



ТЕМА 12

Від моноліту до мікросервісів

Принципи · переваги й недоліки · комунікація між сервісами · коли доцільно

Бекенд-розробка, сервіси та мікросервіси

Комп'ютерна інженерія · 3 курс · Лекція 12 з 18

7 питань сьогоднішньої теми

01

Монолітна архітектура

02

Проблеми моноліту при зростанні

03

Принципи мікросервісів

04

Переваги мікросервісів

05

Недоліки та складнощі

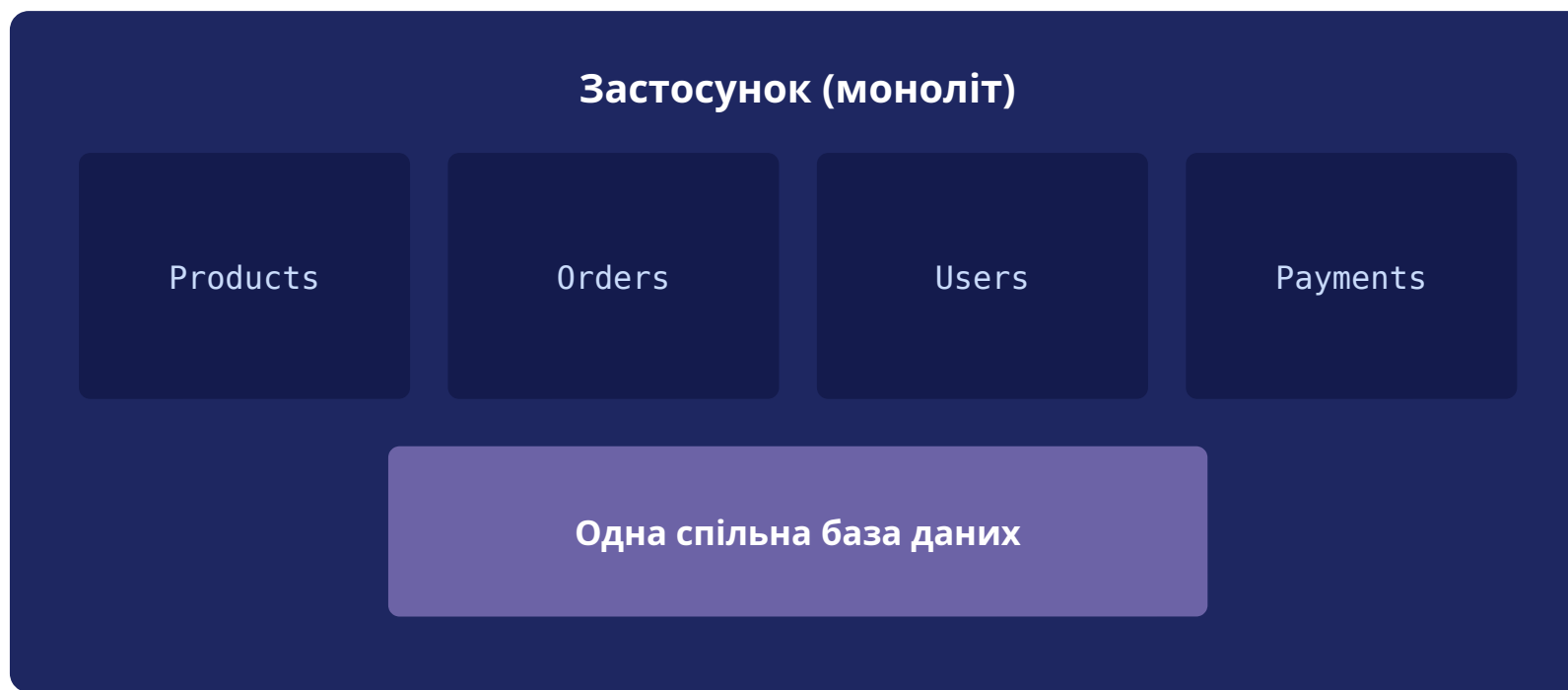
06

Комунікація між сервісами

07

Коли доцільно переходити

Один застосунок, одна база даних

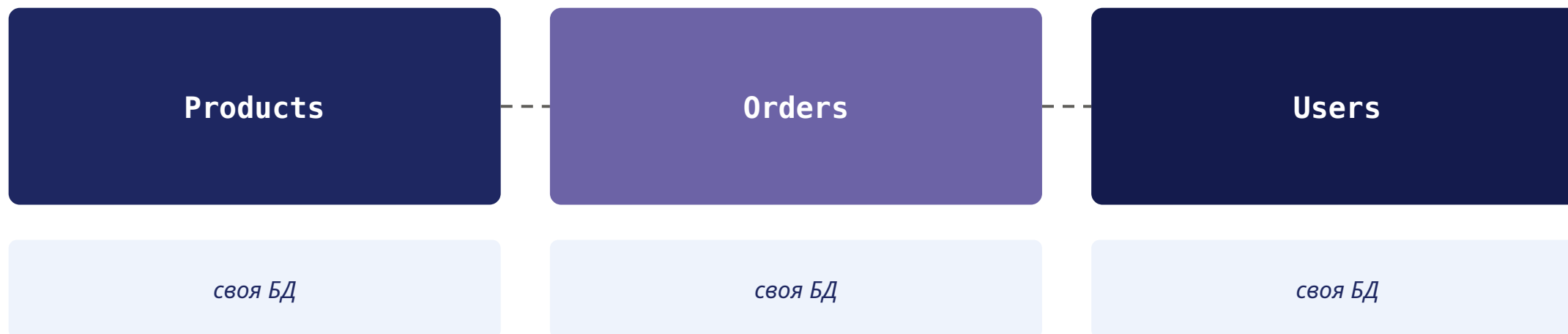


Розгортається й виконується як ЄДИНИЙ процес — саме так був побудований застосунок протягом усього курсу.

Шість типових проблем

Проблема	Прояв
Складність розгортання	Зміна в одному модулі вимагає передеплою всього
Масштабування "всього"	Навантажений лише один модуль — масштабуємо все
Технологічна прив'язка	Увесь застосунок — одна мова й версія фреймворку
Конфлікти команди	Кілька команд в одному репозиторії — часті конфлікти
Крихкість	Помилка в одному модулі може "вивести з ладу" все

Незалежні сервіси, власні дані



Кожен сервіс — окрема бізнес-можливість, розгортається й масштабується незалежно, комунікує через мережу (API), а не прямі виклики функцій.

П'ять ключових принципів

Принцип	Суть
Єдина відповідальність	Кожен сервіс — одна бізнес-можливість
Незалежне розгортання	Оновлення без зупинки інших сервісів
Децентралізовані дані	Власна БД у кожного сервісу
Комунікація через мережу	API (HTTP), а не прямі виклики функцій
Незалежність технологій	Різні сервіси — різні мови/БД

Якщо два сервіси звертаються до однієї БД, вони насправді залишаються прихованим монолітом.

Переваги мікросервісів

- Незалежне масштабування навантажених частин системи
- Незалежне розгортання — команди не блокують одна одну
- Ізоляція збоїв: падіння одного сервісу не валить усе
- Свобода вибору технологій під конкретну задачу
- Менші, зрозуміліші кодові бази для нових розробників

Не безкоштовне покращення

Складність	Опис
Розподілені транзакції	Операція через кілька сервісів — не одна проста транзакція БД
Мережева ненадійність	Виклик іншого сервісу може зависнути чи не дійти
Складність відлагодження	Запит проходить через 5–10 сервісів — важче знайти помилку
Операційне навантаження	Десятки сервісів замість одного застосунку
Узгодженість даних	Дані можуть тимчасово розходитись між сервісами

Синхронна vs асинхронна

REST API (синхронна)

```
GET http://products-service  
    /api/products/42  
  
// Orders чекає на відповідь,  
// перш ніж продовжити
```

Потрібна негайна відповідь

Черга повідомлень (асинхронна)

```
publish('order.created', [  
    'orderId' => 501,  
]);  
  
// Notifications обробить,  
// коли забажає
```

Стійкість до тимчасової недоступності

Пастка: "розподілений моноліт"

Надмірне використання синхронних викликів між сервісами створює приховану залежність: якщо Products недоступний, Orders теж перестає працювати.

Формально сервіси окремі, але фактично так само жорстко зв'язані, як методи одного класу — тільки повільніше й ненадійніше через мережу.

Мікросервіси ЙМОВІРНО ВИПРАВДАНІ, якщо:



Кілька незалежних команд (5+ розробників) заважають одна одній



Різні частини мають кардинально різні вимоги до масштабування



Потрібно розгортати частини системи в різному темпі



Команда вже має досвід і інфраструктуру для розподілених систем

Мікросервіси ЙМОВІРНО ЗАЙВІ, якщо:

- ✗ Проєкт малий (1–5 розробників) — операційні витрати переважають користь
- ✗ Моноліт поки не створює реальних проблем
- ✗ У команди немає досвіду роботи з розподіленими системами
- ✗ Бізнес-логіка ще активно змінюється, межі сервісів неочевидні

Принцип "MonolithFirst"

Мартін Фаулер: більшість команд повинні починати з моноліту й переходити до мікросервісів лише тоді, коли моноліт справді почав створювати проблеми — а не з першого дня "про всяк випадок".

Практична порада: будувати добре структурований модульний моноліт (тема 9) із чіткими межами між модулями — це залишає можливість виокремити мікросервіс пізніше, значно дешевше.

Підсумок теми 12

- Моноліт — не синонім поганого коду; добре структурований моноліт цілком нормальний
- Проблеми моноліту проявляються при зростанні команди й навантаження
- Кожен мікросервіс має власну базу даних — спільна БД означає прихований моноліт
- Мікросервіси обмінюють одні проблеми на інші (мережа, розподілені транзакції)
- Синхронна комунікація (REST) для негайної відповіді, асинхронна (черги) для стійкості
- MonolithFirst: переходьте на мікросервіси за реальної потреби, а не заздалегідь