рендерінга, такі як Server-Side Rendering (SSR) та Static Site Generation (SSG), мають вирішальне значення для покращення SEO.

SSR та SSG надають стабільний та індексований контент для пошукових систем, що істотно покращує SEO. Використання SSR дозволяє генерувати HTML-сторінки на сервері під час кожного запиту, надаючи користувачам і пошуковим роботам повністю сформований контент. Це означає, що весь вміст сторінки доступний одразу після завантаження, що полегшує процес індексації для пошукових систем. Таким чином, пошукові роботи можуть легко аналізувати та зберігати інформацію про сторінки, що покращує видимість сайту у пошукових результатах.

SSG, у свою чергу, також сприяє покращенню SEO, генеруючи статичні сторінки на етапі збору проекту. Це дозволяє створити повністю сформовані HTML-сторінки, які можуть бути миттєво завантажені користувачам та пошуковим роботам. Перевага SSG полягає в тому, що сторінки генеруються заздалегідь і не потребують додаткового часу для обробки на сервері під час запиту. Це забезпечує

високу продуктивність та швидке завантаження, що позитивно впливає на користувацький досвід і, відповідно, на рейтинг у пошукових системах.

Важливим аспектом для SEO є можливість стабільно індексувати контент. З SSR та SSG пошукові системи можуть отримувати доступ до повноцінних HTML-сторінок без необхідності виконання JavaScript, що є критичним для ефективної індексації. Це особливо важливо для динамічних веб-додатків, де вміст часто змінюється або залежить від дій користувача. Використання цих методів дозволяє забезпечити стабільний доступ до оновлюваного контенту, покращуючи видимість сайту у пошукових результатах.

Крім того, SSR та SSG можуть значно поліпшити технічну SEO-оптимізацію, зокрема оптимізацію мета-тегів, заголовків та описів. Генерація повноцінних HTML-сторінок дозволяє більш точно контролювати та оптимізувати ці елементи, що сприяє покращенню рейтингу у пошукових системах. Оптимізація зображень, використання структурованих даних та покращення швидкості завантаження також є важливими аспектами, які можна ефективно реалізувати за допомогою SSR та SSG.

Отже, вибір методу рендерінга, такого як SSR або SSG, має вирішальне значення для SEO-оптимізації веб-додатків. Ці методи забезпечують створення стабільного та індексованого контенту, що покращує видимість сайту у пошукових системах і сприяє залученню більшої кількості користувачів. Ефективне використання SSR та SSG дозволяє досягти високої продуктивності, швидкого завантаження та технічної оптимізації, що є ключовими факторами для успішної SEO-стратегії.

Вибір Server-Side Rendering (SSR) для конкретного веб-додатку є стратегічно обґрунтованим рішенням, яке враховує кілька ключових аспектів, таких як актуальність та оновлюваність контенту, покращення SEO, використання сучасних технологій, зокрема React та Next.js, а також створення серверних компонентів для досягнення оптимальної продуктивності та швидкого завантаження сторінок. Актуальність та оновлюваність контенту є критичними для багатьох веб-додатків, особливо тих, що надають користувачам постійно оновлювану інформацію, новини

чи дані в реальному часі. Використання SSR дозволяє динамічно генерувати сторінки на сервері для кожного запиту, що забезпечує доступ до найсвіжішого контенту без затримок. Це особливо важливо для додатків, які залежать від швидкого реагування на зміни, забезпечуючи користувачам актуальну інформацію та покращуючи загальний користувацький досвід.

Покращення SEO є ще однією вагомою причиною для вибору SSR. Пошукові системи надають перевагу сайтам, які можуть надавати повноцінний HTML-контент без необхідності виконання JavaScript. З SSR всі сторінки генеруються на сервері, що забезпечує їхню миттєву доступність для пошукових роботів. Це значно покращує видимість сайту в пошукових результатах, сприяє кращій індексації та, відповідно, залучає більше органічного трафіку. Завдяки цьому веб-додаток може досягти вищих позицій у пошукових системах, що є критично важливим для його успіху.

Використання React та Next.js для реалізації SSR є сучасним підходом, який дозволяє максимально ефективно використовувати можливості серверного рендерінгу. React, завдяки своїй компонентній архітектурі, надає потужні інструменти для створення динамічних та інтерактивних користувацьких інтерфейсів. Next.js, у свою чергу, спрощує процес інтеграції SSR у проект, забезпечуючи розширені можливості для налаштування серверного рендерінгу. Ця комбінація дозволяє розробникам легко створювати додатки з високою продуктивністю та відмінним користувацьким досвідом.

Створення серверних компонентів для оптимальної продуктивності та швидкого завантаження сторінок є важливим аспектом вибору SSR. Серверні компоненти дозволяють генерувати контент безпосередньо на сервері, що зменшує навантаження на клієнтську частину та забезпечує швидке завантаження сторінок. Це не тільки покращує загальний користувацький досвід, але й позитивно впливає на продуктивність додатку, зменшуючи час завантаження та оптимізуючи використання ресурсів. В результаті користувачі отримують швидкий та плавний досвід взаємодії з додатком, що є ключовим фактором для його успіху.

Таким чином, вибір SSR для конкретного веб-додатку базується на

необхідності забезпечення актуальності та оновлюваності контенту, покращення SEO, використання передових технологій, таких як React та Next.js, а також створення серверних компонентів для досягнення оптимальної продуктивності та швидкого завантаження сторінок. Це комплексне рішення дозволяє створювати ефективні, продуктивні та конкурентоспроможні веб-додатки, які задовольняють потреби сучасних користувачів та відповідають високим стандартам веб-розробки.

Server-Side Rendering (SSR) має численні переваги, які роблять його привабливим вибором для сучасних веб-додатків. Однією з основних переваг є висока відгукливість та швидкість завантаження сторінок. Генерація контенту на сервері дозволяє доставляти користувачам вже готові HTML-сторінки, що значно скорочує час їх завантаження. Це особливо важливо для користувачів з повільнішим інтернет-з'єднанням або менш потужними пристроями, де кожна секунда затримки може призвести до втрати користувачів. Завдяки цьому веб-додатки стають більш доступними та привабливими, забезпечуючи швидкий та плавний досвід взаємодії.

Ще однією важливою перевагою SSR є легкість управління станом додатку. Серверне рендерінг дозволяє централізовано управляти станом додатку, що значно спрощує його підтримку та розширення. Веб-додатки, побудовані з використанням SSR, можуть ефективніше використовувати ресурси сервера для обробки складних бізнес-логік та управління даними, залишаючи клієнтську частину більш легкою та швидкою. Це також дозволяє розробникам зосередитися на створенні більш інтуїтивних та інтерактивних користувацьких інтерфейсів, не турбуючись про складнощі управління станом на клієнтській стороні.

Підтримка індексації контенту для соціальних мереж є ще однією суттєвою перевагою SSR. Генерація повноцінних HTML-сторінок на сервері дозволяє забезпечити коректне відображення та роботу мета-тегів для соціальних мереж, що важливо для поділу контенту та привертання нових користувачів. Це означає, що коли користувачі діляться посиланнями на ваш контент у соціальних мережах, інші користувачі отримують привабливі попередні перегляди з правильними заголовками, описами та зображеннями. Така підтримка покращує залучення нових

відвідувачів і сприяє зростанню аудиторії вашого веб-додатку. Вигляд створеної екосистеми при розгортанні Next.js пакетів (див. рис. 3.1)

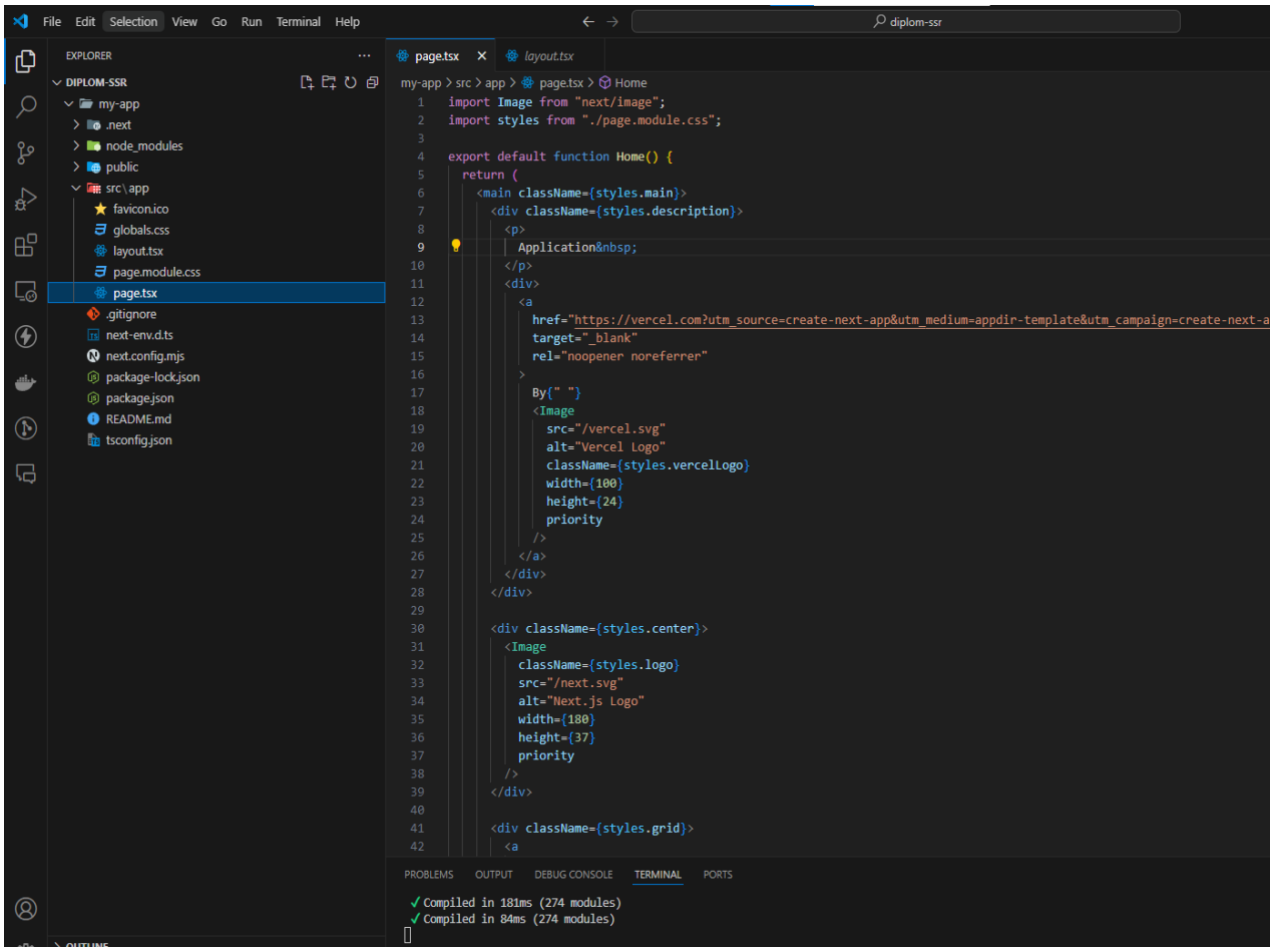

Рисунок 3.1 – Вигляд нового Next.js додатку (виконаний самостійно)

Покращення продуктивності через ефективне використання ресурсів сервера є ще однією значущою перевагою SSR. Серверне рендерінг дозволяє більш раціонально використовувати серверні потужності для обробки запитів та генерування контенту, зменшуючи навантаження на клієнтські пристрої. Це особливо важливо для веб-додатків з великою кількістю одночасних користувачів, де ефективне використання серверних ресурсів може значно підвищити загальну продуктивність системи. Завдяки цьому веб-додатки стають більш масштабованими та здатними обслуговувати більшу кількість користувачів без погіршення якості обслуговування.

Легкість інтеграції зі сторонніми сервісами та додатками також є важливою перевагою SSR. Серверне рендерінг спрощує процес інтеграції з різноманітними

API та зовнішніми сервісами, дозволяючи швидко та ефективно обробляти дані на сервері та надавати їх користувачам. Це відкриває нові можливості для розширення функціональності веб-додатків, включаючи інтеграцію з сервісами аналітики, платіжними системами, маркетинговими інструментами та багатьма іншими. Завдяки цьому веб-додатки можуть надавати користувачам більше цінних можливостей та забезпечувати більш повний та зручний користувацький досвід.

Таким чином, переваги серверного рендерингу (SSR) включають високу відгукливість та швидкість завантаження сторінок, що є критично важливим для сучасних веб-додатків, які прагнуть забезпечити найкращий досвід користувача. Крім того, SSR спрощує управління станом додатку, оскільки весь вміст обробляється на сервері, що може значно полегшити роботу розробників. Важливо також відзначити, що SSR покращує можливості індексації контенту в пошукових системах та соціальних мережах, що є надзвичайно корисним для підвищення видимості та органічного трафіку.

Крім перелічених переваг, SSR також сприяє підвищенню загальної продуктивності додатку через більш ефективне використання ресурсів сервера. Це дозволяє додаткам швидше обробляти запити та зменшувати час завантаження для кінцевих користувачів. Ще однією значущою перевагою є легкість інтеграції зі сторонніми сервісами та додатками, що робить SSR ідеальною технологією для розробки багатофункціональних веб-систем. У сукупності, ці переваги роблять SSR вкрай привабливим вибором для розробки сучасних веб-додатків, які прагнуть досягти високої ефективності, швидкості та зручності використання.

4 РОЗРОБКА ІЗ ВИКОРИСТАННЯМ SSR

4.1 Опис сучасного методу розробки

Вибір технологій для розробки веб-сайту з використанням Server-Side Rendering (SSR) є важливим етапом, який визначає ефективність та продуктивність кінцевого продукту. Використання фреймворків та бібліотек, таких як React та Next.js, забезпечує потужний інструментарій для роботи з SSR. React, завдяки своїй компонентній архітектурі, дозволяє створювати динамічні та інтерактивні користувацькі інтерфейси, тоді як Next.js спрощує процес інтеграції SSR у проект, додаючи функціональність для серверного рендерінгу та статичного генерування сторінок.

Створення та налаштування проекту починається з встановлення необхідних пакетів та конфігурації середовища для забезпечення відповідності вимогам SSR. Це включає встановлення бібліотек React та Next.js, налаштування конфігураційних файлів та підготовку проекту до роботи з серверним рендерінгом.

Розробка React-компонентів є наступним кроком у процесі створення веб-сайту. Це передбачає створення реюзабельних UI-елементів та компонентів, які можуть бути використані як на стороні клієнта, так і на сервері. Такі компоненти повинні бути розроблені з урахуванням можливості їх використання у різних частинах додатку, що дозволяє забезпечити їх ефективність та зручність використання. Використання статичного генерування сторінок, яке дозволяє попередньо створювати HTML для певних сторінок під час збору проекту, також є важливим аспектом. Це допомагає зменшити навантаження на сервер та покращує швидкість завантаження сторінок, що позитивно впливає на користувацький досвід.

Наступним кроком є налаштування серверної частини для обробки запитів та виконання SSR. Це включає розробку механізмів для передачі даних з сервера до клієнта, забезпечуючи повноцінний SSR. Важливо налаштувати сервер таким чином, щоб він міг ефективно обробляти запити та генерувати необхідний контент у відповідь на кожен запит користувача. Оптимізація продуктивності також є

критичним етапом, який включає використання методів кешування, стиснення ресурсів та інших оптимізаційних технік. Це допомагає забезпечити швидке завантаження сторінок та ефективну роботу веб-сайту на різних пристроях та браузерах.

Тестування веб-додатку є невід'ємною частиною процесу розробки, яке включає перевірку роботи React-компонентів, оцінку SEO-оптимізації та загальної продуктивності веб-сайту [11]. Це дозволяє переконатися, що додаток працює коректно та ефективно на всіх етапах взаємодії з користувачами.

Після завершення розробки та тестування веб-сайт готовий до розгортання на сервері. Вибір надійного середовища для розгортання, такого як хмарні платформи або власний сервер, є важливим для забезпечення стабільної роботи веб-сайту [12]. Регулярне оновлення та підтримка веб-сайту після розгортання включають виправлення помилок, оновлення залежностей та розширення функціональності. Використання JavaScript та TypeScript у розробці веб-додатків з React та Next.js має численні переваги.

TypeScript забезпечує статичну типізацію, яка дозволяє виявляти помилки на етапі розробки, покращуючи загальну безпеку та надійність коду. Це особливо корисно при розробці великих та складних проектів, де точність та стабільність є важливими аспектами [13]. Додатково, TypeScript підвищує читабельність коду та забезпечує активну підтримку в інтегрованих середовищах розробки (IDE), що полегшує роботу розробників та підвищує ефективність процесу кодування. Використання TypeScript також сприяє підвищенню продуктивності, зменшуючи кількість помилок та дозволяючи розробникам більш впевнено вносити зміни та розширювати функціональність проекту. Це полегшує масштабування великих проектів та забезпечує зрозумілість і легкість управління кодом у майбутньому, що є особливо важливим для сучасного веб-розвитку.

4.2 Схема створення додатків з використанням SSR

Спрощення процесу розробки є однією з головних переваг використання сучасних фреймворків та бібліотек, таких як React та Next.js. Завдяки готовим

рішенням, багато типових задач, з якими стикаються розробники, вже вирішені. Це дозволяє їм зосередитися на логіці та функціональності своїх програм, замість того, щоб витрачати час на реалізацію базових функцій. Потужність та популярність інструментів, таких як React та Next.js, які розроблені та підтримувані Facebook, забезпечують високу продуктивність та надійність. Вони добре задокументовані, що робить їх доступними та зрозумілими для розробників по всьому світу. Компонентний підхід у React дозволяє створювати багаторазові компоненти, що полегшує розробку та підтримку великих додатків. Це забезпечує можливість повторного використання коду, зменшуючи кількість помилок та підвищуючи ефективність розробки. Висока продуктивність React досягається завдяки віртуальному DOM, який мінімізує операції з реальним DOM. Це дозволяє додаткам швидко реагувати на зміни та забезпечує плавну взаємодію користувача з інтерфейсом.

Можливості Next.js включають автоматичне розділення коду, генерацію статичних сайтів та серверний рендеринг. Це робить Next.js відмінним вибором для створення високопродуктивних та SEO-оптимізованих додатків. Автоматичне розділення коду дозволяє зменшити розмір завантажуваних файлів, що прискорює завантаження сторінок. Генерація статичних сайтів забезпечує швидке завантаження та високу продуктивність, особливо для сторінок з постійним вмістом. Серверний рендеринг покращує SEO, надаючи пошуковим системам готові HTML-сторінки для індексації. Велика спільнота користувачів та розробників, які використовують React та Next.js, забезпечує безліч навчальних матеріалів, прикладів коду та готових рішень для різних задач. Це значно полегшує навчання та використання цих інструментів, а також надає можливість отримати швидку підтримку у разі виникнення проблем.

Прискорення розробки завдяки використанню цих інструментів дозволяє розробникам швидше створювати якісні продукти. Вони можуть використовувати найкращі практики та інструменти, розроблені спільнотою, що забезпечує високу якість та надійність кінцевого продукту. Використання завчасно створених пакетів та бібліотек сприяє значному скороченню часу розробки, дозволяючи зосередитися

на унікальних аспектах проекту та досягти швидшого випуску продукту на ринок. Таким чином, поєднання потужних інструментів, великої спільноти та компонентного підходу робить процес розробки веб-додатків ефективним, продуктивним та зручним для розробників (див. рис. 4.1).

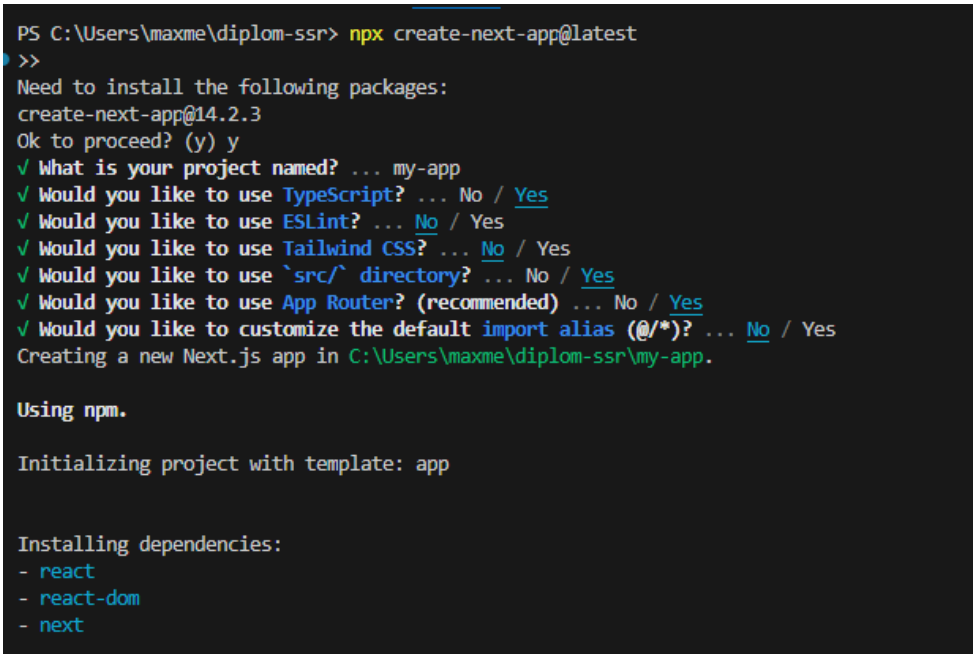

Рисунок 4.1 – Створення Next.js додатку через npm (виконаний самостійно)

Для створення веб-додатку з використанням Next.js необхідно спочатку встановити необхідні пакети та налаштувати проект. Перший крок – переконатися, що на вашому комп'ютері встановлені Node.js та npm (Node Package Manager). Якщо вони ще не встановлені, їх можна завантажити та встановити з офіційного сайту Node.js. Node.js забезпечує виконання JavaScript на сервері, а npm дозволяє легко керувати пакетами та залежностями проекту. Після встановлення Node.js та npm відкрийте командний рядок або термінал і скористайтеся командою для створення нового проекту на базі Next.js. Наприклад, команда npx create-next-app my-app дозволяє швидко налаштувати новий проект зі стандартними налаштуваннями. У процесі створення проекту Next.js автоматично генерує всі необхідні файли та структури для початкового додатку, включаючи файли конфігурації, початкові сторінки, компоненти та інші ресурси.

Наступний крок – запуск сервера розробки. Після створення проекту

перейдіть у папку проекту за допомогою команди cd my-app і запустіть сервер розробки командою npm run dev. Це дозволяє побачити додаток у дії за адресою http://localhost:3000. Відкрийте веб-браузер і перейдіть за цією адресою, щоб побачити початкову сторінку вашого Next.js додатку. Це базовий шаблон, який можна використовувати як відправну точку для подальшої розробки.

Для додавання нових сторінок та компонентів можна створювати нові файли у папці pages. Наприклад, щоб створити сторінку "About Us", створіть файл about.js у папці pages, де визначте простий React-компонент, який відображатиме заголовок та текст на сторінці "About Us". Після збереження цього файлу ви зможете відвідати сторінку за адресою http://localhost:3000/about і побачити новий контент. Next.js також підтримує динамічні маршрути, що дозволяє створювати сторінки з параметрами у URL. Наприклад, можна створити сторінки для окремих статей блогу, користувачів або продуктів, використовуючи шаблонні файли та динамічні маршрути.

Можливості Next.js включають серверний рендеринг, статичну генерацію сайтів, оптимізацію продуктивності та багато інших потужних функцій. Серверний рендеринг покращує SEO, оскільки сторінки генеруються на сервері і надаються пошуковим системам у готовому вигляді. Статична генерація дозволяє попередньо створювати HTML для певних сторінок під час збірки проекту, що зменшує навантаження на сервер та покращує швидкість завантаження сторінок. Оптимізація продуктивності включає методи кешування та стиснення ресурсів, що забезпечує швидке завантаження сторінок і ефективну роботу веб-сайту на різних пристроях та браузерах.

Переваги використання Next.js полягають у тому, що готові інструменти та бібліотеки значно спрощують процес розробки, дозволяючи зосередитися на створенні унікального контенту та функціоналу. Велика спільнота користувачів та розробників забезпечує доступ до багатьох навчальних матеріалів, прикладів коду та готових рішень для різних задач. Це робить Next.js відмінним вибором для розробників будь-якого рівня, від новачків до професіоналів. Завдяки високій продуктивності та оптимізації для пошукових систем, Next.js забезпечує створення

сучасних веб-додатків, які можуть масштабуватися і задовольняти потреби будь-якого проекту. Рисунок 4.2 демонструє, як виглядає ця сторінка у веб-браузері при запуску серверу на локальному хості.

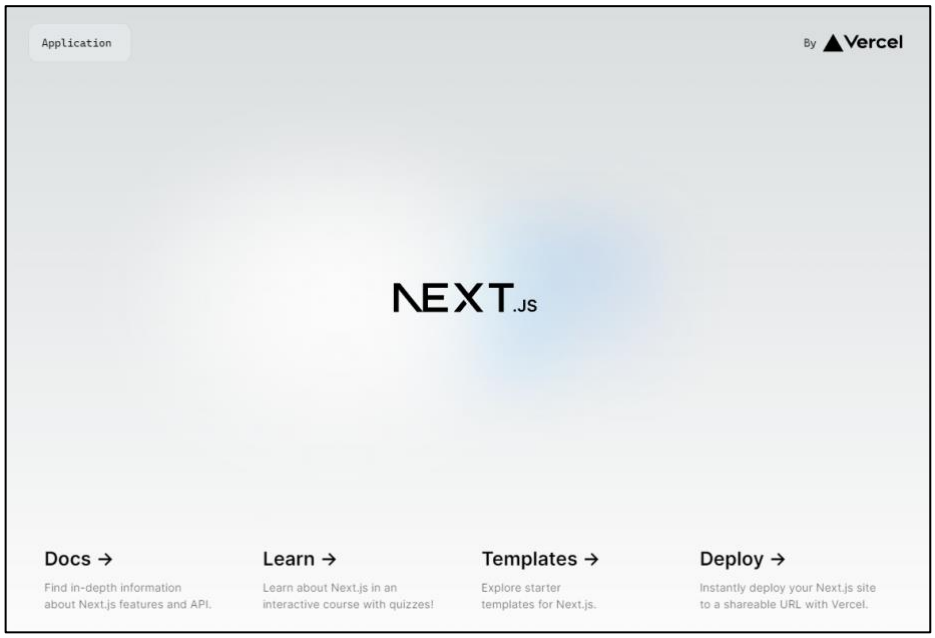

Рисунок 4.2 – Результат створеного додатку (виконаний самостійно)

На даний момент наш додаток працює у форматі Static Site Generation (SSG), що означає, що всі сторінки генеруються заздалегідь і зберігаються як статичні HTML-файли. Це дозволяє нам забезпечити швидке завантаження сторінок для користувачів, оскільки контент вже згенерований та готовий до відображення без необхідності додаткових обчислень на стороні сервера під час кожного запиту. Завдяки SSG ми також досягаємо покращеної SEO-оптимізації, оскільки пошукові системи можуть легко індексувати весь контент, що позитивно впливає на видимість нашого додатку в результатах пошуку.

Проте, існують сценарії, де нам необхідно динамічно завантажувати дані або рендерити сторінки на сервері залежно від запиту користувача. Наприклад, у випадках, коли вміст змінюється часто або залежить від користувацьких дій, статичне генерування не завжди є оптимальним рішенням. У таких ситуаціях ми можемо використовувати інші методи рендерінгу, які пропонує Next.js, для забезпечення гнучкості та динамічності нашого додатку.

Next.js надає декілька альтернативних методів рендерінгу, що дозволяють

задовольнити специфічні потреби нашого проекту. Один із них – це Server-Side Rendering (SSR), при якому сторінки генеруються на сервері під час кожного запиту. Це дозволяє нам забезпечити актуальний контент у реальному часі, що особливо корисно для додатків з динамічним вмістом або частими оновленнями. Інший метод – Incremental Static Regeneration (ISR), який дозволяє нам поєднувати переваги SSG та SSR. З ISR ми можемо генерувати статичні сторінки, які будуть автоматично оновлюватися на сервері через певні інтервали часу або за запитом, забезпечуючи свіжий контент без втрати продуктивності.

Вибір правильного методу рендерінгу залежить від специфіки нашого додатку та вимог до продуктивності і актуальності контенту. Використовуючи комбінацію цих методів, ми можемо забезпечити оптимальну роботу нашого веб-додатку, задовольняючи потреби як користувачів, так і пошукових систем. Завдяки гнучкості Next.js, ми можемо ефективно управляти рендерингом сторінок, обираючи найкращий підхід для кожного конкретного випадку, що дозволяє нам створювати продуктивні, SEO-оптимізовані та динамічні веб-додатки.

На рисунку 4.3 зображено код стандартного підходу до реалізації SSR.

```jsx
import React from 'react';

// Функція для отримання статичних пропсів
export async function getStaticProps() {
  // Отримання даних, які будуть передані як пропси

  const data = await fetchDataFromAPI();

  return {
    props: {
      data,
    },
  };
}

// Компонент, який використовує статичні пропси
const MyPage = ({ data }) => {
  return (
    <div>
      <h1>My Static Page</h1>
      <p>{data.content}</p>
    </div>
  );
};

export default MyPage;
```

Рисунок 4.3 – підхід SSG у Next.js (виконаний самостійно)

Тепер нам потрібно це змінити на підхід SSR, для цього необхідно змінити функції поучення серверних пропсів (див. рис. 4.4).

```
import React from 'react';

// Функція для отримання серверних пропсів
export async function getServerSideProps() {
  // Отримання даних на сервері
  const data = await fetchDataFromAPI();

  return {
    props: {
      data,
    },
  };
}

// Компонент, який використовує серверні пропси
const MyPage = ({ data }) => {
  return (
    <div>
      <h1>My Server-Side Rendered Page</h1>
      <p>{data.content}</p>
    </div>
  );
};

export default MyPage;
```

Рисунок 4.4 – підхід SSR у Next.js (виконаний самостійно)

Використання цієї функції дозволяє забезпечувати постійне оновлення даних на сторінці, що є невід'ємною частиною багатьох сучасних застосунків, особливо коли мова йде про дашборди, профілі користувачів, новинні портали або блоги. Ця можливість важлива для адміністративних панелей і дашбордів, де відображаються статистика, метрики та інші важливі параметри системи, які потрібно постійно оновлювати.

На прикладі, представленому на рисунку, ми бачимо, як у коді функція getServerSideProps виконує завантаження даних з зовнішнього API, що забезпечує не тільки актуальність інформації, але і підвищує інтерактивність сайту, роблячи взаємодію з ним більш персоналізованою та відповідною до потреб користувача.

Такий підхід також сприяє інтеграції з зовнішніми API, забезпечуючи безперервний потік актуальної інформації, що критично важливо для сайтів, де вміст швидко змінюється або де необхідне відображення найновіших даних.

Вивчаючи приклад використання функції getServerSideProps у React, ми виявили, як ця функція стала ключовим елементом у забезпеченні динамічного оновлення контенту на веб-сторінках. Вона була викликана на кожен запит до сторінки, що дозволяло завжди містити актуальні дані. Ця особливість була особливо важливою для дашбордів та профілів користувачів, а також для сторінок з часто оновлюваним контентом, таких як новини або блоги.

В цьому контексті, завдяки getServerSideProps, сторінки завжди відображали найсвіжішу інформацію, що було критично важливим для адміністративних панелей та дашбордів, де постійно оновлювалась статистика та метрики. Завантаження найновіших даних про користувачів під час кожного запиту робило взаємодію з сайтом більш інтерактивною та персоналізованою.

Крім того, інтеграція з зовнішніми API за допомогою цієї функції забезпечувала відображення найактуальнішої інформації, сприяючи ефективності та релевантності контенту. Це дозволило веб-сайтам не тільки підтримувати високий рівень актуальності даних, але й значно покращувати загальний досвід користувачів.

4.3 Експериментальне порівняння підходів до рендерингу

Порівняння швидкості рендерингу з використанням серверного рендерингу (SSR) може надати нам цінну інформацію про ефективність веб-застосунків, зокрема як швидко вони можуть відповідати на запити користувачів та забезпечувати актуальний контент. SSR дозволяє серверу обробити всю інформацію та згенерувати веб-сторінку до її відправлення клієнту, що може

значно прискорити відображення сторінки, оскільки більшість даних уже оброблені і не потрібно чекати завершення завантаження JavaScript та інших ресурсів [14]. Такий підхід особливо корисний для динамічних застосунків, де важливо швидко оновлювати інформацію, наприклад, в новинних порталах або платформах електронної комерції, де затримки в завантаженні можуть вплинути на досвід користувачів та втрату потенційних продажів.

Водночас, порівняння може виявити та підкреслити недоліки SSR, зокрема його вплив на продуктивність сервера при високому навантаженні, що може стати критичним для великих застосунків з великою кількістю користувачів. В такому випадку, сервер може перевантажуватися, відповідно підвищуючи загальний час відгуку та знижуючи загальну продуктивність. Порівняння може також показати, як альтернативні підходи, такі як статична генерація (SSG) або клієнтський рендеринг (CSR), можуть зменшити навантаження на сервер і покращити загальну швидкість застосунків за рахунок передачі частини обробки на клієнтський бік або завчасної підготовки сторінок.

Щоб ефективно використовувати сильні сторони різних методів рендерингу, можна застосувати гібридні стратегії. Одним із таких підходів є комбінування статичної генерації (SSG) з Інкрементною Статичною Регенерацією (ISR), яке дозволяє періодично оновлювати статично згенеровані сторінки. Це допомагає підтримувати контент актуальним, не призводячи до зайвого навантаження на сервер. Ще одна стратегія полягає у використанні клієнтського рендерингу (CSR) для обробки менш важливих сегментів контенту. Це звільняє серверні ресурси для опрацювання більш критичних запитів, сприяючи оптимізації загальної продуктивності системи. Таким чином, використання гібридних технік дозволяє досягти оптимального балансу між швидкістю, ефективністю та актуальністю веб-додатків.

Отже далі подивимось на детальний аналіз рендерингу на рисунку 4.5.

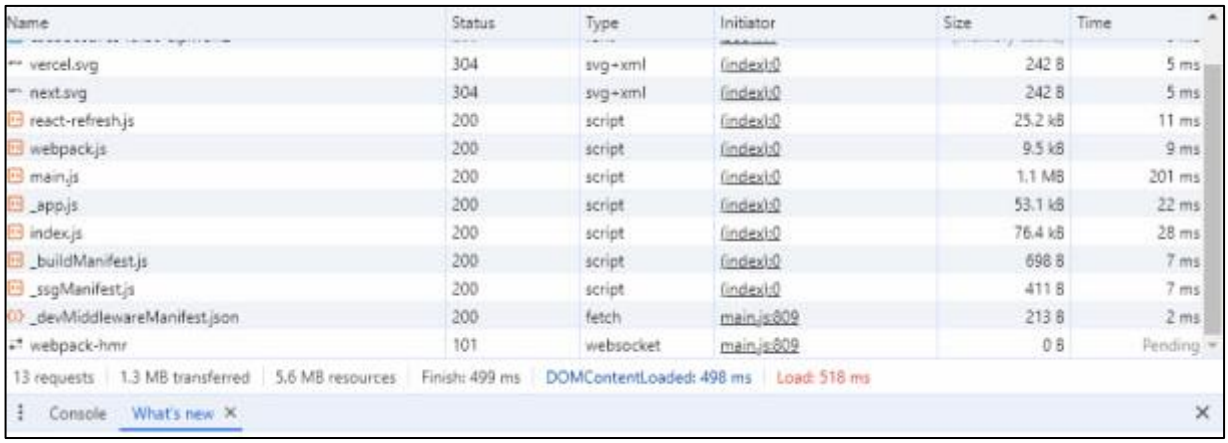

Рисунок 4.5 – Аналіз швидкості рендерінгу сторінки з використанням SSR (виконаний самостійно)

З результатів можемо зробити аналіз швидкості завантаження веб-сторінки, яка, ймовірно, використовує серверний рендеринг SSR. Тестування показує список запитів, їх статус, тип, ініціатор, розмір та час відповіді.

Загальний час завантаження: за результатами загальний час завантаження сторінки складає близько 518 мілісекунд, що є досить швидким для багатьох веб-застосунків, особливо коли врахувати, що DOMContentLoaded відбувається за 499 мілісекунд. Це свідчить про ефективність серверного рендерінгу у забезпеченні швидкого першого відображення.

Об'єми переданих даних та ресурсів: загалом було передано 1.3 МБ даних та використано 5.6 МБ ресурсів. Це може вказувати на те, що сторінка досить ресурсномістка або містить значну кількість скриптів і стилів.

main.js є найбільшим файлом (1.1 МБ) і має найбільший час завантаження (201 мс). Великий розмір і час завантаження цього скрипту може впливати на загальну продуктивність сторінки.

Файли, як react-refresh.js і webpack.js, мають малий час завантаження (9 мс і 9 мс відповідно), що вказує на ефективність кешування та передачі цих скриптів.

Натомість результат швидкості завантаження SSG зображено на рисунку 4.6.

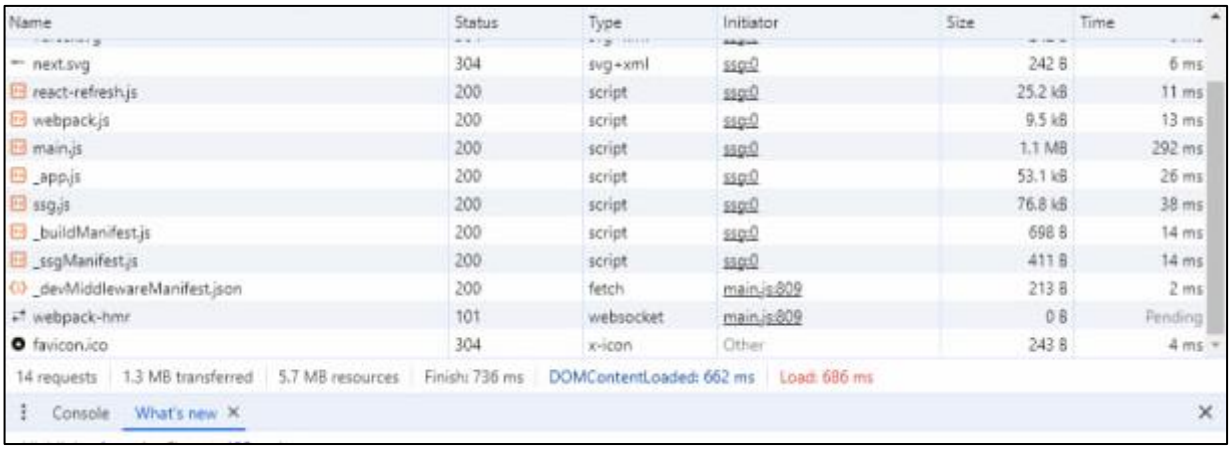

Рисунок 4.6 – Швидкість рендерінгу сторінки з використанням SSG (виконаний самостійно)

Загальний час завантаження сторінки становить 686 мілісекунд, а час, за який сторінка стала доступною для користувача (DOMContentLoaded), складає 662 мілісекунди. Ці показники свідчать про досить швидке завантаження, що є важливим фактором у забезпеченні хорошого користувацького досвіду.

За даними скріншоту, на сторінці загалом виконано 14 запитів, із загальним обсягом передачі даних 1.3 МБ та використанням 5.7 МБ ресурсів. Це вказує на наявність значної кількості контенту та скриптів, які обробляються на сторінці. Серед основних елементів, які споживають час при завантаженні, виділяється файл main.js з розміром 1.1 МБ і часом завантаження 292 мілісекунди, що є найбільшим індивідуальним вкладом у загальний час завантаження.

Порівняно з динамічним серверним рендерингом, статична генерація дозволяє знизити час відповіді сервера, оскільки більшість ресурсів генеруються заздалегідь і кешуються. Це забезпечує швидке завантаження сторінок навіть при високих навантаженнях, що особливо корисно для сайтів із великим об'ємом відвідувань.

Загалом, використання SSG на цьому проєкті забезпечує ефективне використання серверних ресурсів і зниження часу завантаження сторінок. Однак, існує потенціал для подальшої оптимізації, зокрема шляхом мінімізації та асинхронного завантаження JavaScript-файлів, що може зменшити обсяг передаваних даних і прискорити взаємодію користувачів зі сторінкою.

ВИСНОВКИ

Дослідження моделей та технологій завантаження сторінок у веб-застосунках з метою визначення оптимізованих підходів рендерингу виявилося невід'ємним етапом у сучасному веб-розвитку. Оптимізація процесу рендерингу є важливою для забезпечення ефективного та швидкого завантаження веб-сторінок, що в свою чергу впливає на користувацький досвід, SEO, а також загальний успіх веб-додатка. У цих висновках розглянемо ключові висновки та висвітлимо важливі практики, які можуть використовуватися для розробки оптимізованих веб-застосунків.

Вибір моделі рендерингу є стратегічним рішенням, яке визначає подальшу архітектуру веб-додатка. Дослідження SSR, SSG та CSR показує, що немає універсального рішення для всіх випадків. SSR може бути корисним для покращення SEO та зниження часу завантаження, SSG - для статичних контентних сторінок, а CSR - для динамічного контенту на боці клієнта. Важливо враховувати специфіку проекту та вимоги до продуктивності для вибору оптимальної моделі.

Ми провели ретельне дослідження стратегій рендерингу сторінок у веб-застосунках, що допомогло нам зрозуміти та оцінити важливість та вплив серверного рендерингу (SSR) та статичної генерації сторінок (SSG). Це дослідження виявило, що обидва підходи відіграють ключову роль у сучасному веб-розробстві, враховуючи їхні особливості та застосування для досягнення максимальної продуктивності та якості користувацького досвіду.

Особливу увагу ми приділили SSR через його здатність швидко обробляти та відображати контент, що критично для застосунків із високою динамікою оновлень, таких як електронна комерція чи інтерактивні платформи. Це сприяє не тільки поліпшенню взаємодії з користувачем, але й покращує SEO, оскільки пошукові системи отримують повністю сформовані сторінки, що сприяє кращій індексації.

Також ми оцінили переваги SSG, який забезпечує високу швидкість завантаження сторінок завдяки попередньому генеруванню HTML-файлів. Це особливо ефективно для сайтів зі сталим контентом, де частота оновлень є

мінімальною, але вимоги до швидкості завантаження високі.

На основі отриманих даних ми визначили, що інтеграція SSG та SSR може бути оптимальним рішенням для додатків, що вимагають гнучкості в обслуговуванні великої кількості користувачів із різними потребами. Такий гібридний підхід дозволяє максимально використовувати переваги обох методів, оптимізуючи загальну продуктивність та задовольняючи специфічні вимоги проектів.

Наше дослідження підтвердило, що розуміння контексту застосування та правильний вибір між SSR та SSG є критичними для успіху сучасних веб-застосунків, дозволяючи розробникам забезпечити високу швидкість завантаження, ефективність і задоволення користувачів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Gatsby.js vs. Next.js — Which is the Best React Framework? (+ SSG/SSR Rendering Option) / URL: https://serinryu.medium.com/gatsby-js-vs-next-js-which-is-the-best-react-framework-ssg-ssr-rendering-option-55672aa6e321 (дата звернення: 10.05.2024).

2. Peter Kent. Search Engine Optimization for Dummies / Publisher: For Dummies, 2012, – 456 p.

3. Перов О.С., Афанасьєва І.В. ФРЕЙМВОРК ДЛЯ РЕНДЕРІНГУ 3D СЦЕН НА ПЛАТФОРМАХ, ЩО ПІДТРИМУЮТЬ METAL API//Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління. Тези доповідей дванадцятої міжнародної науково-технічної конференції 27-28 квітня 2022 року. -Том2:секція 5. Баку-Харків-Жиліна-2022. -С. 117

4. Michele Riva. Real-World Next.js: Build scalable, high-performance, and modern web applications using Next.js, the React framework for productio / Publisher: Packt Publishing, 2022, – 366 p.

5. Stoyan Stefanov. React: Up & Running: Building Web Applications / Publisher: O'Reilly Media, 2016, – 219 p.

6. Douglas Crockford. The Good Parts: The Good Parts / Publisher: Yahoo Press, 2020, – 170 p.

7. Dudar Z., Shubin I., Skovorodnikova V., Litvin S. Research of Ways to Increase the Efficiency of Functioning Between Firewalls in the Protection of Information Web-Portals in Telecommunications Networks Lecture Notes in Networks and Systems, 2021, 212 LNNS, стр. 272–292 DOI 10.1007/978-3-030-76343-5_14

8. Хомицкий И.А., Новіков Ю. С. Проблема обновления контента при использовании cdn// 20-Й Юбилейный Международний молодежний форум «Радиоэлектроника и молодеж в XXI ВЕКЕ».

9. JavaScript Testing Best Practices to Follow / URL: https://www.browserstack.com/guide/javascript-testing-best-practices (дата звернення: 13.05.2024).

10. Jon Duckett. HTML and CSS: Design and Build Websites / Publisher: John Wiley & Sons, 2011 – 490 p.

11. Gerardus Blokdyk. Web Application Testing A Complete Guide – 2023 Edition / Publisher: The Art of Service - Web Application Testing Publishing, 2023, – 313 p.

12. Turevska O., Shubin, I. Improving the automated testing of Web-based services by reflecting the social habits of target audiences//2015 Information Technologies in Innovation Business Conference, ITIB 2015 - Proceedings, 2015, c. 93-96, 7355062

13. Yakov F, Anton M. TypeScript Quickly / Publisher: Manning, 2020 –350 p.

14. Marc G, Will F. Redux in Action / Publisher: Manning, 2018 – 312 p.