1. Propósito Principal de Clean Architecture

Clean Architecture busca crear sistemas que sean independientes de frameworks, bases de datos, interfaces de usuario y agentes externos. Su objetivo principal es separar las preocupaciones del negocio de los detalles técnicos, creando código que sea testeable, mantenible y flexible ante cambios.

2. Beneficios en Microservicios Spring Boot

Testabilidad mejorada: La lógica de negocio se puede probar sin dependencias externas,Flexibilidad tecnológica: Cambiar de JPA a MongoDB sin afectar el negocio, Mantenibilidad: Código más organizado y fácil de entender, Reutilización: Los casos de uso pueden ser invocados desde diferentes controladores,Evolución independiente: Cada capa puede evolucionar sin afectar las demás

3. Principales Capas y Responsabilidades

Capa de Dominio (Domain):

Entidades de negocio

Reglas de negocio fundamentales

Interfaces de repositorios

Value Objects

Capa de Aplicación (Application):

Casos de uso (Use Cases)

Orquestación de la lógica de negocio

DTOs de entrada y salida

Interfaces de servicios externos

Capa de Infraestructura (Infrastructure):

Implementación de repositorios

Configuración de base de datos

Clientes HTTP

Mappers entre entidades y DTOs

Capa de Presentación (Presentation):

Controladores REST

Manejo de excepciones

Validaciones de entrada

Serialización/deserialización

4. Desacople de Lógica de Negocio e Infraestructura

El desacople es crucial porque:

Permite testing unitario sin dependencias externas, Facilita cambios tecnológicos sin afectar reglas de negocio, Mejora la portabilidad del código

5. Rol de la Capa de Aplicación

La capa de aplicación orquesta los casos de uso y contiene:

Casos de uso específicos del negocio

Coordinación entre diferentes entidades de dominio

Manejo de transacciones

Validaciones de flujo (no de negocio)

Transformación de datos entre capas

No debe contener reglas de negocio puras ni detalles de infraestructura.

6. Diferencia entre UseCase y Service

UseCase:

Representa una acción específica del usuario, entrada y salida claramente definidas, , atomic (una sola responsabilidad)

Ejemplo: `CreateUserUseCase`, `ProcessPaymentUseCase`

Service:

Agrupa funcionalidades relacionadas, puede contener múltiples operaciones, más genérico y reutilizable

Ejemplo: `UserService`, `PaymentService`

Los UseCases son más granulares y expresivos sobre la intención del negocio.

7. Repositories como Interfaces en Dominio

Definir repositories como interfaces en dominio (no usar directamente JpaRepository) porque:

Inversión de dependencias: El dominio no depende de Spring Data

Testabilidad: Se pueden crear implementaciones mock fácilmente

Flexibilidad: Cambiar de JPA a otra tecnología sin afectar dominio

Abstracción: El dominio define qué necesita, no cómo se implementa

8. Implementación de UseCase en Spring Boot

```java
// Dominio
public interface UserRepository {
    User save(User user);
    Optional<User> findByEmail(String email);
}

// Aplicación
@Component
public class CreateUserUseCase {
    private final UserRepository userRepository;

    public CreateUserUseCase(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    public UserResponse execute(CreateUserRequest request) {
        // Validaciones de negocio
        // Crear entidad
        // Persistir
        // Retornar respuesta
    }
}
```

Ventajas:

Fácil testing con mocks, código expresivo y claro, reutilizable desde diferentes controladores, separación clara de responsabilidades

9. Problemas sin Clean Architecture

Código acoplado: Cambios en infraestructura afectan negocio, Testing complejo: Necesitas base de datos para probar lógica, Mantenimiento costoso: Código mezclado y difícil de entender

10. Escalabilidad y Mantenibilidad en Microservicios

Clean Architecture facilita:

Escalabilidad:

Equipos independientes trabajando en diferentes capas, Deployment independiente de componentes, Optimización específica por capa, Scaling horizontal más sencillo

Mantenibilidad:

Código organizado y predecible,Testing automatizado más efectivo,Refactoring seguro con tests de regresión,Documentación implícita a través de la estructura, Onboarding rápido de nuevos desarrolladores