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

El objetivo fundamental de Clean Architecture es organizar el software en capas bien diferenciadas, donde cada una tiene responsabilidades específicas y no depende de las demás de manera directa. Esto significa separar completamente la lógica de negocio de los detalles técnicos como bases de datos, frameworks o interfaces de usuario. El resultado es un sistema donde puedes cambiar la tecnología sin afectar las reglas de negocio centrales, lo que hace que el software sea más resistente al cambio y más fácil de mantener a largo plazo.

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

Implementar Clean Architecture en microservicios Spring Boot genera varios beneficios importantes:

- La lógica de negocio se vuelve completamente independiente de Spring Boot o cualquier framework específico
- Las pruebas unitarias se simplifican enormemente porque no necesitas cargar el contexto completo de Spring para probar la lógica central
- El código se organiza de manera mucho más clara y modular, facilitando el trabajo en equipo
- Puedes escalar diferentes partes del sistema independientemente sin riesgo de romper funcionalidades existentes
- Cambiar tecnologías se vuelve sencillo, por ejemplo migrar de JPA con MySQL a MongoDB sin tocar la lógica de negocio

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

Capa de Dominio (Enterprise Business Rules): Esta es la capa más interna y contiene las entidades del negocio y las reglas que nunca cambian, independientemente de la tecnología que uses. Aquí no hay dependencias externas de ningún tipo.

Capa de Aplicación (Application Business Rules): Aquí se definen todos los casos de uso del sistema. Esta capa se encarga de orquestar las operaciones del dominio pero no sabe nada sobre cómo se implementan los detalles técnicos.

Capa de Infraestructura (Adapters & Frameworks): Contiene todas las implementaciones técnicas como acceso a bases de datos, integración con APIs externas, configuraciones de Spring, y todo lo relacionado con tecnologías específicas.

Capa de Presentación (Delivery Mechanisms): Maneja toda la comunicación externa del sistema, incluyendo controladores REST, validaciones de entrada, formateo de respuestas y manejo de errores.

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

El desacoplamiento es fundamental porque permite que el sistema evolucione de manera más fluida. Cuando tu lógica de negocio no depende de detalles técnicos específicos, puedes hacer cambios significativos en la infraestructura sin riesgo de romper funcionalidades. Por ejemplo, si decides migrar de MySQL a PostgreSQL, o incluso a una base de datos NoSQL, solo necesitas cambiar la implementación en la capa de infraestructura. Además, esto facilita enormemente las pruebas porque puedes validar tu lógica de negocio sin necesidad de tener una base de datos corriendo o servicios externos disponibles.

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

La capa de aplicación actúa como coordinadora de todo el flujo del sistema a través de los casos de uso. Su responsabilidad principal es implementar la lógica de aplicación específica, que incluye validaciones de flujo, cálculos complejos, y coordinación entre diferentes entidades del dominio. Esta capa no se preocupa por cómo se almacenan los datos ni cómo se presentan al usuario, simplemente orquesta las operaciones necesarias para completar cada caso de uso del sistema.

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

UseCase: Representa una operación muy específica y bien definida del sistema, como "Registrar nuevo usuario" o "Calcular descuento de producto". Cada UseCase tiene una responsabilidad única y está enfocado en resolver un problema particular del negocio.

Service: Es un concepto más amplio que tradicionalmente puede contener múltiples responsabilidades diferentes. En proyectos pequeños puede funcionar bien, pero conforme el sistema crece, los Services tienden a acumular demasiadas responsabilidades, mezclando lógica de diferentes flujos de negocio. Por eso en sistemas complejos es preferible dividir la funcionalidad en UseCases específicos.

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

Definir interfaces propias en el dominio te permite mantener la lógica de negocio completamente independiente de cualquier tecnología de persistencia específica. Cuando usas directamente JpaRepository, estás acoplando tu dominio a Spring Data JPA. En cambio, al definir tu propia interfaz, puedes implementarla con JPA hoy, pero mañana cambiar a MongoDB, servicios REST, o cualquier otra tecnología sin modificar ni una línea de tu lógica central. Esto te da flexibilidad total para evolucionar la arquitectura técnica sin impactar el corazón 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 específica en la capa de aplicación que se encarga de orquestar el acceso a los repositorios del dominio y coordinar todas las operaciones necesarias para completar un caso de uso particular.

Las ventajas principales son:

- La lógica de negocio queda completamente aislada de detalles técnicos
- Los cambios en reglas de negocio se pueden hacer sin afectar controladores o implementaciones de infraestructura
- Las pruebas unitarias se vuelven mucho más efectivas y rápidas
- El código se organiza mejor porque cada UseCase tiene responsabilidades claramente definidas

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

Sin Clean Architecture, los proyectos de microservicios enfrentan varios problemas serios:

- Acoplamiento fuerte entre capas, donde la lógica de negocio se mezcla con controladores, repositorios y configuraciones técnicas
- Dificultad extrema para realizar pruebas efectivas porque todo está interconectado
- Cambios tecnológicos se vuelven proyectos complejos y riesgosos que pueden romper funcionalidades existentes
- La mantenibilidad se degrada rápidamente conforme el sistema crece
- La escalabilidad se ve limitada porque no puedes modificar partes específicas sin riesgo de afectar otras
- Alto riesgo de introducir errores al agregar nuevas funcionalidades

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

Clean Architecture es especialmente valiosa en microservicios porque permite que cada servicio mantenga una estructura interna modular, desacoplada y bien organizada. Esto significa que cada microservicio puede escalar, probarse, desplegarse y mantenerse de forma completamente independiente de los demás. En arquitecturas distribuidas, esta independencia es fundamental porque permite que diferentes equipos trabajen en paralelo sin interferencias, facilita la adopción de nuevas tecnologías de manera gradual, y reduce significativamente el riesgo de que cambios en un servicio afecten a otros componentes del sistema.