JBOSS-EAP-EXAMPLE

DATOS GENERALES

Identificación del documento	**ADR-001_ARQUITECTURA**			
Título del documento	**Arquitectura**			
Proyecto	JBOSS-EAP-EXAMPLE			
Gerente				
Área demandante				
Director				
Responsables	**Área**		**Responsable**	
	Arquitectura			
Control de versionado	**Versión**	**Fecha**	**Autor**	**Comentarios**
	1.0.0	11/10/2025	Federico Martín Lara	Versión inicial

Contenido

Introducción

Necesitamos establecer un patrón de arquitectura fundamental para JBOSS-EAP-EXAMPLE. Esta decisión es crítica, ya que definirá cómo se estructura el código, cómo interactúan sus componentes y cómo evolucionará el sistema. El patrón elegido debe el desarrollo, facilitar la mantenibilidad a largo plazo y permitirnos ser independientes de la tecnología externa, considerando que tenemos múltiples puntos de entrada y salida. Como prioridad, debe permitir probar e ilustrar acerca de las distintas funcionalidades de la plataforma donde se va a ejecutar este proyecto, ya que la misión de dicho proyecto es mostrar y servir de aprendizaje de la plataforma JBOSS EAP.

Documentos de referencia

ADR-000_DOC_GENERAL

Opciones consideradas

1. Arquitectura en capas

- **Descripción:** El enfoque tradicional que organiza el código en capas horizontales (Presentación, Lógica de Negocio, Acceso a Datos). La comunicación es estrictamente jerárquica y hacia abajo.
- **Ventajas:** Es un patrón simple, muy conocido y rápido de implementar para proyectos sencillos.
- **Inconvenientes:** Genera un fuerte acoplamiento de la lógica de negocio con la capa de datos. Dificulta las pruebas unitarias aisladas del dominio y hace que cambiar tecnologías (como la base de datos) sea una tarea compleja y arriesgada.

2. Arquitectura hexagonal (Puertos y Adaptadores)

- **Descripción:** Aísla el núcleo de la aplicación (el dominio con la lógica de negocio) del mundo exterior. El núcleo define "puertos" (interfaces) para la comunicación, y las tecnologías externas se "enchufan" a través de "adaptadores" que implementan esos puertos.
- **Ventajas:** La lógica de negocio es totalmente independiente de la infraestructura, lo que permite una testeabilidad completa y aislada (ideal para TDD). Facilita enormemente el cambio o la adición de tecnologías (ej: añadir un nuevo tipo de consumidor de eventos) sin tocar el dominio.
- **Inconvenientes:** Requiere más disciplina y puede introducir más clases/interfaces (boilerplate) al principio.

3. CQRS (Command Query Responsability Segregation)

- **Descripción:** Un patrón que separa completamente el modelo utilizado para escribir

datos (Comandos) del modelo utilizado para leerlos (Consultas).

- **Ventajas:** Permite optimizar de forma independiente las escrituras y las lecturas, lo cual es muy útil en un leaderboard (muchas escrituras de puntuaciones, muchas lecturas del ranking). Puede mejorar drásticamente el rendimiento y la escalabilidad.
- **Inconvenientes:** Aumenta significativamente la complejidad del sistema, ya que a menudo requiere gestionar dos modelos de datos y su sincronización (consistencia eventual).

Decisión

Se ha decidido adoptar la **Arquitectura en capas** como el patrón principal para el proyecto.

Justificación: Esta arquitectura se alinea perfectamente con nuestros objetivos principales. Nos permite **aislar la lógica de negocio**. Esto es fundamental para poder aplicar **TDD** de manera efectiva sobre el dominio.

Se descarta CQRS en esta fase inicial. Aunque el patrón encaja bien con el problema, la complejidad que introduce se considera excesiva para el MVP. La Arquitectura Hexagonal no impide que, en el futuro, si los requisitos de rendimiento lo exigen, se pueda implementar CQRS dentro de la estructura ya existente.

Consecuencias

- La estructura del proyecto se organizará en torno a uno o varios módulos de presentación y un módulo de negocio.
- El desarrollo inicial puede ser ligeramente más lento debido a la necesidad de establecer los distintos módulos del proyecto.
- A largo plazo, se espera que la mantenibilidad sea mayor y que la adición de nuevas funcionalidades o tecnologías sea más sencilla y segura.