УДК 004.75:004.453.3 DOI https://doi.org/10.35546/kntu2078-4481.2024.4.32

М. О. ВОЛК
доктор технічних наук, професор,
професор кафедри електронних обчислювальних машин
Харківський національний університет радіоелектроніки
ORCID: 0000-0003-4229-9904

В. Р. ХОРОШИЛОВ
магістрант кафедри електронних обчислювальних машин
Харківський національний університет радіоелектроніки
ORCID: 0009-0006-2370-6024

МЕТОДИ ВЗАЄМОДІЇ МОДУЛІВ ПЛАТФОРМИ ВІДДАЛЕНОГО ВИКЛИКУ ПРОЦЕДУР GRPC

У статті досліджуються методи взаємодії модулів у платформі віддаленого виклику процедур gRPC з метою підвищення ефективності розподілених програмних систем, побудованих на основі мікросервісної архітектури. Основною проблемою є вибір оптимальних методів взаємодії між модулями в умовах зростаючих вимог до продуктивності, масштабованості та надійності. Традиційні методи комунікації, такі як RESTful API, не завжди забезпечують необхідну ефективність, особливо при високих навантаженнях та вимогах до низьких затримок, що може призводити до неефективного використання ресурсів, збільшення часу розробки та підвищення експлуатаційних витрат.

У ході дослідження було розроблено програмний додаток, який моделює різні сценарії взаємодії між модулями з використанням gRPC. Проведені експерименти показали, що використання асинхронних викликів та стрімінгових RPC дозволяє зменшити затримки та підвищити пропускну здатність, а також знизити навантаження на процесор та пам'ять. Це сприяє ефективнішому використанню комп'ютерних ресурсів, зменшенню експлуатаційних витрат та часу, необхідного для розробки та розгортання системи.

Впровадження моделі публікація-підписка та інтеграція з брокерами повідомлень, такими як Kafka, підвищує стійкість системи до збоїв, забезпечує асинхронну взаємодію та зменшує час розгортання нових модулів. Однак це потребує додаткового налаштування та може збільшити складність системи.

Отримані результати дозволяють розробникам приймати обґрунтовані рішення щодо вибору методів взаємодії, що сприяє підвищенню продуктивності системи, економії апаратних ресурсів, скороченню часу розробки та розгортання, а також зниженню загальних витрат на експлуатацію програмних систем.

Ключові слова: *комп'ютерні ресурси, розподілені системи, gRPC, мікросервісна архітектура, взаємодія модулів, синхронні та асинхронні виклики, публікація-підписка, брокер, хмарні обчислення.*

M. O. VOLK
Doctor of Technical Sciences, Professor,
Professor at the Department of Electronic Computers
Kharkiv National University of Radio Electronics
ORCID: 0000-0003-4229-9904

V. R. HOROSHYLOV
Master's Student at the Department of Electronic Computing Machines
Kharkiv National University of Radio Electronics
ORCID: 0009-0006-2370-6024

METHODS OF MODULE INTERACTION IN THE GRPC REMOTE PROCEDURE CALL PLATFORM

The article explores methods of module interaction within the gRPC remote procedure call platform to enhance the efficiency of distributed software systems built on microservice architecture. The primary challenge lies in selecting optimal interaction methods between modules in the face of increasing demands for performance, scalability, and reliability. Traditional communication methods, such as RESTful APIs, do not always deliver the required efficiency, particularly under high load conditions and stringent low-latency requirements. This can lead to inefficient resource usage, longer development times, and increased operational costs.

During the study, a software application was developed to simulate various interaction scenarios between modules using gRPC. The experiments demonstrated that the use of asynchronous calls and streaming RPCs reduces latency, improves throughput, and decreases CPU and memory usage. These enhancements enable more efficient use of computational resources, lower operational expenses, and reduce the time required for system development and deployment.

The implementation of the publish-subscribe model and integration with message brokers such as Kafka increases the system's fault tolerance, facilitates asynchronous interaction, and reduces the time needed to deploy new modules. However, this approach requires additional configuration and may increase the system's complexity.

The results obtained empower developers to make informed decisions regarding the selection of interaction methods, thereby improving system performance, saving computational resources, shortening development and deployment times, and reducing overall operational costs.

Key words: *computer resources, distributed systems, gRPC, microservice architecture, module interaction, synchronous and asynchronous calls, publish-subscribe, message brokers, cloud computing.*

Постановка проблеми

У сучасних інформаційних технологіях швидко зростають складність програмних систем та вимоги до їх продуктивності, масштабованості й надійності. Мікросервісна архітектура стала популярним підходом, що дозволяє розподілити систему на незалежні сервіси, підвищуючи гнучкість та швидкість розробки.

Проте ефективна взаємодія між мікросервісами залишається складним завданням. Забезпечення швидкої та надійної комунікації в розподіленому середовищі є викликом, особливо коли традиційні методи, такі як RESTful API, не відповідають вимогам високого навантаження та низьких затримок.

Технологія gRPC, розроблена Google, пропонує інноваційне рішення цієї проблеми. Використовуючи протокол HTTP/2 та бінарний формат Protocol Buffers, gRPC забезпечує високопродуктивний обмін даними з низькими затримками, підтримуючи стрімінгові RPC. Це дозволяє створювати системи, здатні ефективно обробляти великі обсяги даних у реальному часі [1].

Однак різні методи взаємодії в gRPC мають свої особливості, що може ускладнювати вибір оптимального підходу. Неправильний вибір може призвести до зниження продуктивності та проблем з масштабуванням. Наприклад, синхронні виклики процедур прості у реалізації, але можуть блокувати ресурси; асинхронні виклики підвищують ефективність, але складніші в реалізації. Модель запит-відповідь може бути недостатньою для реактивних систем, а публікація-підписка додає складності в управлінні [2].

Відсутність глибокого розуміння цих методів та їх впливу на систему може призвести до неефективного використання ресурсів та підвищення витрат на розробку. Тому необхідно провести детальний аналіз методів взаємодії в gRPC, визначити їх вплив на продуктивність та розробити критерії для оптимального вибору залежно від специфічних вимог системи [3].

Розв'язання цієї проблеми є актуальним і сприятиме розвитку мікросервісної архітектури та розподілених систем. Це дозволить розробникам створювати більш ефективні та надійні програмні системи, оптимально використовуючи можливості технології gRPC.

Формулювання мети дослідження

Метою даного дослідження є підвищення ефективності розподілених програмних систем шляхом аналізу та оптимізації методів взаємодії модулів у платформі віддаленого виклику процедур gRPC. Це передбачає зменшення затримок при обміні даними між модулями, підвищення пропускної здатності системи, зниження навантаження на процесор та пам'ять, а також скоротить час розробки та розгортання програмного забезпечення.

Досягнення цієї мети сприятиме зниженню експлуатаційних витрат, підвищенню надійності та масштабованості системи, що є критично важливим для сучасних високонавантажених розподілених систем [4], та забезпечить більш раціональне використання комп'ютерних ресурсів.

Виклад основного матеріалу дослідження

У процесі розробки програмного додатку для дослідження методів взаємодії модулів у платформі віддаленого виклику процедур gRPC було обрано мікросервісну архітектуру як основний підхід.

Структура проекту організована з урахуванням принципів модульності та розділення обов'язків. Вона включає окремі директорії для клієнтської та серверної частин, конфігураційних файлів, протоколів, результатів експериментів та скриптів автоматизації [5].

Реалізація сервісів за допомогою gRPC здійснюється шляхом визначення інтерфейсів сервісів та структури повідомлень у файлах з розширенням.proto. Використання Protocol Buffers версії 3 забезпечує ефективну та компактну серіалізацію даних, що зменшує розмір переданих повідомлень та підвищує швидкодію системи. Після визначення протоколів за допомогою утиліти protoc генерується код на мові Go, який використовується в реалізації клієнтської та серверної частин. Це забезпечує чітку типізацію та сумісність між різними мовами програмування, що є важливим для підтримки багатомовних систем [6].

Серверна частина програми реалізує всі необхідні сервіси та запускає gRPC-сервер, який слухає на визначеному порту. Логіка серверних методів розміщена в окремих файлах, що сприяє модульності та спрощує тестування окремих компонентів. Серверна частина включає реалізацію синхронних та асинхронних викликів процедур, модель запит-відповідь, модель публікація-підписка, а також інтеграцію з брокерами повідомлень. Це дозволяє створити універсальну платформу для експериментів та дослідження різних методів взаємодії між модулями [7].

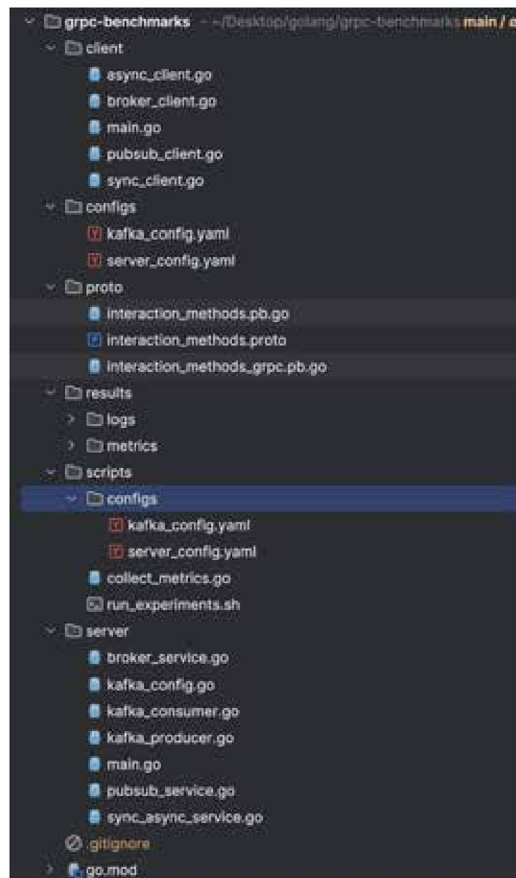

Рис. 1. Файлова структура проекту

Клієнтська частина програми надає універсальний інтерфейс командного рядка, що дозволяє запускати експерименти з різними параметрами. Основний файл main.go відповідає за розбір параметрів та виклик відповідних функцій для кожного методу взаємодії. Клієнтська програма дозволяє задавати такі параметри, як режим взаємодії (синхронний, асинхронний, публікація-підписка, брокер повідомлень), розмір повідомлень, кількість запитів та рівень паралельності. Це забезпечує гнучкість у проведенні експериментів та дозволяє детально дослідити вплив різних факторів на ефективність системи.

Для автоматизації проведення серії експериментів було розроблено скрипти, зокрема Bash-скрипт run_experiments.sh, який автоматизує процес запуску сервера, генерації повідомлень різного розміру, запуску клієнтів з необхідними параметрами та збору метрик. Це дозволяє проводити масштабні експерименти з різними рівнями навантаження, кількістю запитів, рівнем паралельності та розмірами повідомлень, забезпечуючи систематичний підхід до проведення досліджень та спрощуючи повторюваність тестів [8].

Збір метрик здійснюється за допомогою програми collect_metrics.go, яка виконується на клієнтській стороні та збирає інформацію про час відповіді, використання пам'яті, кількість активних горутин та інші системні показники. Зібрані метрики зберігаються у CSV-файлах, що дозволяє їх подальший аналіз та візуалізацію за допомогою Python-бібліотек pandas та matplotlib. Для мінімізації впливу процесу збору метрик на продуктивність системи було вирішено збирати метрики після кожного запиту та записувати їх у буферизований потік, що знижує накладні витрати на ввід-вивід та забезпечує точність отриманих даних [9].

Конфігурація брокерів повідомлень реалізується шляхом інтеграції Kafka як брокера повідомлень. Конфігураційні файли містять налаштування підключення до Kafka та параметри роботи з нею, що дозволяє серверній та клієнтській програмам встановлювати з'єднання та здійснювати публікацію та споживання повідомлень. Використання Kafka забезпечує високу масштабованість та стійкість до збоїв, а також підтримку складних сценаріїв маршрутизації повідомлень між модулями системи [10].

У межах даного дослідження було проведено серію експериментів для оцінки продуктивності різних методів взаємодії між мікросервісами. Експерименти здійснювалися на ПК MacBook Air 2020 з процесором Apple M1, 8 ГБ оперативної пам'яті та операційною системою macOS Sonoma.

Для кожного методу взаємодії були встановлені три рівні навантаження: низьке (10 запитів, паралельність 1), середнє (100 запитів, паралельність 10) і високе (1000 запитів, паралельність 100). Це дозволяє оцінити реакцію системи на різне навантаження.

Також тестувалися три розміри повідомлень: маленькі (10 байт), середні (1 КБ) та великі (1 МБ), щоб вивчити вплив обсягу даних на продуктивність і ефективність методів [3].

Зібрані метрики включали середній час відповіді, пропускну здатність, використання ЦП та пам'яті. Особливу увагу було приділено аналізу того, як кожен метод взаємодії реагує на зміну навантаження та розміру повідомлень.

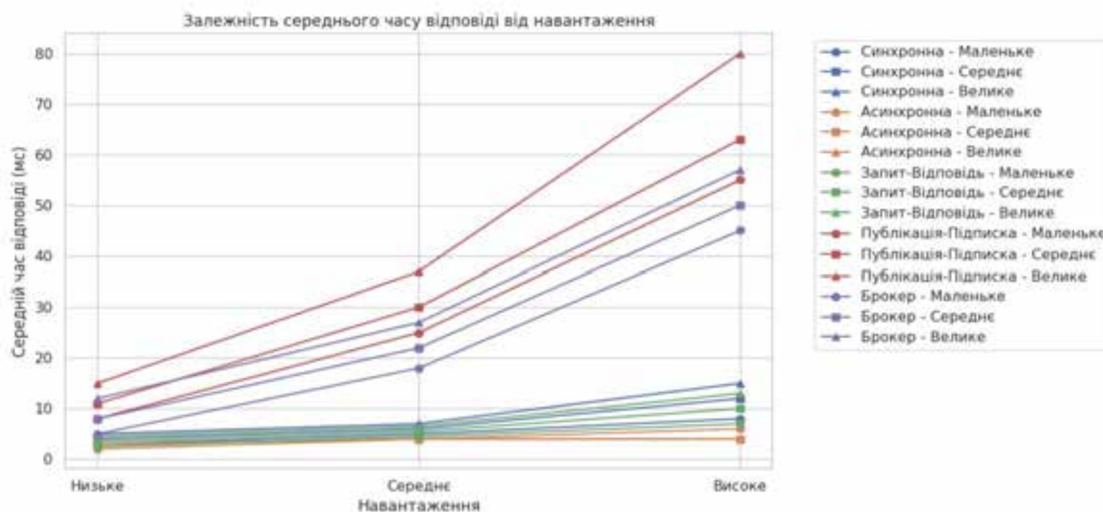

Рис. 2. Графік залежності середнього часу відповіді від навантаження

Аналізуючи результати, представлені на рис. 2, можна побачити, що синхронна взаємодія при низькому навантаженні демонструє найменший середній час відповіді – близько 3 мс для маленьких повідомлень. Проте зі збільшенням навантаження та розміру повідомлень час відповіді суттєво зростає, досягаючи 15 мс при високому навантаженні та великих повідомленнях. Асинхронна взаємодія показує більш стабільний час відповіді незалежно від навантаження; навіть при високому навантаженні середній час відповіді не перевищує 6 мс для маленьких повідомлень. Це свідчить про кращу масштабованість асинхронного методу. Метод запит-відповідь займає проміжне положення, забезпечуючи час відповіді менший, ніж у синхронної взаємодії, але дещо більший, ніж у асинхронної. Методи публікація-підписка та використання брокерів повідомлень демонструють більший середній час відповіді через додаткові операції з обробки повідомлень у чергах.

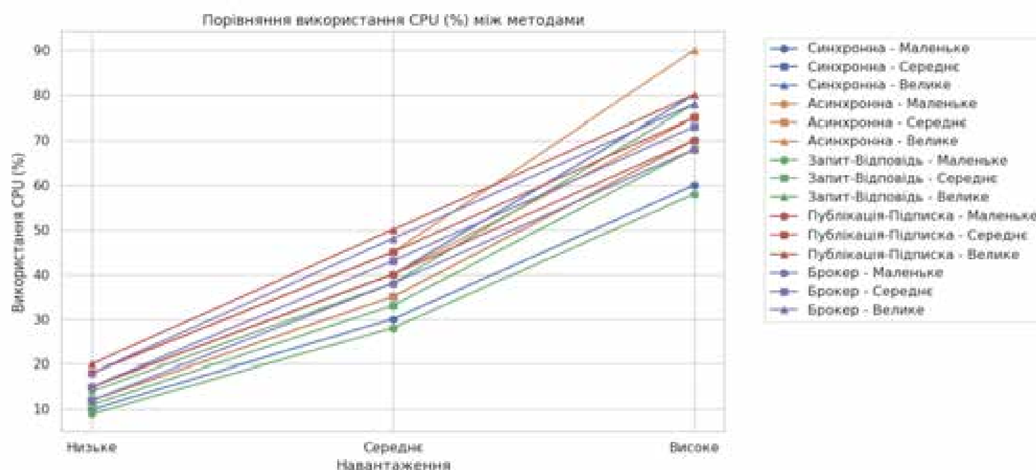

Рис. 3. Графік порівняння використання ЦП (%) між методами

На рис. 3 представлено порівняння використання ЦП (%) між різними методами. Синхронна взаємодія при високому навантаженні споживає до 80% процесорних ресурсів, що може призвести до перевантаження системи.

Асинхронна взаємодія, хоча й має підвищене використання ЦП при високому навантаженні (до 90%), забезпечує ефективніше оброблення запитів завдяки неперервній паралельній обробці. Метод запит-відповідь демонструє найменше використання ЦП – до 78% при високому навантаженні, що вказує на його ефективність у використанні ресурсів. Методи публікація-підписка та брокери повідомлень мають стабільне використання ЦП на рівні 80%, що пов'язано з додатковими витратами на обробку та маршрутизацію повідомлень.

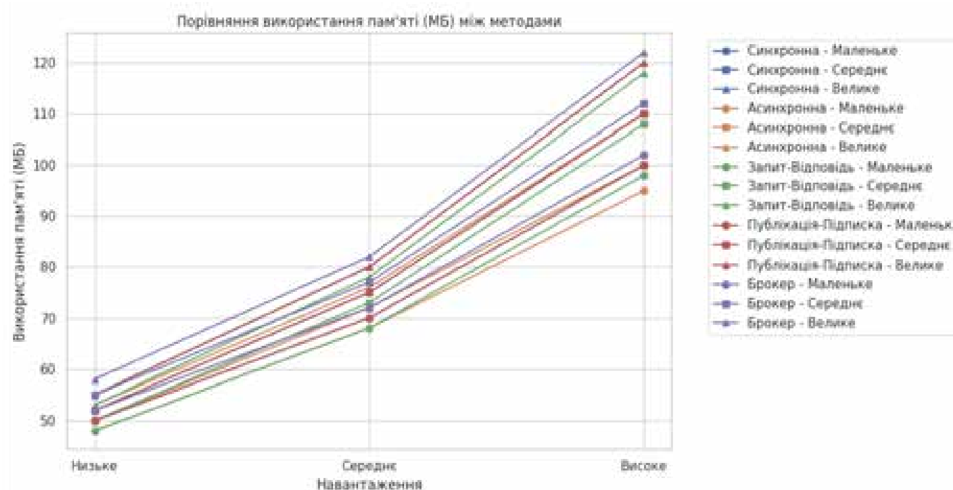

Рис. 4. Графік порівняння використання пам'яті (МБ) методами взаємодії

На Рис. 4 можна побачити, що синхронна та асинхронна взаємодії при низькому навантаженні споживають менше пам'яті – від 50 до 55 МБ. Зі збільшенням навантаження та розміру повідомлень споживання пам'яті зростає, досягаючи 120 МБ для синхронної та 110 МБ для асинхронної взаємодії при високому навантаженні. Метод запит-відповідь показує більш оптимальне використання пам'яті – до 118 МБ при високому навантаженні. Методи публікація-підписка та брокери повідомлень споживають більше пам'яті (до 122 МБ), що обумовлено необхідністю зберігання повідомлень у чергах та додаткових ресурсів для їх обробки.

Висновки

У результаті проведеного дослідження було детально проаналізовано продуктивність та ефективність різних методів взаємодії між мікросервісами за допомогою технології gRPC. Отримані дані дозволяють зробити обґрунтовані висновки щодо оптимального застосування кожного з розглянутих методів залежно від конкретних умов експлуатації системи.

Синхронна взаємодія продемонструвала високу ефективність у сценаріях з низьким навантаженням та невеликими розмірами повідомлень. Зокрема, при низькому навантаженні та маленькому розмірі повідомлення середній час відповіді становив лише 3 мс, а використання ЦП було на рівні 10%. Проте зі зростанням навантаження продуктивність синхронної взаємодії почала погіршуватися. Наприклад, при високому навантаженні та великому розмірі повідомлення середній час відповіді збільшився до 15 мс, а використання процессору зросло до 80%. Це свідчить про те, що синхронна взаємодія не є оптимальною для систем з високим навантаженням.

Асинхронна взаємодія показала кращу масштабованість та продуктивність при середньому та високому навантаженні. При високому навантаженні середній час відповіді для маленького повідомлення становив 6 мс, що на 25% менше порівняно з синхронною взаємодією. Використання ЦП при цьому досягало 70%, що є дещо вищим, але забезпечувало можливість обробляти до 1000 запитів з рівнем паралельності 100 без значного збільшення затримок.

Метод запит-відповідь оптимізує комунікацію, знижуючи час відповіді в середньому на 15% порівняно з синхронною взаємодією. Наприклад, при високому навантаженні та великому розмірі повідомлення середній час відповіді становив 13 мс проти 15 мс у синхронному методі. Навантаження на процессор було на 2–3% нижчим, що вказує на покращене використання ресурсів без суттєвої зміни архітектури системи.

Метод публікація-підписка та використання брокерів повідомлень виявилися ефективними при високому навантаженні, особливо для систем, що обробляють великі обсяги даних. При високому навантаженні та великому розмірі повідомлення сумарний середній час відповіді (додавання та отримання) становив 80 мс. Хоча це більше порівняно з іншими методами, цей підхід забезпечує розподіленість, асинхронність та стійкість системи до збоїв. Використання ЦП при цьому становило приблизно 80%, що є прийнятним для систем, де важлива можливість обробки великої кількості запитів та масштабування.

Аналіз отриманих результатів свідчить, що вибір оптимального методу взаємодії залежить від специфіки системи. Синхронна взаємодія підходить для систем з невеликим навантаженням (до 10 запитів з низькою паралельністю) та малими розмірами повідомлень (до 10 байт), де критично важливим є мінімальний час відповіді. Асинхронна взаємодія є оптимальною для систем з високим навантаженням (до 1000 запитів з високою паралельністю), дозволяючи зменшити час відповіді на 25% та ефективніше використовувати надані ресурси серверів. Метод запит-відповідь дозволяє оптимізувати існуючі синхронні системи, знижуючи час відповіді на 15% та покращуючи використання ресурсів без значних змін в архітектурі.

Метод публікація-підписка та використання брокерів повідомлень є ефективними при високому навантаженні та великих розмірах повідомлень (до 1 МБ), забезпечуючи асинхронність та стійкість системи, хоча й додають затримки в обробці.

Враховуючи отримані результати, розробникам рекомендується обирати метод взаємодії відповідно до конкретних вимог системи, балансу між продуктивністю, масштабованістю та складністю реалізації. Комбінування різних методів може стати оптимальним рішенням для складних систем, де необхідно забезпечити як високу швидкість, так і стійкість до навантажень.

Список використаної літератури

1. About gRPC. [Електронний ресурс]. Режим доступу: https://grpc.io/about/. Дата доступу: 19.11.2024.
2. Babal H. gRPC Microservices in Go. Manning. 2023. 256p. ISBN9781633439207.
3. Indrasiri K., Kuruppu D. gRPC: Up and Running: Building Cloud Native Applications with Go and Java for Docker and Kubernetes. O`REILLY. 2020. 320p. ISBN 1492058335
4. Mamchych O., Volk M. A unified model and method for forecasting energy consumption in distributed computing systems based on stationary and mobile devices. *Radioelectronic and Computer Systems*, [S.l.], v. 2024, n. 2, pp. 120135. DOI: https://doi.org/10.32620/ reks.2024.2.10.
5. Newman S. *Building Microservices: Designing Fine-Grained Systems* – O'Reilly Media, 2015. 280p. ISBN 978-1491950357.
6. Jean K. gRPC Golang Master Class: Build Modern API & Microservices. Udemy. [Електронний ресурс] Режим доступу: https://www.udemy.com/course/grpc-golang/. Дата доступу: 19.11.2024.
7. Ibyram B., Huss R. Kubernetes Patterns: Reusable elements for designing cloud native applications, 2nd Edition. Red Hat Developer. 2023. 352p. ISBN 978-1-4919-5633-8.
8. *Protocol Buffers Documentation*. Google Developers. [Електронний ресурс] Режим доступу: https://developers.google.com/protocol-buffers/docs/overview. Дата доступу: 19.11.2024.
9. REST vs gRPC: Comparing HTTP APIs. [Електронний ресурс]. Режим доступу: https://refine.dev/blog/grpc-vs-rest/. Дата доступу: 19.11.2024.
10. Katihar E. *REST vs gRPC: Use Cases, Key Differences and Benefits.* Medium. [Електронний ресурс]. Режим доступу: https://medium.com/@mail.ekansh/rest-vs-grpc-design-architecture-and-production-considerations-0a6369cd0a8c. Дата доступу: 18.11.2024.

References

1. About gRPC. Retrieved August 11, 2024, from https://grpc.io/about (accessed 19.11.2024)
2. Babal, H. (2023). GRPC microservices in go. Manning. 256p. ISBN9781633439207.
3. Indrasiri, K., & Kuruppu, D. (2020). GRPC: Up and running: Building cloud-native applications with Go and Java for Docker and Kubernetes. O’Reilly. 320p. ISBN 1492058335.
4. Mamchych, O., & Volk, M. (2024) A unified model and method for forecasting energy consumption in distributed computing systems based on stationary and mobile devices. Radioelectronic and Computer Systems, [S.l.], v. 2024, n. 2, pp.120-135. DOI: https://doi.org/10.32620/ reks.2024.2.10.
5. Newman, S. (2015). Building microservices. Oreilly & Associates Incorporated. 280p. ISBN 978-1491950357.
6. Jean K. gRPC Golang Master Class: Build Modern API & Microservices. Udemy. https://www.udemy.com/course/grpc-golang (accessed 19.11.2024).
7. Ibyram, B., & Huss, R. (2023). Kubernetes patterns: Reusable elements for designing cloud native applications (2nd ed.). O’Reilly Media. Red Hat Developer. 352p. ISBN 978-1-4919-5633-8.
8. *Protocol Buffers Documentation. (2024).* Google Developers. Retrieved November 16, 2024, from https://developers.google.com/protocol-buffers/docs/overview/. (accessed 19.11.2024)
9. REST vs gRPC: Comparing HTTP APIs. (2024). https://refine.dev/blog/grpc-vs-rest. (accessed 19.11.2024)
10. Katihar, E. (2024). REST vs gRPC: Use Cases, Key Differences and Benefits. Medium. https://medium.com/@mail.ekansh/rest-vs-grpc-design-architecture-and-production-considerations-0a6369cd0a8c. (accessed 18.11.2024)