Documento de investigación - Grupo 8

Integrantes:
Gerardo Hidalgo Carroll
Santiago Vásquez Sánchez

Profesor:
ANDRES MARTIN SANCHEZ
Asignatura Arquitectura De Software

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA DE SISTEMAS
BOGOTÁ, D.C.
Marzo 18
2024

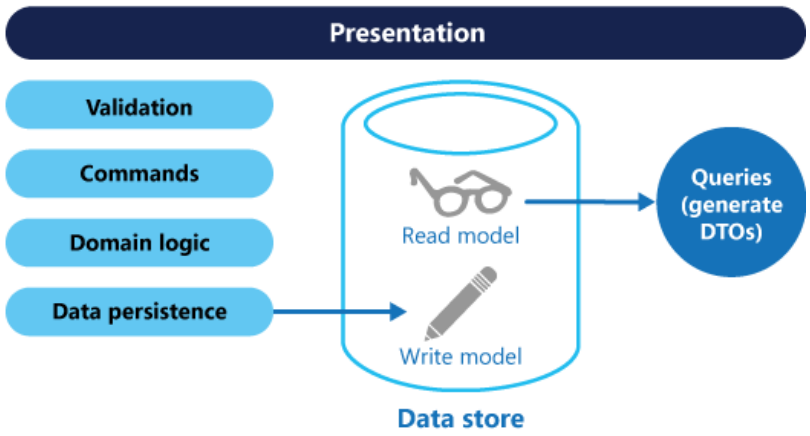
Temas

- CQRS / Blackboard
- SVELTE
- Rest / XML
- Ruby on rails
- Cassandra

Qué es:

- **CQRS:** Command Query Responsibility Segregation, es un patrón de arquitectura de software que separa las responsabilidades de lectura y escritura de datos en una aplicación.
- **Blackboard:** El patrón Blackboard, también conocido como patrón Pizarra, es una técnica usada en la arquitectura de software para resolver problemas complejos donde no se conoce una solución determinista (paso a paso).
- **Rest / XML:** Un servicio REST/XML es un tipo de servicio web que utiliza la arquitectura RESTful y el formato de datos XML para intercambiar información entre aplicaciones.
- **Ruby on rails:** Es un marco de trabajo de código abierto para el desarrollo de aplicaciones web del lado del servidor. Está escrito en el lenguaje de programación Ruby y sigue el patrón del Modelo Vista Controlador (MVC).
- **Cassandra:** Es un sistema de gestión de bases de datos (SGBD) de código abierto, distribuido y diseñado para manejar grandes cantidades de datos con alta disponibilidad.

CQRS



Historia y evolución:

- 2002: Greg Young introduce el concepto de CQRS en un artículo titulado "CQRS: Command-Query Responsibility Segregation".

- 2005: Udi Dahan publica un artículo titulado "CQRS: The Event Sourcing Alternative". En este artículo, Dahan describe cómo CQRS se puede utilizar junto con el patrón Event Sourcing para crear aplicaciones más escalables y confiables.
- 2010: CQRS comienza a ganar popularidad entre la comunidad de desarrollo de software.
- 2014: Se publica el libro "CQRS in Action" de Jimmy Bogard. Este libro proporciona una guía completa para implementar CQRS en aplicaciones del mundo real.

Ventajas y desventajas

- Permite escalar las operaciones de lectura y escritura de forma independiente.
- Reduce la contención de bloqueos y optimiza el acceso a los datos.
- Facilita el desarrollo y mantenimiento del código.
- Permite aplicar diferentes niveles de seguridad a las operaciones de lectura y escritura.
- Facilita la evolución de la aplicación y la implementación de nuevas funcionalidades.

Desventajas

- Puede ser complejo de implementar y comprender.
- Requiere una inversión en infraestructura y desarrollo.
- Puede haber un desfase temporal entre las lecturas y las escrituras.
- Puede ser difícil identificar la causa de errores en aplicaciones CQRS.

Casos de aplicación

- Twitter: Utiliza CQRS para separar las operaciones de lectura y escritura en su base de datos. Esto permite a Twitter escalar su aplicación para manejar millones de usuarios y tweets.
- Facebook: Utiliza CQRS para separar las operaciones de lectura y escritura en su base de datos de mensajes. Esto permite a Facebook escalar su aplicación para manejar miles de millones de mensajes por día.
- Amazon: Utiliza CQRS para separar las operaciones de lectura y escritura en su base de datos de productos. Esto permite a Amazon escalar su aplicación para manejar millones de productos y pedidos.
- eBay: Utiliza CQRS para separar las operaciones de lectura y escritura en su base de datos de subastas. Esto permite a eBay escalar su aplicación para manejar millones de subastas por día.
- Netflix: Utiliza CQRS para separar las operaciones de lectura y escritura en su base de datos de recomendaciones. Esto permite a Netflix escalar su aplicación para manejar millones de usuarios y recomendaciones.
- Google Analytics: Utiliza CQRS para separar las operaciones de lectura y escritura en su base de datos de análisis. Esto permite a Google Analytics escalar su aplicación para manejar miles de millones de eventos por día.
- Microsoft Azure: Utiliza CQRS para implementar su servicio de Event Hubs.

- Salesforce: Utiliza CQRS para implementar su servicio de Salesforce Wave Analytics.

Principios SOLID
Ejemplo

SRP:

- Crear una clase CommandHandler separada para cada tipo de comando.
- Crear una clase QueryHandler separada para cada tipo de consulta.

OCP:

- Definir interfaces para los CommandHandlers y QueryHandlers.
- Implementar estas interfaces en clases concretas.

LSP:

- Asegurarse de que las subclases de CommandHandlers y QueryHandlers se comporten de la misma manera que sus superclases.

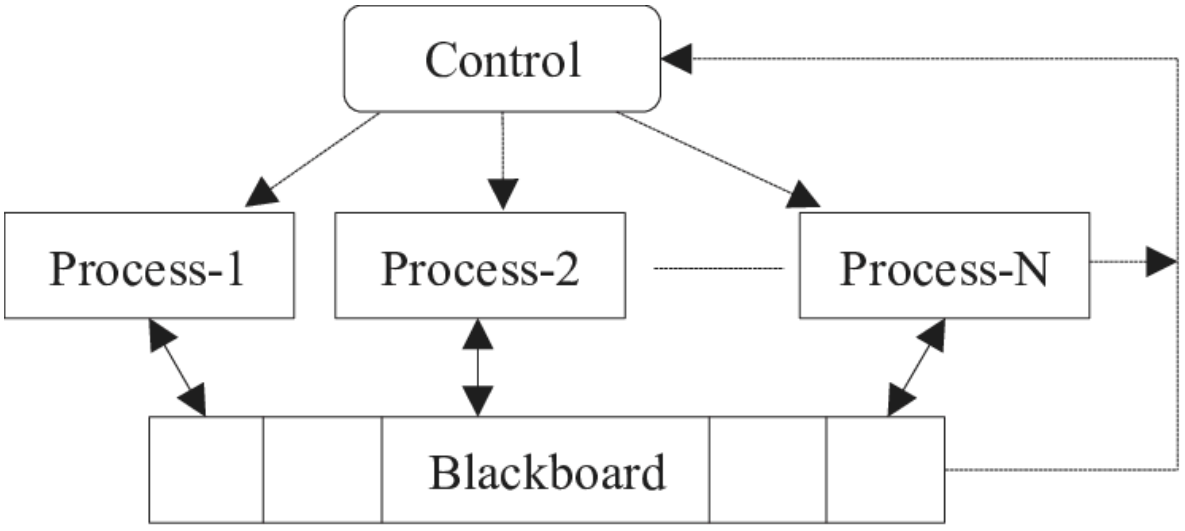
ISP:

- Definir interfaces separadas para las operaciones de lectura y escritura.
- Implementar estas interfaces en clases concretas.

DIP:

- Inyectar las dependencias necesarias en los CommandHandlers y QueryHandlers.

Blackboard



Historia y evolución

- Años 80: Se introdujeron nuevos mecanismos de control para mejorar la eficiencia y la coordinación de los módulos de conocimiento.
- Años 90: Se comenzó a utilizar el patrón Blackboard en otras áreas como la planificación y el diagnóstico de fallos.
- Años 2000: Se desarrolló la arquitectura Blackboard de próxima generación (Next Generation Blackboard - NGBB), que incorpora técnicas de inteligencia artificial más modernas como las redes neuronales y la computación evolutiva.
- El patrón Blackboard sigue siendo un importante paradigma en la arquitectura de software para sistemas de inteligencia artificial.

Ventajas y desventajas:

Ventajas

- El patrón Blackboard es muy flexible y puede ser adaptado a una amplia variedad de problemas.
- La arquitectura modular del patrón Blackboard facilita el desarrollo y mantenimiento del software.
- El patrón Blackboard puede ser escalado para manejar grandes cantidades de datos y tareas complejas.
- Los módulos de conocimiento del patrón Blackboard pueden ser reutilizados en diferentes aplicaciones.
- El patrón Blackboard es tolerante a fallos en los módulos de conocimiento.
- El patrón Blackboard facilita la explicación de las decisiones tomadas por el sistema.

Desventajas

- El patrón Blackboard puede ser complejo de implementar y entender.
- La eficiencia del patrón Blackboard puede verse afectada por la cantidad de módulos de conocimiento y la complejidad de las interacciones entre ellos.
- El mantenimiento del patrón Blackboard puede ser complejo debido a la gran cantidad de módulos de conocimiento y las interacciones entre ellos.
- El desarrollo e implementación del patrón Blackboard puede ser costoso.
- Se requiere un conocimiento profundo del problema para desarrollar e implementar el patrón Blackboard de manera efectiva.

Casos de aplicación:

Reconocimiento de voz, diagnóstico de fallos, planificación, análisis de datos y traducción automática han empleado con éxito el patrón Blackboard. Este enfoque permite que diversos módulos de conocimiento aborden aspectos específicos de cada tarea, como la extracción de características, el diagnóstico de fallos, la planificación de recursos, el análisis de datos y la traducción lingüística.

Principios SOLID

- SRP : Cada módulo de conocimiento en el Blackboard debe tener una sola responsabilidad definida para simplificar su entendimiento y mantenimiento.
- OCP : El Blackboard debe ser abierto para la extensión, permitiendo la adición de nuevos módulos sin modificar los existentes, adaptándose a nuevas necesidades.
- LSP: Los módulos de conocimiento deben ser sustituibles por subtipos que sigan el mismo contrato, mejorando la flexibilidad del sistema.
- ISP : Definir interfaces pequeñas y bien definidas para la comunicación entre módulos, reduciendo el acoplamiento y facilitando el mantenimiento.
- DIP : Los módulos no deben depender directamente de otros módulos, sino de abstracciones, lo que facilita las pruebas y el mantenimiento del sistema.

SVELTE

Historia y evolución

- Svelte nació en 2016 de la mente de Rich Harris, un desarrollador web familiarizado con React y Vue. La complejidad y tamaño de estos frameworks lo motivaron a crear uno más ligero y accesible.
- La primera versión de Svelte debutó en 2016, siendo rudimentaria pero ya destacando por su enfoque sin DOM virtual y su sintaxis intuitiva.
- A lo largo de los años, Svelte ha crecido y evolucionado rápidamente. Nuevas versiones han traído consigo funcionalidades mejoradas y un rendimiento optimizado. Asimismo, una comunidad activa ha contribuido al desarrollo del framework, generando una amplia gama de bibliotecas y herramientas.

Ventajas y desventajas

Ventajas

- Rápido y eficiente: Sin DOM virtual, aplicaciones Svelte son rápidas incluso en dispositivos con ancho de banda limitado.
- Fácil de aprender y usar: Sintaxis intuitiva similar a HTML, CSS y JavaScript.
- Flexible: Adaptable a diferentes arquitecturas.
- Comunidad activa: Soporte y recursos disponibles.
- Tamaño pequeño: Sin dependencias externas, código final reducido.
- Componentes reactivos: Sistema de componentes para interfaces de usuario reutilizables.

Desventajas

- Menos popular: Relativamente nuevo, menos recursos disponibles.
- Ecosistema limitado: Menos bibliotecas y herramientas.
- Curva de aprendizaje: Aunque fácil, requiere cierto tiempo.
- Soporte de navegadores: Compatibilidad en mejora constante.
- Depuración más desafiante: Comparado con otros frameworks.
- Falta de algunas funcionalidades comunes: Como enrutamiento y gestión de estado.

Casos de aplicación

Grandes empresas como The New York Times, GitHub, Spotify, IBM, Vercel, Khan Academy, Auth0 y DigitalOcean utilizan Svelte para crear componentes web, dashboards, interfaces de usuario y aplicaciones internas. Svelte les ha permitido mejorar el rendimiento, crear interfaces interactivas y receptivas, desarrollar aplicaciones seguras y confiables, y ofrecer una experiencia de usuario fluida y moderna.

Principios SOLID

- SRP: Svelte fomenta componentes con una única tarea para facilitar su reutilización y comprensión.
- OCP: Svelte permite la extensibilidad de componentes a través de slots y props, adaptándose a diferentes necesidades sin modificar su código fuente.
- LSP: Los componentes de Svelte pueden extenderse y reutilizarse, permitiendo una organización jerárquica y la reutilización de funcionalidades.
- ISP: Las props actúan como interfaces en Svelte, permitiendo una comunicación clara entre componentes y la definición de interfaces claras.
- DIP : Aunque Svelte no tiene inyección de dependencias integrada, es posible implementarla con bibliotecas de terceros para un código desacoplado y testable.

REST/XML

Historia y evolución

- 2000: Roy Fielding publica su tesis doctoral sobre la arquitectura REST.
- 2001: Se publica la primera especificación de SOAP, un protocolo de mensajería para servicios web que utiliza XML.
- 2002: Se publica la especificación WSDL, que define un lenguaje para describir servicios web.
- 2004: Se publica la especificación RESTful Web Services, que establece prácticas óptimas para el desarrollo de APIs REST.
- 2007: Se publica la especificación Atom, un formato de datos para feeds web.
- 2010: Se publica la especificación JSON, un formato de datos ligero y fácil de leer.

Ventajas y desventajas

Ventajas

- REST permite adaptarse a distintas necesidades.
- Los servicios REST/XML pueden ampliarse fácilmente.
- Son compatibles con diversas aplicaciones y plataformas.
- Son relativamente simples de implementar y entender.
- XML garantiza un intercambio de datos seguro.
- Permite verificar la precisión de los datos.

Desventajas

- XML puede generar mensajes extensos.
- Su uso puede resultar complicado para algunos.
- Requiere software específico para manipularlo.
- Puede ser menos eficiente que otros formatos, como JSON.

Casos de aplicación

- APIs de comercio electrónico: Permiten a los usuarios comprar productos en línea.
- APIs de redes sociales: Permiten a los usuarios interactuar entre sí.
- APIs de mapas: Permiten a los usuarios obtener información sobre mapas y direcciones.
- APIs de noticias: Permiten a los usuarios obtener información sobre noticias y eventos actuales.
- APIs meteorológicas: Permiten a los usuarios obtener información sobre el tiempo.
- Amazon Web Services: Ofrece APIs REST/XML para servicios en la nube.
- Google Maps Platform: Proporciona una API REST/XML para información geográfica.
- Facebook Graph API: Ofrece una API REST/XML para datos de redes sociales.

Ruby on rails

Historia y evolución

En 1993, Yukihiro Matsumoto crea Ruby, un lenguaje simple y orientado a objetos. En 2003, David Heinemeier Hansson crea Rails, un framework ágil. Desde su lanzamiento en 2004, Rails gana popularidad. A lo largo de los años, se lanzaron varias versiones con mejoras, como la 1.0 en 2005 y la 2.0 en 2006. Rails se consolida como el framework líder en aplicaciones web y ha inspirado otros como Django y Laravel. Las versiones posteriores, como la 5.0 en 2016 y la 6.0 en 2018, continúan mejorando el rendimiento y la seguridad. Se espera que Rails siga evolucionando y manteniendo su posición destacada en el desarrollo web.

Ventajas y desventajas

Ventajas

- Rails permite desarrollar código de manera rápida y con menos errores.
- Su sintaxis clara y legible facilita la comprensión y escritura del código.
- Sigue un enfoque de "convención sobre configuración", reduciendo la cantidad de código necesario.
- Cuenta con una gran comunidad de desarrolladores dispuestos a ayudar.
- Es capaz de crecer para soportar grandes volúmenes de tráfico y usuarios.

- Proporciona herramientas integradas para pruebas, depuración y generación de código.
- Se actualiza periódicamente para abordar vulnerabilidades de seguridad.

Desventajas

- Aunque es relativamente fácil de aprender, requiere tiempo para dominarlo.
- Puede ser demasiado complejo para proyectos pequeños y simples.
- Adaptar aplicaciones Rails a necesidades específicas puede ser complicado.
- No siempre es el framework más veloz disponible.
- Requiere muchas gemas de terceros, lo que puede aumentar la complejidad y los riesgos de seguridad.
- Su tamaño considerable puede resultar en aplicaciones más pesadas.

Casos de aplicación

- Airbnb: El uso de Rails ha facilitado a Airbnb expandirse rápidamente y atender a una amplia base de usuarios.
- Twitter: Gracias a Rails, Twitter ha logrado gestionar eficazmente grandes volúmenes de tráfico y datos.
- Hulu: La implementación de Rails ha posibilitado que Hulu ofrezca una experiencia de usuario fluida y altamente personalizada.
- Shopify: El uso de Rails ha permitido a Shopify expandirse y brindar soporte a numerosas tiendas en línea.
- Basecamp: Gracias a Rails, Basecamp se ha convertido en una herramienta confiable y fácil de usar para equipos de todo el mundo.

Principios SOLID

- SRP : En Rails, cada clase o módulo debe tener una sola tarea definida, evitando acoplamientos con funciones no relacionadas.
- OCP: Las entidades en Rails deben ser extensibles pero no modificables, permitiendo añadir funcionalidades sin alterar el código existente.
- LSP: Los subtipos en Rails deben ser intercambiables con sus tipos base sin errores.
- ISP : Las interfaces en Rails deben ser pequeñas y específicas, evitando interfaces grandes con múltiples métodos.
- DIP: Los módulos de alto nivel no deben depender directamente de los de bajo nivel, sino de abstracciones.

Cassandra

Historia y evolución

- Cassandra fue inicialmente desarrollado por Facebook en 2008 para mejorar la búsqueda en la bandeja de entrada.
- Lanzado como proyecto de código abierto en el mismo año. En 2010, se convirtió en un proyecto de la Fundación Apache.
- Durante los años 2008-2010, el enfoque principal estuvo en la escalabilidad y la disponibilidad del sistema.
- Entre 2011 y 2013, se introdujeron nuevas funcionalidades como la replicación de datos y el soporte para diversos tipos de datos.
- Desde 2014 hasta la actualidad, el desarrollo de Cassandra se ha centrado en mejorar el rendimiento, la seguridad y la usabilidad del sistema.

Ventajas y desventajas

Ventajas

- Cassandra puede distribuirse en múltiples servidores, permitiendo el manejo eficiente de grandes volúmenes de datos con una alta disponibilidad.
- Utiliza un modelo de datos tipo clave-valor, similar a una hoja de cálculo, con estructuras de filas, columnas y valores, lo que brinda versatilidad en el almacenamiento.
- Aunque proporciona una sólida consistencia a nivel de fila, la consistencia entre filas puede ser eventual, adecuada para aplicaciones donde la disponibilidad y el rendimiento superan la necesidad de consistencia estricta.
- Como proyecto de código abierto, Cassandra es gratuito para su uso y modificación, brindando acceso libre a su código fuente.

Desventajas

- La consistencia eventual puede no ser adecuada para todos los casos de uso.

- Configuración y administración pueden ser complejas.
- No está diseñada para manejar consultas complejas utilizadas en el análisis de datos.
- Puede resultar difícil de aprender y utilizar, especialmente para aquellos sin experiencia previa en bases de datos NoSQL.

Casos de aplicación

- Netflix: Cassandra se utiliza para almacenar datos de visualización de usuarios, recomendaciones de películas y metadatos de contenido.
- Spotify: Cassandra se utiliza para almacenar datos de escucha de usuarios, metadatos de música y datos de artistas.
- Twitter: Cassandra se utiliza para almacenar tweets, datos de usuarios y relaciones entre usuarios.
- Instagram: Cassandra se utiliza para almacenar fotos, videos, datos de usuarios y relaciones entre usuarios.
- Uber: Cassandra se utiliza para almacenar datos de viajes, datos de conductores y datos de pasajeros.

Principios SOLID

- SRP: Cassandra se centra en almacenar y recuperar datos, siguiendo el principio de responsabilidad única.
- OCP: Si bien es de código abierto y puede ser extendido por la comunidad, el núcleo central de Cassandra es relativamente cerrado y difícil de modificar.
- LSP: Cassandra no implementa completamente este principio debido a inconsistencias en su modelo de datos, lo que dificulta la sustitución de implementaciones.
- ISP: Cassandra ofrece una única interfaz para acceder a los datos, lo cual puede no ser adecuado para todas las aplicaciones, incumpliendo parcialmente este principio.
- DIP: Cassandra utiliza interfaces para permitir la inyección de dependencias, aumentando la flexibilidad del código.

Atributos de calidad

Tema	Atributos de calidad
CQRS	Escalabilidad, Rendimiento, Seguridad
BLACKBOARD	Escalabilidad, Eficiencia, Tolerancia a fallos
SVELTE	Escalabilidad, Seguridad, Accesibilidad
REST/XML	Escalabilidad, Seguridad, Confiabilidad, Interoperabilidad, Mantenibilidad
RUBY ON RAILS	Escalabilidad, Eficiencia, Seguridad, Portabilidad
CASSANDRA	Escalabilidad, Disponibilidad, Rendimiento, Seguridad

Referencias:

- https://docs.aws.amazon.com/es_es/prescriptive-guidance/latest/modernization-data-persistence/cqrs-pattern.html
- https://aprendearquitecturasoftware.wordpress.com/2018/10/03/grupo-3-patron-de-arquitectura-blackboard-pizarra/
- https://www.youtube.com/watch?v=DJCWpUVf5E0
- https://www.w3schools.com/xml/
- https://platzi.com/blog/svelte-un-nuevo-framework-para-crear-aplicaciones-web-mejoradas/
- https://www.toptal.com/ruby-on-rails/tips-and-practices
- https://cassandra.apache.org/
-