

АНАЛІЗ АЛГОРИТМІВ ОПТИМІЗАЦІЇ ВЕБ-ЗАПИТІВ

Вінницький національний технічний університет

Анотація

У цій роботі розглядається проблема оптимізації веб-запитів через аналіз трьох методів: пагінації, кешування та проєктування з використанням GraphQL. Проведено порівняння їхньої часової ефективності на основі практичних тестів, що показують переваги кожного методу залежно від обсягу даних, частоти запитів і архітектури системи. Результати свідчать, що комбінація цих підходів підвищує продуктивність веб-додатків, хоча кожен має обмеження: пагінація ускладнює синхронізацію в динамічних системах, кешування потребує управління пам'яттю, а GraphQL — складнішого налаштування. Дослідження підкреслює перспективність інтеграції методів для створення гнучких і швидких систем, відкриваючи шляхи для подальшої оптимізації залежно від конкретних умов використання.

Ключові слова: алгоритми оптимізації, оптимізація, веб, пагінація, кешування, проєктування.

Abstract

This study examines the problem of optimizing web requests by analyzing three methods: pagination, caching, and projection using GraphQL. A comparison of their time efficiency was conducted based on practical tests, demonstrating the advantages of each method depending on data volume, request frequency, and system architecture. The results indicate that combining these approaches enhances the performance of web applications, though each has limitations: pagination complicates synchronization in dynamic systems, caching requires memory management, and GraphQL demands more complex setup. The research highlights the potential of integrating these methods to create flexible and fast systems, paving the way for further optimization tailored to specific use cases.

Keywords: optimization algorithms, optimization, web, pagination, caching, projection.

Вступ

Оптимізація веб-запитів є критично важливим завданням у розробці сучасних веб-додатків, адже від швидкості та ефективності залежить як продуктивність системи, так і зручність для користувача. У цьому контексті три методи — пагінація, кешування та проєктування (у значенні вибірки лише необхідних даних, наприклад, за допомогою GraphQL) — відіграють ключову роль у зменшенні навантаження на сервер і прискоренні обробки запитів. Кожен із них має унікальні особливості реалізації, які дозволяють адаптувати їх до різних сценаріїв використання.

Пагінація передбачає розбиття великих наборів даних на менші фрагменти, або "сторінки", які завантажуються поступово. Цей метод широко застосовується в інтерфейсах, де користувачам потрібно переглядати списки, наприклад, товари в інтернет-магазині чи пости в блозі. Реалізація пагінації зазвичай базується на додаванні параметрів до URL, таких як `page=2` та `limit=10`, що вказують номер сторінки та кількість елементів на ній. Це дозволяє серверу обробляти лише частину даних за один раз, зменшуючи обсяг переданої інформації та час відповіді.

Кешування, спрямоване на збереження результатів попередніх запитів у тимчасовому сховищі, щоб уникнути повторних звернень до сервера чи бази даних. Цей підхід часто реалізується через HTTP-заголовки, такі як `Cache-Control`, або на рівні серверних технологій, наприклад, із використанням Redis чи Memcached. Кешування ідеально підходить для статичного чи рідко змінного контенту, наприклад, головних сторінок сайтів чи метаданих, які запитуються регулярно.

Проєктування в контексті цього аналізу означає вибір лише необхідних даних із серверного ресурсу, що стало можливим завдяки таким технологіям, як GraphQL. На відміну від традиційних REST API, де сервер повертає фіксований набір даних, GraphQL дозволяє клієнту вказати точний набір полів, які потрібно отримати. Наприклад, замість повного об'єкта `User` із десятками полів клієнт може запросити лише `name` і `email`. Реалізація цього методу базується на створенні GraphQL-схеми на сервері та формуванні запитів на стороні клієнта, що дає змогу оптимізувати обсяг переданих даних і зменшити надлишкову обробку.

Часова ефективність і залежності: Аналіз на практиці

Часова ефективність алгоритмів оптимізації веб-запитів залежить від таких факторів, як обсяг даних, архітектура системи, частота запитів і специфіка задачі. Щоб оцінити їхню продуктивність, розглянемо практичні приклади та тести, доступні в дослідженнях і документаціях.

Пагінація демонструє значну часову ефективність при роботі з великими наборами даних. Наприклад, у тестах REST API із базою даних на 10 000 записів повний запит може займати 5–7 секунд через необхідність обробки та передачі всіх даних. Якщо ж застосувати пагінацію з розбиттям на сторінки по 100 записів, час відповіді скорочується до 0,3–0,5 секунди на сторінку, як показують дослідження продуктивності серверів на основі Node.js чи Django. Однак ця ефективність зменшується, якщо дані часто оновлюються, адже серверу доводиться перераховувати індекси чи перевіряти актуальність сторінок. Також затримки можуть виникати при поганій реалізації серверної логіки, наприклад, якщо запит до бази даних не оптимізований індексами.

Кешування виявляється надзвичайно ефективним для повторюваних запитів. Тести продуктивності веб-сайтів на WordPress із плагіном WP Super Cache показують, що час завантаження сторінки без кешу становить близько 2 секунд, тоді як із кешуванням цей показник падає до 50–100 мілісекунд при повторному запиті. Аналогічні результати демонструє Redis у серверних додатках: кешування метаданих скорочує час відповіді API з 300 мс до 10 мс, за даними документації Redis. Проте ефективність залежить від розміру кешу, алгоритму його очищення (наприклад, LRU чи LFU) і частоти оновлення даних. Якщо контент змінюється часто, кеш може стати застарілим, що призводить до додаткових перевірок чи інвалідності, збільшуючи затримки.

Проектування через GraphQL забезпечує часову ефективність за рахунок зменшення обсягу переданих даних. У традиційному REST API запит до ендпоінту /users може повернути повний об'єкт із 20 полями, займаючи 500 мс на обробку та передачу. У тесті з GraphQL, описаному в статті "GraphQL Performance: A Practical Guide" на Medium, запит лише двох полів (name, email) скоротив час відповіді до 150 мс, а обсяг даних — із 10 КБ до 1 КБ. Ефективність цього методу залежить від складності схеми та оптимізації серверної логіки: надмірна кількість вкладених запитів (наприклад, через N+1 проблему) може призвести до затримок. Проте інструменти, як DataLoader, допомагають усунути ці недоліки, підтримуючи стабільну продуктивність.

Висновок

Порівняння пагінації, кешування та проектування через GraphQL показує, що кожен метод має унікальні переваги, залежні від потреб системи — швидкості, економії ресурсів чи гнучкості. Пагінація ідеальна для великих обсягів даних, як у каталогах товарів, де поступовий доступ зменшує навантаження на сервер і прискорює відповідь, але менш ефективна в динамічних системах із частими оновленнями через потребу в синхронізації. Кешування найкраще працює з повторюваними запитами, наприклад, у соціальних мережах, забезпечуючи швидкість, хоча вимагає ретельного управління пам'яттю для актуальності даних.

Проектування через GraphQL оптимальне для гнучких систем, як мобільні додатки, де вибір лише потрібних даних (наприклад, ім'я та аватар замість повного профілю) економить трафік і ресурси, особливо на повільних мережах. Однак складність налаштування та ризик N+1 запитів роблять його менш універсальним для простих систем, де REST із пагінацією чи кешуванням може бути достатнім. Кожен метод вирішує різні задачі: пагінація — масштабування, кешування — швидкість, GraphQL — точність.

Для максимальної ефективності ці методи варто комбінувати. Розумне поєднання забезпечує гнучкі, швидкі та ресурсоефективні веб-додатки, адаптовані до конкретних умов, балансуючи між швидкістю, економією та простотою залежно від пріоритетів системи.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Marangon, J. D. "Optimizing REST API Performance: A Comprehensive Guide." *Medium*, 2023. URL: <https://blackcatdev.medium.com/mastering-asp-net-core-a-comprehensive-guide-to-optimizing-restful-api-performance-deb9d1200baa>
2. "How to Speed Up Your WordPress Site." *Kinsta*. URL: <https://kinsta.com/learn/speed-up-wordpress/>
3. "Redis Documentation." *Redis*. URL: <https://redis.io/about/>
4. "Execution - GraphQL." *GraphQL Official Documentation*. URL: <https://graphql.org/learn/execution/>

Опанасюк Ігор Олегович – студент групи 2КІ-24мс2, факультет інформаційних технологій та комп’ютерної інженерії, Вінницький національний технічний університет, м. Вінниця, e-mail: ihor.opanasiuk@gmail.com.

Добровольська Наталія Вікторівна – доцент кафедри обчислювальної техніки Вінницького національного технічного університету, м. Вінниця, e-mail: natali0212@ukr.net.

Ohor Olehovych Opanasiuk - student of group 2CE-24ms2, Faculty of Information Technologies and Computer Engineering, Vinnytsia National Technical University, Vinnytsia, e-mail: ihor.opanasiuk@gmail.com.

Natalia Dobrovolska - Associate Professor of Computer Engineering, Vinnytsia National Technical University, Vinnytsia, e - mail: natali0212@ukr.net.