Aplicando Clean Architecture en Microservicios con Spring Boot

Jesús Estenllos Loaiza Serrano 2313021

Desarrollo de Software III

Docente: Ing. Álvaro Salazar

Grupo 50

Sede Tuluá

17/06/2025

1. ¿Cuál es el propósito principal de Clean Architecture en el desarrollo de software?

El propósito principal de Clean Architecture es separar las responsabilidades del software en capas bien definidas, desacoplando la lógica de negocio de la infraestructura y la presentación. Esto permite que los sistemas sean más mantenibles, escalables y adaptables al cambio tecnológico.

2. ¿Qué beneficios aporta Clean Architecture a un microservicio en Spring Boot?

Aplicar Clean Architecture en un microservicio con Spring Boot permite:

- Independencia tecnológica (la lógica no depende de frameworks).
- Mayor facilidad para realizar pruebas unitarias.
- Código más modular y mantenible.
- Escalabilidad: permite agregar nuevas funcionalidades sin romper otras.
- Reemplazo sencillo de tecnologías (por ejemplo, cambiar de JPA a MongoDB sin afectar la lógica central).

3. ¿Cuáles son las principales capas de Clean Architecture y qué responsabilidad tiene cada una?

- Dominio (Enterprise Business Rules): Contiene las entidades y reglas de negocio puras, sin dependencia de frameworks.
- Aplicación (Application Business Rules): Define los casos de uso (UseCases) y orquesta la lógica del dominio.
- Infraestructura (Adapters & Frameworks): Implementa servicios externos (bases de datos, colas, APIs) y configura tecnologías como JPA o Spring.
- Presentación (Delivery Mechanisms): Controladores REST y validaciones, gestiona la entrada/salida del sistema.

4. ¿Por qué se recomienda desacoplar la lógica de negocio de la infraestructura en un microservicio?

Porque así se facilita el mantenimiento y evolución del sistema. Si la infraestructura cambia (por ejemplo, de MySQL a PostgreSQL), no es necesario modificar la lógica de negocio. Además, se pueden realizar pruebas unitarias más efectivas sin necesidad de

instanciar dependencias externas.

5. ¿Cuál es el rol de la capa de aplicación y qué tipo de lógica debería contener?

La capa de aplicación orquesta los flujos del sistema mediante casos de uso. Contiene la lógica de aplicación, como validaciones, cálculos y reglas que coordinan las entidades del dominio, sin preocuparse por cómo se almacenan o muestran los datos.

6. ¿Qué diferencia hay entre un UseCase y un Service en Clean Architecture?

● UseCase: Representa una operación específica del sistema (ej. "Listar productos") y se enfoca en una lógica de negocio bien definida.

● Service: Es un concepto más amplio que puede contener múltiples responsabilidades.

En proyectos pequeños puede ser suficiente, pero en sistemas complejos puede mezclarse lógica de distintos flujos, por lo que es mejor dividir en UseCases.

7. ¿Por qué se recomienda definir Repositories como interfaces en la capa de dominio en lugar de usar directamente JpaRepository?

Porque usar interfaces en el dominio permite desacoplar la lógica de negocio de cualquier tecnología de persistencia. De esta forma, se puede cambiar la implementación (JPA, Mongo, REST, etc.) sin modificar la lógica central del sistema.

8. ¿Cómo se implementa un UseCase en un microservicio con Spring Boot y qué ventajas tiene?

Un UseCase se implementa como una clase de servicio en la capa de aplicación que orquesta el acceso a los repositorios del dominio. Ventajas:

● Aisla la lógica de negocio.

● Facilita cambios en reglas sin afectar controladores o infraestructura.

● Favorece pruebas unitarias.

● Mejora la organización del código por responsabilidades.

9. ¿Qué problemas podrían surgir si no aplicamos Clean Architecture en un proyecto de microservicios?

● Acoplamiento entre capas (por ejemplo, lógica de negocio mezclada con controladores o repositorios).

● Dificultad para realizar pruebas.

● Problemas al cambiar la base de datos o la tecnología.

● Menor mantenibilidad y escalabilidad.

● Riesgo de errores al introducir nuevas funcionalidades.

10. ¿Cómo Clean Architecture facilita la escalabilidad y mantenibilidad en un entorno basado en microservicios?

Clean Architecture permite que cada microservicio tenga una estructura modular, desacoplada y bien organizada. Esto hace que cada parte del sistema pueda escalar, testearse, desplegarse y mantenerse de forma independiente, lo cual es clave en arquitecturas distribuidas.

11. ¿Qué es Attribute-Driven Design (ADD) y cuál es su propósito en el diseño de software?

Attribute-Driven Design (ADD) es un método sistemático para diseñar arquitecturas de software basado en atributos de calidad (requisitos no funcionales como escalabilidad, seguridad o desempeño). Su propósito es:

- Definir una arquitectura que cumpla con los requisitos clave del sistema.
- Tomar decisiones basadas en prioridades de negocio y restricciones técnicas.
- Evitar soluciones genéricas, enfocándose en necesidades específicas.

12. ¿Cómo se relaciona ADD con Clean Architecture en el proceso de diseño de sistemas?

ADD y Clean Architecture son complementarios:

- ADD guía el diseño inicial basado en atributos de calidad (ej: si la seguridad es crítica, sugiere capas de autenticación).
- Clean Architecture implementa esos requisitos mediante una estructura organizada (capas independientes, inversión de dependencias).
- Juntos, ADD define el "qué" (requisitos) y Clean Architecture el "cómo" (implementación limpia y mantenible).

13. ¿Cuáles son los pasos principales del método ADD para definir una arquitectura de software?

Los pasos clave son:

1. Identificar requisitos y atributos de calidad (ej: rendimiento, modularidad).
2. Priorizar atributos según importancia para el negocio.
3. Seleccionar patrones arquitectónicos (ej: microservicios si escalabilidad es clave).

4. Definir elementos arquitectónicos (módulos, componentes, conexiones).

5. Iterar y refinar hasta cumplir los requisitos.

14. ¿Cómo se identifican los atributos de calidad en ADD y por qué son importantes?

Se identifican mediante:

- Entrevistas con stakeholders (ej: el equipo de negocio prioriza disponibilidad).
- Escenarios de calidad (ej: "El sistema debe manejar 10,000 usuarios concurrentes").
- Estándares del dominio (ej: HIPAA para salud).

Importancia:

- Determinan las decisiones técnicas (ej: usar caché si el rendimiento es crítico).
- Evitan sobreingeniería o soluciones inadecuadas.

15. ¿Por qué Clean Architecture complementa ADD en la implementación de una solución?

- ADD puede sugerir alta mantenibilidad como atributo clave → Clean Architecture la garantiza con:
  - Separación clara de capas (Entities, Use Cases, Interfaces).
  - Independencia del framework y la base de datos.
  - Facilidad para cambiar componentes sin impacto global.
- Ejemplo: Si ADD prioriza testabilidad, Clean Architecture la facilita con interfaces desacopladas.

16. ¿Qué criterios se deben considerar al definir las capas en Clean Architecture dentro de un proceso ADD?

Basarse en los atributos de calidad identificados:

- Seguridad: Capa de autenticación independiente.
- Escalabilidad: Microservicios en lugar de monolito.
- Mantenibilidad: Reglas de negocio en capas internas (Entities, Use Cases).
- Flexibilidad: Inversión de dependencias (DIP) para adaptarse a cambios.

17. ¿Cómo ADD ayuda a tomar decisiones arquitectónicas basadas en necesidades del negocio?

- Traduce requisitos no funcionales a diseños concretos:

o Ejemplo: Si el negocio requiere bajo tiempo de mercado (time-to-market), ADD puede sugerir arquitecturas modulares con componentes reutilizables.

- Evita soluciones técnicas incongruentes con las metas del negocio.

18. ¿Cuáles son los beneficios de combinar ADD con Clean Architecture en un sistema basado en microservicios?

- ADD define:

o Microservicios como patrón para escalabilidad y despliegue independiente.

- Clean Architecture asegura:

o Cada microservicio sea mantenible (con sus propias capas internas).

o Acoplamiento mínimo entre servicios.

- Resultado: Sistema escalable, pero también robusto y fácil de modificar.

19. ¿Cómo se asegura que la arquitectura resultante cumpla con los atributos de calidad definidos en ADD?

- Técnicas de validación:

o Prototipado: Pruebas de concepto para atributos críticos (ej: pruebas de carga).

o Reviews arquitectónicas: Evaluar el diseño contra los requisitos.

o Herramientas: Simuladores de rendimiento, análisis de seguridad (SonarQube, OWASP).

20. ¿Qué herramientas o metodologías pueden ayudar a validar una arquitectura diseñada con ADD y Clean Architecture?

- Modelado:

o UML o C4 Model para visualizar componentes.

- Pruebas:

o Chaos Engineering (Gremlin) para resiliencia.

o JMeter para rendimiento.

- Análisis estático:

o SonarQube para calidad de código.

- Metodologías:

- ATAM (Architecture Tradeoff Analysis Method) para evaluar compensaciones.