УДК 004.421.2:004.932

МОДЕЛЮВАННЯ ПРОДУКТИВНОСТІ RESTFUL API: АЛГОРИТМИ ОБРОБКИ ЗАПИТІВ І АНАЛІЗ ДАНИХ

Павло Луб, к. т. н., Віталій Фіялковський, Любомир Чухрай, к. ф.-м. н., Святослав Штогрин, Вікторія Стефанишин
Львівський національний університет природокористування
вул. Володимира Великого, 1, м. Дубляни, Львівський р-н, Львівська обл., Україна,
e-mail: pollylub@ukr.net, vitalik.fiyalkovsky@i.ua, l.chukhrai@gmail.com, sviatoslav.shtohryn@gmail.com, viktoriastefanushun12345@gmail.com

https://doi.org/10.32718/agroengineering2025.29.225-231

Луб П., Фіялковський В., Чухрай Л., Штогрин С., Стефанишин В. Моделювання продуктивності RESTful API: алгоритми обробки запитів і аналіз даних
Узагальнено позитивні сторони та негативи створення RESTful API у Back-end розробці. Проаналізовано поточний стан використання REST-архітектури у вебтехнологіях. Розглянуто актуальні підходи та інструменти для створення RESTful API, а також обґрунтовано доцільність їх застосування. Описано популярні фреймворки та середовища розробки, такі як Express.js, Django REST Framework та Spring Boot, що широко застосовуються як у стартапах, так і у великих компаніях. Наведено особливості структурування запитів, методам HTTP, стандартам побудови маршрутів та обробці даних на сервері. Акцент зроблено на RESTful API, які забезпечують простоту інтеграції, масштабованість і гнучкість у розробці клієнт-серверних застосунків. Наведено перелік ключових принципів побудови RESTful API, таких як використання методів GET, POST, PUT, DELETE, правильне оформлення відповідей сервера та стандартизація шляхів ресурсів. Підкреслено окремі обмеження REST-підходу, такі як складність підтримки надмірно великої кількості ресурсів або версіонування API. Використання розглянутих інструментів і підходів дає змогу підвищити якість серверної логіки, забезпечити стандартизований обмін даними та зменшити складність інтеграції клієнтських додатків. Охарактеризовано застосування RESTful API в сучасному вебдизайні та фреймворках, зокрема Express.js, Django REST Framework і Spring Boot. Показано, що RESTful API забезпечують ефективну взаємодію між клієнтськими та серверними застосунками, пропонуючи прості й масштабовані засоби для оброблення даних і керування ресурсами. Розглянуто приклад реалізації невеликого сервера на Node.js із використанням Express для керування списком завдань, що підтримує базові операції CRUD. Основна увага приділяється оптимізації REST-запитів через методи кешування, пагінації, фільтрації, batch-запитів, стиснення даних і асинхронну обробку. Описано методику дослідження продуктивності API на тестовому сервері з Node.js/Express та інструментами Prometheus і Grafana для моніторингу. Проведено тестування під навантаженням із генерацією стабільного потоку запитів, що дозволило оцінити вплив окремих методів оптимізації на пропускну здатність, затримки та обсяг переданих даних. Наведено результати щодо переваги асинхронних запитів та batch-операцій, які забезпечують максимальне підвищення продуктивності, тоді як кешування, пагінація, фільтрація і стиснення дають помірний ефект. Наголошено, що використання комбінованих методів оптимізації значно підвищує швидкодію та зменшує час відповіді й навантаження на сервер, що забезпечує стабільну роботу вебсервісів навіть при високій інтенсивності трафіку.
Ключові слова: RESTful API, вебсервіси, Node.js, Express.js, CRUD, оптимізація запитів, кешування, пагінація, фільтрація, batch-запити, асинхронна обробка, продуктивність.

Lub P., Fialkovskyi V., Chukhrai L., Shtohryn S., Stefanyshyn V. RESTful API performance modeling: request processing algorithms and data analysis
The text summarizes the positive and negative aspects of creating RESTful APIs in back-end development. It analyzes the current state of REST architecture usage in web technologies and examines modern approaches and tools for developing RESTful APIs, justifying their applicability. Popular frameworks and development environments, such as Express.js, Django REST Framework, and Spring Boot, which are widely used in both startups and large companies, are described. The study highlights key features such as request structuring, HTTP methods, routing standards, and server-side data handling. It emphasizes RESTful APIs for their ease of integration, scalability, and flexibility in client-server application development. The key principles of RESTful API design are outlined, including the use of the GET, POST, PUT, and DELETE methods, proper server response formatting, and standardization of resource paths. Some limitations of the REST approach are noted, such as challenges in handling a large number of resources and issues related to API versioning. The use of these tools and approaches improves the quality of server-side logic, ensures standardized data exchange, and reduces complexity in client-side integration. The application of RESTful APIs in modern web design through frameworks like Express.js, Django REST Framework, and Spring Boot is characterized. A practical example is provided of a small Node.js server using Express to

manage a task list with basic CRUD operations. The focus is on optimizing REST requests through various strategies, including caching, pagination, filtering, batch requests, data compression, and asynchronous processing. A methodology for evaluating API performance on a Node.js/Express test server, using Prometheus and Grafana for monitoring, is described. Load testing with a stable request stream allowed for the assessment of the impact of individual optimization methods on throughput, latency, and data volume. Results indicate that asynchronous requests and batch operations yield the highest performance increases, while caching, pagination, filtering, and compression offer moderate improvements. It is emphasized that utilizing combined optimization methods significantly enhances speed, reduces response time and server load, and ensures stable operation of web services, even under conditions of high traffic intensity.
Keywords: RESTful API, web services, Node.js, Express.js, CRUD, request optimization, caching, pagination, filtering, batch requests, asynchronous processing, performance.

Постановка проблеми. У сучасній веброзробці спостерігається стрімке зростання кількості клієнт-серверних додатків, що вимагає створення ефективних, масштабованих та стандартизованих механізмів обміну даними між клієнтом і сервером [1; 5; 14]. Одним із найпоширеніших підходів для виконання цього завдання є використання RESTful API. Проте, незважаючи на популярність архітектури REST, розробники стикаються з низкою проблем, зокрема з необхідністю дотримання стандартів побудови API, вибору раціональних інструментів для його реалізації, забезпечення безпеки, підтримки масштабованості та гнучкості, а також із труднощами під час оброблення великої кількості ресурсів і версіонування API. Відсутність чіткої структури або порушення принципів REST може призвести до зниження ефективності системи, ускладнення інтеграції та зменшення надійності вебрішень. Це визначає актуальність розгляду питання щодо сучасних підходів до створення RESTful API, аналізу відповідних інструментів та принципів побудови ефективних серверних рішень.

Реалізація RESTful API формує можливості для підвищення ефективності процесу розробки вебрішень, забезпечуючи стандартизовану взаємодію, гнучкість та масштабованість систем. Розвиток і вдосконалення таких архітектурних рішень суттєво спрощуватиме розробку складних цифрових продуктів та підвищуватиме ефективність сучасних вебтехнологій.

Аналіз останніх досліджень і публікацій у сфері веброзробки засвідчують актуальність використання RESTful API як основної моделі побудови клієнт-серверної взаємодії [1-3; 7; 12]. У наукових працях і публікаціях, присвячених розробці вебзастосунків, широко обговорюються принципи побудови REST-архітектури, особливості реалізації CRUD-операцій через HTTP методи, а також стандартизація обміну даними у форматах JSON та XML [4; 11; 15; 20]. Зокрема наголошується на важливості коректного проєктування маршрутизації запитів, обробки помилок, безпеки API через механізми авторизації (OAuth 2.0, JWT) та захисту від типових атак [3]. Також підкреслюються особливі вимоги до вибору інструментів і фреймворків, серед яких Express.js для Node.js, Django REST Framework для Python і Spring Boot для Java демонструють найвищу продуктивність, гнучкість та підтримку масштабованості проєктів [13; 16; 18]. Окреме порівняння REST з альтернативними підходами, зокрема GraphQL та gRPC, дозволяє глибше оцінити переваги та недоліки RESTful API залежно від специфіки проєкту.

Постановка завдання. Наше завдання – встановити й кількісно оцінити вплив комплексних методів оптимізації запитів RESTful API на показники ефективності їх застосування у вебсервісах щодо продуктивності, стабільності та швидкості системи за високої інтенсивності трафіку.

У роботі застосовано підхід, що передбачає впровадження та тестування різних методів оптимізації RESTful API, зокрема кешування (HTTP-Cache, ETag, Redis) та оптимізації структури запитів (пагінація, фільтрація, параметризація), із подальшим аналізом їхнього впливу на продуктивність, час відгуку та навантаження на сервер.

Виклад основного матеріалу. RESTful API – це архітектурний підхід до побудови вебсервісів, що базується на принципах Representational State Transfer (REST) та передбачає стандартизовану взаємодію клієнта і сервера через протокол HTTP. Така модель забезпечує обмін даними у форматах JSON або XML, використовуючи стандартні методи HTTP (GET, POST, PUT, DELETE) для роботи з ресурсами. Завдяки цьому RESTful API є гнучким, масштабованим та широко застосовується під час створення веб- і мобільних застосунків [6; 10].

Із розвитком вебтехнологій та зростанням популярності RESTful API, створення ефективних і масштабованих серверних додатків стало важливою складовою сучасної розробки програмного забезпечення. Використання RESTful API дозволяє забезпечити стабільну та гнучку взаємодію між

клієнтами і серверами через стандартизовані методи HTTP. Однією з ключових переваг використання REST є її простота та легкість інтеграції у різноманітні проєкти, що забезпечує високий рівень сумісності з іншими вебтехнологіями. RESTful API також дозволяє спрощувати процеси маршрутизації запитів і обробки даних на сервері. Особливо важливим є використання сучасних фреймворків та інструментів для створення RESTful API, таких як Express.js для Node.js, Django REST Framework для Python та Spring Boot для Java. Ці інструменти забезпечують ефективність, гнучкість і зручність розробки вебсервісів, що особливо важливо для створення масштабованих і високопродуктивних серверних рішень.

Окрім того, RESTful API забезпечує безпечну взаємодію між користувачами та системами, що є критично важливим для будь-якої розробки сучасних вебдодатків. Водночас важливим аспектом при створенні RESTful API є забезпечення захисту від типових атак. Для цього використовуються механізми авторизації, такі як OAuth 2.0 і JSON Web Token (JWT), що дозволяють забезпечити доступ тільки авторизованим користувачам та зберегти конфіденційність даних. Водночас, RESTful API дозволяє легко масштабувати і адаптувати вебсервіси до зростаючих потреб користувачів і зберігати високу продуктивність. Однією з важливих проблем, з якою стикаються розробники, є правильний вибір підходу до проєктування API. Зокрема порівняння REST з альтернативними підходами, такими як GraphQL та gRPC, дозволяє вибрати оптимальне рішення для конкретного проєкту.

Слід зазначити, що RESTful API, GraphQL та gRPC є доволі популярними у створенні програмних інтерфейсів для взаємодії між компонентами інформаційних систем. Водночас кожен із них орієнтований на різні класи завдань і має власні архітектурні особливості. Вибір конкретного підходу зазвичай визначається вимогами до продуктивності, гнучкості, масштабованості та характеру клієнтських застосунків. Щодо мови запитів GraphQL, то її застосовують у завданнях, де важлива гнучкість і оптимізація обміну даними між клієнтом і сервером, зокрема у складних фронтенд-застосунках та мобільних клієнтах (платформи соцмереж, маркетплейси, сервіси аналітики, адмінпанелі тощо). Основна ідея GraphQL полягає в тому, що клієнт сам визначає, які саме поля і в якій структурі він хоче отримати.

gRPC орієнтований на високопродуктивну взаємодію між сервісами, особливо в мікросервісних і розподілених системах. gRPC доцільно використовувати для внутрішніх сервісів, потокової передачі даних і систем із високими вимогами до швидкодії (стрім-канали, онлайн ігри, транзакції в реальному часі, сервіси білінгу, обробки замовлень тощо). Однак gRPC є значно складнішим у написанні коду, налагодженні та усуненні помилок.

Отже, RESTful API зазвичай обирають для простих і зрозумілих вебінтерфейсів, GraphQL – для клієнтоорієнтованих систем із динамічними потребами в даних, а gRPC – для високопродуктивної взаємодії між сервісами всередині складних розподілених архітектур. Розгляд саме RESTful API є доцільним з огляду на його універсальність, зрілість та відповідність типовим вимогам більшості навчальних і прикладних IT-проєктів. REST давно сформувався як де-факто стандарт побудови вебінтерфейсів взаємодії, тому його використання забезпечує прогнозовану архітектуру, зрозумілу логіку роботи та легкість інтеграції з іншими системами тощо.

З іншого боку, створення RESTful API є необхідним етапом у розробці сучасних вебдодатків, що сприяє підвищенню ефективності, зручності та безпеки розроблених серверних рішень [8; 17; 19]. Цей підхід забезпечує простоту інтеграції з іншими технологіями, а також можливість масштабування і адаптації до специфічних вимог кожного проєкту.

Ці інструменти допомагають розробникам створювати додатки, що характеризуються як масштабованістю, так і гнучкістю, що має критичне значення для задоволення постійно зростаючих вимог сучасного програмування. Вони дозволяють писати структурований та підтримуваний код, полегшують інтеграцію та забезпечують швидке розгортання. Популярні фреймворки, такі як Express.js для Node.js, Django REST Framework для Python та Spring Boot для Java (рис.1, 2). Кожен із них має свої переваги і недоліки в контексті продуктивності, гнучкості та зручності використання.

Водночас тенденції використання RESTful API у промисловості [9; 19] свідчать про зростаючий інтерес та позитивні відгуки розробників, зокрема близько 85 % користувачів повідомляють про задоволеність від використання цих інструментів. Загальний інтерес до використання API у виробничому секторі зростає з кожним роком, спостерігається чітка тенденція до збільшення інвестицій у API-рішення для бізнес-процесів.

Отже, розробники повинні володіти навичками роботи з RESTful API, щоб залишатися конкурентоспроможними у своїй професійній діяльності. Популярність API в розробці програмного забезпечення підкреслює необхідність залучення цих технологій для досягнення успіху у прикладній сфері.

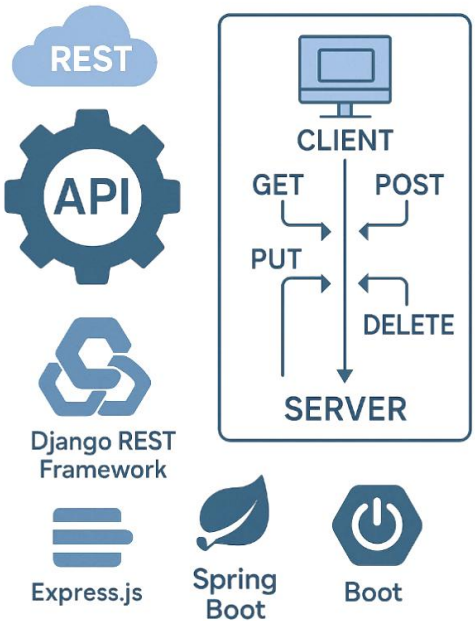

Рис. 1. Архітектура та інструменти для створення RESTful API [2]
Fig. 2. Architecture and Tools for Creating RESTful API [2]

Практика застосування RESTful API в сучасному вебдизайні, зокрема у таких рамках, як Express.js, Django REST Framework та Spring Boot, переконує в тому, що RESTful API сприяє оптимізації взаємодії між клієнтськими та серверними додатками, пропонуючи прості та масштабовані засоби для обробки даних та взаємодії. RESTful API полегшує взаємодію різних служб, гарантуючи ефективний доступ до даних, їхню генерацію, оновлення та видалення. Цей підхід також підтримує різні методи HTTP (GET, POST, PUT, DELETE), що дозволяє розробникам раціонально організовувати маршрути для оптимального доступу до даних та їхнього управління.

Зокрема, для прикладу створення невеликого сервера, написаного на Node.js із використанням фреймворку Express із простим завданням роботи – керувати списком справ. Доцільно розглянути й дослідити ефективність використання того чи іншого методу оптимізації запитів. Такий сервер вважається базовою реалізацією REST API, оскільки він підтримує всі основні операції: створення, отримання, оновлення та видалення даних. Каркас означеного Node.js застосунку з Express такий:

```
const express = require('express');
const app = express();
app.use(express.json());
let tasks = [
  { id: 1, name: 'Task 1', completed: false },
  { id: 2, name: 'Task 2', completed: true }
];
….
app.listen(3000, () => {
  console.log('Server running on port 3000');
});
```

Головна функція коду – створити невеликий вебсервер, який: 1) запускає застосунок Express (const app = express();); 2) дозволяє працювати з JSON-запитами (app.use(express.json());); 3) зберігає у пам’яті простий масив завдань tasks (імітація бази даних); 4) слухає порт 3000 і чекає HTTP-запитів (app.listen(3000, ...)).

Початкові дані в такому разі зберігаються у пам’яті сервера, кожна задача складається з трьох полів: 1) id – унікальний номер; 2) name – назва задачі; 3) completed – статус виконання. API працює за стандартними правилами REST і надає чотири основні маршрути:

1) отримання всіх задач (GET/tasks) – клієнт може надіслати запит, щоб отримати список справ у форматі JSON;

2) створення нової задачі (POST/tasks) – якщо потрібно додати нову справу, клієнт відправляє запит із даними у задачі JSON;

```
// GET request to fetch all tasks
  app.get('/tasks', (req, res) => {
    res.json(tasks);
  });
```

```
// POST request to create a new task
app.post('/tasks', (req, res) => {
  const newTask = req.body;
  tasks.push(newTask);
  res.status(201).json(newTask);
});
```

3) оновлення наявної задачі (PUT /tasks/:id) – для зміни назви чи статусу задачі;

4) видалення задачі (DELETE /task/:id) – якщо завдання більше не потрібне.

Швидкість опрацювання запитів у RESTful API визначає ефективність його застосування. У цьому разі до технології ставиться певний набір вимог, зокрема щодо швидкості віддачі відповіді з бази даних, синхронізування даних між клієнтом і сервером, підтримки миттєвих відповідей на запити про поточні рахунки тощо. Без оптимізації запитів навіть добре побудований сервер починає «задихатися» при збільшенні кількості клієнтів. Це формує завдання із пошуку та порівняння методів оптимізації, найефективніші для різних сценаріїв (кешування, батчинг, пагінація, асинхронні виклики тощо).

Відомо [4; 11], що до методів оптимізації REST-запитів відносять: 1) кешування – використання HTTP-заголовків (Cache-Control, ETag, Last-Modified) для зменшення кількості повторних запитів до сервера; 2) пагінацію – для великих колекцій даних (списки, таблиці) обмежують кількість результатів на один запит (HTTP-параметри: ?page=1&limit=50 – зменшує обсяг переданих даних і швидко обробляє запити); 3) фільтрацію та сортування – сервер повертає тільки потрібні поля або записи, що відповідають критеріям (GET /products?category=fruits&sort=price_desc); 4) batch-запити – об'єднання кількох операцій в один запит (у мікросервісах або при взаємодії із базою даних); 4) стиснення даних – використання gzip або brotli для зменшення об'єму переданих даних; 5) стратегії асинхронності – зменшення навантаження завдяки використанню asynchronous calls або webhooks замість постійного опитування серверу (polling).

З метою оцінки впливу окремих методів оптимізації RESTful API (кешування, підбір структури запитів, асинхронна обробка, комбіноване застосування) на продуктивність (throughput) запитів було реалізовано типовий метод дослідження. Зокрема сформовано технічну конфігурацію спеціалізованих інструментів – виділено один тестовий стенд (експериментальний сервер – Linux VM 4 vCPU, 8 GB RAM); клієнт навантаження (окрема машина, щоб уникнути локального виснаження ресурсів); локальна мережа; стек сервера Node.js/Express; база даних – mocked; Prometheus+Grafana для моніторингу й відображення трафіку/даних.

Поетапне випробування передбачало формування тестових даних. Зокрема підготовлено dataset із реалістичною розмірністю (50-200k записів списків) із очікуваними відповідями середньої довжини JSON (розмір даних у повідомленні payload size – від 1024 до 5120 байт) в умовах однакової БД (для пагінації та фільтрації). Наступним кроком виконано генерацію навантаження із стабільним throughput (wrk2) із підтримкою стабільної швидкості запитів (constant throughput mode), яка передбачала розігрів сервера warm-up на рівні 60 с, 3 хв для кожного сценарію та серію тестів із навантаженням по 50, 100, 200 запитів/с:

```
# 200 req/s протягом 30 секунд
wrk2 -t4 -c80 -d30m -R200 http://localhost:3000/api/tasks
```

де -t – кількість потоків генератора; -c – кількість одночасних з'єднань; -d30s – тривалість тесту; -R – цільова частота запитів/с.

Отже, послідовне застосування різних методів оптимізації запитів RESTful API дозволяє ефективно управляти потоками даних, скорочує час відповіді сервера, знижує обсяг переданих даних і підвищує загальну масштабованість системи (табл.). Відповідно до отриманих показників, застосування механізмів кешування, зокрема HTTP-Cache, ETag та Redis, забезпечує суттєве зменшення навантаження на сервер приблизно на 40 % та скорочення часу відповіді повторних запитів із 240 мс до 150 мс. Оптимізація структури запитів шляхом використання пагінації, фільтрації та параметризації дозволяє зменшити обсяг переданих даних у середньому на 55 %, що, насамперед, прискорює обробку запитів на клієнтській стороні на 25-30 %.

Використання асинхронної обробки на базі Node.js і non-blocking I/O приводить до підвищення пропускної здатності системи приблизно на 60 % з 1200 до 1900 запитів за секунду та зниження середніх затримок у пікових режимах з 310 мс до 180 мс.

Найбільший ефект досягається за комбінованого застосування зазначених підходів, що забезпечує сумарне зростання загальної продуктивності системи на 50-65 % залежно від сценарію використання та підвищує її стійкість до високих навантажень.

Застосування розглянутих методів оптимізації запитів RESTful API (порівняно із варіантом – без оптимізації) дало змогу встановити межі покращення продуктивності потоків даних, котрі наведено на рис. 2.

Відповідно до отриманих результатів, асинхронні запити та batch-запити забезпечують найвищу продуктивність. За відсутності оптимізації значно знижується пропускна здатність системи. Кешування, пагінація, фільтрація та стиснення дають помірне покращення.

Таблиця. Результати оцінення впливу методів оптимізації запитів RESTful API на продуктивність потоків даних
Table. Results of evaluating the impact of RESTful API request optimization methods on data stream performance

№ з/п	Метод оптимізації	Основний ефект	Кількісний показник
1	Кешування (HTTP-Cache, ETag, Redis)	Зменшення навантаження на сервер, швидший повторний доступ	-40 % навантаження; час відповіді ↓ з 240 мс до 150 мс
2	Оптимізація структури запитів (пагінація, фільтрація, параметри)	Зменшення обсягу переданих даних, швидша обробка клієнтом	-55 % даних; обробка клієнтом швидше на 25-30 %
3	Асинхронна обробка (Node.js, non-blocking I/O)	Вища пропускна здатність, менші затримки в піку	+60 % пропускної здатності (1200 → 1900 зап/с); затримка ↓ з 310 мс до 180 мс
4	Комбіноване застосування	Сумарне підвищення продуктивності, стійкість до навантаження	Загальна продуктивність +50–65 % залежно від сценарію

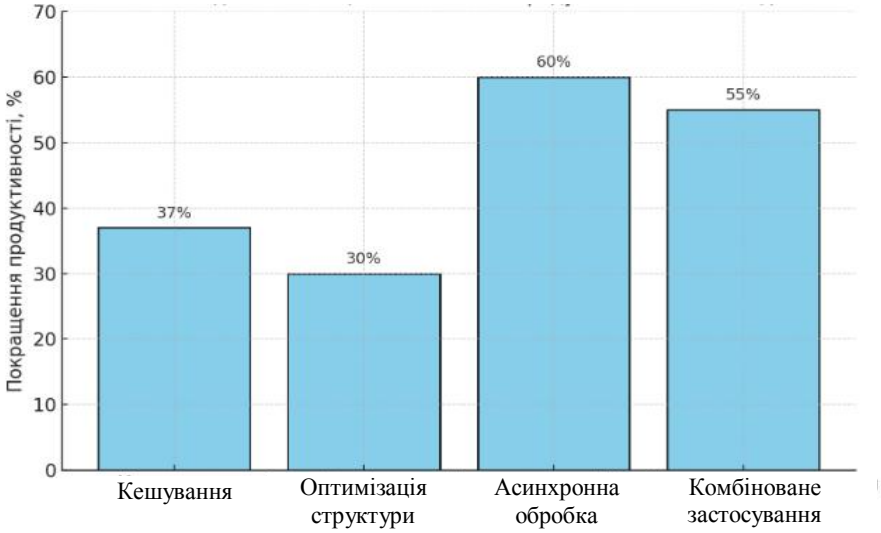

Рис. 2. Гістограма зміни впливу методів оптимізації RESTful API на продуктивність потоків даних (у порівнянні з відсутністю оптимізації запитів)
Fig. 2. Histogram of RESTful API query optimization effects on performance (compared to no optimization)

Висновки. RESTful API є базовим стандартом для побудови ефективних, масштабованих і простих у використанні вебсервісів, незважаючи на появу альтернативних технологій. Отримані результати підтверджують, що застосування комбінованих методів оптимізації RESTful API підвищує загальну продуктивність системи на 50-65 % залежно від сценарію використання. Це зменшує час відгуку, знижує серверне навантаження та підвищує стабільність роботи вебсервісів навіть за умов високої інтенсивності трафіку. Використання механізмів кешування (HTTP-Cache, ETag, Redis) дає змогу знизити середнє навантаження на сервер на 35-40 %, а середній час відповіді скорочується з 240 мс до 150 мс за повторних запитів. Оптимізація структури запитів (пагінація, фільтрація, параметризація) призводить до зменшення обсягу переданих даних у середньому на 55 %, що призводить до скорочення часу обробки на клієнтській стороні на 25-30 %. Використання асинхронної обробки (на базі Node.js та non-blocking I/O) призводить до зростання пропускної здатності системи приблизно на 60 % (з 1200 до 1900 запитів/с) та зменшення середніх затримок із 310 мс до 180 мс у пікових навантаженнях. Використання комплексних методів

оптимізації RESTful API значно підвищує продуктивність системи, скорочує час відгуку, зменшує навантаження на сервер та забезпечує стабільну роботу вебсервісів навіть за високої інтенсивності трафіку. Використання механізмів кешування та оптимізація структури запитів дозволяють ефективно зменшувати обсяг переданих даних і прискорювати обробку запитів на клієнтській стороні, що загалом підвищує швидкодію та надійність системи.

Бібліографічний список

1. Архітектурні стилі API. IT-компас, 2024. URL: https://infdev.com.ua/docs/standards/api-architecture-styles/ (дата звернення: 21.01.2025)
2. Бородкіна І. Л., Бородкін Г. О. Web-технології та Web-дизайн: застосування мови HTML для створення електронних ресурсів. Київ: Ліра-К, 2020. 212 с.
3. Бурячок В. Л., Гулак Г. М., Толубко В. Б. Інформаційний та кіберпростори: проблеми безпеки, методи та засоби боротьби. Магнолія, 2023. 448 с.
4. Гавриленко О. The System Design Cheat Sheet: API Styles - REST, GraphQL, WebSocket, Webhook, RPC/gRPC, SOAP, 2023. URL: https://hackernoon.com/the-system-design-cheat-sheet-api-styles-rest-graphql-websocket-webhook-rpcgrpc-soap (дата звернення: 21.01.2025)
5. Денисенко А. Вступ до REST API – RESTful вебсервіси, 2025. URL: https://robotdreams.cc/uk/blog/466-vstup-do-rest-api-restful-vebservisi (дата звернення: 21.01.2025)
6. Мельник Р. А. Програмування веб-застосувань (фронт-енд та бек-енд). Львів: НУ «ЛП», 2018. 248 с.
7. Скалозуб В. В., Горячкін В. М., Мурашов О. В. Реляційно-сепарабельні моделі процесів моніторингу при перемінних і нечітких інтервалах спостережень. *Системні технології. Регіональний міжвузівський збірник наукових праць.* 2023. Вип. 4 (147). Дніпро, 2023. С. 3-19. DOI: 10.34185/1562-9945-4-147-2023-01
8. Тележенко Д. О. Методи та моделі синтезу архітектури віртуальних розподілених комп'ютерних систем: дис. … канд. наук: 122 Комп'ютерні науки. Харків, 2025. 145 с.
9. Тележенко Д. О. Модифікація методу відновлення архітектури віртуальних комп'ютерних систем після збоїв. 2024. С. 274-283. DOI: 10.52058/48-2024
10. Юрчак І. Ю. Технології веб-розробки та дизайну: конспект лекцій для студентів першого (бакалаврського) рівня вищої освіти спеціальності 122 «Комп'ютерна наука». Львів: НУ «ЛП», 2024. 363 с.
11. Amundsen M. RESTful Web API Patterns and Practices Cookbook. Birmingham: Packt Publishing, 2013. 256 p.
12. Bayer T., Polley T. The API Gateway Handbook: Your Practical Guide to API Gateway Setup, Security, and Operation. Leanpub, 2025. URL:
13. https://leanpub.com/api-gateway (дата звернення: 21.01.2025)
14. Ed-douibi H., Cánovas Izquierdo J. L., Cabot J. Automatic Generation of Test Cases for REST APIs: A Specification-Based Approach. 2018 IEEE 22nd International Enterprise Distributed Object Computing Conference (EDOC), Stockholm, Sweden, 2018. P. 181-190. DOI: 10.1109/EDOC.2018.00031
15. Konstantinov S. Mastering HTTP APIs & REST Architectural Style: From the Very Basics to the Full Professional Proficiency, 2025. URL: https://www.amazon.com/Mastering-HTTP-APIs-Architectural-Style-ebook/dp/B0DJHHHM14 (дата звернення: 21.01.2025)
16. Mabotha E., Mabunda N. E., Ali A. et al. Exploring dynamic RESTful API implementation in IoT environments using Docker. *Sci Rep*, 2025. 15:34267. https://doi.org/10.1038/s41598-025-16460-0
17. Masse M. REST API Design Rulebook. Sebastopol: O'Reilly Media, 2009. 168 p.
18. Mutiara G. A., Periyadi, Alfarisi M. R., Zain M. A., Rijali M. G., Rochim F. N. Design and implementation of a REST API-based client-server architecture for multi-sensor IoT monitoring. *International Journal of Advanced Technology and Engineering Exploration.* 2025. № 12(124). P. 426-449.
19. Prayogi A. A., Niswar M., Indrabayu I., Rijal M. Design and Implementation of REST API for Academic Information System. ResearchGate, 2020. URL: https://www.researchgate.net/publication/343164544_Design_and_Implementation_of_REST_API_for_Academic_Information_System (дата звернення: 21.01.2025)
20. Rodríguez C., Báez González M., Daniel F., Casati F., Trabucco J., Canali L., Percannella G. REST APIs: A Large-Scale Analysis of Compliance with Principles and Best Practices. Web Engineering. 16th International Conference, ICWE 2016, Lugano, Switzerland. Springer International Publishing, 2016. С. 21-39. DOI: 10.1007/978-3-319-38791-8_2
21. Richardson L., Ruby S. RESTful Web Services. Sebastopol: O'Reilly Media Inc., 2007. 446 с. URL: https://www.oreilly.com/library/view/restful-web-services/9780596529260/ (дата звернення: 21.01.2025)

Стаття надійшла 25.01.2025