Watch: **Comunicar modules y Bounded Contexts: Repositories vs Application Service**

Comunicación entre Modules y BC via AS vs Repos 🤔

Vamos a ver los pros y contras que suponen inyectar repositorios e inyectar servicios, aunque existen más posibilidades que veremos más adelante, nos centraremos en este video en esas dos

Inyectando VideoRepository

Clase `VideoCommentPublisher`:

```
public class VideoCommentPublisher
{
    private $videoCommentRepository;
    private $videoRepository
    private $publisher;

    public function __construct(VideoCommentRepository
$videoCommentRepository, VideoRepository $videoRepository,
DomainEventPublisher $pubblisher)
    {
        $this->$videoCommentRepository =
$videoCommentRepository;
        $this->videoRepository = $videoRepository;
        $this->publisher = $publisher;

    }

    public function publish(VideoCommentId $id, VideoId
$videoiD, VideoCommentContent $content)
    {
        $this->ensureVideoExist($videoId);

        $comment = VideoComment::publish($id, $videoId,
$content);

        $this->repository->sasve($comment)
```

```
        $this->publisher->publish(...$comment-
>pullDomainEvents())
    }

    private function ensureVideoExist(VideoId $id): void
    {
        $video = $this->videoRepository->search($id)

        if(null ==== $video)
        {
            throw new VideoNotFound($id)
        }
    }
}
```

En una primera iteración vamos implementar el método `ensureVideoExist` inyectando el repositorio. Pero ¡ojo! 👀 estamos inyectando el repositorio de *Video*, referenciando desde un módulo al dominio de otro módulo

Como pros de esta implementación tendremos, para empezar, que ya tendríamos completada la feature propuesta y podremos comprobar si el video existe. Sin embargo, estaríamos duplicando esta lógica en distintos puntos del código. Además, otro problema con esta implementación es que estamos generando un mayor acoplamiento con otros módulos (recordemos que nuestro objetivo es que Bounded Contexts y Módulos sean fácilmente modificables)

Inyectando VideoFinder

Clase `VideoCommentPublisher`:

```
public class VideoCommentPublisher
{
    private $repository;
    private $finder
    private $publisher;

    public function __construct(VideoCommentRepository
$repository, VideoFinder $finder, DomainEventPublisher
$pubblisher)
    {
        $this->$repository = $repository;
```

```
            $this->finder = $finder;
            $this->publisher = $publisher;

        }

        public function publish(VideoCommentId $id, VideoId
$videoiD, VideoCommentContent $content)
        {
            $this->ensureVideoExist($videoId);

            $comment = VideoComment::publish($id, $videoId,
$content);

            $this->repository->sasve($comment)

            $this->publisher->publish(...$comment-
>pullDomainEvents())
        }

        private function ensureVideoExist(VideoId $id): void
        {
            $this->finder->__invoke($id)
        }
}
```

Aunque sigue habiendo acoplamiento con el módulo de *Video*, el hecho de inyectar el Servicio de Dominio nos hace ganar en términos de no tener la lógica duplicada en diferentes puntos

Por otro lado, estas iteraciones nos estan dejando ver que `VideoId` es un candidato perfecto para ser empujado a *Shared* puesto que estamos haciendo uso de él desde distintos módulos. En el caso de que fueramos a seguir compartiendo `VideoFinder` entre distintos módulos, también sería un buen candidato, pero como veremos en los siguientes videos, dejaremos de llamar a este servicio desde otros módulos distintos a Video

¿Alguna Duda?

Si tienes cualquier duda o sugerencia relacionada con este video no dudes en abrir una nueva discusión aquí abajo 👇👇👇

Nos vemos en la siguiente Lección: 📥 ¡Eventos de Dominio!