Лекція 7

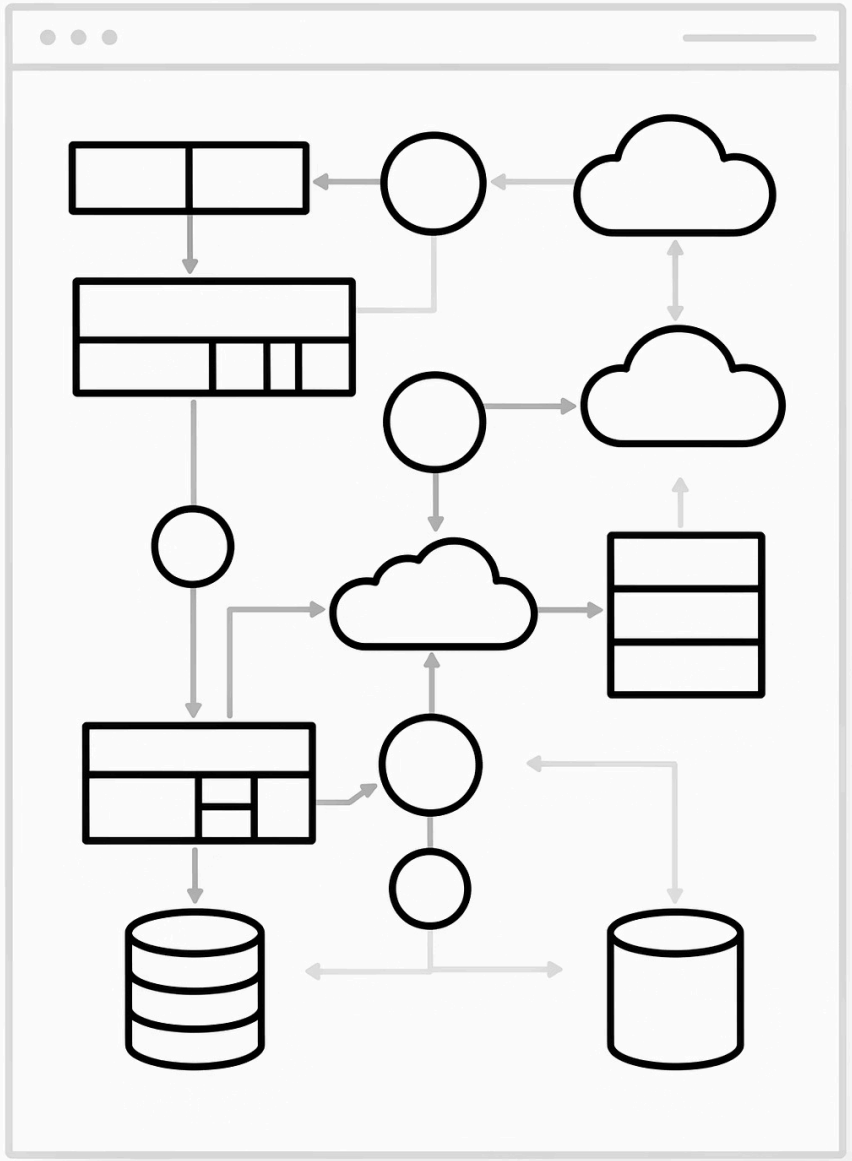
Архітектурне проєктування

Криза "білих дошок"

UML — потужний інструмент, але має проблему надмірної складності. Коли архітектори намагаються намалювати архітектуру хмарного сервісу (AWS/Azure) за допомогою UML Deployment Diagram, виходить схема, яку неможливо прочитати без лупи.

Часто команди малюють "квадратики зі стрілочками" на дошці без жодних правил. Проблема: ви намалювали квадрат "Сервер". Що це? Залізо? Java-процес? Docker-контейнер? База даних? Через місяць ніхто не пам'ятає, що означала ця схема.

C4 Model (автор Саймон Браун) — це сучасний підхід, створений для вирішення проблеми комунікації. Це не заміна UML, це спосіб навести порядок в абстракціях.



Метафора Google Maps

Найкращий спосіб зрозуміти C4 — це аналогія з картами. Коли ви відкриваєте Google Maps, ви не бачите все одразу. Ви використовуєте зум.

Level 1: Context

Вигляд з космосу. Континенти і країни. Як система вписується в IT-ландшафт компанії.

Level 2: Containers

Вигляд країни. Міста і дороги. Які сервери та бази даних у нас є.

Level 3: Components

Вигляд міста. Вулиці та будинки. З яких модулів складається серверний код.

Level 4: Code

Вигляд кімнати. Меблі. Класи та інтерфейси.

Суть C4: Ніколи не змішуйте рівні. Не малюйте SQL-таблицю на схемі, де зображені користувачі. Це як малювати стілець на карті Європи.

Чотири рівні "C"

Context (Контекст)

Для кого: Нетехнічні люди (Замовники, Менеджери)

Containers (Контейнери)

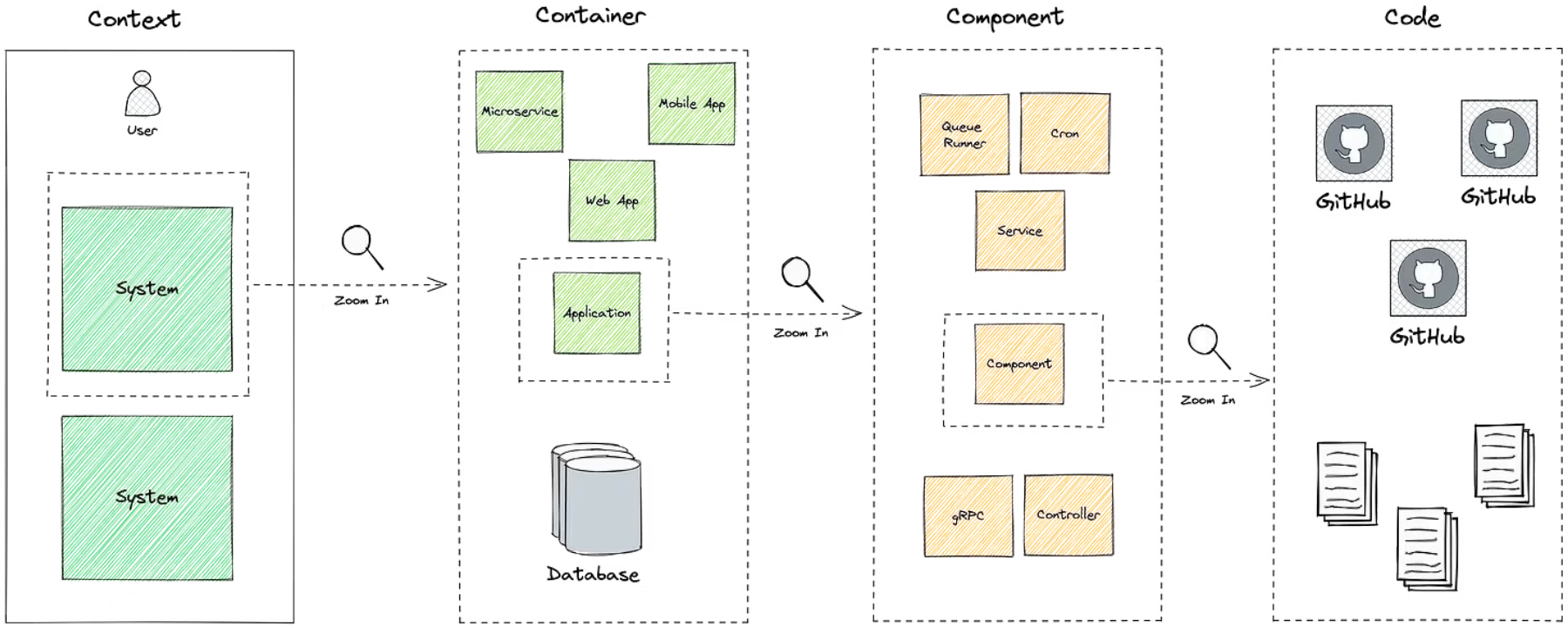
Для кого: Розробники, DevOps, Архітектори

Components (Компоненти)

Для кого: Тільки для розробників конкретного сервісу

Code (Код)

Для кого: Ні для кого



Порівняння: UML vs C4

C4 не має суворого стандарту форми стрілок (як UML), але має суворі правила текстового опису. Кожен елемент на діаграмі повинен мати:

Назву

Наприклад, "API Gateway"

Технологію

Наприклад, [Java / Spring Boot]

Опис відповідальності

Наприклад, "Маршрутизує запити до мікросервісів"

> **Золоте правило C4:** Жоден "квадратик" не може бути пустим.

Критерій	UML	C4 Model
Фокус	Детальне проєктування класів і логіки	Візуалізація архітектури та розгортання
Аудиторія	Переважно розробники	Від бізнесу до розробників (залежно від рівня)
Складність	Висока (сотні типів стрілок)	Низька (ієрархія квадратів)
Використання	"Code Design" (дизайн коду)	"System Design" (дизайн системи)

Level 1: System Context

КОНТЕКСТ СИСТЕМИ

Це діаграма найвищого рівня. Вона відповідає на питання:
"Як наша система вписується в світ?"

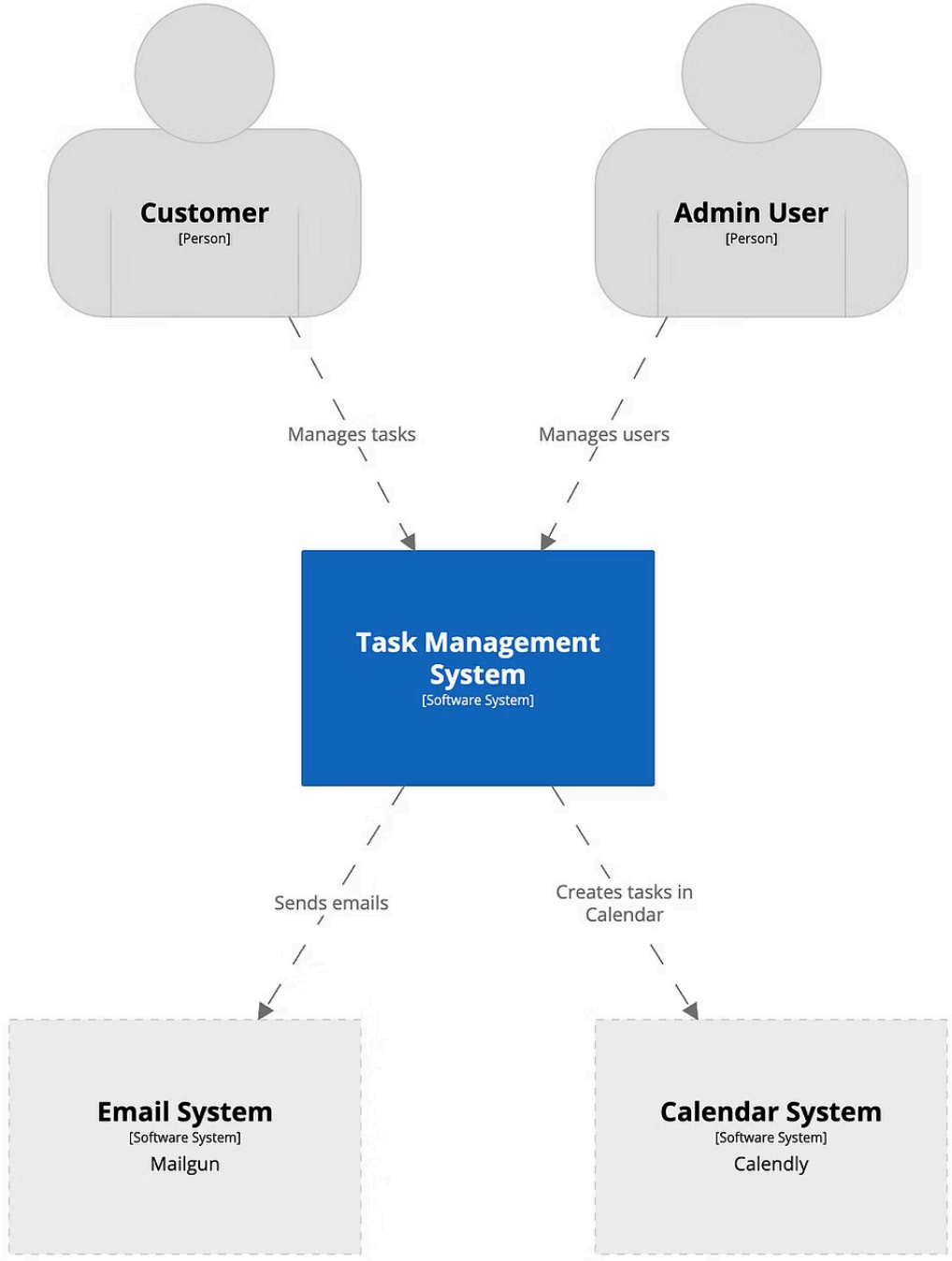
Характеристики

- **Центральний елемент:** Вся ваша система зображена як один прямокутник (Black Box)
- **Користувачі (Actors):** Студент, Викладач, Адмін
- **Зовнішні системи:** Email-сервіс (SendGrid), Платіжна система (Stripe), Google Analytics
- **Зв'язки:** Показують бізнес-взаємодію (наприклад, "Надсилає листи", "Отримує звіт")

Аудиторія

Бізнес-аналітики, Product Owner, Замовник, нові співробітники

Технічні протоколи тут писати не обов'язково.



Level 2: Containers

НАЙВАЖЛИВІША ДІАГРАМА

Single-Page Application (SPA)

React/Angular додаток, що працює в браузері

Mobile App

iOS/Android додаток

Server-side Web Application

Java/Spring, Python/Django, Node.js API

Database

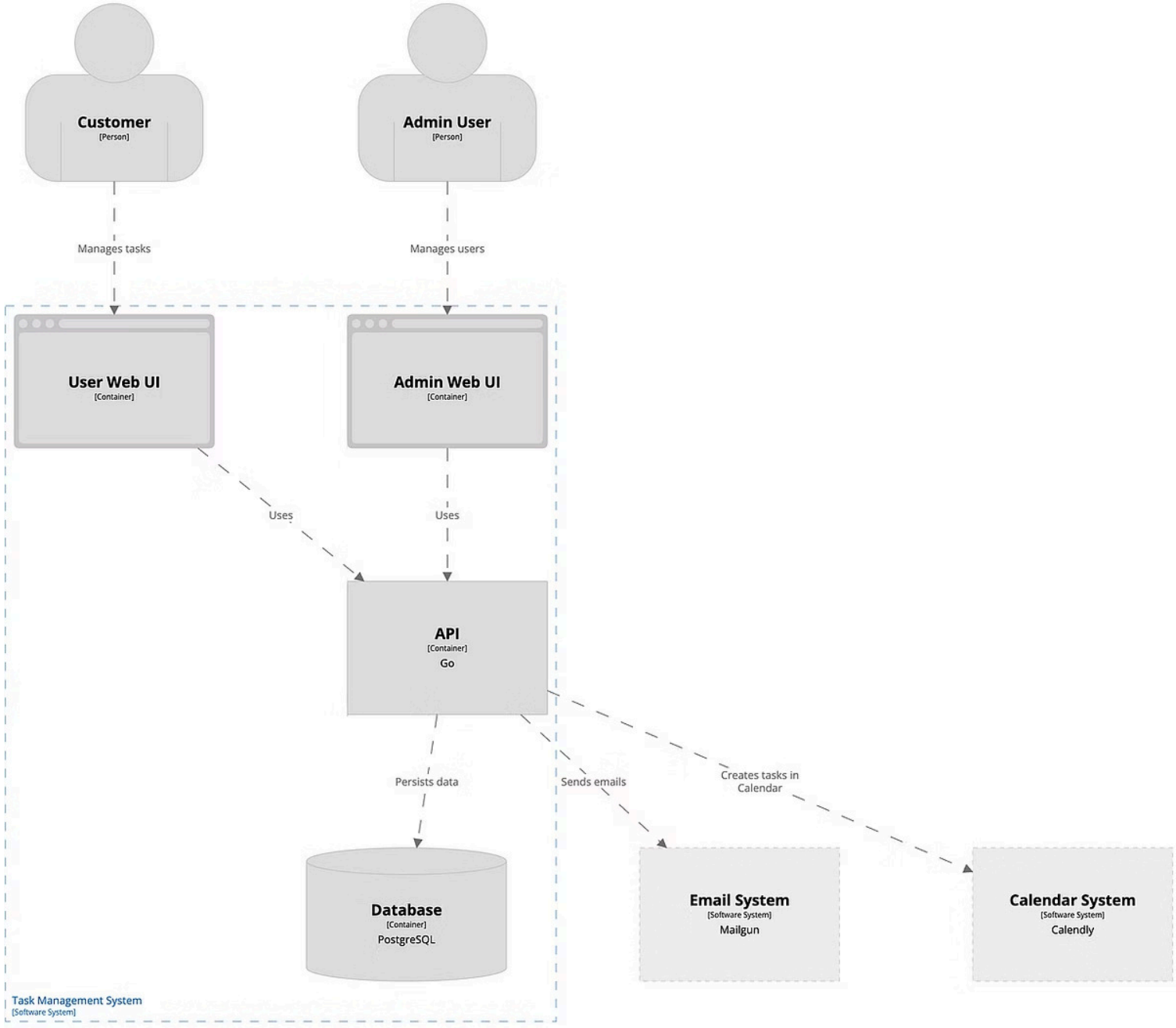
Схема бази даних (PostgreSQL, MongoDB)

File System / Blob Storage

AWS S3

Message Broker

RabbitMQ, Kafka



Level 3: Components

Ми вибираємо один контейнер (зазвичай це Backend API) і дивимось, з чого він складається всередині.

Приклад компонентів у бекенді

- Sign In Controller (дозволяє увійти)
- Password Reset Component (скидає пароль)
- Email Component (обгортка над зовнішнім API)
- Security Component

Взаємодія модулів

Показує взаємодію модулів всередині процесу

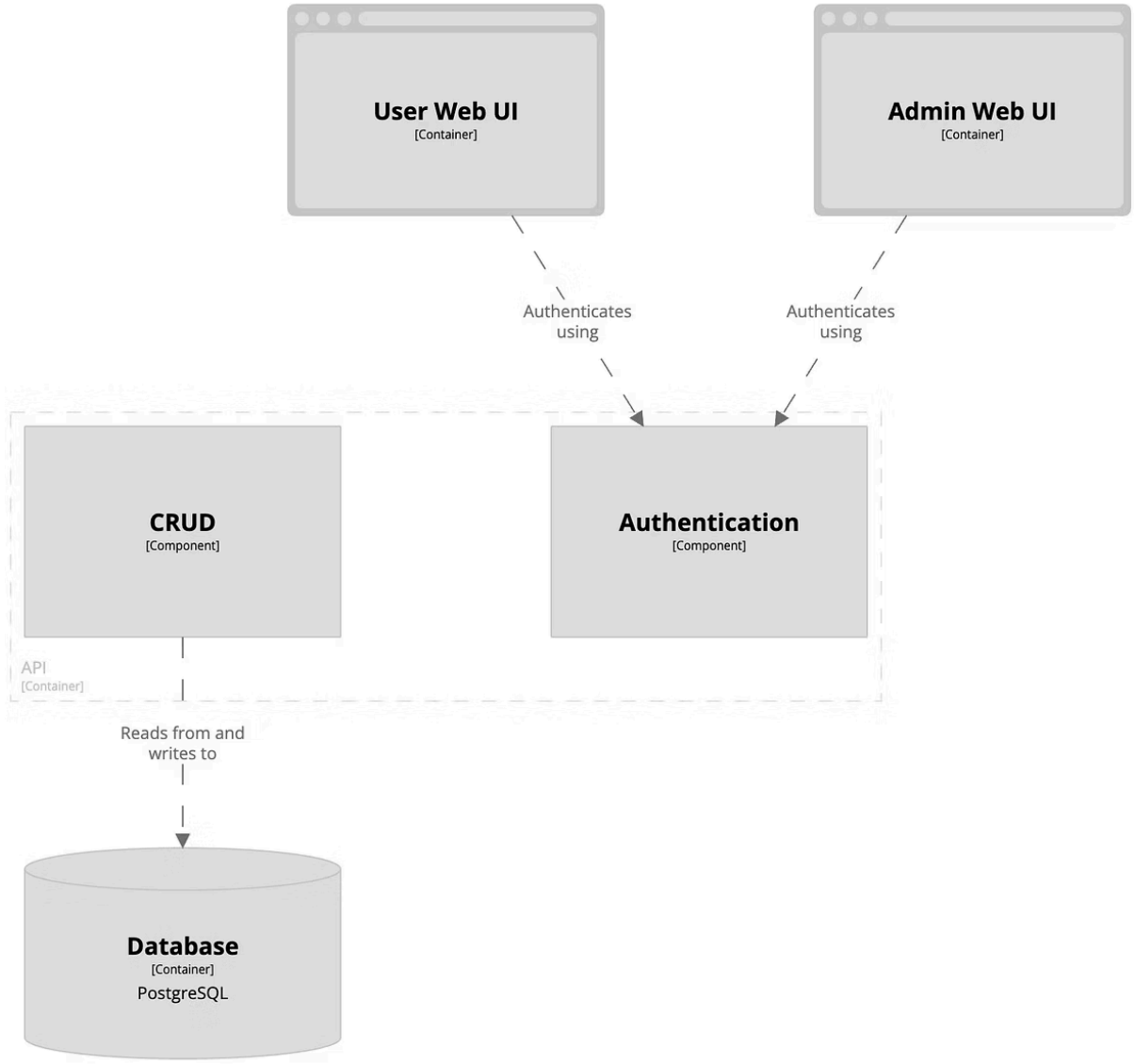
Залежності

Показує залежності (Dependency Injection)

Патерни

Тут ми бачимо реалізацію патернів (Facade, Adapter, Controller-Service-Repository)

Аудиторія: Тільки розробники, які пишуть код цього конкретного сервісу.



Level 4: Code

РІДКО ВИКОРИСТОВУЄТЬСЯ

Сучасний підхід

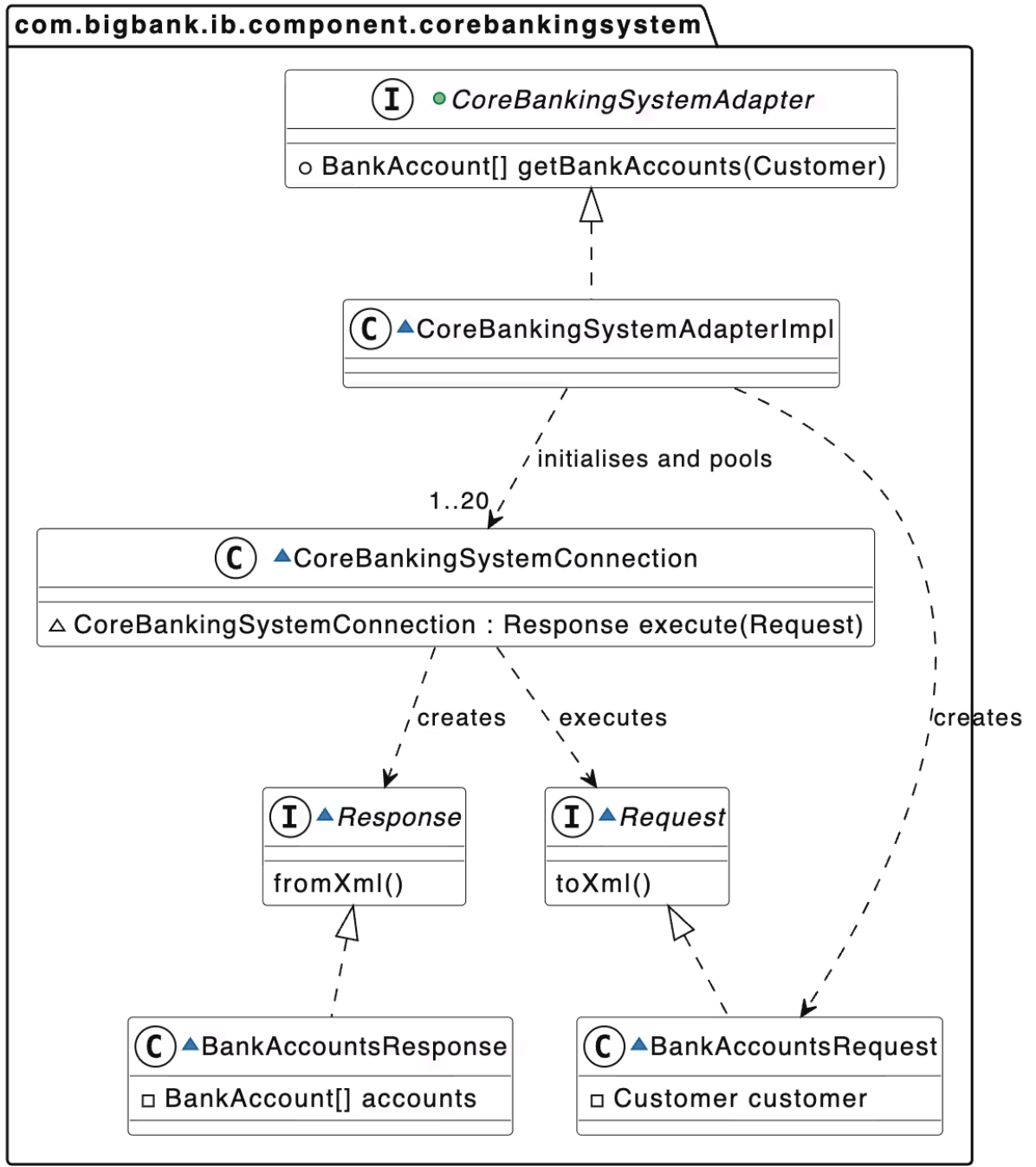
У C4 Model цей рівень майже ніколи не малюють.

Чому?

Деталізація настільки висока, що діаграма застаріває в момент натискання "Save" в IDE.

Рекомендація

Цей рівень має генеруватися автоматично з коду (JavaDoc, Doxygen) за потребою, а не малюватися архітектором.



Зведена таблиця рівнів

Рівень	Аналогія (Карта)	Що показує?	Для кого?	Технічні деталі
Context	Країна	Люди + Системи	Всі (вкл. Замовника)	Немає
Container	Місто	Додатки + БД	IT-команда	Протоколи (HTTP, TCP)
Component	Будинок	Модулі / Сервіси	Backend/Frontend Devs	Назви пакетів/класів
Code	Меблі	Класи / Інтерфейси	Ніхто (Автогенерація)	Типи даних, методи

Ця таблиця показує, як кожен рівень C4 Model відповідає різному масштабу деталізації та різній аудиторії.

ТЕХНОЛОГІЧНИЙ СТЕК

Складові технологічного стеку

Стек (Tech Stack) — це набір інструментів, які працюють разом. Зазвичай його ділять на 4 рівні:

Frontend (Клієнтська частина)

Варіанти: React, Angular, Vue, Svelte, або нативний Mobile (Swift/Kotlin)

Критерій: Інтерактивність інтерфейсу та швидкість розробки

Backend (Серверна логіка)

Варіанти: Java (Spring), C# (.NET), Node.js (Express/Nest), Python (Django/FastAPI), Go

Критерій: Продуктивність, типізація, екосистема

Database (Зберігання даних)

Варіанти: Relational (PostgreSQL, MySQL) vs NoSQL (MongoDB, Cassandra, Redis)

Критерій: Структура даних та вимоги до транзакцій (ACID)

Infrastructure (Розгортання)

Варіанти: AWS, Azure, Google Cloud, Heroku, власні сервери (On-Premise)

Критерії вибору технологій

DECISION DRIVERS

Як обрати між Python і Java? Використовуйте ці 4 фільтри:

А. Вимоги проєкту

- **Real-time:** Чат або гра → Node.js або Go
- **Enterprise / Banking:** Транзакційність → Java або C#
- **AI / Data Science:** Обробка даних → Python
- **High Load:** Мільйони запитів → C++, Rust або Go

Б. Компетенція команди

Це найважливіший фактор. У вас є 5 крутих розробників на C#? Пишіть на C#, навіть якщо Python кращий для задачі. Команда витратить місяць на вивчення Python і напише поганий код. На C# вони зроблять це за тиждень якісно.

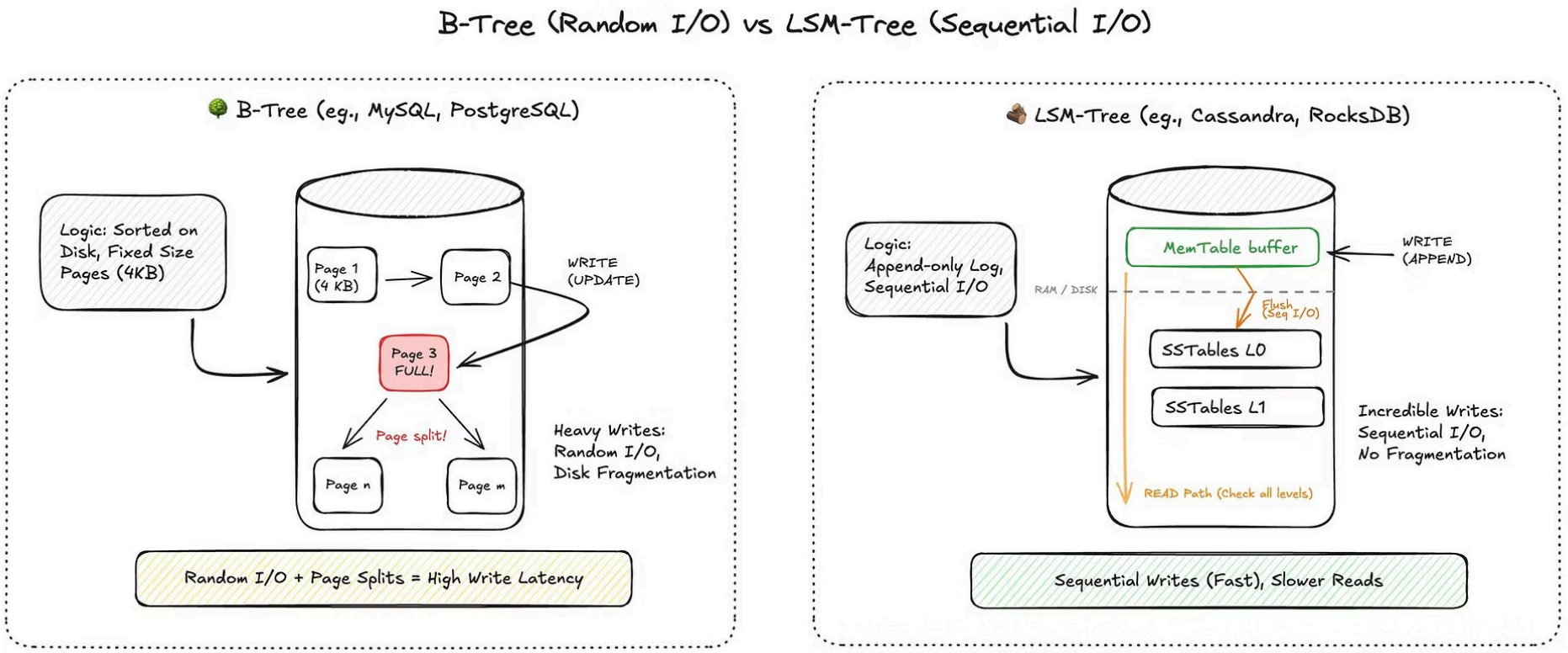
В. Time-to-Market

- **Стартап (MVP):** Треба "на вчора" → Python, JS, Ruby
- **Довгобуд (LTS):** Система на 10 років → Java, TypeScript, Go

Г. Екосистема та Спільнота

Перевірте бібліотеки. Якщо для нової модної мови X немає драйвера для вашої бази даних або бібліотеки для оплати картками — ви будете писати їх самі. Це дорого. Обирайте "нудні" технології з великою спільнотою.

SQL vs NoSQL: Вічна битва



Компроміс: Сучасні SQL бази (PostgreSQL) чудово вміють зберігати JSON (тип jsonb). Тому в 90% випадків починайте з PostgreSQL.

Триланкова архітектура

3-TIER ARCHITECTURE

Найпоширеніший патерн взаємодії в веб-розробці — це триланкова архітектура. Це класика, яку ви будете використовувати у 99% своїх проєктів.

Presentation Tier (Клієнт)

"Обличчя". Браузер (React) або Мобільний додаток. Відповідає за відображення та збір даних від користувача.

Application Tier (Сервер)

"Мізки". Java/Python/Node.js. Тут живе бізнес-логіка, валідація, розрахунки.

Data Tier (База даних)

"Пам'ять". SQL/NoSQL. Відповідає за надійне зберігання.

Золоте правило безпеки: Клієнт НІКОЛИ не спілкується з Базою Даних напряму. Чому: Якщо ви зашиєте пароль від БД у JavaScript-код фронтенду, будь-який школяр знайде його через "Inspect Element" і видалить вашу базу.

Ланка Клієнт - Сервер

API

Як браузер просить сервер надати дані? Через протокол HTTP (HyperText Transfer Protocol).

А. Стилі API (Як формулювати прохання)

REST (Representational State Transfer)

Стандарт індустрії. Використовує HTTP-методи: GET (дай), POST (створи), PUT (онови), DELETE (видали).

Приклад: GET /api/users/123 — дай юзера 123

GraphQL

Гнучкий підхід (розроблений Facebook). Клієнт просить тільки ті поля, які йому треба (економія трафіку).

Приклад: "Дай мені тільки ім'я та фото юзера 123"

WebSocket

Постійний двосторонній канал. Використовується для чатів, ігор, бірж (де треба миттєва реакція).

Б. Формат даних (Мова спілкування)

Майже завжди це **JSON** (JavaScript Object Notation). Це легкий текстовий формат. Legacy: XML (використовується в старих банківських системах SOAP).

Stateless vs Stateful

ПАМ'ЯТЬ СЕРВЕРА

Як сервер пам'ятає, що ви залогінилися?

Stateful (Сесії)

Сервер тримає в оперативній пам'яті список залогінених користувачів.

Проблема: Якщо сервер перезавантажиться, всі "вилетять". Важко масштабувати (якщо у вас 10 серверів, вони не знають про сесії одне одного).

Stateless (Токени / JWT)

Сучасний стандарт. При логіні сервер видає клієнту зашифрований "бейдж" (JWT Token).

Клієнт прикріплює цей токен до кожного запиту. Сервер не пам'ятає нічого. Він просто перевіряє підпис токена.

Перевага: Це дозволяє мати хоч 100 серверів.

Висновки:

Ми розглянули архітектуру від "погляду з космосу" (C4 Context) до "дротів під капотом" (HTTP/SQL).

Основні моменти:

Починайте з Context

Далі Containers і вже потім Components.

Діаграма Container

Найбільш робоча діаграма, що висітиме у вас в офісі на стіні.

SQL vs NoSQL

SQL: Чітка схема, транзакції, складні JOIN, вертикальне масштабування.
NoSQL: Змінна схема, висока пропускна здатність, горизонтальне масштабування.