Лекція: Моделювання архітектури ПЗ за допомогою C4-моделі

Мета: Зрозуміти принципи, рівні деталізації та практичне застосування нотації C4 для візуалізації та документування архітектури програмного забезпечення.

---

1. Вступ: Проблема, яку вирішує C4

Часто технічна документація або:

- Відсутня взагалі ("діра в пам'яті").
- Застаріла і не відповідає реальності.
- Надто детальна і складна для розуміння нетехнічними спеціалістами (менеджерами, замовниками).
- Надто абстрактна і не дає відповіді на конкретні питання розробників.

C4-модель — це **набір конвенцій** для створення карт архітектури вашого ПЗ, подібно до того, як Google Maps показує різні рівні деталізації: від країни до окремої вулиці.

Основні принципи C4:

1. **Контекст:** Система в оточенні.
2. **Контейнери:** Як система складається з основних "шматків".
3. **Компоненти:** Як влаштований кожен "шматок".
4. **Код:** Детальна реалізація компонентів (за допомогою UML діаграм класів тощо).

Назва "C4" походить від перших літер перших чотирьох рівнів: Context, Containers, Components, Code.

---

2. Чотири рівні C4-моделі

Рівень 1: Діаграма контексту (System Context Diagram)

- **Аудиторія:** Будь-хто, від замовника до нетехнічного керівника та нового розробника.
- **Відповідає на питання:** Що робить система? З ким/чим взаємодіє?
- **Що зображає:**

    - **Один головний** Прямокутник — ваша система, що розглядається.
    - **Люди-користувачі (Actors)** та **зовнішні системи**, з якими взаємодіє ваша система.
    - **Стрілки**, що показують потік даних і напрямок взаємодії.
- **Не показує:** Технології, протоколи, деталі реалізації.

Рівень 2: Діаграма контейнерів (Container Diagram)

- **Аудиторія:** Архітектори, розробники, DevOps-інженери.
- **Відповідає на питання:** Як система влаштована "під капотом"? Які основні технологічні блоки?
- **Що зображає:**
    - **Контейнери.** Контейнер — це окремий процес, що виконує код або зберігає дані (наприклад, веб-додаток, мобільний додаток, серверна API, база даних, файлова система, мікросервіс). Кожен контейнер живе окремо і може бути розгорнутий самостійно.
    - Взаємодії між цими контейнерами.
    - Технологічний стек для кожного контейнера (наприклад, "Angular", "Spring Boot", "PostgreSQL").
- **Не показує:** Деталі внутрішньої структури кожного контейнера.

Рівень 3: Діаграма компонентів (Component Diagram)

- **Аудиторія:** Архітектори та розробники.
- **Відповідає на питання:** Як влаштований один конкретний контейнер? Які логічні блоки (компоненти) всередині нього відповідають за конкретні функції?
- **Що зображає:**
    - **Компоненти** всередині одного контейнера. Компонент — це група пов'язаних функцій (модуль, клас, група класів), наприклад, UserController, OrderService, EmailRepository.
    - Взаємодії між компонентами та зовнішніми сутностями (іншими контейнерами або системами).
- **Не показує:** Детальну реалізацію в коді.

Рівень 4: Діаграма коду (Code Diagram)

- **Аудиторія:** Розробники.
- **Відповідає на питання:** Як реалізований конкретний компонент?
- **Що зображає:** Детальні діаграми, створені за допомогою інструментів (наприклад, UML діаграми класів, ER-діаграми для бази даних). Цей рівень рідко малюється вручну, його часто генерують автоматично з коду (за допомогою IDE або спеціальних інструментів).

---

3. Практичний приклад: Розробка системи "Інтернет-Магазин"

Розглянемо створення архітектурної документації для простого інтернет-магазину.

Крок 1: Діаграма Контексту (Рівень 1)

Система: Інтернет-Магазин "TechStore"

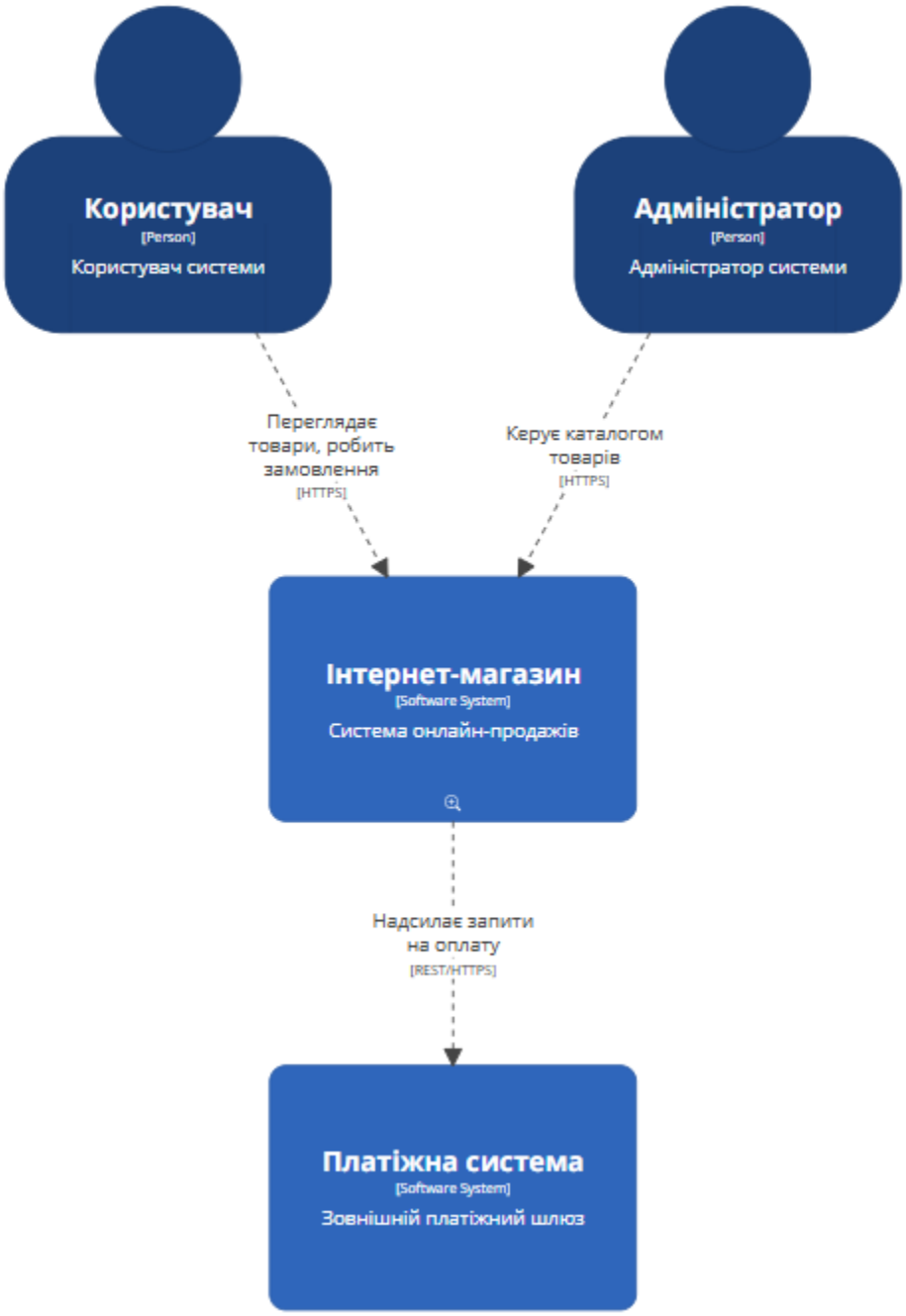
Користувачі:

- **Клієнт:** Переглядає товари, робить замовлення.
- **Менеджер з продажу:** Оновлює каталог товарів, переглядає замовлення.

Зовнішні системи:

- **Платіжний шлюз (Payment Gateway):** Для обробки платежів (наприклад, Stripe).
- **Служба доставки (Email Service):** Для відправки листів-підтверджень (наприклад, SendGrid).
- **Система складського обліку (Warehouse System):** Для перевірки наявності товару.

Діаграма Контексту:



(Стрілки підписані: "Переглядає каталог / Робить замовлення", "Сповіщає про статус", "Оновлює каталог", "Надсилає платіж", "Надсилає email", "Перевіряє наявність")

```
@startuml TechStore Context Diagram
!includeurl https://raw.githubusercontent.com/plantuml-stdlib/C4-PlantUML/master/C4.puml
!includeurl https://raw.githubusercontent.com/plantuml-stdlib/C4-PlantUML/master/C4_Context.puml

title System Context Diagram for "TechStore" Internet Shop

Person(customer, "Клієнт", "Користувач, який переглядає товари та робить замовлення")
Person(manager, "Менеджер з продажу", "Працівник, який оновлює каталог та переглядає замовлення")

System(techstore, "Інтернет-Магазин \"TechStore\"", "Онлайн-система для продажу товарів")

System_Ext(payment_gateway, "Платіжний шлюз", "Stripe API")
System_Ext(email_service, "Служба доставки email", "SendGrid API")
System_Ext(warehouse_system, "Система складського обліку", "Внутрішня система управління складом")

' Взаємодії
Rel(customer, techstore, "Переглядає каталог, робить замовлення")
Rel(techstore, customer, "Надсилає підтвердження замовлення")

Rel(manager, techstore, "Оновлює каталог товарів, переглядає замовлення")

Rel(techstore, payment_gateway, "Надсилає запит на оплату", "HTTPS")
Rel(payment_gateway, techstore, "Повертає статус платежу", "HTTPS")

Rel(techstore, email_service, "Надсилає email-сповіщення", "SMTP/API")

Rel(techstore, warehouse_system, "Перевіряє наявність товару", "REST API")
Rel(warehouse_system, techstore, "Повертає інформацію про наявність", "REST API")

@enduml
```

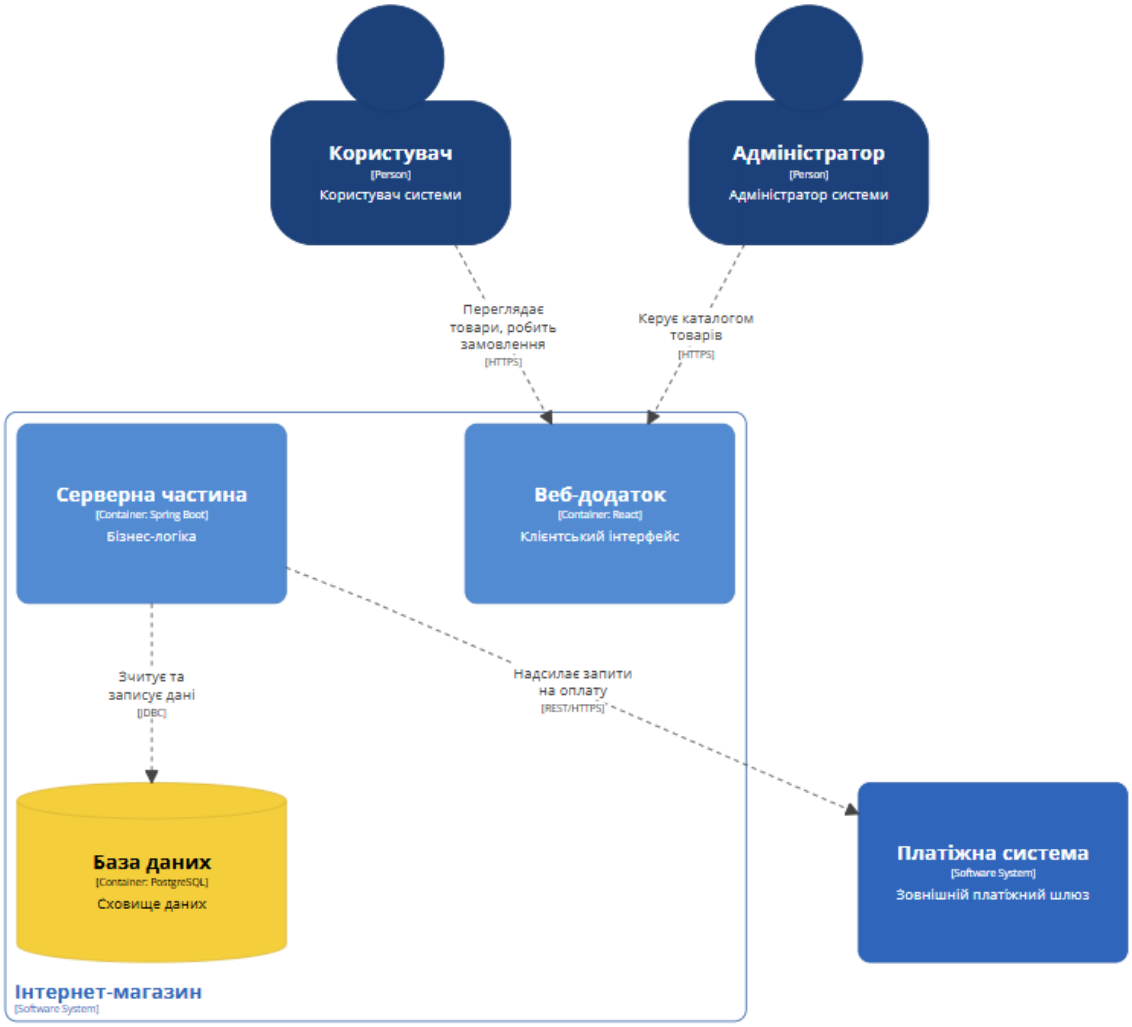
Висновок: Ми визначили межі системи та всі ключові зовнішні точки контакту.

Крок 2: Діаграма Контейнерів (Рівень 2)

Розкриваємо "TechStore". Він складається з таких контейнерів:

1. **Веб-додаток (Single-Page Application):** Клієнтська частина, написана на React. Взаємодіє з сервером через API.
2. **Серверна частина (Backend API):** Основний бізнес-логік, написаний на Spring Boot. Обробляє запити від веб-додатку.
3. **База даних (Database):** Зберігає дані про товари, замовлення, клієнтів. Використовує PostgreSQL.
4. **Мобільний додаток (Mobile App):** Додаток для iOS/Android (наприклад, на React Native), який також використовує Backend API.

Діаграма Контейнерів:



(Стрілки підписані відповідними HTTP-запитами та діями)

@startuml TechStore Container Diagram
!includeurl https://raw.githubusercontent.com/plantuml-stdlib/C4-PlantUML/master/C4.puml
!includeurl https://raw.githubusercontent.com/plantuml-stdlib/C4-PlantUML/master/C4_Container.puml

title Container Diagram for "TechStore" Internet Shop

Person(customer, "Клієнт", "Користувач інтернет-магазину")
Person(manager, "Менеджер з продажу", "Працівник магазину")

```
System_Boundary(techstore_boundary, "Інтернет-Магазин \"TechStore\"") {
  Container(spa, "Веб-додаток", "React", "Single-Page Application, клієнтський інтерфейс")
  Container(mobile_app, "Мобільний додаток", "React Native", "Мобільний додаток для iOS/Android")
  Container(backend, "Серверна частина", "Spring Boot", "REST API, бізнес-логіка")
  ContainerDb(database, "База даних", "PostgreSQL", "Зберігання даних про товари, замовлення, користувачів")
}

System_Ext(payment_gateway, "Платіжний шлюз", "Stripe API")
System_Ext(email_service, "Служба доставки email", "SendGrid API")
System_Ext(warehouse_system, "Система складського обліку", "REST API")

' Взаємодії між контейнерами
Rel(customer, spa, "Використовує", "HTTPS")
Rel(customer, mobile_app, "Використовує", "HTTPS")

Rel(spa, backend, "Викликає API", "REST/HTTPS")
Rel(mobile_app, backend, "Викликає API", "REST/HTTPS")

Rel(manager, backend, "Використовує адмін-панель", "HTTPS")

Rel(backend, database, "Зчитує/записує дані", "JDBC")
Rel(backend, payment_gateway, "Ініціює платежі", "REST/HTTPS")
Rel(backend, email_service, "Надсилає сповіщення", "SMTP/API")
Rel(backend, warehouse_system, "Перевіряє наявність", "REST/HTTPS")

@enduml
```

Висновок: Тепер ми бачимо технологічний стек і основні архітектурні рішення. Зрозуміло, що бізнес-логіка централізована в Backend API.

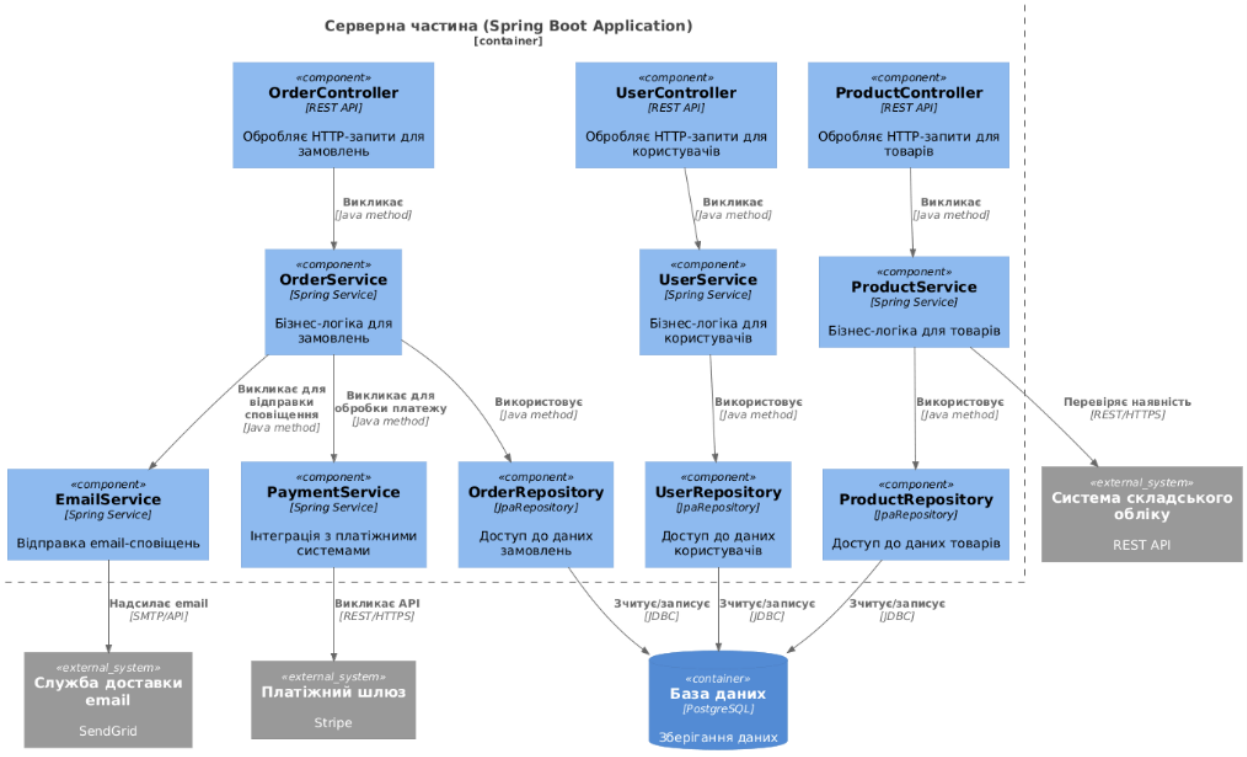
Крок 3: Діаграма Компонентів (Рівень 3)

Деталізуємо найважливіший контейнер — **Серверна частина (Spring Boot)**. Він містить такі компоненти:

- UserController**:** Обробляє HTTP-запити, пов'язані з користувачами (реєстрація, логін).
- ProductController**:** Обробляє запити на отримання каталогу товарів.
- OrderController**:** Обробляє створення та перегляд замовлень.

- OrderService**:** Основний клас бізнес-логіки для роботи з замовленнями (створення, перевірка, оновлення статусу).
- PaymentService**:** Відповідає за взаємодію з платіжним шлюзом.
- EmailService**:** Відповідає за відправку email-сповіщень.
- UserRepository **/** ProductRepository **/** OrderRepository**:** Компоненти, що відповідають за звернення до бази даних.

Діаграма Компонентів (для Backend API):



Висновок: Ця діаграма дає розробнику чітке уявлення про структуру коду серверної частини, ключові залежності між компонентами та їх відповідальність.

@startuml TechStore Component Diagram
!includeurl https://raw.githubusercontent.com/plantuml-stdlib/C4-PlantUML/master/C4.puml
!includeurl https://raw.githubusercontent.com/plantuml-stdlib/C4-PlantUML/master/C4_Component.puml

title Component Diagram for Backend API

Container_Boundary(backend, "Серверна частина (Spring Boot Application)") {

```
  Component(user_controller, "UserController", "REST API", "Обробляє HTTP-запити для користувачів")
  Component(product_controller, "ProductController", "REST API", "Обробляє HTTP-запити для товарів")
  Component(order_controller, "OrderController", "REST API", "Обробляє HTTP-запити для замовлень")

  Component(user_service, "UserService", "Spring Service", "Бізнес-логіка для користувачів")
  Component(product_service, "ProductService", "Spring Service", "Бізнес-логіка для товарів")
  Component(order_service, "OrderService", "Spring Service", "Бізнес-логіка для замовлень")

  Component(payment_service, "PaymentService", "Spring Service", "Інтеграція з платіжними системами")
  Component(email_service, "EmailService", "Spring Service", "Відправка email-сповіщень")

  Component(user_repository, "UserRepository", "JpaRepository", "Доступ до даних користувачів")
  Component(product_repository, "ProductRepository", "JpaRepository", "Доступ до даних товарів")
  Component(order_repository, "OrderRepository", "JpaRepository", "Доступ до даних замовлень")
}

ContainerDb(database, "База даних", "PostgreSQL", "Зберігання даних")

System_Ext(payment_gateway, "Платіжний шлюз", "Stripe")
System_Ext(external_email_service, "Служба доставки email", "SendGrid")
System_Ext(warehouse_system, "Система складського обліку", "REST API")

' Взаємодії між компонентами
Rel(user_controller, user_service, "Викликає", "Java method")
Rel(product_controller, product_service, "Викликає", "Java method")
Rel(order_controller, order_service, "Викликає", "Java method")

Rel(user_service, user_repository, "Використовує", "Java method")
Rel(product_service, product_repository, "Використовує", "Java method")
Rel(order_service, order_repository, "Використовує", "Java method")

Rel(order_service, payment_service, "Викликає для обробки платежу", "Java method")
Rel(order_service, email_service, "Викликає для відправки сповіщення", "Java method")
```

```
Rel(user_repository, database, "Зчитує/записує", "JDBC")
Rel(product_repository, database, "Зчитує/записує", "JDBC")
Rel(order_repository, database, "Зчитує/записує", "JDBC")

Rel(payment_service, payment_gateway, "Викликає API", "REST/HTTPS")
Rel(email_service, external_email_service, "Надсилає email", "SMTP/API")
Rel(product_service, warehouse_system, "Перевіряє наявність", "REST/HTTPS")

@enduml
```

---

4. Інструменти для створення C4-діаграм

- **Structurizr:** Офіційний інструмент (як SaaS, так і локальна версія), що підтримує текстове описання діаграм (DSL).
- **Draw.io / Diagrams.net:** Безкоштовний і популярний інструмент з готовими бібліотеками шаблонів C4.
- **PlantUML:** Текстовий інструмент для створення діаграм, має підтримку C4 через додаткові бібліотеки.
- **Miro / Lucidchart:** Онлайн-дошки з можливістю створення C4-діаграм.

5. Висновки

- **C4-модель — це мова спілкування.** Вона дозволяє ефективно комунікувати про архітектуру між різними групами стейкхолдерів.
- **"Карти, або картини"**. C4 створює саме "карти" архітектури, які можна "збільшувати" та "зменшувати", а не розрізнені художні картини.
- **Простота та стандартизація.** Використання стандартного набору абстракцій (Система, Контейнер, Компонент) робить діаграми зрозумілими та послідовними.
- **Починайте з контексту.** Це найважливіший крок, який визначає всю подальшу структуру.

C4-модель не вимагає складних інструментів — почати можна з простого блокнота або маркера на дошці. Головне — це філософія поступового розкриття деталей.