Clean Architecture en microservicios Spring Boot

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

El propósito principal de **Clean Architecture** es **organizar el código de forma que esté desacoplado, testeable y fácil de mantener**, permitiendo que la lógica de negocio esté aislada de detalles técnicos como bases de datos, frameworks o interfaces de usuario. Busca que los cambios en infraestructura o tecnología **no afecten el núcleo de la aplicación**.

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

- **Separación clara de responsabilidades**, lo que facilita el mantenimiento.
- **Desacoplamiento de tecnologías**, permitiendo cambiar frameworks o bases de datos sin modificar la lógica de negocio.
- **Mejor estructura para pruebas unitarias**.
- **Escalabilidad natural**, ya que cada microservicio puede evolucionar independientemente.
- **Legibilidad y organización** del código, ideal para equipos grandes.

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

1. **Dominio**: Contiene entidades y lógica de negocio pura.
2. **Aplicación**: Gestiona los casos de uso y orquesta el flujo entre capas.
3. **Infraestructura**: Implementa detalles técnicos como bases de datos, clientes HTTP, etc.
4. **Presentación**: Controladores REST, UI o interfaces expuestas al usuario o sistema externo.

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

Porque así se logra:

- **Independencia tecnológica**: se puede cambiar la base de datos, motor de mensajería o framework sin reescribir reglas de negocio.

- **Reutilización**: la lógica de negocio puede probarse o reutilizarse en otros contextos.
- **Pruebas más efectivas**: permite probar lógica sin depender de entornos externos como bases de datos.

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

La **capa de aplicación** se encarga de los **casos de uso** del sistema.
Debe contener lógica que **orquesta operaciones**, como coordinar acciones entre entidades del dominio y servicios de infraestructura, pero sin incluir lógica de negocio compleja ni lógica técnica.

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

- Un **UseCase** representa un **flujo de negocio concreto** (por ejemplo: "crear pedido"), y vive en la **capa de aplicación**.
- Un **Service** puede estar en diferentes capas:
    - En **dominio**, representa lógica de negocio específica.
    - En **infraestructura**, implementa interacciones técnicas (por ejemplo, enviar un correo).

En Clean Architecture, los **UseCases orquestan servicios** del dominio y colaboran con interfaces para acceder a recursos externos.

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

Porque así se mantiene el **dominio independiente del framework**.
JpaRepository pertenece a Spring Data (infraestructura), y si se usa directamente en el dominio, se rompe el principio de inversión de dependencias.
En cambio, al definir interfaces en el dominio y su implementación en infraestructura, se logra **flexibilidad y bajo acoplamiento**.

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

Un **UseCase** se implementa como una clase en la **capa de aplicación**, que:

- Usa interfaces del dominio (como repositorios).
- Contiene el flujo del negocio.
- Es llamado desde un controlador o desde otros casos de uso.

Ventajas:

- **Código más limpio y enfocado**.
- **Facilidad de prueba unitaria**.
- Permite aplicar **inyección de dependencias** para desacoplar infraestructura.

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

- **Código acoplado** a frameworks y tecnologías.
- **Dificultad para hacer pruebas** unitarias sin base de datos real o servidor.
- **Problemas de mantenimiento** a medida que crece el código.
- Alta **dependencia entre módulos**, dificultando escalabilidad.
- **Duplicación de lógica** entre servicios por falta de estructura.

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

- Permite que **cada microservicio evolucione de forma independiente**.
- Al separar lógica, facilita el trabajo de **múltiples equipos en paralelo**.
- Hace más fácil **cambiar tecnologías** sin afectar el dominio.
- Mejora la **observabilidad, trazabilidad y pruebas**, claves en sistemas distribuidos.