ADR - Microservicios - Desarrollo de capa de acceso a datos - Consultas

ADR - 000002 - Microservicios

Estado	VIGENTE
Partes interesadas	AREA GOBIERNO TECNOLOGICO DESARROLLO SOFTWARE ASISTENCIAL (NTTDATA) DESARROLLO SOFTWARE ECONOMICO-FINANCIERO (NTTDATA) DES ARROLLO SOFTWARE RRHH (AYESA)
TAGS	ADR Microservicios Repositorio Consultas
Tomada el	17 may 2024

Version	Date	Author	Comment
16	12-11-02024	LUIS MARTINEZ FONTIVEROS	Publicado en área de Gobernanza y Calidad

Responsable	LUIS MARTINEZ FONTIVEROS CARLOS GONZALEZ SANTIAGO
Documentación Asociada	Patrón Repositorio y especificación - Martin Fowler
Solución Afectada	N/A

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 las consultas y operaciones con filtrados (updates) de los datos
	
	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
	• Reutilizablo
	• 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 opta por una solución basada en el **Patrón Especificación** 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

Specification Pattern

Características

- patrón de diseño que se utilizan para abstraer y encapsular criterios de búsqueda o reglas de negocio

Pros:

- **Abstracción de las Condiciones de Consulta**: Permite definir condiciones de consulta de una manera abstracta y reutilizable, lo que facilita la construcción de consultas complejas.
- **Composición de Condiciones**: Las especificaciones pueden ser combinadas mediante operaciones lógicas como AND, OR, NOT, lo que permite construir consultas más elaboradas y flexibles.

- **Reutilización de Código**: Al definir condiciones de consulta como objetos reutilizables, se evita la duplicación de lógica de consulta en diferentes partes del código.
- **Facilita la Mantenibilidad**: Proporciona una manera clara y estructurada de representar condiciones de consulta, lo que facilita la comprensión y el mantenimiento del código.

Contras:

- **Complejidad Adicional**: Puede introducir una capa adicional de complejidad, especialmente en aplicaciones pequeñas o simples, donde la implementación del patrón puede ser excesiva.
- **Posible Sobrecarga de Abstracción**: En algunos casos, la abstracción proporcionada por el patrón Especificación puede resultar en una sobrecarga cognitiva para los desarrolladores, especialmente si no están familiarizados con el patrón.
- **Rendimiento Potencialmente Menor**: Dependiendo de la implementación y la cantidad de condiciones evaluadas, el patrón Especificación puede tener un impacto en el rendimiento de las consultas, especialmente en sistemas con grandes volúmenes de datos.
- **Dificultad para Expresar Algunas Consultas**: Algunas consultas complejas pueden ser difíciles de expresar utilizando el patrón Especificación, lo que puede

Criterio	OK/KO	Notas
Estándar	OK	Es un patrón de diseño que se puede implementar de forma estandarizada en cualquier lenguaje de programación.
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**: Beneficioso para encapsular reglas de negocio reutilizables y validaciones dentro de servicios, facilitando la coherencia y la reutilización de reglas comunes. • **Clean Architecture**: Encajaría en la capa de dominio, describiendo reglas de negocio que pueden ser combinadas y reutilizadas, manteniendo el dominio limpio y libre de lógica específica de la infraestructura • **Domain Centric**: Es especialmente poderoso en Domain Centric para encapsular reglas de negocio complejas y combinables, asegurando que las invariantes del dominio se mantengan de manera declarativa y reutilizable.
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.

Query Object Pattern

Características

- patrón de diseño que se utilizan para abstraer y encapsular criterios de búsqueda o reglas de negocio

Pros

- **Abstracción de Consultas**: Permite encapsular consultas complejas en objetos específicos, lo que facilita su reutilización y mantenimiento.
- **Separación de Responsabilidades**: Ayuda a mantener una separación clara entre la lógica de negocio y la lógica de consulta, lo que mejora la modularidad y la legibilidad del código.
- **Composición de Consultas**: Los objetos de consulta pueden ser combinados y encadenados fácilmente para construir consultas más elaboradas y flexibles.
- **Facilita la Prueba Unitaria**: Al encapsular consultas en objetos específicos, se facilita la prueba unitaria de la lógica de consulta sin necesidad de depender de una conexión a la base de datos.

Contras

- **Complejidad Adicional**: Introducir una capa adicional de abstracción puede aumentar la complejidad del código, especialmente en aplicaciones pequeñas o simples donde la implementación del patrón puede ser excesiva.
- **Posible Sobrecarga de Abstracción**: Dependiendo de la implementación, el patrón Query Object puede introducir una sobrecarga de abstracción si no se aplica correctamente, lo que puede dificultar la comprensión del código.
- **Rendimiento Potencialmente Menor**: En sistemas con grandes volúmenes de datos o consultas complejas, el uso del patrón Query Object puede tener un impacto en el rendimiento debido a la necesidad de instanciar objetos adicionales y realizar operaciones de mapeo.
- **Dificultad para Expresar Algunas Consultas**: Algunas consultas complejas pueden ser difíciles de expresar utilizando el patrón Query Object, especialmente si implican operaciones que van más allá de la simple selección y filtrado de datos.

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	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**: Útil para encapsular consultas específicas dentro de microservicios que manejan datos, permitiendo que cada servicio tenga consultas específicas según sus necesidades. • **Clean Architecture**: Encajaría en la capa de infraestructura, proporcionando una forma limpia y centralizada de construir consultas. • **Domain Centric**: Puede ser usado para consultas específicas a repositorios dentro del dominio, facilitando la construcción de consultas complejas sin mezclar lógica de negocio.
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	KO	La capa de abstracción de consultas no es lo suficientemente genérica para poder ser reutilizable en otros ámbitos, ya que finalmente está acoplado a la consulta y el dominio concreto para el que se crea

Native Queries

Pros:

1. **Flexibilidad Completa**: Las consultas nativas permiten escribir consultas SQL complejas y específicas que pueden aprovechar todas las características y funciones del motor de base de datos subyacente.
2. **Optimización de Rendimiento**: En algunos casos, las consultas nativas pueden ofrecer un mejor rendimiento al permitir optimizaciones específicas de la base de datos, como el uso de índices o características específicas del motor de base de datos.
3. **Migración sin Problemas**: Si ya tienes consultas SQL existentes que deseas utilizar en tu aplicación, las consultas nativas te permiten reutilizar fácilmente ese código SQL sin necesidad de traducirlo a otro formato.

Contras:

1. **Portabilidad Limitada**: Al escribir consultas directamente en SQL nativo, tu aplicación se vuelve menos portátil entre diferentes sistemas de bases de datos. Esto puede complicar la migración a una base de datos diferente en el futuro si es necesario.
2. **Mayor Exposición a Vulnerabilidades**: Las consultas nativas pueden exponer tu aplicación a vulnerabilidades de seguridad como inyecciones SQL si no se manejan adecuadamente los parámetros y la sanitización de datos.
3. **Menor Mantenibilidad**: Las consultas nativas pueden ser más difíciles de mantener y actualizar, ya que el código SQL se mezcla directamente con el código de la aplicación, lo que dificulta la comprensión y la depuración.

Criterio	OK/KO	Notas
Estándar	KO	No es un patrón ni algo estandarizable ya que es una query definida para un caso de uso concreto que cubra una necesidad de un negocio concreto
Sostenible	KO	Al ser una query que se define en un string, no se puede validar en tiempo de compilación lo que dificulta su mantenimiento y evolutivo en caso de error. Además al estar muy ligado a la tecnología del acceso al dato, cualquier cambio al respecto impacta de forma directa en la consulta
Arquitectura de Referencia	OK	• **Microservicios**: Aporta Optimización en las consultas cuando • **Clean Architecture**: Al estar aislado en la capa de acceso al dato no produce ninguna incompatibilidad con la Arquitecturas Limpias • **Domain Centric**: Al estar aislado en la capa de acceso al dato, siempre que esté implementado dentro del patrón repositorio (como una operación específica del negocio) , no interfiere con la metodología de diseño domain centric

Desacoplado	KO	Al estar aislado en la capa de acceso al dato, siempre que esté implementado dentro del patrón repositorio (como una operación específica del negocio) , estará desacoplado de las diferentes capas Se produce un **acoplamiento a la tecnología** del acceso al dato, cualquier cambio al respecto impacta de forma directa en la consulta.
Reutilizable	KO	Al ser una query definida para un caso de uso concreto que cubra una necesidad de un negocio concreto, no se puede reutilizar en otros negocios solo tiene cabida en aquél para el que fue definido

Named Queries

Pros:

1. **Abstracción de la Capa de Datos**: Las consultas con nombre proporcionan una abstracción de la capa de datos al definir las consultas en un lugar centralizado, lo que facilita la gestión y la actualización de las consultas.
2. **Mejor Portabilidad**: Al utilizar consultas con nombre, puedes escribir consultas en un lenguaje de consulta independiente de la base de datos, lo que hace que tu aplicación sea más portátil entre diferentes sistemas de bases de datos.
3. **Seguridad Mejorada**: Las consultas con nombre pueden proporcionar un mecanismo para prevenir la inyección SQL al utilizar parámetros vinculados y otras técnicas de prevención de seguridad.

Contras:

1. **Limitaciones de Funcionalidad**: Las consultas con nombre pueden no admitir todas las características y funciones avanzadas que ofrece el lenguaje de consulta SQL nativo. Esto puede limitar la flexibilidad al escribir consultas complejas.
2. **Rendimiento Potencialmente Menor**: En algunos casos, las consultas con nombre pueden resultar en un rendimiento ligeramente inferior en comparación con las consultas nativas, especialmente para consultas complejas que requieren optimizaciones específicas de la base de datos.
3. **Menos Flexibilidad**: Aunque proporcionan una capa de abstracción, las consultas con nombre pueden ser menos flexibles para adaptarse a requisitos específicos o casos de uso únicos que pueden requerir consultas SQL personalizadas.

Criterio	OK/KO	Notas
Estándar	KO	No es un patrón ni algo estandarizable ya que es una query definida para un caso de uso concreto que cubra una necesidad de un negocio concreto
Sostenible	KO	Al ser una query que se define en un string, no se puede validar en tiempo de compilación lo que dificulta su mantenimiento y evolutivo en caso de error. Con respecto a las native queries ofrece una capa de aislamiento tecnológico lo que aumentaría su sostenibilidad.

Arquitectura de Referencia	OK	• **Microservicios**: Aporta Optimización en las consultas cuando • **Clean Architecture**: Al estar aislado en la capa de acceso al dato no produce ninguna incompatibilidad con la Arquitecturas Limpias • **Domain Centric**: Al estar aislado en la capa de acceso al dato, siempre que esté implementado dentro del patrón repositorio (como una operación específica del negocio) , no interfiere con la metodología de diseño domain centric
Desacoplado	OK	Al estar aislado en la capa de acceso al dato, siempre que esté implementado dentro del patrón repositorio (como una operación específica del negocio) , estará desacoplado de las diferentes capas Ofrece una capa de aislamiento tecnológico lo que se desacoplaría de la tecnología de acceso al dato
Reutilizable	KO	Al ser una query definida para un caso de uso concreto que cubra una necesidad de un negocio concreto, no se puede reutilizar en otros negocios solo tiene cabida en aquél para el que fue definido

Decisión Tomada

Después del análisis realizado se opta por el **patrón especificación**:

- 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
- 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

Es **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 espacio de Gobernanza y calidad