Watch: **Role interfaces vs Header interfaces**

Role interfaces vs Header interfaces 🤯

Para entender mejor los conceptos de Role interfaces y Header interfaces continuaremos con refactorizando el caso de uso Student Sign Up, concretamente invertiremos la dependencia del repositorio que estábamos inyectando en el constructor

Clase `StudentSignUpper`:

```
final class StudentSignUpper
{
    private $repository;

    public function __construct(StudentRepositoryInmemory $repos
    {
        $this->repository = $repository;
    }

    // ...
}
```

Tal como vimos en el curso de SOLID, para invertir la dependencia lo que haremos será extraer una interface donde vamos a definir las cabeceras de los métodos (firmas o headers), y será esta interface la que recibirá el caso de uso por parámetro

interface `StudentRepository`:

```
interface StudentRepository
{
    public function save(Student $student): void;

    public function search(StudentId $id): ?Student
}
```

Clase `StudentSignUpper`:

```
final class StudentSignUpper
{
    private $repository;

    public function __construct(StudentRepository $repository)
    {
        $this->repository = $repository;
    }

    // ...
}
```

Este acoplamiento que teníamos de Infraestructura dentro del Application Service, además de suponer un 'smell', nos suponía una falta de tolerancia al cambio y un alto coste de Entrada/Salida a BD en tiempo de Test. Gracias a este refactor si que podremos mockear esa Entrada/Salida, por lo que nuestros tests se ejecutarán mucho más rápido y se limitarán a comprobar que el comportamiento sea correcto

Pero no se trata simplemente de extraer una interface para invertir esas dependencias, a nivel estructural tampoco debemos mantenernos acoplados por unas firmas de métodos que estén condicionadas por implementaciones concretas (como podría ser pasar por parámetro una *key* porque estemos pensando en una implementación con *Redis*), puesto que el cliente no tiene que conocer los detalles de nuestra implementación

Algo que nos ayudará a hacer que los métodos del Repositorio sean agnósticos a la implementación es el uso del **Patrón Criteria** que vimos en el curso de SOLID

Lo importante de todo esto es que **el cómo definimos las interfaces venga establecido por los clientes**, las interfaces cumplen con un rol, en este caso el de ser el repositorio de `Student`, y deben ser agnósticas a cualquier implementación que haya por debajo. **No son Header Interfaces** a nivel de generar unas interfaces en base a las cabeceras de las implementaciones

A modo de síntesis podemos condensar la idea que queremos transmitir en este video en que **Las interfaces pertenecen a los clientes**, para evitar Leaks de infraestructura en nuestra aplicación, lo mejor es no conocer la infraestructura.

- El Arte del Patadón Pa'lante: Presentación de Eduardo Ferro acerca de la importancia de postergar las decisiones de detalles de implementación al último momento responsable

¿Alguna Duda?

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

Nos vemos en el próximo video: ¡Comunicar modules y Bounded Contexts: Repositories vs Application Service!