УДК 004.42(043.2)

А.О. Шевчук
О.Н. Романюк

ВИКОРИСТАННЯ МІКРОСЕРВІСНОЇ АРХІТЕКТУРИ У РОЗРОБЦІ МАСШТАБОВАНИХ ВЕБЗАСТОСУНКІВ

Вінницький національний технічний університет

Анотація

У тезах розглянуто використання мікросервісної архітектури у процесі розробки масштабованих вебзастосунків. Описано основні принципи поділу програмної системи на незалежні сервіси, особливості їх взаємодії та вплив такого підходу на підтримку, розгортання і розвиток програмного забезпечення. Наведено переваги мікросервісів порівняно з монолітною архітектурою, а також визначено обмеження, які необхідно враховувати під час проєктування розподілених систем.

Ключові слова: мікросервісна архітектура, вебзастосунок, масштабованість, програмне забезпечення, API, DevOps, розподілена система.

Abstract

This paper examines the use of microservice architecture in the development of scalable web applications. It describes the basic principles of dividing a software system into independent services, the specifics of their interaction, and the impact of this approach on software maintenance, deployment, and development. The advantages of microservices over monolithic architecture are presented, and the limitations that must be considered when designing distributed systems are identified.

Keywords: microservices architecture, web application, scalability, software, API, DevOps, distributed system.

Вступ

Сучасні вебзастосунки дедалі частіше працюють із великими обсягами даних, значною кількістю користувачів та необхідністю швидкого впровадження нового функціоналу. У таких умовах архітектура програмної системи безпосередньо впливає на її надійність, продуктивність, зручність супроводу та можливість подальшого розвитку. Для невеликих проєктів традиційна монолітна архітектура може бути зручною, оскільки вся логіка зосереджена в одному застосунку. Проте зі зростанням складності системи та кількості функціональних модулів монолітний підхід часто ускладнює масштабування й оновлення окремих компонентів.

Одним із перспективних напрямів в інженерії програмного забезпечення є використання мікросервісної архітектури. Вона передбачає поділ застосунку на набір автономних сервісів, кожен із яких відповідає за окрему бізнес-функцію та взаємодіє з іншими компонентами через чітко визначені інтерфейси. Такий підхід дозволяє розробляти, тестувати, розгортати та масштабувати сервіси незалежно один від одного, що є важливою перевагою для сучасних вебсистем [1].

Мета дослідження: обґрунтувати доцільність використання мікросервісної архітектури у розробці масштабованих вебзастосунків, проаналізувати її основні переваги й обмеження, а також визначити практичні підходи до проєктування та впровадження таких систем.

Особливості мікросервісної архітектури

Мікросервісна архітектура розглядається як спосіб побудови програмної системи з набору невеликих незалежних сервісів. Кожен сервіс має власну логіку, відповідає за конкретну частину предметної області та може розгортатися окремо від інших компонентів. На відміну від монолітної архітектури, де всі модулі пов’язані в межах одного застосунку, мікросервіси дають змогу зменшити залежність між частинами системи.

У вебзастосунках окремими сервісами можуть бути модуль автентифікації, каталог товарів, обробка замовлень, платіжний модуль, система повідомлень, аналітика або сервіс керування користувачами. Кожен із них виконує конкретну задачу, але разом вони формують єдиний програмний продукт. Взаємодія між сервісами зазвичай здійснюється через REST API, gRPC або асинхронні черги повідомлень [2].

Важливою особливістю мікросервісної архітектури є можливість використання різних технологій для різних компонентів. Наприклад, один сервіс може бути реалізований мовою Python, інший - Java або Go, а для збереження даних можуть застосовуватися як реляційні, так і нереляційні бази даних. Такий підхід забезпечує технологічну гнучкість, але водночас потребує узгоджених правил взаємодії між компонентами.

Переваги та обмеження використання мікросервісів

Основною перевагою мікросервісів є незалежне розгортання. Якщо потрібно змінити або оновити окремий сервіс, немає необхідності повністю перевипускати весь застосунок. Це зменшує ризики під час релізів, прискорює доставку нового функціоналу та дозволяє командам працювати паралельно над різними частинами системи.

Другою важливою перевагою є гнучке масштабування. У монолітній архітектурі зазвичай доводиться масштабувати весь застосунок, навіть якщо навантаження зростає лише на один модуль. У мікросервісній системі можна збільшити кількість екземплярів саме того сервісу, який має найбільше навантаження, наприклад сервісу пошуку, оплати або обробки повідомлень.

Мікросервіси також підвищують відмовостійкість системи. Помилка в одному сервісі не обов'язково призводить до повної зупинки всього застосунку. За умови правильної реалізації механізмів повторних запитів, обмеження часу очікування та резервних сценаріїв система може продовжувати працювати навіть у разі часткових збоїв.

Разом із тим мікросервісна архітектура має низку обмежень. Вона ускладнює інфраструктуру, потребує якісного моніторингу, централізованого логування, автоматизованого тестування та налаштованого CI/CD-процесу. Додатковими викликами є узгодженість даних, безпека міжсервісної взаємодії, обробка мережевих помилок та складність налагодження розподілених запитів [3].

Порівняння монолітної та мікросервісної архітектур

Вибір архітектурного підходу залежить від масштабу проєкту, кількості функціональних модулів, вимог до продуктивності та можливостей команди. Для невеликих застосунків монолітна архітектура може бути простішою й дешевшою у впровадженні. Для великих систем, що активно розвиваються, мікросервіси можуть забезпечити кращу гнучкість і масштабованість.

Таблиця 1 - Порівняння монолітної та мікросервісної архітектур

Критерій	Монолітна архітектура	Мікросервісна архітектура	Практичне значення
Розгортання	Оновлюється весь застосунок	Оновлюються окремі сервіси	Зменшення ризиків під час релізів
Масштабування	Масштабується вся система	Масштабуються потрібні сервіси	Економія ресурсів
Супровід	Складніший у великих проєктах	Простіший для окремих модулів	Зручніша командна робота
Інфраструктура	Простіша в налаштуванні	Потребує DevOps-процесів	Необхідність автоматизації

Проєктування та впровадження мікросервісних систем

Проєктування мікросервісної системи доцільно починати з аналізу предметної області та визначення меж відповідальності окремих сервісів. Неправильне розбиття системи може призвести до надмірної кількості залежностей, дублювання логіки та складної взаємодії між компонентами. Тому під час проєктування важливо дотримуватися принципу високої зв'язності всередині сервісу та слабкої зв'язаності між сервісами.

Для керування даними часто застосовується підхід, за якого кожен сервіс має власне сховище. Це зменшує залежність між компонентами, але ускладнює виконання операцій, що охоплюють кілька сервісів. У таких випадках можуть використовуватися шаблони Saga, Event Sourcing або асинхронна синхронізація даних. Вибір конкретного підходу залежить від вимог до узгодженості, швидкодії та надійності системи [4].

Важливу роль у впровадженні мікросервісів відіграють контейнеризація та оркестрація. Контейнери дають змогу запускати сервіси в ізольованому середовищі з необхідними залежностями, а системи оркестрації забезпечують масштабування, перезапуск у разі збоїв і керування розгортанням. Це робить мікросервісну архітектуру зручною для хмарних середовищ і сучасних DevOps-процесів.

Практичне значення для інженерії програмного забезпечення

У межах спеціальності «Інженерія програмного забезпечення» мікросервісна архітектура має важливе практичне значення, оскільки поєднує питання проєктування, програмування, тестування, розгортання та супроводу програмних систем. Вивчення цього підходу допомагає краще розуміти принципи побудови складних вебзастосунків, роботу серверної інфраструктури та взаємодію між різними компонентами програмного продукту.

Для навчальних і практичних проєктів мікросервіси доцільно використовувати тоді, коли система має кілька незалежних функціональних модулів, потребує масштабування або може розвиватися різними командами. Водночас для невеликих застосунків монолітна архітектура часто залишається простішою та ефективнішою. Тому вибір архітектури повинен ґрунтуватися не лише на популярності підходу, а й на реальних вимогах до проєкту.

Висновок

Застосування мікросервісної архітектури у розробці масштабованих вебзастосунків відкриває значні можливості для підвищення гнучкості, надійності та ефективності програмних систем. Мікросервіси забезпечують незалежне розгортання компонентів, гнучке масштабування окремих функцій і кращу організацію командної розробки. Водночас цей підхід потребує ретельного проєктування, автоматизації процесів розгортання, якісного моніторингу та продуманої роботи з даними.

Отже, мікросервісна архітектура є доцільною для складних систем із високими вимогами до масштабованості та розвитку, але не повинна використовуватися без аналізу потреб конкретного проєкту. Подальші дослідження доцільно спрямувати на порівняння ефективності монолітної та мікросервісної архітектур у навчальних і комерційних вебзастосунках, а також на вивчення методів автоматизованого тестування розподілених програмних систем.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Fowler M., Lewis J. Microservices. URL: https://martinfowler.com/articles/microservices.html (дата звернення: 29.04.2026).
2. Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd ed. Sebastopol : O’Reilly Media, 2021. 616 p.
3. Richardson C. Microservices Patterns: With examples in Java. Shelter Island : Manning Publications, 2018. 520 p.
4. Bass L., Clements P., Kazman R. Software Architecture in Practice. 4th ed. Boston : Addison-Wesley Professional, 2021. 464 p.

Шевчук Артем Олександрович - студент спеціальності 121 «Інженерія програмного забезпечення», факультет інформаційних технологій та комп’ютерної інженерії, Вінницький національний технічний університет, м. Вінниця, e-mail: arten701@gmail.com.

Романюк Олександр Никифорович – доктор технічних наук, професор, професор кафедри програмного забезпечення, завідувач кафедри програмного забезпечення, Вінницький національний технічний університет, м. Вінниця, e–mail: rom8591@gmail.com.

Shevchuk Artem O. – student of group 1PI-22b, Faculty of Information Technologies and Computer Engineering, Vinnytsia National Technical University, Vinnytsia, e-mail: arten701@gmail.com

Romanyuk Oleksandr N. – Doctor of Technical Sciences, Professor, Professor of the Software Engineering Department, Head of the Software Engineering Department, Vinnytsia National Technical University, Vinnytsia, e–mail: rom8591@gmail.com.