УДК 004.415:004.72 DOI https://doi.org/10.30837/IYF.PDICIMT.2024.152

ПОРІВНЯННЯ МОНОЛІТНОЇ ТА МІКРОСЕРВІСНОЇ АРХІТЕКТУРИ У РОЗРОБЦІ У ВЕБЗАСТОСУНКІВ

Скворцов В. Х.
Науковий керівник – к.т.н., доц. Золотарьов В. А.
Харківський національний університет радіоелектроніки, каф. IMI,
м. Харків, Україна
e-mail: vladyslav.skvortsov@nure.ua.

This paper is dedicated to the analysis and comparison of two approaches to software architecture in modern web development: monolithic and microservices. It discusses the characteristics, advantages, and challenges of each approach with the aim of determining the optimal solution for specific projects or organizations. The principles and advantages of microservices architecture are examined, along with the challenges faced by developers, compared to the main principles and advantages of the monolithic approach. Various aspects are considered during the analysis, including flexibility, scalability, and risk management, to provide a comprehensive assessment of both architectural models.

На сьогодні існує два підходи до архітектури програмного забезпечення у вебзастосунках: монолітна та мікросервісна (див. рисунок 1). На зображенні представлені такі елементи, як монолітна (Monolith) та мікросервісна (Microservices) архітектури, інтерфейс користувача (UI), бізнес-логіка (Business Logic) та доступ до даних (Data Access).
Метою доповіді є проаналізувати та порівняти два підходи до архітектури програмного забезпечення. Через огляд їх особливостей, переваг та викликів прагнемо з'ясувати, який підхід може бути найвигіднішим для конкретного проєкту чи організації.

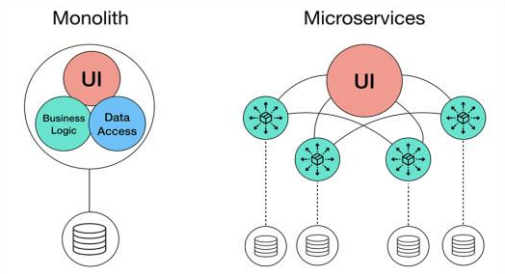

Рисунок 1 – Порівняння мікросервісної та монолітної архітектур
Мікросервісна архітектура (Microservices architecture) – це сучасний підхід до розробки програмного забезпечення, що базується на розбитті великих застосунків на невеликі, незалежні, добре визначені та автономні сервіси, які спілкуються між собою через API [1].

Мікросервісна архітектура ґрунтується на кількох принципах, які формують його основу. Перш за все, це розбиття застосунку на невеликі сервіси, кожен з яких відповідає за конкретну функціональність, і, важливо, ці сервіси повинні бути максимально незалежними та автономними. Далі, сервіси взаємодіють між собою через API, яке може працювати на різних протоколах, таких як HTTP, gRPC, та інші, що забезпечує легкість інтеграції та зміни. Кожен сервіс може мати свою власну базу даних або використовувати різні системи збереження даних, такі як SQL або NoSQL бази, що зменшує залежність між ними та сприяє легшому масштабуванню [2].

Крім того, кожен сервіс може бути розгорнутий незалежно від інших, що дозволяє швидко впроваджувати зміни та оновлення без необхідності перекомпіляції всього застосунку. Нарешті, сервіси можуть бути масштабовані незалежно один від одного залежно від навантаження, що дозволяє ефективно використовувати ресурси.

Щодо переваг мікросервісної архітектури, вона забезпечує гнучкість у внесенні змін та розвитку окремих сервісів, масштабованість, що дозволяє легко реагувати на потреби, швидкість розгортання нових функцій та оновлень через незалежність сервісів, та зменшення ризиків виникнення великих збоїв.

Однак, існують і виклики, зокрема, складність управління та моніторингу багатьох сервісів, збільшення мережевих витрат через збільшення комунікації, а також ускладнене тестування та налагодження через розподілений характер архітектури. Таким чином, мікросервісний підхід може бути ефективним для великих, складних застосунків, але потребує додаткових зусиль у проєктуванні та управлінні порівняно з традиційними монолітними архітектурами.

Монолітна архітектура (Monolithic architecture) – це традиційний підхід до розробки програмного забезпечення, при якому весь застосунок побудований як єдине ціле, з усіма його компонентами та функціональністю, які взаємодіють між собою в межах одного кодового базису [3].

Основні принципи монолітної архітектури полягають у тому, що застосунок розробляється та підтримується як єдине ціле. Усі компоненти та функціональність зазвичай знаходяться в межах одного кодового базису, що робить розробку простішою [4].

Проте така єдність також призводить до того, що всі компоненти взаємозалежні, а це своєю чергою ускладнює управління ризиками. Проблеми в одному компоненті можуть мати каскадний ефект на всю систему, роблячи виявлення та виправлення проблем складнішими. Крім того, масштабування такої архітектури також може бути проблематичним, оскільки всі компоненти знаходяться в межах одного кодового базису. Хоча є переваги, наприклад, простота розробки та зменшення накладних

витрат, через меншу потребу у комунікації та інтеграції, проте виклики монолітної архітектури, такі як складність масштабування та залежність компонентів, можуть стати перешкодою для великих та складних систем.
Тому монолітний підхід може бути ефективним для менших проєктів або проектів з обмеженими ресурсами, але не завжди підходить для великих та складних систем, які вимагають гнучкості та масштабованості.
У монолітної архітектури є переваги, зокрема простота розробки та зменшення накладних витрат на комунікацію та інтеграцію, оскільки весь код знаходиться в одному місці. Однак це може призвести до проблем з масштабуванням та управлінням ризиками, оскільки всі компоненти взаємозалежні. З іншого боку, мікросервіси дозволяють розділити застосунок на окремі сервіси, що полегшує розробку та підтримку, а також дозволяє більш гнучко масштабувати окремі частини системи та управляти ризиками.
Вибір між монолітною та мікросервісною архітектурою залежить від потреб та вимог проєкту. Монолітний підхід підходить для менших проєктів або тих, що мають обмежені ресурси, де важлива простота та швидкість розробки. Однак він може бути непрактичним для великих та складних систем, які потребують гнучкості та масштабованості. Мікросервіси відповідають складним та розподіленим системам, де ключові гнучкість, масштабованість та управління ризиками.
На закінчення відзначимо, що обидві архітектури мають свої переваги та недоліки, і рішення має ґрунтуватися на конкретних вимогах проєкту, що розглядається. Ретельно оцінюючи компроміси між монолітною та мікросервісною архітектурою, можна вибрати найбільш відповідне рішення для досягнення цілей.

Список використаних джерел:

1. What are microservices IBM: вебсайт. URL: https://www.ibm.com/topics/microservices (дата звернення: 10.02.2024).
2. What are microservices AWS: вебсайт. URL: https://aws.amazon.com/ru/microservices/ (дата звернення: 10.02.2024).
3. Monolithic Architecture: вебсайт. URL: https://www.geeksforgeeks.org/monolithic-architecture/ (дата звернення: 10.02.2024).
4. Understanding Monolith Architecture: A Guide for Product Managers: вебсайт. URL: https://bootcamp.uxdesign.cc/understanding-monolith-architecture-a-guide-for-product-managers-1c7bd3c3a3cf (дата звернення: 10.02.2024).