ADR - Microservicios - Desarrollo de capa de acceso a datos

ADR-BACKEND-REPOSITORY-CAPA-DE-ACCESO-A-DATOS

Estado	VIGENTE
Partes interesadas	AREA GOBIERNO TECNOLOGICO DESARROLLO SOFTWARE ASISTENCIAL (NTTDATA) DESARROLLO SOFTWARE ECONOMICO-FINANCIERO (NTTDATA) DESARROLLO SOFTWARE RRHH (AYESA)
TAGS	
Tomada el	17 may 2024
Responsable	LUIS MARTINEZ FONTIVEROS CARLOS GONZALEZ SANTIAGO
Documentación Asociada	Patrón Repositorio - Martín Fowler Arquitectura de referencia SAS
Solución Afectada	N/A

Version	Date	Author	Comment
2	06-04-02026	ADMINISTRADOR 6	Actualización automática de enlaces Jira/Confluence: ws001 -> nuevos dominios

Responsable	Aprobado por	Consultados	Informados
LUIS MARTINEZ FONTIVEROS CARLOS GONZALEZ SANTIAGO	LUIS MARTINEZ FONTIVEROS		DESARROLLO SOFTWARE ASISTENCIAL (NTTDATA) DESARROLLO SOFTWARE RRHH (AYESA) DESARROLLO SOFTWARE ECONOMICO-FINANCIERO (NTTDATA)

Asunto a Decidir	El objetivo de este ADR es establecer los patrones de diseño y arquitectónicos a implementar en la capa de acceso al dato.
	
	Criterios:
	
	• Que sea una solución estandarizada o al menos un estándar de facto dentro de la comunidad de desarrollo e ingeniería de software
	• Favorece el desacoplamiento
	• Sostenible
	• Reutilizable
	• Alineado con la arquitectura de referencia de contenedores.
	  ◦ Como encaja con la arquitectura de microservicios
	  ◦ Como encaja con clean Architecture
	  ◦ Como encaja con el desarrollo Domain Centric

Identificación de la decisión	Se decide basar la implementación en el **Patrón Repository** Como impulso para alcanzar el objetivo de mayor sostenibilidad del software, para los proyectos desarrollados en java, será **obligatorio** el uso del **framework de arquitectura** para la implementación de dicho patrón en los siguientes escenarios: • Implementaciones basadas en JPA • Implementaciones basadas en memoria

Contexto	Para facilitar las implementaciones dentro de esta transformación tecnológica, se busca definir un patrón común para desarrollar la capa de acceso a datos, con el objetivo de que sea reconocible por cualquier proveedor, y en cualquier aplicación. Esto permitirá al SAS, afrontar con mayores garantías situaciones como el mantenimiento de una funcionalidad, sus evolutivos, el cambio de los desarrolladores entre proyectos, la incorporación de nuevos desarrolladores o cambios de proveedor

Alternativas Consideradas

Repository Pattern

- **Características**
  - Centraliza el acceso a los datos y proporciona una abstracción de las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre las entidades del dominio.
- **Pros**:
  - Abstracción de operaciones CRUD.
  - Facilita pruebas unitarias.
  - Desacopla la lógica de negocio de la lógica de persistencia.

- **Contras**:
    - Puede volverse demasiado genérico.
    - Puede introducir una capa adicional de complejidad.

Criterio	OK/KO	Notas
Estándar	OK	Es un patrón de diseño que se puede implementar de forma estandarizada en cualquier lenguaje de programanción. Centrándose en java, lenguaje mayoritario, Frameworks como Sprint Boot lo ofrecen y dentro de la estandarización de Jakarta (Jakarta Data) actualmente se encuentra en desarrollo
Sostenible	OK	Al ser un patrón con gran capacidad de generalización se puede utilizar y reutilizar un diferentes desarrollos. Sus pautas son reconocibles por lo que se reduce los riesgos provocados por situaciones de mantenimiento de una funcionalidad, evolutivos, cambio de los desarrolladores entre proyectos, incorporación de nuevos desarrolladores o cambios de proveedor.
Arquitectura de Referencia	OK	• **Microservicios**: Ideal para encapsular la lógica de acceso a datos dentro de cada servicio. • **Clean Architecture**: Repositorios como interfaces en la capa de dominio, con implementaciones en la capa de infraestructura. • **Domain Centric**: Repositorios representan colecciones de agregados, manteniendo el enfoque en el dominio.
Desacoplado	OK	Haciendo uso de las interfaces adecuadas y de DTOS, se puede tener una implementación desacoplada de la tecnología y aislada de las definiciones del dominio
Reutilizable	OK	Su diseño genérico favorece la reutilización reduciendo los costes de desarrollo y mantenimiento.

DAO

- **Características**
    - **Abstracción**: DAO proporciona una abstracción de alto nivel para las operaciones de base de datos, de modo que los clientes no necesitan conocer los detalles de la implementación.
    - **Encapsulación**: Encapsula todos los accesos a la base de datos en una capa separada, lo que facilita el mantenimiento y la evolución de la lógica de acceso a datos.
    - **Separación de Responsabilidades**: Separa la lógica de acceso a datos de la lógica de negocio, promoviendo una arquitectura más modular y fácil de mantener.
- **Pros**
    - **Encapsulación de la Lógica de Acceso a Datos**: Centraliza todas las operaciones de acceso a datos en un lugar, facilitando el mantenimiento y la evolución de la lógica de acceso a datos.
    - **Separación de Responsabilidades**: Separa la lógica de negocio de la lógica de acceso a datos, promoviendo una arquitectura limpia y modular.

- **Facilita la Prueba y el Mocking**: Al encapsular el acceso a datos en interfaces y clases específicas, es más fácil crear mocks o stubs para pruebas unitarias.
- **Reutilización de Código**: Las operaciones de acceso a datos comunes se encapsulan en métodos reutilizables, reduciendo la duplicación de código.
- **Cambio Transparente de Implementación**: Si se necesita cambiar la base de datos o la tecnología de acceso a datos, solo el DAO necesita ser modificado sin afectar al resto de la aplicación.

- **Contras**
  - **Complejidad Adicional**: Introducir DAOs puede añadir una capa adicional de abstracción, aumentando la complejidad del diseño.
  - **Sobrecarga para Proyectos Pequeños**: En proyectos pequeños o simples, el uso de DAOs puede ser innecesario y sobrecargar el desarrollo.
  - **Falta de Flexibilidad en Consultas Dinámicas**: Los DAOs tradicionales pueden ser menos flexibles cuando se necesita construir consultas dinámicas complejas, lo que puede llevar a un aumento en el número de métodos específicos.

Criterio	OK/KO	Notas
Estándar	OK	Es un patrón de diseño que se puede implementar de forma estandarizada en cualquier lenguaje de programanción.
Sostenible	OK	Es un patrón más cercano a la tecnología de acceso al dato, aunque se puede implementar como una abstracción. Se suele hacer uso de los DAO dentro de los patrones repositorio. Sus pautas son reconocibles por lo que se reduce los riesgos provocados por situaciones de mantenimiento de una funcionalidad, evolutivos, cambio de los desarrolladores entre proyectos, incorporación de nuevos desarrolladores o cambios de proveedor.
Arquitectura de Referencia	OK	• **Microservicios**: Ayuda a mantener una separación clara de la lógica de acceso a datos dentro de cada microservicio, facilitando la independencia y escalabilidad de cada servicio. • **Clean Architecture**: Mantiene la independencia de la lógica de negocio respecto a la infraestructura, permitiendo cambiar la tecnología de persistencia sin afectar al dominio • **Domain Centric**: Encapsula la lógica de persistencia, manteniendo el dominio limpio y enfocado en la lógica de negocio. Facilita la implementación de repositorios que manejan agregados y aplican las reglas de negocio necesarias.
Reutilizable	OK	Su diseño genérico favorece la reutilización reduciendo los costes de desarrollo y mantenimiento.
Desacoplado	OK	Haciendo uso de las interfaces adecuadas y de DTOS, se puede tener una implementación desacoplada de la tecnología y aislada de las definiciones del dominio

Active Record Pattern

- **Características**:
  - Combina la lógica de negocio y el acceso a los datos en la misma clase (la de dominio), donde cada objeto es responsable de persistir o consultar sus propios datos.
- **Pros**:
  - Fácil de usar y rápido de implementar.
  - Conveniente para aplicaciones simples.
- **Contras**:
  - Mezcla lógica de negocio con acceso a datos.
  - Dificulta pruebas y escalabilidad.

Criterio	OK/KO	Notas
Estándar	OK	Es un patrón de diseño que se puede implementar de forma estandarizada en cualquier lenguaje de programanción.
Sostenible	KO	Los cambios producidos en la tecnología de acceso al dato impacta en el objeto de negocio de forma directa, aumentando los riesgos de que en un evolutivo se pueda romper un funcionalidad ya existente
Arquitectura de Referencia	KO	• **Microservicios**: Adecuado para servicios muy simples y autoconclusivos. • **Clean Architecture**: No encaja bien debido a la falta de separación de responsabilidades. • **Domain Centric**: Contradice el principio de separación de responsabilidades y encapsulación del dominio.
Reutilizable	KO	Por su simplicidad, no busca una abstracción reutilizable si no que se implementa en base a la necesidad concreta
Desacoplado	KO	Como se ha comentado como una de sus contras establece un acoplamiento directo entre el dominio y la tecnología de acceso al dato.

Comparando los DAO y Repository

- **DAO** es una abstracción de la persistencia de datos. Sin embargo, **Repository** es una abstracción de una colección de objetos.
- **DAO** es un concepto de nivel inferior, más cercano a los sistemas de almacenamiento. Sin embargo, **Repository** es un concepto de nivel superior, más cercano a los objetos del dominio.
- **DAO** funciona como una capa de acceso/mapeo de datos, ocultando consultas complejas. Sin embargo, un **Repository** es una capa entre los dominios y las capas de acceso a datos, ocultando la complejidad de reunir datos y preparar un objeto de dominio.

- **DAO NO** puede ser implementado usando un **Repository**. Sin embargo, un **Repository SI** puede usar un **DAO** p ara acceder al almacenamiento subyacente.
- Además, si tenemos un dominio anémico, **el Repository será simplemente un DAO**.
- Adicionalmente, el patrón Repository fomenta un diseño orientado al dominio, proporcionando una comprensión fácil de la estructura de datos incluso para los miembros del equipo no técnicos.

Comparando Active Record y Repository

- El patrón **Repository** externaliza la responsabilidades de acceso al dato (persistencia y consulta), mientras que **Active Record** las internaliza.
- Existen donde las clases **Active Record** son inyectadas como una instancia de repositorio que delegaba sus responsabilidades de acceso al dato de forma internamente.
- Los objetos de dominio **NO** deberían tener la responsabilidad de cómo (o incluso si) se persisten, tal y como el **A ctive Record** asume
- El patrón **Repository**, en cambio, proporciona ignorancia del mecanismo de acceso al dato a tus objetos de dominio y externaliza las preocupaciones de persistencia y proporcionarlas como un servicio externo.

Decisión Tomada

- El patrón elegido es el **Repository**
- Se considera el patrón más adecuado para alcanzar los objetivos de transformación tecnológica de la casa y alineado con la arquitectura de referencia.
- Está alineado con todos requisitos
- En la comparativa con el resto de patrones se determina que es el más agnóstico a los mecanismos de acceso al dato y el que más se acerca a una implementación basada en el dominio, es el patrón elegido
- Por simplificación de la solución, **no se hará uso** del patrón **DAO** dentro del patrón **Repostory** para implementar la capa más cercana a la tecnología del acceso al dato
- Como impulso para alcanzar el objetivo de mayor sostenibilidad del software, para los proyectos desarrollados en java, será **obligatorio** el uso del **framework de arquitectura** para la implementación de dicho patrón en los siguientes escenarios:
    - Implementaciones basadas en JPA
    - Implementaciones basadas en memoria
- Para la necesidad de acceso a la capa de datos a través de otros mecanismos se debe consultar con el departamento de arquitectura (l-arquitectura.stic.sspa@juntadeandalucia.es) sobre la estrategia de implementación para seguir fomentando los objetivos de reutilización y sostenibilidad.

Consecuencias

Será **obligatorio** el uso del **framework de arquitectura** para la implementación de dicho patrón en los siguientes escenarios:

- Implementaciones basadas en JPA
- Implementaciones basadas en memoria

Estado Actual

- 12 nov 2024 Se publica en el espacio de Gobernanza y calidad.