Codelab Virtual Threads

Desarrollo de Software III

Jhojan Stiven Castaño Jejen-2259476

Universidad Del Valle

Sede Tuluá

Septimo Semestre

2025

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

   R/= El propósito principal de Clean Architecture es encargarse de la estructuración del software en capas claramente diferenciadas, haciendo que la lógica central del negocio este separada de detalles como interfaz de usuario o la base de datos. Logrando facilitar el sistema sea flexible, fácil de extender y mas resistente a cambios tecnológicos.

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

R/= A la hora de aplicar clean architecture en un microservicio en el cual es desarrollado con spring boot, se tienen ventajas como:

- Independencia del framework
- Código más organizado y fácil de comprender
- Mejor capacidad para evolucionar el sistema sin romper funcionalidades existentes
- Permitir cambios tecnológicos de soporte sin impactar el núcleo del sistema

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

R/= Dominio: Alberga las entidades y reglas del negocio que son esenciales, sin la dependencia tecnológica.

● Aplicación: Define los casos de uso que dictan cómo se comporta el sistema.

● Infraestructura: Se encarga de conectar el sistema con tecnologías externas.

● Presentación: Gestiona la interacción con el usuario o cliente.

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

R/= Al mantener separada la lógica del negocio de la tecnología utilizada, es más sencillo poder adaptar el sistema a nuevos requerimientos o plataformas. Ademas, esto facilita las pruebas, ya que se pueden probar la lógica sin depender de componentes externos.

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

R/= La capa de plicacion se encagar de actuar como coordinador de las operaciones del sistema. Es el lugar donde se define que acciones se deben ejecutar, aplicando reglas de negocios y validaciones, sin preocupaciones por detalles de persistencia o presentación.

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

R/=**UseCase** se encarga de representar una funcionalidad especifica que el sistema ofrece, por ejemplo, registrar un usuario.

En comparación del **un Service** que es un termino mas amplio y genérico, por lo general agrupa diversas responsabilidades. En modo que el sistema crece, es mas preferible estructurar la lógica del **UseCase** más pequeños y específicos.

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

R/= Diseñar los repositorios como inferfaces en el dominio logra evitar que la lógica del negocio dependa directamente de tecnologías concretas como JPA. Permitiendo cambiar la tecnología de persistencia sin necesidad de modificaciones la lógica del sistema.

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

R/= Un UseCase suele ser una clase que actúa como servicio en la capa de aplicación. Esto se encarga de coordinar el acceso a los repositorios y aplica reglas de negocio. Entre sus beneficios están: mejor aislamiento de la lógica, mayor facilidad para realizar cambios, pruebas unitarias sencilla y un código mucho mas sencillo y estructurado

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

R/= Problemas comunes serian que el código termine muy acoplado, donde la lógica el negocio se mezcle con los controladores o detalles de persistencia. Dificultando las pruebas, complicando cambios tecnológicos y hace que el mantenimiento y la evolución del sistema sean mas costosos y propensos a errores.

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

R/= El clean architecture permite que cada microservicio evolucione de forma independiente. Esto mejora la capacidad de escalar componentes individuales y facilita la implementación de nuevas funcionalidades, el mantenimiento y las pruebas, lo cual es fundamental en sistemas distribuidos.