Лекція 2: Архітектурні стилі та патерни

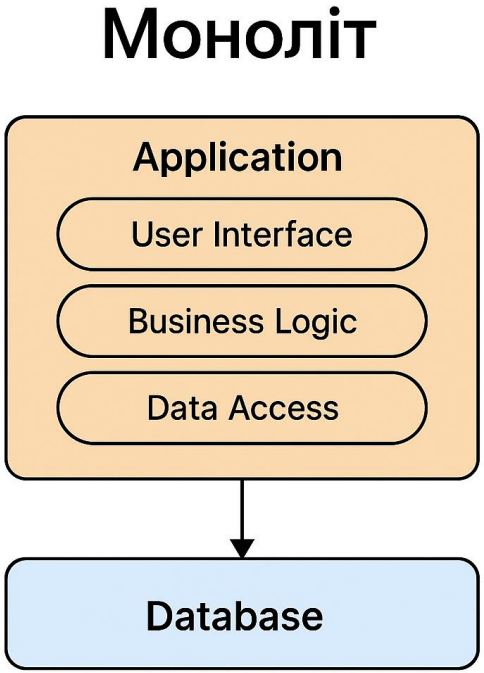
Моноліт, SOA, мікросервіси, REST, GraphQL, патерни

Поняття архітектурного стилю

- - Структурна організація системи
- - Визначає взаємодію компонентів
- - Впливає на масштабованість і продуктивність

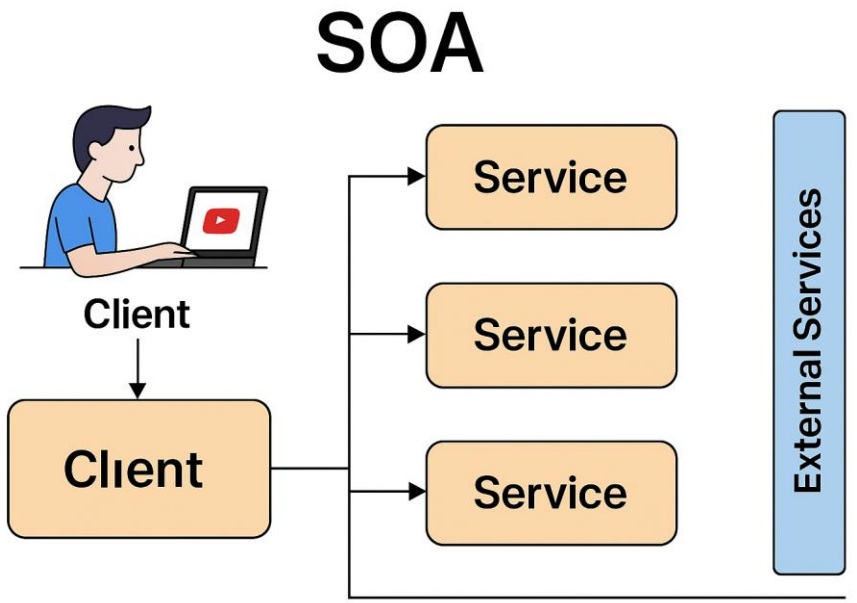
Моноліт

- - Один великий застосунок
- - Плюси: простота, швидкий старт
- - Мінуси: складність масштабування



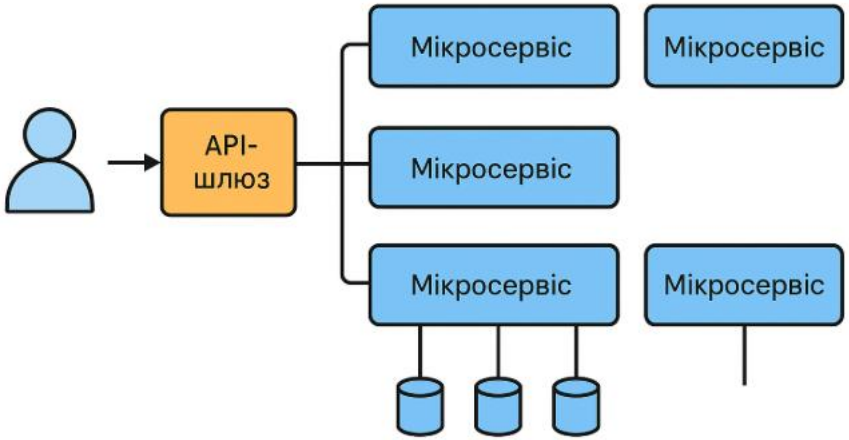
SOA

- - Сервісно-орієнтована архітектура
- - Система складається з сервісів
- - Взаємодія через API



Мікросервіси

- - Незалежні дрібні сервіси
- - Легко масштабуються
- - Приклади: Netflix, Amazon

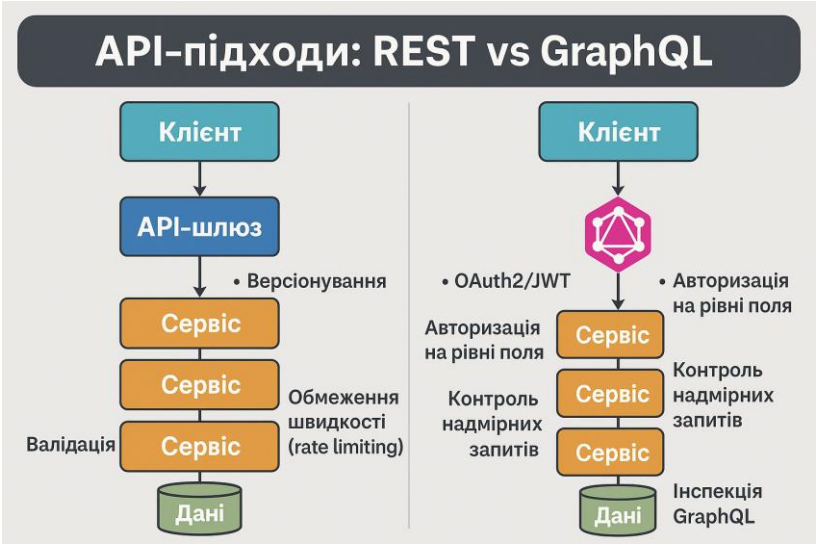


API-підходи: REST vs GraphQL

- - REST: HTTP-методи, ресурси
- - GraphQL: вибіркові дані

REST, приклад:

– Запит GET /users/1 поверне інформацію про користувача з ID 1.
– Запит POST /users дозволить створити нового користувача.



GraphQL,
приклад коду:

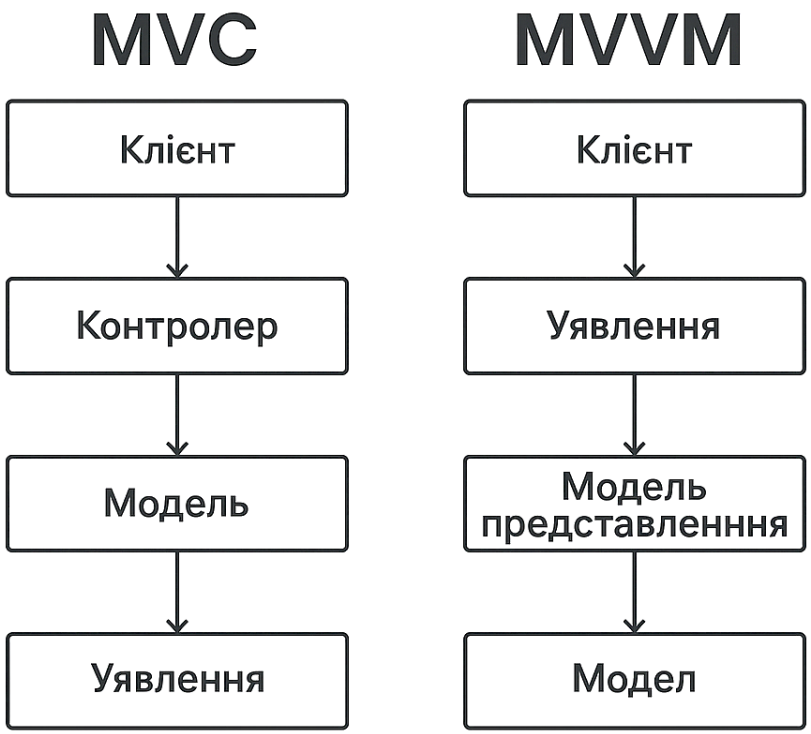
```
{
  user(id: 1) {
    name
    email
  }
}
```

Таблиця порівняння REST і GraphQL

Критерій	REST	GraphQL
Архітектура	- Орієнтований на ресурси (URL → ресурс) - Використовує HTTP-методи (GET, POST, PUT, DELETE) - Один запит → одна відповідь від сервісу	- Орієнтований на запити - Клієнт сам формує структуру відповіді - Один запит може об’єднати кілька джерел даних
Проєктування	- Простий у реалізації - Потребує версіонування (v1, v2…) - Добре підходить для невеликих та середніх систем	- Гнучкий, немає жорсткого версіонування - Економія трафіку (немає overfetching) - Ідеальний для мобільних застосунків та складних UI
Безпека	- Використовує стандартні механізми (OAuth2, JWT) - Контроль доступу до ресурсів - Кешування для оптимізації	- Потребує додаткового захисту: – контроль доступу до окремих полів – обмеження складності запитів - Теж використовує JWT/OAuth2
Приклад з життя	- Запит на профіль користувача → сервер повертає весь профіль (навіть якщо потрібне тільки ім’я)	- Запит на профіль користувача → сервер повертає лише ті поля, які клієнт запросив (наприклад, тільки ім’я й email)

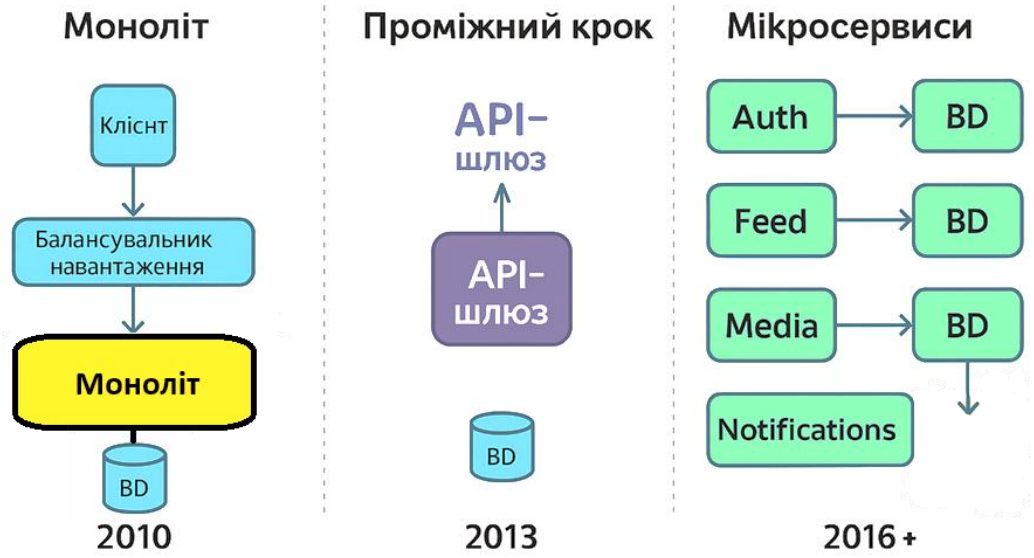
Архітектурні патерни проєктування

- - MVC: Model-View-Controller
- - MVVM: Model-View-ViewModel



Кейс: Instagram

- Спочатку моноліт
- Поступовий перехід на мікросервіси
- Використання REST API



Висновки

- Вибір архітектури визначає гнучкість системи
- Патерни спрощують розробку

Ключові моменти:

- Моноліт — протіше для старту, але має обмеження.
- SOA — компромісний варіант для великих корпоративних систем.
- Мікросервіси — сучасний підхід для масштабних продуктів.
- REST і GraphQL визначають, як сервіси взаємодіють.
- Патерни MVC та MVVM роблять код структурованим і зрозумілим.