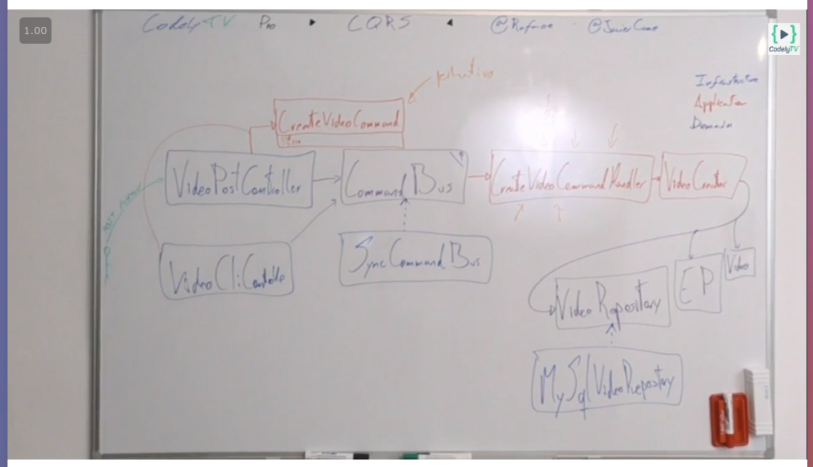
Watch: **Introducción a CQRS - 💸 ¡Disponible sin registro!**
✓ Completed · 17m · 100 Points



Introducción

En este curso **analizaremos un caso práctico**. En concreto será está aplicación siguiendo una arquitectura basada en Command Query Responsibility Segregation (CQRS a partir de ahora).

En esta primera lección haremos un repaso rápido por todos los componentes que intervienen en el flujo de una petición. En concreto será una **petición con el fin de crear un nuevo vídeo** en un sistema de gestión de vídeos, comentarios y demás 😬

Importante destacar que **en las siguientes lecciones estudiaremos con más detenimiento cada uno de estos componentes** y capas que intervienen en nuestra arquitectura. Analizaremos qué beneficio tiene cada uno de ellos para **formarnos un criterio de cuándo nos compensa introducirlos**.

Elementos de la arquitectura

En el vídeo repasamos cada uno de los siguientes elementos:

- `POST /videos/`
  - Responsabilidad: Servir de protocolo de comunicación con los clientes. Petición a través del protocolo de comunicación HTTP revelando la intención de crear un nuevo recurso bajo la colección de recursos videos.
- `VideoPostController`
  - Responsabilidades:
    - Recibir la petición HTTP
    - Recuperar los datos necesarios para la creación del vídeo y encapsularlos en el Comando correspondiente
    - Enviar el comando al bus
  - Capa: Infraestructura
- `CreateVideoCommand`
  - Responsabilidades:
    - Encapsular los datos necesarios para la creación de un vídeo
    - Permitir su fácil transporte entre capas y serialización
  - Capa: Aplicación
  - Consideraciones:
    - Para garantizar el propósito de este elemento (fácil transporte y serialización de datos), los datos que contiene deberían ser primitivos o escalares. En resumen, podremos almacenar cualquier tipo de dato que podamos meter en un JSON (string, int, bool, array, etc)
    - Aquí vemos una aplicación del patrón de diseño Data Transfer Object (DTO)
- `CommandBus`
  - Responsabilidades:
    - Transportar un determinado Command a su CommandHandler correspondiente
    - Almacenar por tanto el mapeo 1 a 1 entre commands y command handlers
  - Beneficios:
    - Al ser un punto de indirección introducido entre la recepción de la petición y la ejecución de la lógica de negocio asociada, permitirá por tanto cortar el flujo de ejecución y por tanto procesar peticiones de forma asíncrona
    - Al tratarse de un mapeo entre intención de realizar una acción, y la propia ejecución de esa acción con su lógica de negocio pertinente, permite ser reutilizado por cualquier pieza que necesite ejecutar dicha lógica de negocio. Esto es, podremos tirar comandos desde distintos puntos de nuestro sistema y tendremos un único punto de cambio en caso de necesitar modificar cómo deberían tratarse estos comandos (los handlers).
  - Capa: Dominio (esto representaría el contrato, la interface)
- `SyncCommandBus`
  - Responsabilidad:
    - Implementación concreta de la interface de dominio anteriormente descrita.
    - En este ejemplo se trata de una implementación síncrona. Bloqueará por tanto el hilo de ejecución desde que recibe el command hasta que obtiene una respuesta que permita liberar el hilo de ejecución.
  - Capa: Infraestructura
- `CreateVideoCommandHandler`
  - Responsabilidades:
    - Recibir Command a través del bus
    - Transformar datos del Command a tipos del dominio (value objects, nunca instanciará

entidades, de eso se encargará el propio caso de uso).
    - Invocar al caso de uso pasándole los parámetros recibidos ya mapeados a Value Objects.
  - Capa: Aplicación
- `VideoCreator`
  - Responsabilidades:
    - Encapsular la lógica de negocio asociada a un determinado caso de uso
    - Recibir los datos necesarios para ejecutar los casos de uso
    - Orquestar las llamadas a los distintos elementos que deberán entrar en acción para la consecución del caso de uso (repositorios, servicios de domino, modificaciones en entidades, etc)
  - Capa: Aplicación
  - Consideraciones:
    - Este tipo de elementos son denominados "Application Services" en el libro "Implementing Domain-Driven Desing" (DDD). No obstante, Sandro Mancuso también los denomina "Actions" en sus charlas sobre "Interaction Driven Design"
- `Video`
  - Responsabilidad: Representar un determinado concepto de dominio. Se puede entender como Entidad o, en términos de DDD, como Aggregate. Más concretamente, en este caso estaríamos hablando de la Entidad que adopta el rol de Aggregate Root dentro del agregado.
  - Capa: Dominio
- `VideoRepository`
  - Responsabilidad:
    - Expresar el contrato de dominio para la interacción con la capa de persistencia.
    - Guardar, actualizar, y eliminar recursos de tipo video
  - Capa: Dominio
  - Consideraciones:
    - Implementa el patrón de diseño "Repository".
    - Patrón de diseño que nos interesará evitar: Data Access Objects (DAO). Motivo: Forzar a que nuestra lógica de negocio se encuentre en los casos de uso o servicios de dominio. Evitar la explosión de métodos en nuestras interfaces de interacción con la persistencia.
- `MySqlVideoRepository`
  - Responsabilidad:
    - Implementación concreta de la interface antes descrita. Proveer de la lógica necesaria para interaccionar con la base de datos MySQL.
  - Capa: Infraestructura

Enlaces relacionados

Os recomendamos encarecidamente seguir el siguiente orden a la hora de hacer los cursos sobre Arquitectura y Diseño de Software:

1. Principios SOLID Aplicados
2. Arquitectura Hexagonal
3. Testing: Introducción y buenas prácticas
4. CQRS
5. Event-Driven Architecture
6. DDD
7. CQRS+Event Sourcing

Conclusión

La mala noticia: ¡Aquí hay mucha tela por cortar! Aún no hemos visto cómo aprovechar todo el potencial que nos ofrece esta arquitectura y seguro que todo resulta muy confuso por ahora

La buena noticia: ¡Estáis en el sitio perfecto! En las próximas lecciones justamente iremos analizando punto por punto cada uno de estos elementos. Veremos detalles de implementación en PHP y Scala, pero lo más importante: Entenderemos qué aporta cada elemento para poder formarnos ese criterio que comentábamos al inicio. ¡Nos vemos!