Embracing Clean Architecture

in Spring Boot with Kotlin

Łukasz Pięta

Senior Software Engineer @ OLX

Kotlin passionate

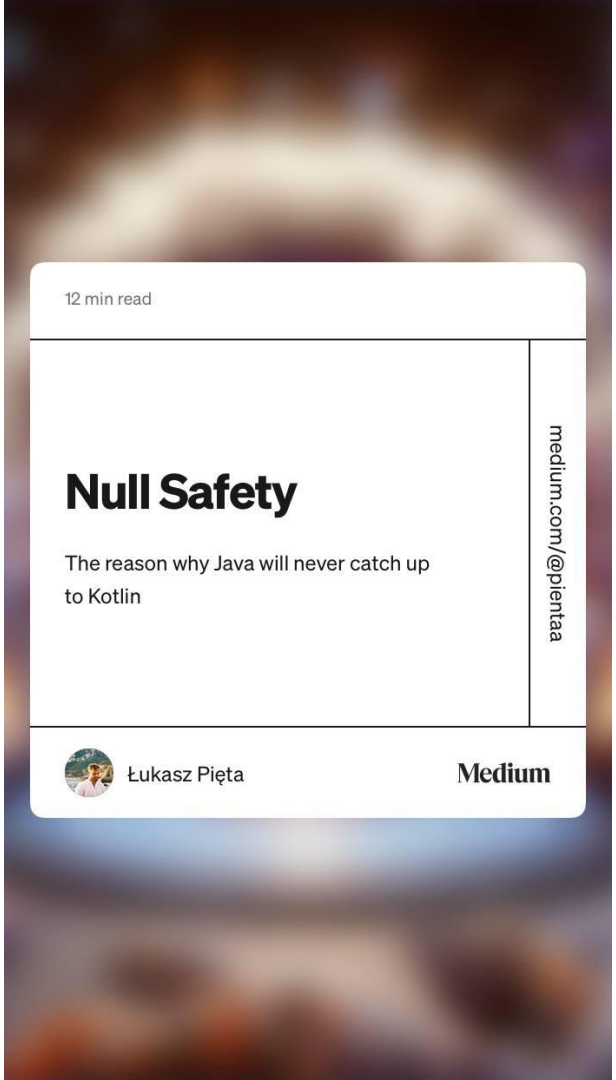
DDD enthusiast

GitHub: @pientaa

Medium: @pientaa

LinkedIn: @pientaa

X (twitter): @_pientaa


12 min read
Null Safety
The reason why Java will never catch up to Kotlin
medium.com/@pientaa
Łukasz Pięta
Medium

Software architecture

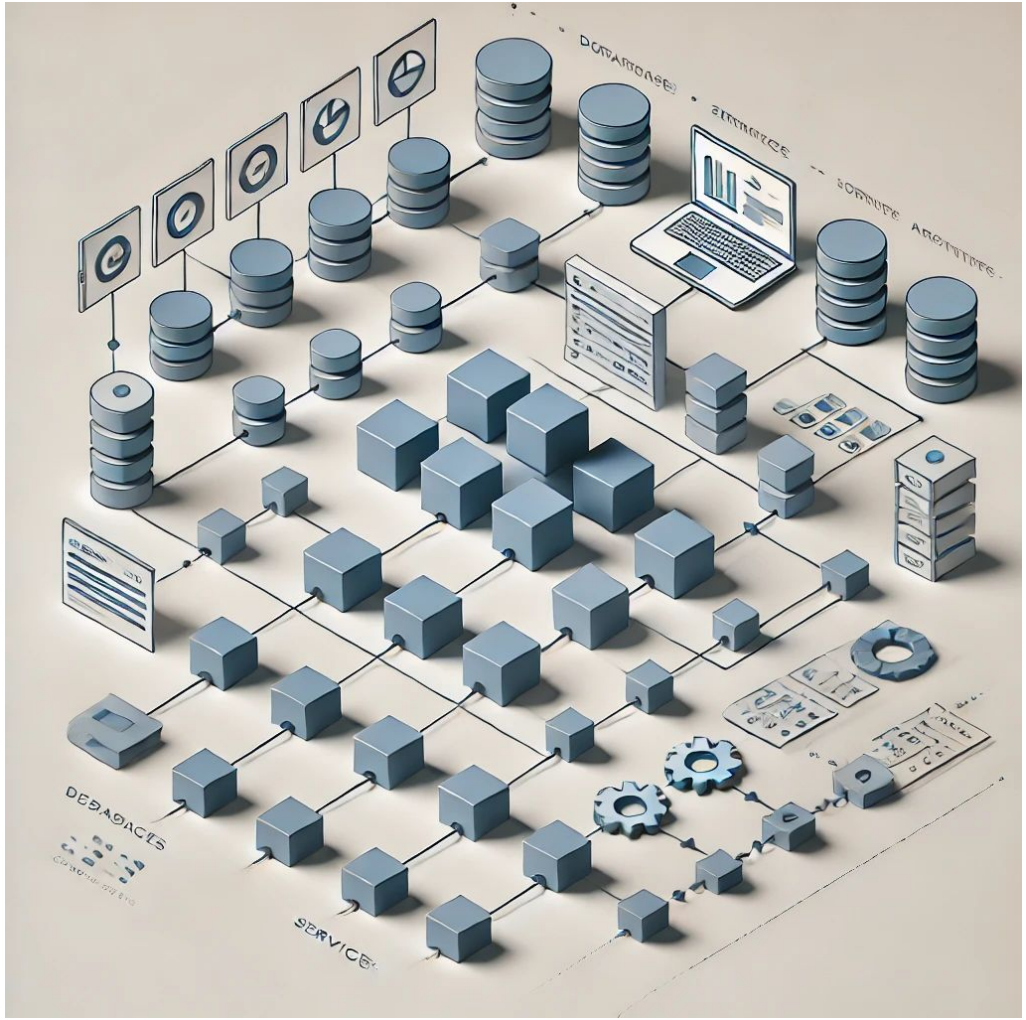
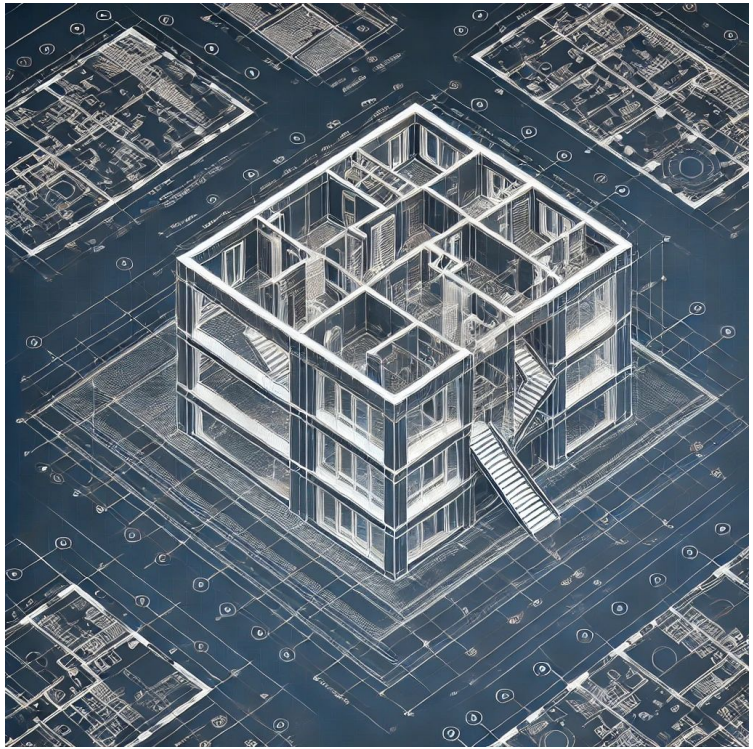


Image generated by AI using OpenAI's DALL-E

“Architecture” metaphor



Images generated by AI using OpenAI's DALL-E

“The set of **significant decisions** about the **organization of a software system**, including the choice of structural elements and their interfaces.”

~ Martin Fowler

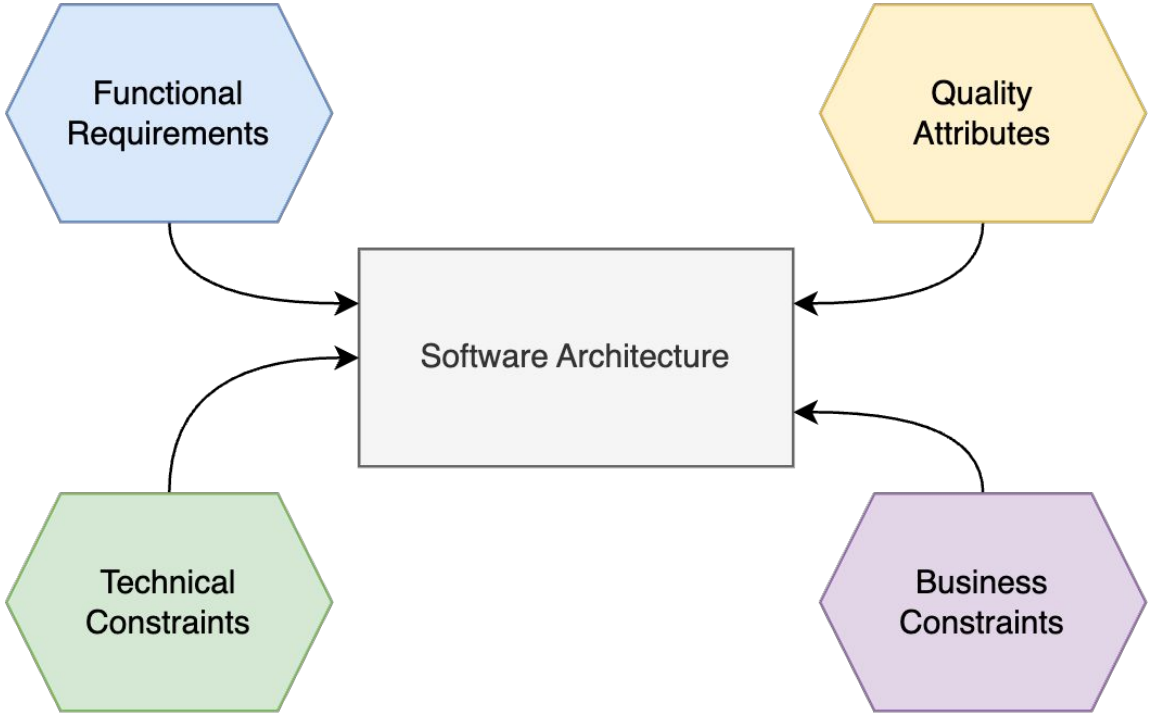
“Architecture is about **the important stuff**. Whatever that is.”

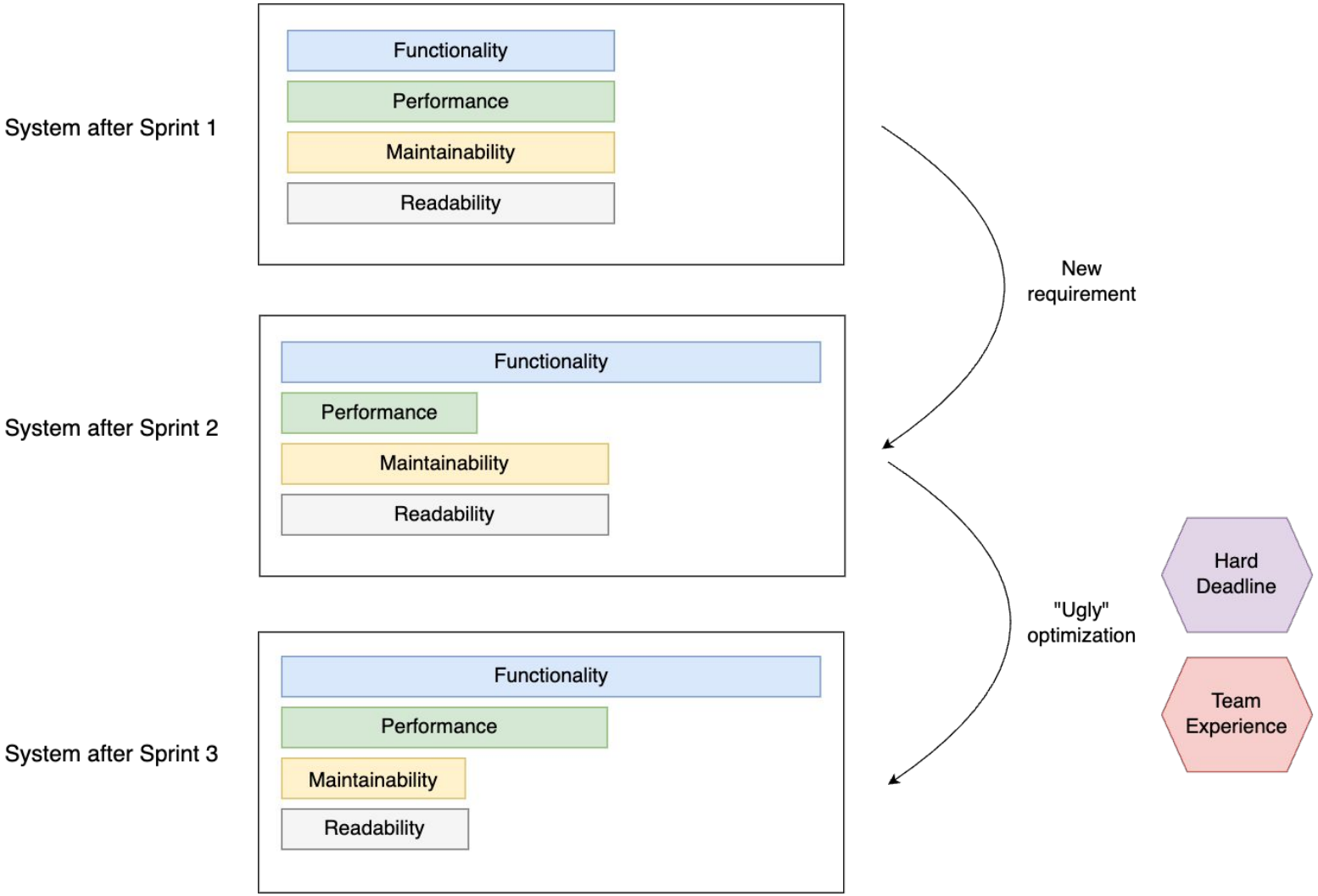
~ Ralph Johnson

Architectural Drivers

"Set of common things that really drive, influence and shape the resulting software architecture"

~ Simon Brown



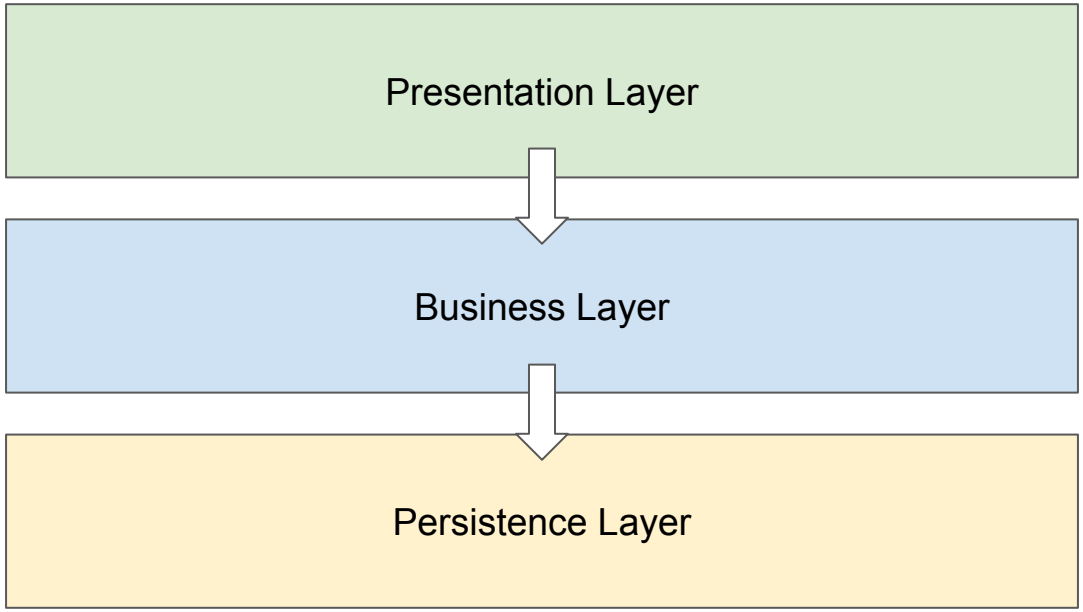

System after Sprint 1
Functionality
Performance
Maintainability
Readability
New requirement
System after Sprint 2
Functionality
Performance
Maintainability
Readability
"Ugly" optimization
Hard Deadline
Team Experience
System after Sprint 3
Functionality
Performance
Maintainability
Readability


TRADE-OFFS
LIGHTYEAR
TRADE-OFFS EVERYWHERE
imgflip.com

Software Architect Toolbox

Image generated by AI using OpenAI's DALL-E

Layered architecture



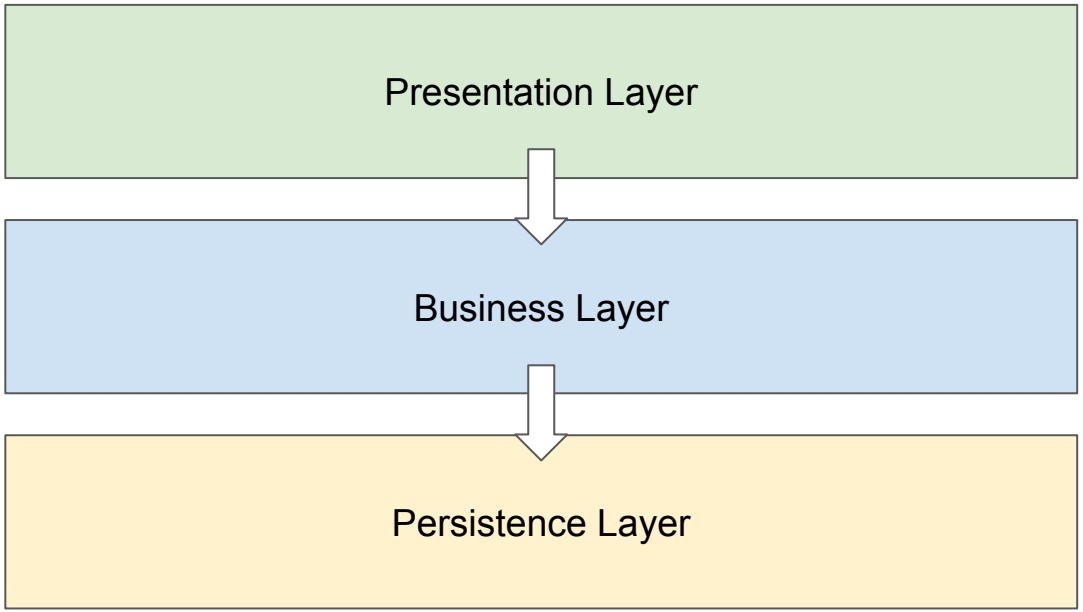
Layered architecture

Pros:

- Just simple

Cons:

- Tight coupling
- Maintainability rapidly decreasing with increasing complexity



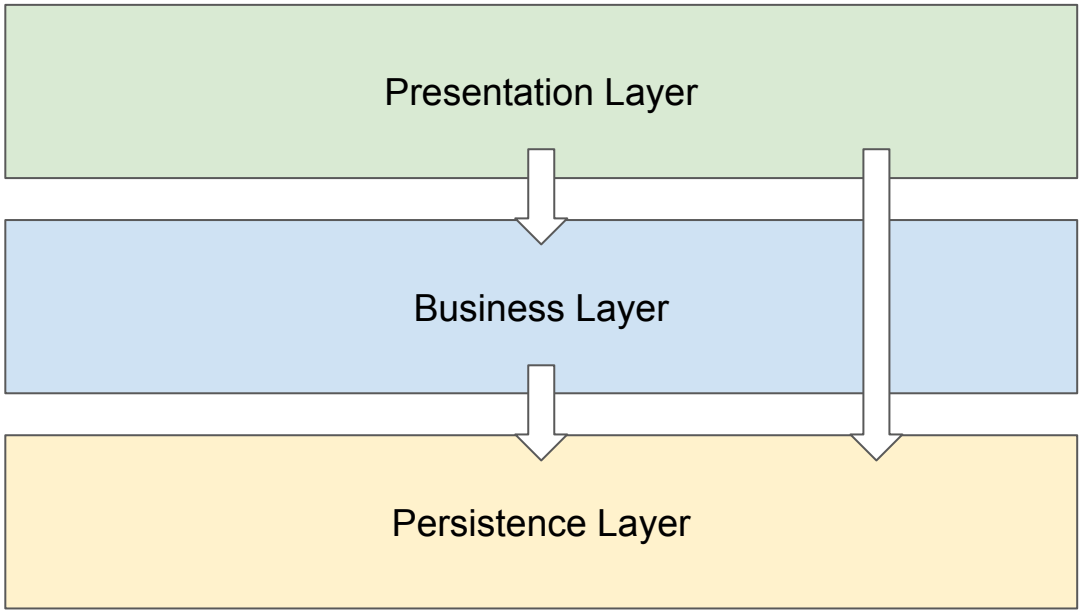
Layered architecture

Pros:

- Just simple

Cons:

- Tight coupling
- Maintainability rapidly decreasing with increasing complexity



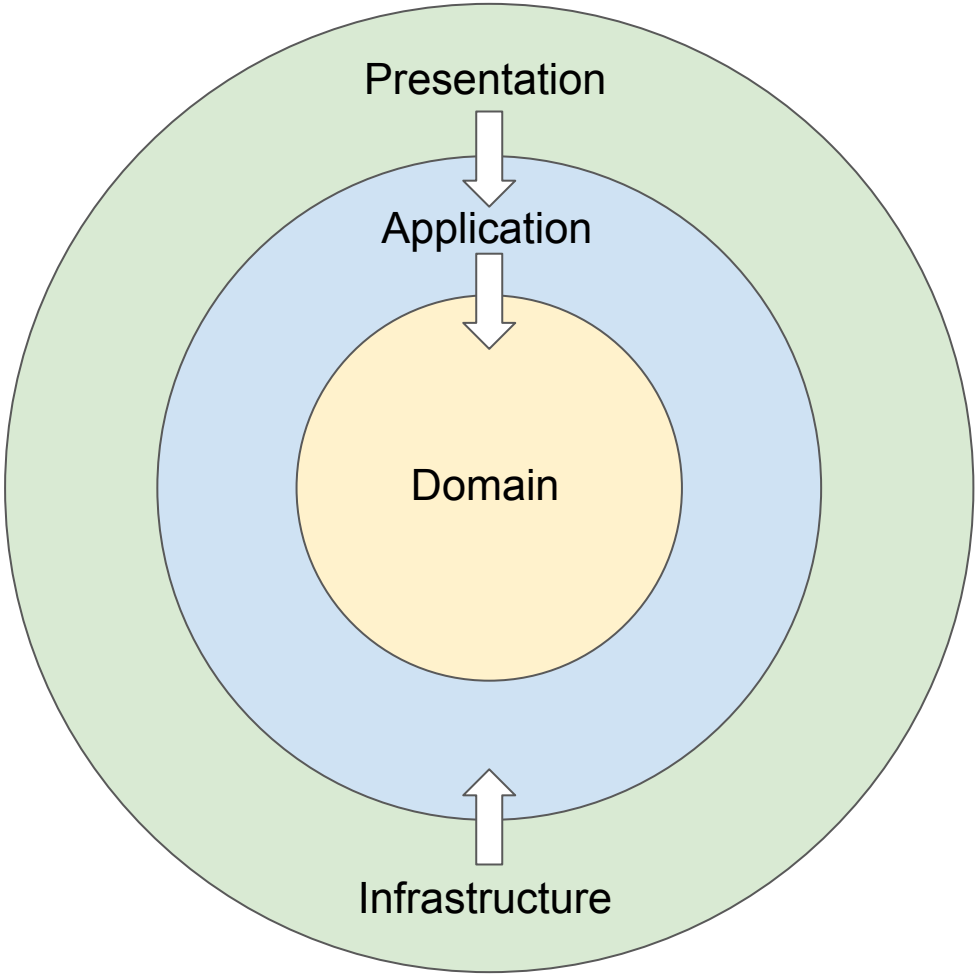
Onion architecture

Pros:

- Testability
- Separation of Concerns
- Flexibility

Cons:

- Initial Complexity
- Learning Curve
- Increased Boilerplate Code



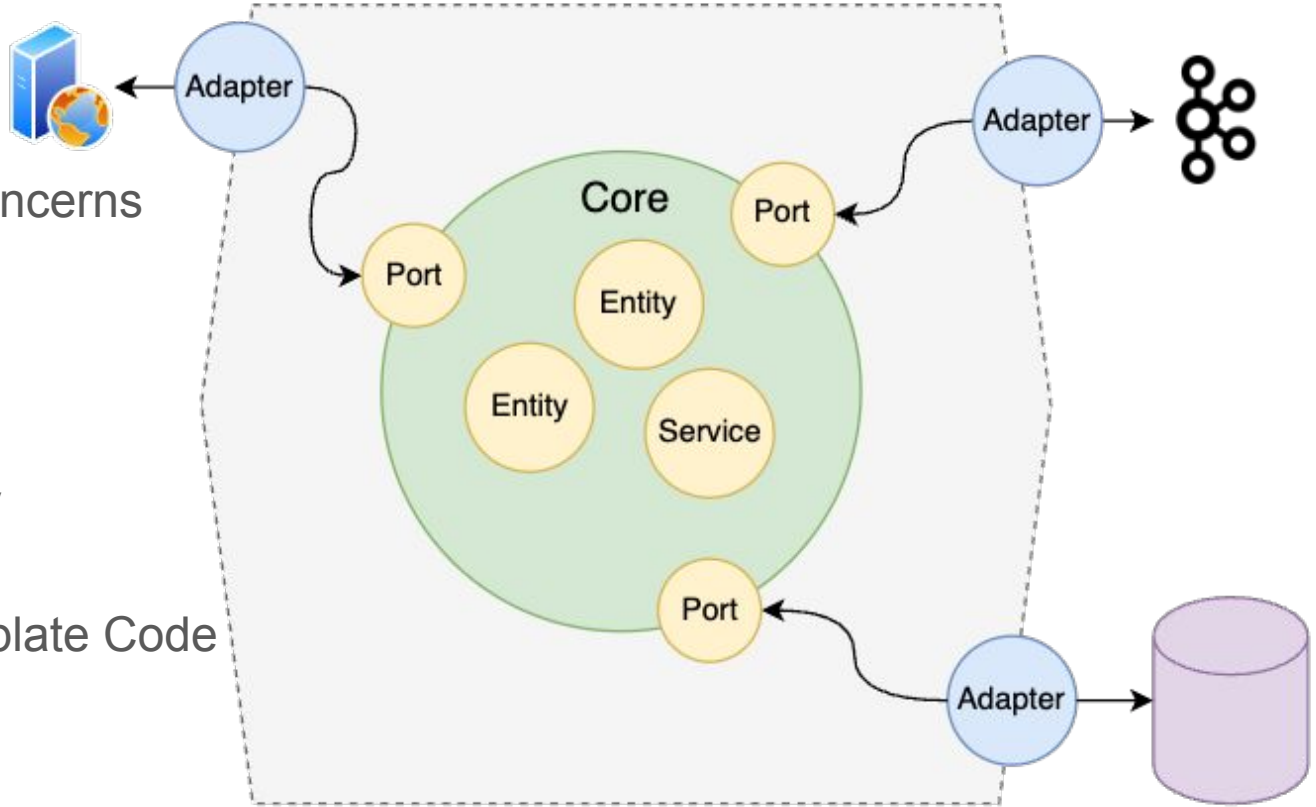
Hexagonal architecture

Pros:

- Testability
- Separation of Concerns
- Flexibility

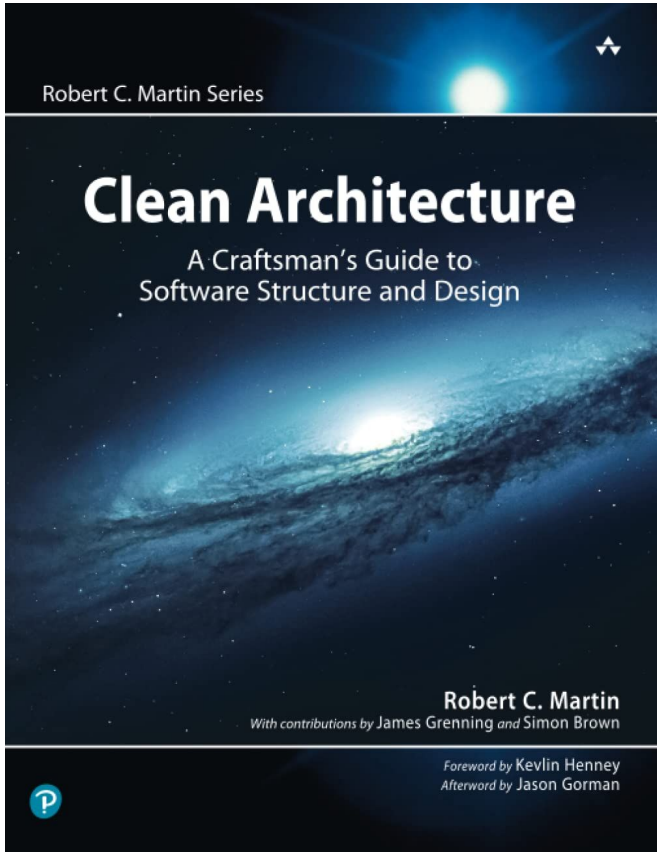
Cons:

- Initial Complexity
- Learning Curve
- Increased Boilerplate Code



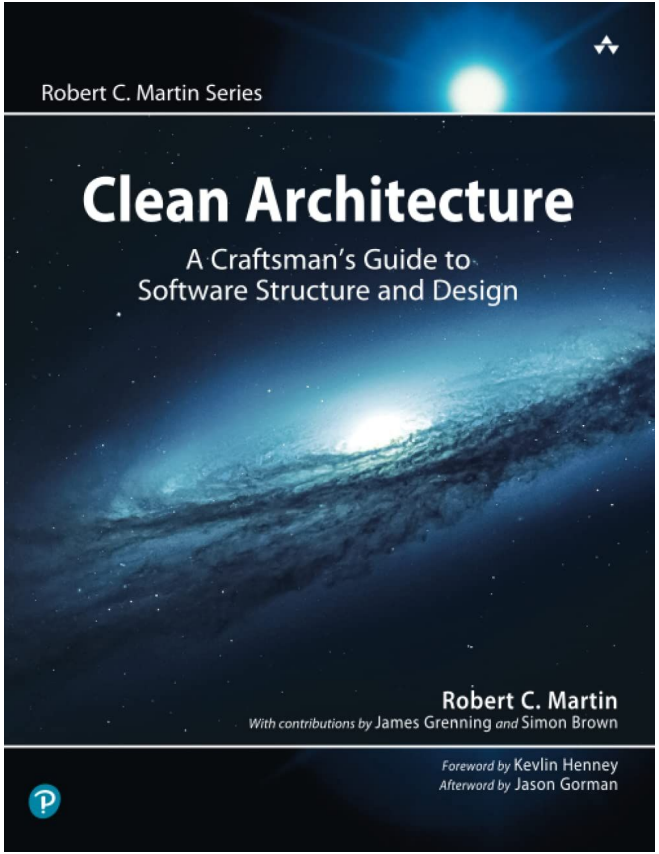
What is “Clean architecture” then?

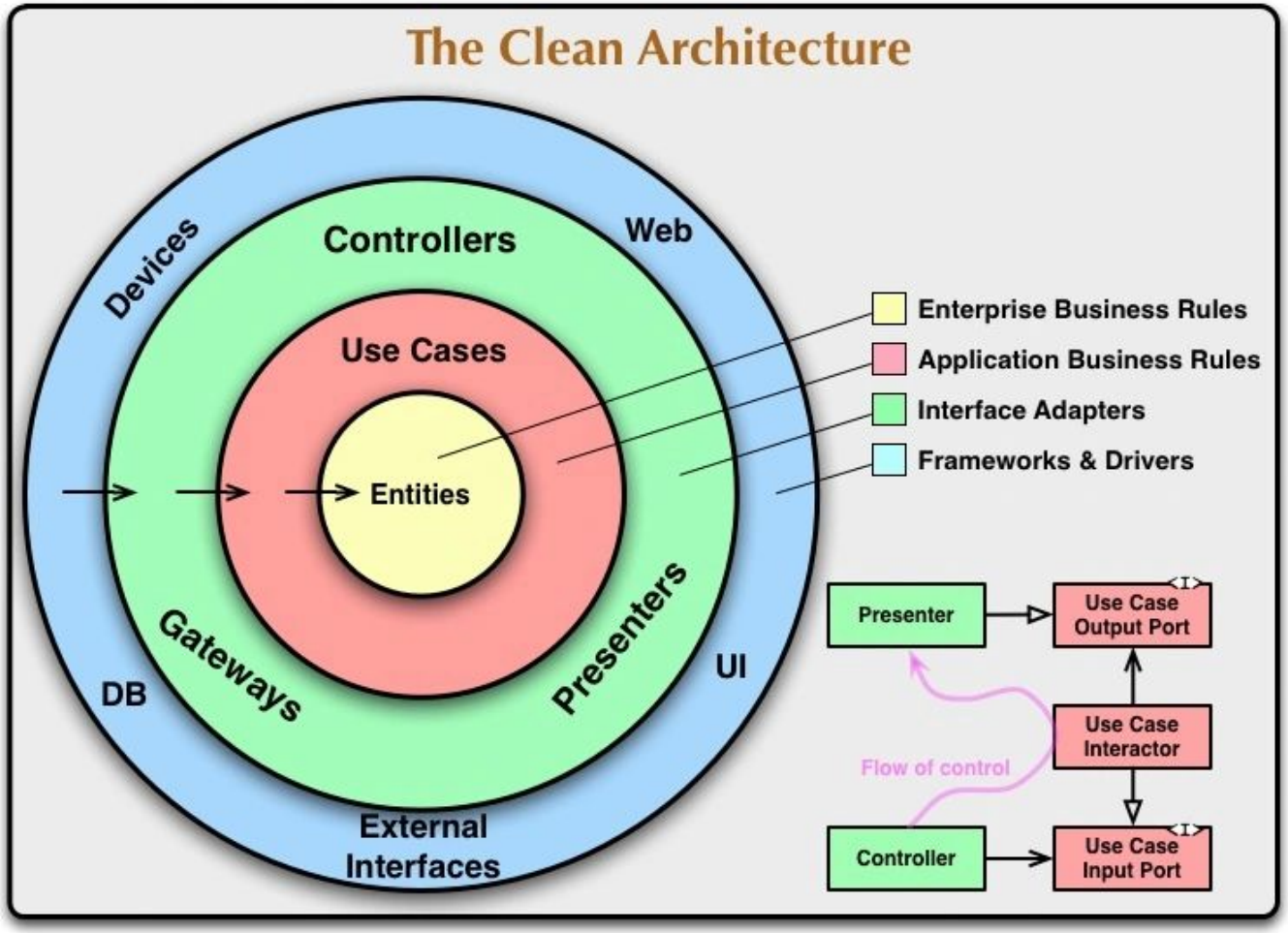
What is “Clean architecture” then?



What is “Clean architecture” then?

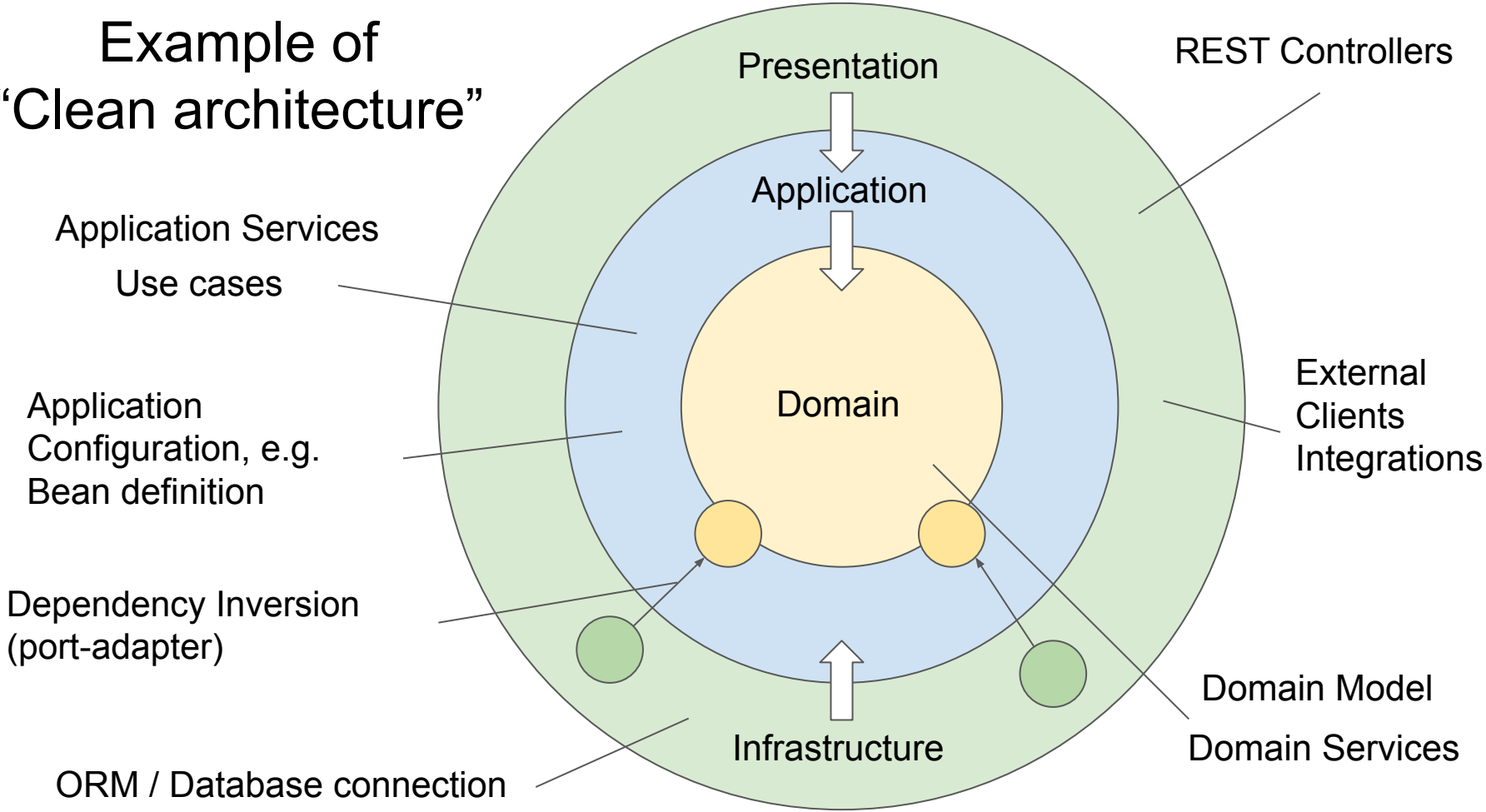
- Highly testable,
- Independent of framework,
- Independent of GUI,
- Independent of database,
- Independent of external dependencies.





https://herbertograca.com/2017/09/28/clean-architecture-standing-on-the-shoulders-of-giants/

Example of “Clean architecture”



Code example

Image generated by AI using OpenAI's DALL-E

Step 1: Product Listings

Step 1: Product Listings

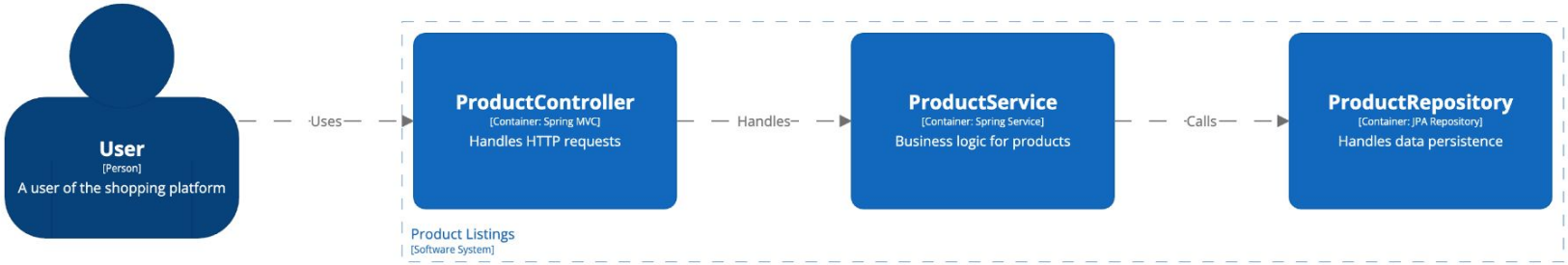
POST /products

Add Product

GET /products

Product Listings

Step 1: Product Listings



[Container] Product Listings
Thursday 19 September 2024 at 18:40 Central European Summer Time

Demo

Step 2: Manage Product Discounts

Two types of discounts:

- “N for the price of 1”
- Count-based percentage

Step 2: Manage Product Discounts

Two types of discounts:

- “N for the price of 1”
- Count-based percentage

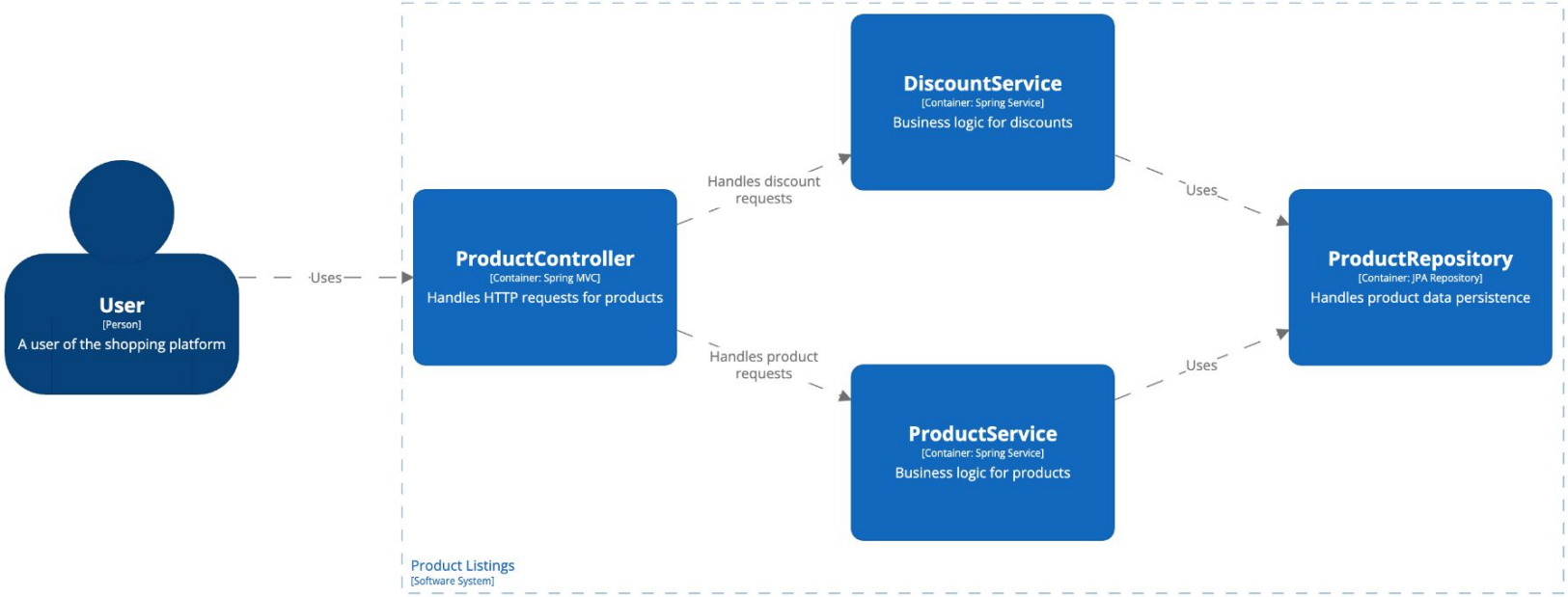
POST /products/{productId}/discounts

Add Discount To Product

POST /products/calculate-price

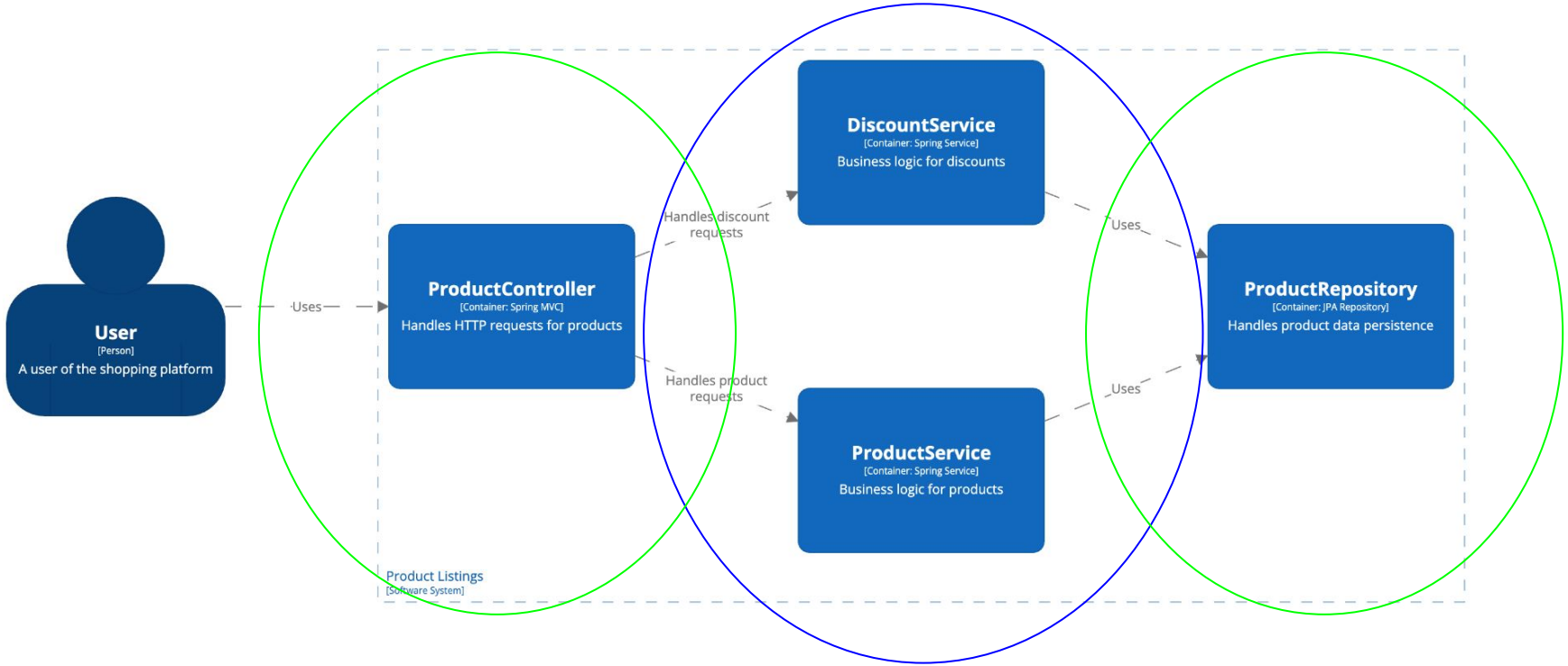
Products Pricing

Step 2: Manage Product Discounts



[Container] Product Listings
Thursday 19 September 2024 at 18:52 Central European Summer Time

Step 2: Manage Product Discounts



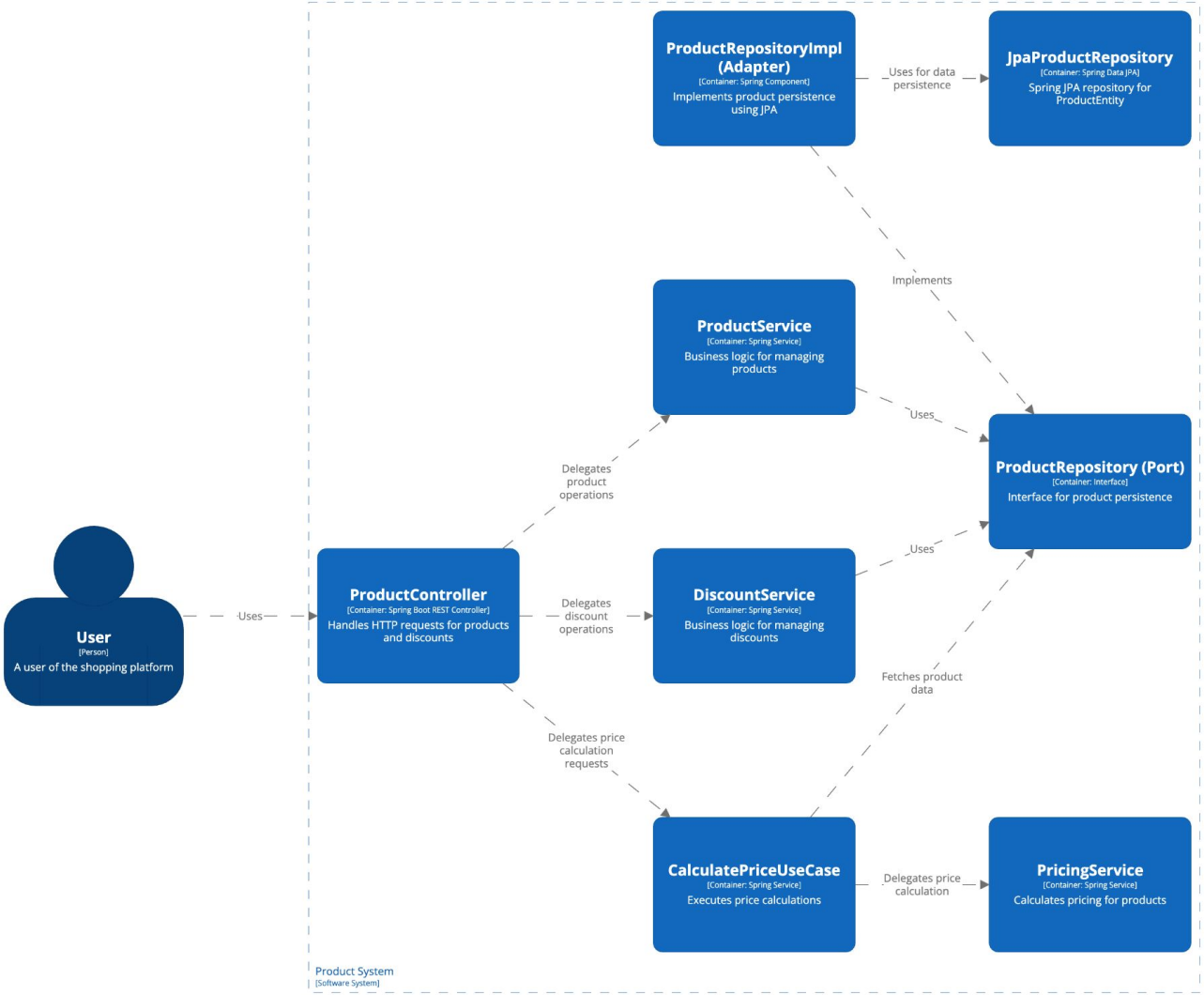
[Container] Product Listings
Thursday 19 September 2024 at 18:52 Central European Summer Time

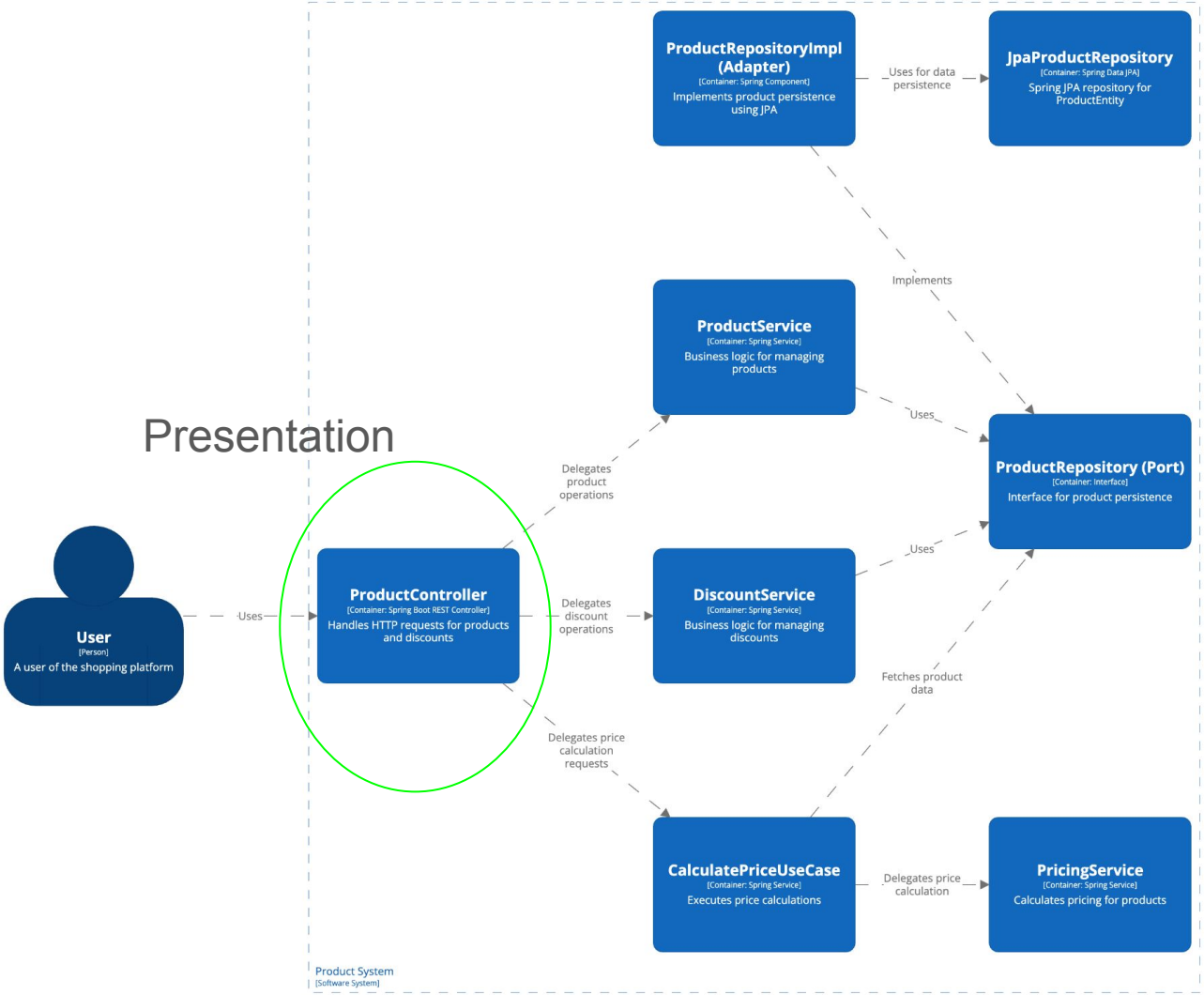
Presentation layer

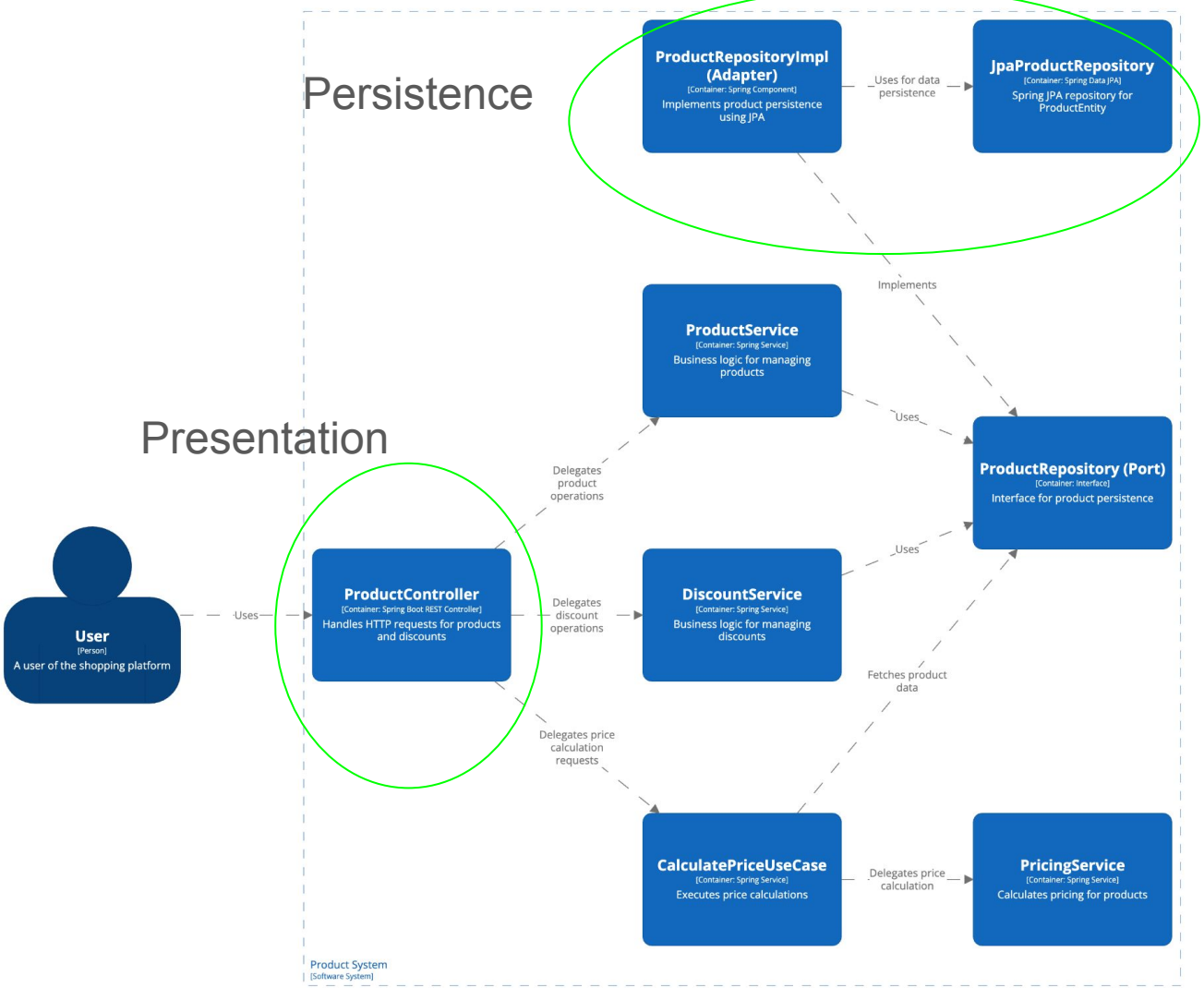
Business layer

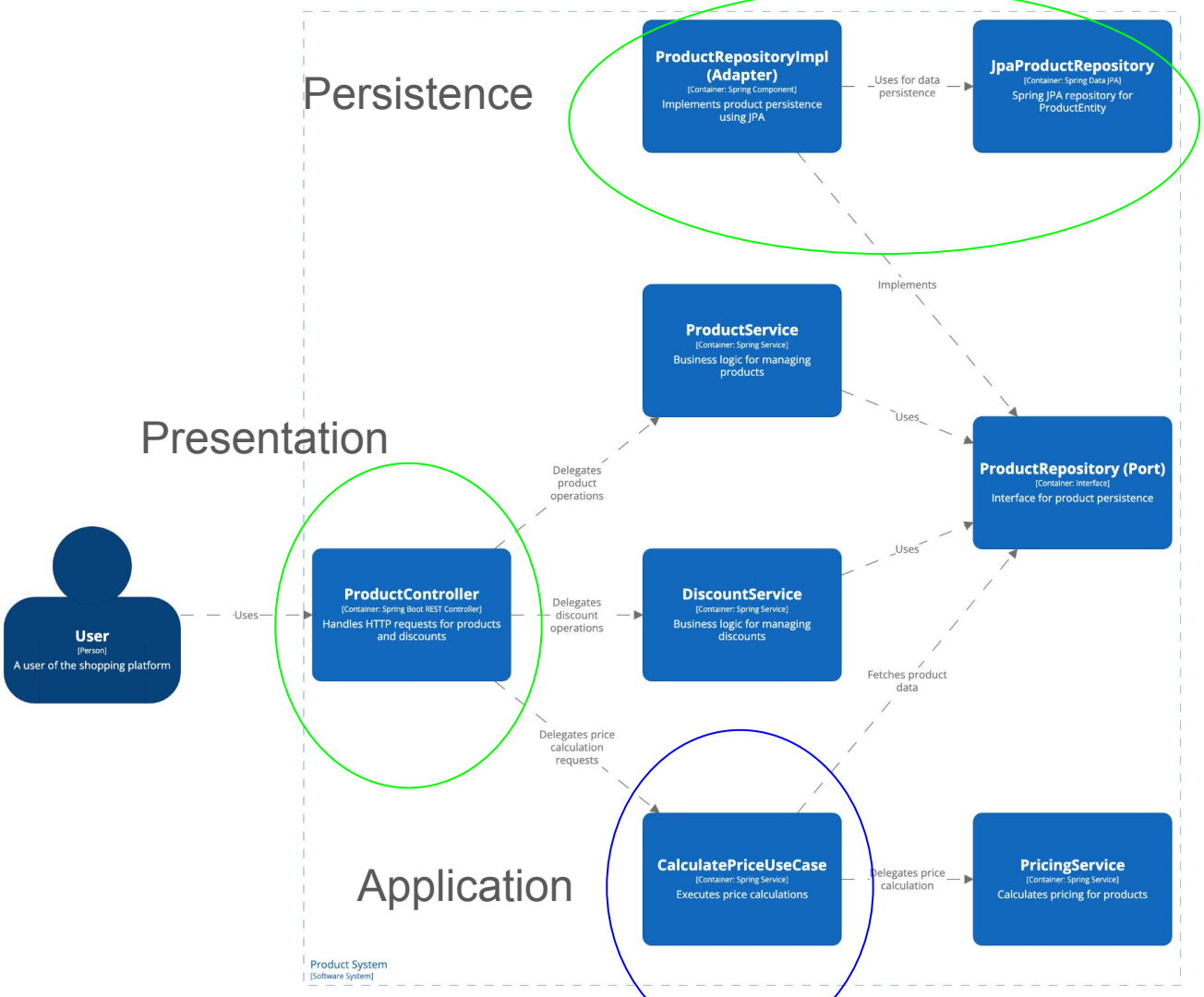
Persistence layer

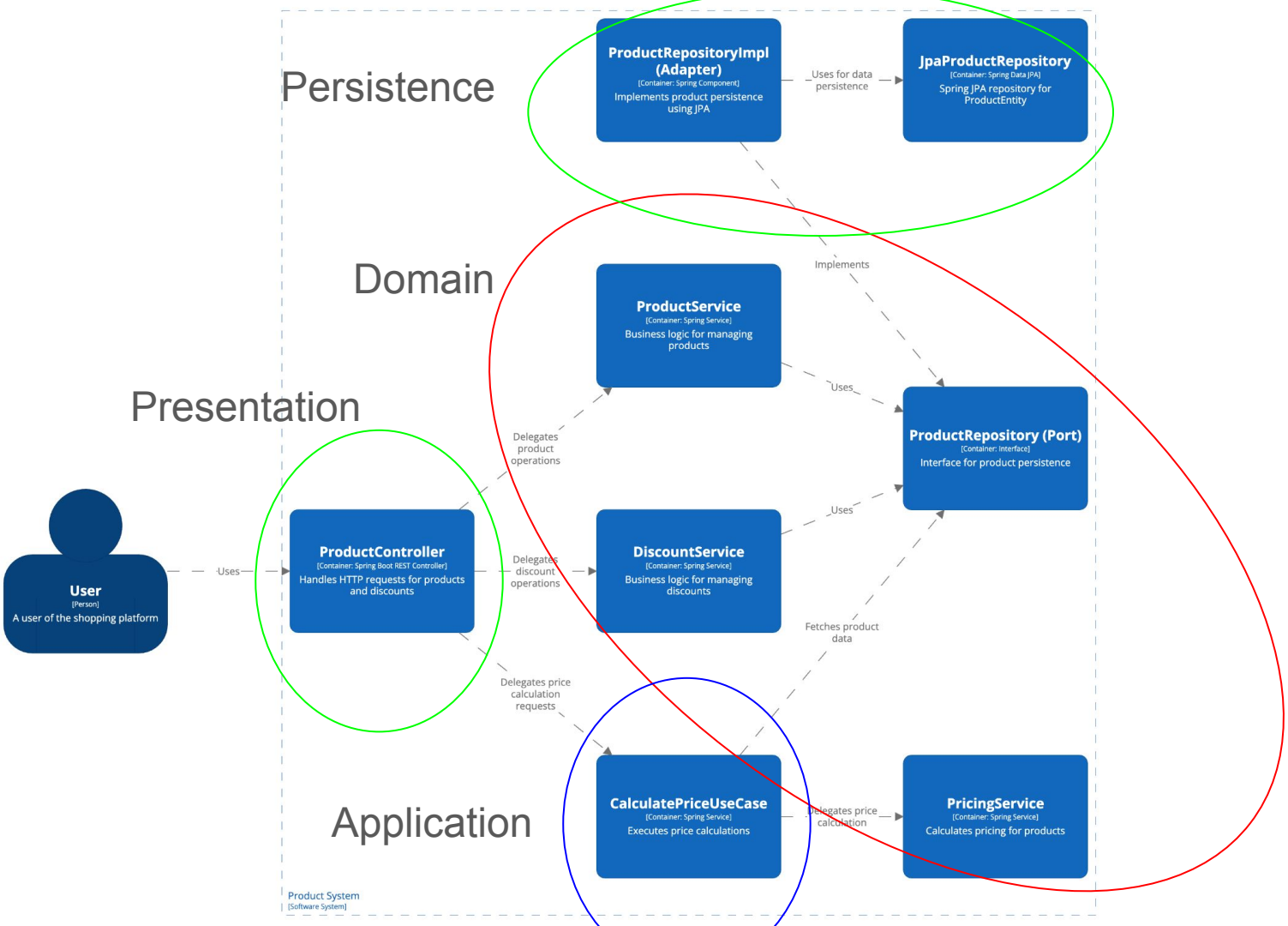
Demo


ProductRepositoryImpl (Adapter)
[Container: Spring Component]
Implements product persistence using JPA
Uses for data persistence
JpaProductRepository
[Container: Spring Data JPA]
Spring JPA repository for ProductEntity
Implements
ProductService
[Container: Spring Service]
Business logic for managing products
Uses
ProductRepository (Port)
[Container: Interface]
Interface for product persistence
Delegates product operations
Uses
User
[Person]
A user of the shopping platform
Uses
ProductController
[Container: Spring Boot REST Controller]
Handles HTTP requests for products and discounts
Delegates discount operations
DiscountService
[Container: Spring Service]
Business logic for managing discounts
Fetches product data
Delegates price calculation requests
CalculatePriceUseCase
[Container: Spring Service]
Executes price calculations
Delegates price calculation
PricingService
[Container: Spring Service]
Calculates pricing for products
Product System
[Software System]


Presentation
User
[Person]
A user of the shopping platform
Uses
ProductController
[Container: Spring Boot REST Controller]
Handles HTTP requests for products and discounts
Delegates product operations
Delegates discount operations
Delegates price calculation requests
ProductRepositoryImpl (Adapter)
[Container: Spring Component]
Implements product persistence using JPA
Uses for data persistence
JpaProductRepository
[Container: Spring Data JPA]
Spring JPA repository for ProductEntity
Implements
ProductService
[Container: Spring Service]
Business logic for managing products
Uses
ProductRepository (Port)
[Container: Interface]
Interface for product persistence
Uses
DiscountService
[Container: Spring Service]
Business logic for managing discounts
Fetches product data
CalculatePriceUseCase
[Container: Spring Service]
Executes price calculations
Delegates price calculation
PricingService
[Container: Spring Service]
Calculates pricing for products
Product System
[Software System]


Persistence
ProductRepositoryImpl (Adapter)
[Container: Spring Component]
Implements product persistence using JPA
Uses for data persistence
JpaProductRepository
[Container: Spring Data JPA]
Spring JPA repository for ProductEntity
Implements
ProductService
[Container: Spring Service]
Business logic for managing products
Uses
ProductRepository (Port)
[Container: Interface]
Interface for product persistence
Presentation
Delegates product operations
Uses
User
[Person]
A user of the shopping platform
Uses
ProductController
[Container: Spring Boot REST Controller]
Handles HTTP requests for products and discounts
Delegates discount operations
DiscountService
[Container: Spring Service]
Business logic for managing discounts
Fetches product data
Delegates price calculation requests
CalculatePriceUseCase
[Container: Spring Service]
Executes price calculations
Delegates price calculation
PricingService
[Container: Spring Service]
Calculates pricing for products
Product System
[Software System]


Persistence
ProductRepositoryImpl (Adapter)
[Container: Spring Component]
Implements product persistence using JPA
Uses for data persistence
JpaProductRepository
[Container: Spring Data JPA]
Spring JPA repository for ProductEntity
Implements
ProductService
[Container: Spring Service]
Business logic for managing products
Uses
ProductRepository (Port)
[Container: Interface]
Interface for product persistence
Presentation
Delegates product operations
User
[Person]
A user of the shopping platform
Uses
ProductController
[Container: Spring Boot REST Controller]
Handles HTTP requests for products and discounts
Delegates discount operations
DiscountService
[Container: Spring Service]
Business logic for managing discounts
Uses
Fetches product data
Delegates price calculation requests
Application
CalculatePriceUseCase
[Container: Spring Service]
Executes price calculations
Delegates price calculation
PricingService
[Container: Spring Service]
Calculates pricing for products
Product System
[Software System]


Persistence
Domain
Presentation
Application
ProductRepositoryImpl (Adapter)
[Container: Spring Component]
Implements product persistence using JPA
Uses for data persistence
JpaProductRepository
[Container: Spring Data JPA]
Spring JPA repository for ProductEntity
Implements
ProductService
[Container: Spring Service]
Business logic for managing products
Uses
ProductRepository (Port)
[Container: Interface]
Interface for product persistence
Delegates product operations
Uses
User
[Person]
A user of the shopping platform
Uses
ProductController
[Container: Spring Boot REST Controller]
Handles HTTP requests for products and discounts
Delegates discount operations
DiscountService
[Container: Spring Service]
Business logic for managing discounts
Fetches product data
Delegates price calculation requests
CalculatePriceUseCase
[Container: Spring Service]
Executes price calculations
Delegates price calculation
PricingService
[Container: Spring Service]
Calculates pricing for products
Product System
[Software System]

Symptoms of “unclean” architecture

Symptoms of “unclean” architecture

- Tight coupling,

Symptoms of “unclean” architecture

- Tight coupling,
- Hard-to-test code,

Symptoms of “unclean” architecture

- Tight coupling,
- Hard-to-test code,
- Mixed concerns,

Symptoms of “unclean” architecture

- Tight coupling,
- Hard-to-test code,
- Mixed concerns,
- Circular dependencies,

Symptoms of “unclean” architecture

- Tight coupling,
- Hard-to-test code,
- Mixed concerns,
- Circular dependencies,
- Difficult to extend,

Symptoms of “unclean” architecture

- Tight coupling,
- Hard-to-test code,
- Mixed concerns,
- Circular dependencies,
- Difficult to extend,
- Unclear structure - where to put my changes?

Q&A