Tema 3. CQRS en sistemas reales

Separando lectura y escritura en nuestro caso práctico

Recap: de dónde venimos

Hemos visto:

- DDD → modelo del dominio y reglas
- Hexagonal → capas, puertos y adaptadores

- **Pregunta clave:**
  ¿Cómo organizamos los casos de uso cuando el sistema crece?

Estado actual tras el Tema 2 (Hexagonal)

- Arquitectura hexagonal funcionando
- Dominio modelado
- Casos de uso que mezclan:
  - lectura
  - validación
  - modificación de estado
- Un mismo modelo para leer y escribir

El problema del CRUD “plano”

- Lecturas cada vez más complejas
- Escrituras con reglas de negocio
- Un mismo servicio hace de todo
- Cambios en lectura rompen escritura (y viceversa)

Contexto real: nuestra aplicacion

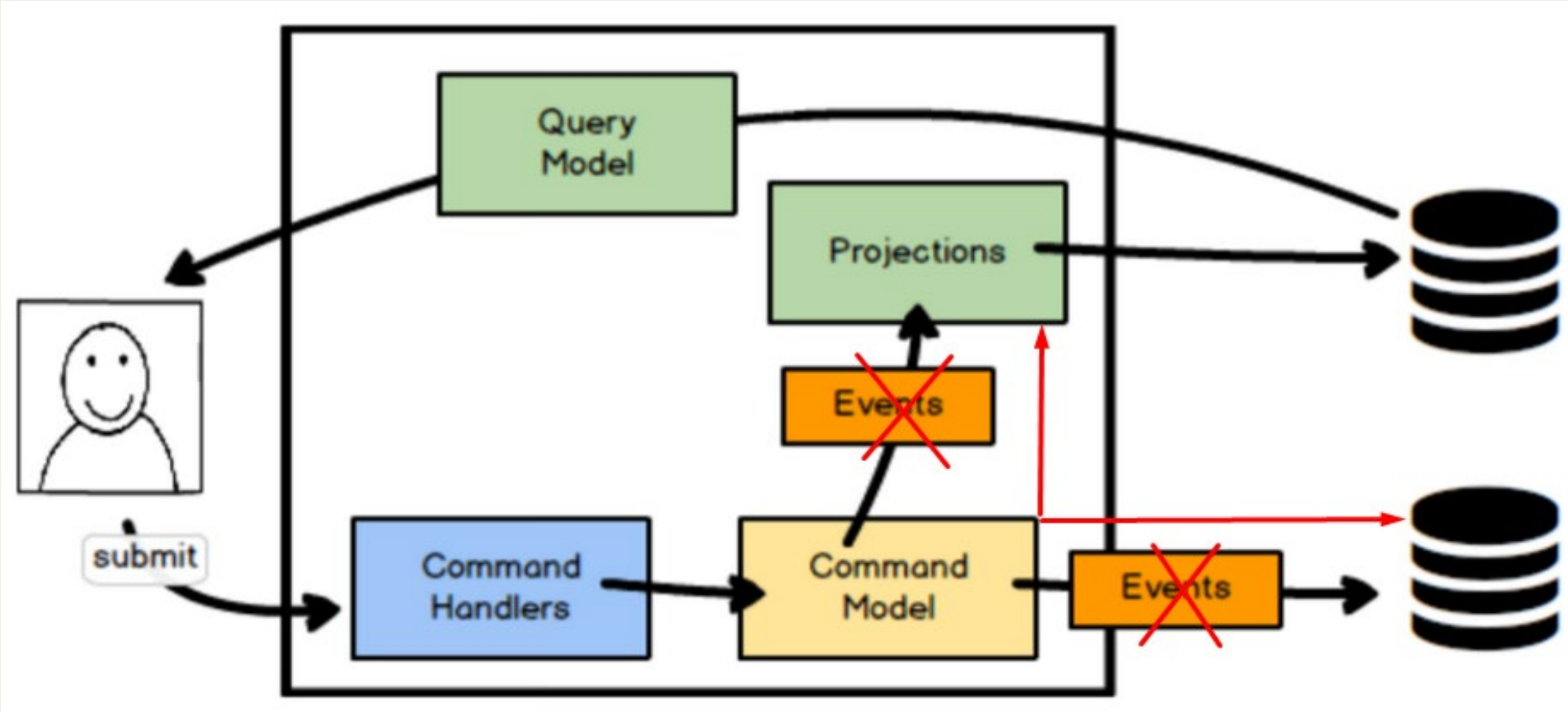
- Sistemas legacy / backoffice como **fuente de verdad**
- Frontoffice como capa de servicio
- Lecturas desde:
  - Vistas/proyecciones del backoffice
  - tablas propias del frontoffice
- Escrituras se materializan en el backoffice
- Necesidad de **proyecciones de lectura**
- **¡¡En la práctica ya son distintas las lecturas y escrituras!!**

Problemas del enfoque sin CQRS

- Casos de uso demasiado grandes
- Modelos forzados a servir para:
  - lectura
  - escritura
- Dificultad para optimizar lecturas
- Acoplamiento UI ↔ modelo interno
- Fricción con legacy como fuente de verdad

Qué es CQRS

- Separar:
  - **Commands** → cambian el estado
  - **Queries** → obtienen información
- Separar:
  - modelos
  - casos de uso
- CQRS como **refactor estructural**
- ~~No entramos aún en comunicación asíncrona~~


Query
Model
Projections
Events
submit
Command
Handlers
Command
Model
Events

CQRS y nuestro contexto legacy

- Escrituras:
  - orientadas al backoffice
  - alineadas con reglas del dominio
- Lecturas:
  - orientadas a la UI
  - combinan datos del backoffice y frontoffice
- Necesidad de **modelos de lectura y escritura distintos**

Refactor de los casos de uso con CQRS

- **Antes (Tema 2):**
  - GestionarXUseCase
- **Después (Tema 3):**
  - CrearXCommandHandler
  - ActualizarXCommandHandler
  - CancelarXCommandHandler
  - ObtenerXQueryHandler
  - ListarXQueryHandler

Commands

- Representan **intenciones de cambio del usuario**
- Cambian el estado del dominio
- No están pensados para construir vistas
- Modelo enfocado a:
  - validar reglas
  - Ejecutar reglas de negocio (a través del dominio)
  - expresar operaciones de negocio
- Command contiene los datos y filtros, el handler la funcionalidad

Queries

- Representan **necesidades de información**
- No modifican estado
- Modelos (optimizados) para lectura
- Pueden combinar/componer distintas fuentes de datos
- Dominio como traductor de significado (No como motor de cambio)
- Query contiene los datos y filtros, el handler la funcionalidad

CQRS dentro de la arquitectura hexagonal

- Adaptadores REST → Commands / Queries
- Capa de aplicación:
  - CommandHandlers
  - QueryHandlers
- Dominio:
  - utilizado por ambos
- Repositorios:
  - conceptualmente separados para lectura y escritura

Estructura actual

es.um.atica.[GROUP].[CONTEXT].application.usercase.[CASOS]

↓

es.um.atica.[GROUP].[CONTEXT].application.usercase.[CASO_CONCRETO]

Por ejemplo:

es.um.atica.umufly.vuelos.application.usercase.vuelos

↓

es.um.atica.umufly.vuelos.application.usercase.obtenervuelos

es.um.atica.umufly.vuelos.application.usercase.listarvuelos

Beneficios reales que obtenemos

- Casos de uso más pequeños
- Modelos más expresivos
- Lecturas optimizadas para consumo
- Escrituras alineadas con reglas de negocio
- Mejor encaje con legacy como fuente de verdad

Errores comunes al aplicar CQRS

- Usar el mismo DTO para Command y Query
- Reutilizar el mismo modelo para todo
- Introducir CQRS sin una necesidad real
- Pensar que CQRS es duplicar bases de datos
- Controllers con lógica de negocio

CQRS + DDD + Hexagonal (visión global)

- **DDD** define el modelo del dominio
- **Arquitectura Hexagonal** protege ese dominio
- **CQRS** estructura los casos de uso

Mensaje clave:
Tres piezas, un mismo objetivo: mantener la lógica de negocio clara, protegida y evolutiva.

RECORDEMOS: CQRS en nuestra arquitectura (visión de alto nivel)

REST
↓
Command / Query
↓
Application (Handlers)
↓
Domain (reglas de negocio)
↓
Legacy / Backoffice (fuente de verdad)
↓
Proyecciones FrontOffice (lecturas)

- Lecturas desde proyecciones y datos auxiliares
- Escrituras alineadas con backoffice

Puente hacia el Tema 4

- CQRS separa responsabilidades
- Aún existe acoplamiento temporal
- Siguiente paso:
  - desacoplar reacciones
  - permitir que el sistema evolucione sin dependencias directas


UNIVERSIDAD DE MURCIA
1272

Apéndice 1: UMUBUS - ¿Qué es? ¿Para que la usamos?

- Librería interna para implementar:
  - CQRS (Commands / Queries / Handlers)
  - Arquitecturas basadas en mensajes (eventos)
- Estándar común en los proyectos backend de la universidad
- Nos permite:
  - Unificar cómo definimos casos de uso (Query/Command y Handlers)
  - ~~Desacoplar controladores de la lógica de aplicación (QueryBus/CommandBus)~~ -> En el siguiente tema

Apéndice 1: UMUBUS - UMUBUS + DDD + Hexagonal + CQRS

- **DDD** → modelo del dominio
- **Hexagonal** → puertos y adaptadores
- **CQRS** → separación de lectura/escritura
- **UMUBUS** → infraestructura para Commands, Queries y Handlers

Apéndice 1: UMUBUS - Componentes

- **Command**
  - Command.java, SyncCommand.java
- **Query**
  - Query.java
- **CommandHandler**
  - CommandHandler.java: Ejecuta un Command
- **QueryHandler**
  - QueryHandler.java: Ejecuta una Query
- **CommandBus / QueryBus** (Tema 4)
  - CommandBus.java, SyncCommandBus.java, QueryBus.java: Capa de despacho de Commands/Queries

Apéndice 1: UMUBUS - Flujo

- Controller REST recibe petición HTTP
- Se construye un Command o Query
- Se delega en el Handler (o Bus)
- El Handler, **orquesta** la operación:
  - Llama al dominio
  - Usa puertos de aplicación
- Se devuelve el resultado al controller

Apéndice 1: UMUBUS – Incluir la librería

- **Dependencia en pom.xml**

```
<repositories>
        <repository>
                <id>fundewebjs.archiva.atica.umu.es</id>
                <name>ATICA - UMU Repository - FundeWebJS</name>
                <url>https://archiva.um.es/archiva/repository/FundeWebJS/</url>
                <releases>
                        <enabled>true</enabled>
                </releases>
                <snapshots>
                        <enabled>true</enabled>
                </snapshots>
        </repository>
</repositories>
```

Apéndice 1: UMUBUS – Incluir la librería

- **Dependencia en pom.xml**

```
<!-- UMUBUS local (CQRS + eventos locales) -->
<dependency>
        <groupId>es.um.atica.fundewebjs</groupId>
        <artifactId>fundewebjs-umubus</artifactId>
        <version>1.0.15-SNAPSHOT</version>
</dependency>
```

Apéndice 1: UMUBUS – Clase Query (Usuario)

```
public class ObtenerUsuarioQuery extends Query<Optional<Usuario>> {
    private String usuarioId;
    //Definimos constructor privado
    public static ObtenerUsuarioQuery of(String usuarioId) {
        return new ObtenerUsuarioQuery(usuarioId);
    }
}
```

Apéndice 1: UMUBUS – Clase QueryHandler (Usuario)

```
@Component
public class ObtenerUsuarioQueryHandler implements
QueryHandler<Optional<Usuario>,ObtenerUsuarioQuery> {
   private UsuarioReadRepository usuariosRepository;
         //Definimos constructor
   @Override
   public Optional<Usuario> handle(ObtenerUsuarioQuery query) throws
Exception {
      return usuariosRepository.findUsuario(query.getUsuarioId());
   }
}
```

Apéndice 1: UMUBUS – Clase SyncCommand (Usuario)

```
public class CrearUsuarioCommand extends SyncCommand<Void> {
   private String id;
         //Definimos constructor privado
   public static CrearUsuarioCommand of(String id) {
      // Validate Command Data for UI
      return new CrearUsuarioCommand(id);
   }
}
```

Apéndice 1: UMUBUS – Clase SyncCommandHandler 1 (Usuario)

```
@Component
public class CrearUsuarioCommandHandler implements
SyncCommandHandler<Void,CrearUsuarioCommand>{
   private UsuarioWriteRepository usuariosWriteRepository;
   private UsuarioReadRepository usuariosReadRepository;
         //Definimos constructor
   @Override
public Void handle(CrearUsuarioCommand command) {
.....
}
```

Apéndice 1: UMUBUS – Clase SyncCommandHandler 2 (Usuario)

```
public Void handle(CrearUsuarioCommand command) {
   // Idempotencia
            usuariosReadRepository.findUsuario(command.getId())
       .ifPresentOrElse(
          (u)-> { throw new UnsupportedOperationException(String.format("..."); },
          () -> {Usuario usr = UsuarioFactory.createUsuario(command.getId());
             usr.createUsuario();
             usuariosWriteRepository.saveUsuario(usr);
          }
       );
   return null;
}
```


UNIVERSIDAD DE MURCIA
1272