Watch: **Aggregate root vs Entity: Ejemplo Video y VideoHighlight**

Ejemplo de Agregados: Solucionando problemas de El Mundo Real™

Como habíamos visto en el video anterior, `Review` había crecido demasiado como para seguir manteniéndolo dentro del Aggregate `Course`, por lo que el primer paso que llevaremos a cabo será extraerlo fuera como un nuevo Aggregate con su propio Aggregate Root `CourseReview`

Dentro de `Course` seguiremos manteniendo el Value Object `Rating`, puesto que es un elemento que vamos a seguir necesitando a la hora de pintar la pantalla del curso

Clase `Course`:

```
final public class Course extends AggregateRoot {
    private final CourseId id;
    private final CourseName name;
    private final CourseRating rating;

    public Course(CourseId id, CourseName name, CourseRating
rating){
        this.id = id;
        this.name = name;
        this.rating = rating;
    }
    // ...
}
```

Clase `Review`:

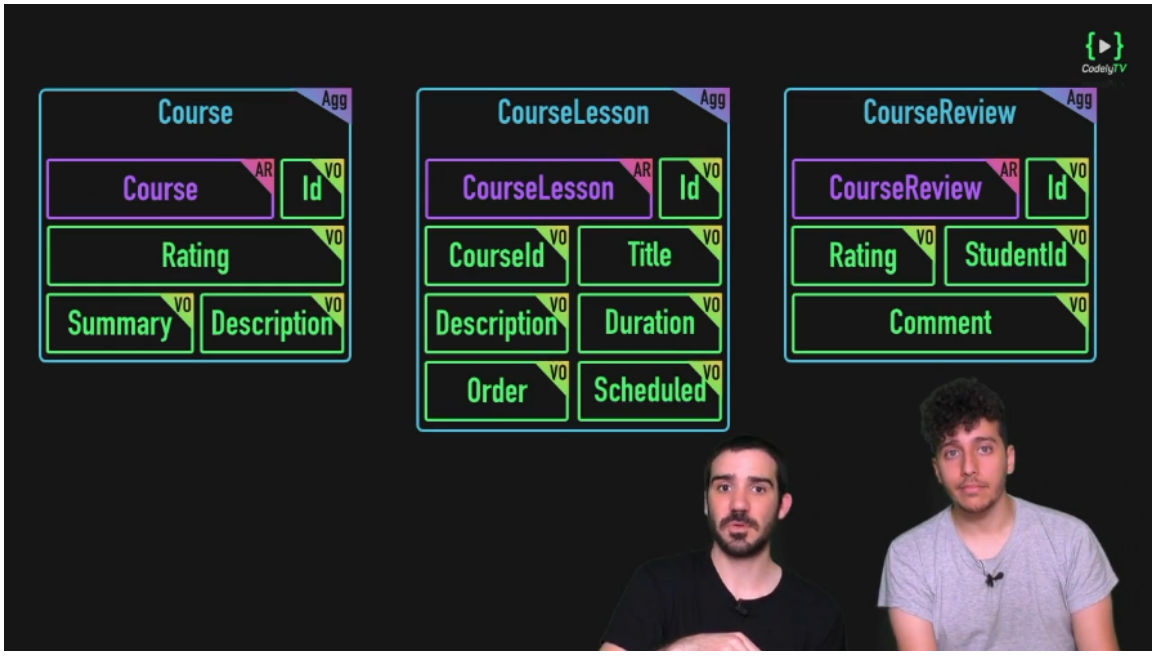
```
public final class Review extends AggregateRoot {
    private final StudentId studentId;
```

```
    private final CourseId courseId;
    private final CourseReviewRating rating;
    private final ReviewComment comment;

    public Review(StudentId studentId, CourseId
courseId,CourseReview Rating rating,ReviewComment comment){
        this.studentId = studentId;
        this.courseId = courseId;
        this.rating = rating;
        this.comment = comment;
    }
    // ...
}
```

Ahora Review es un elemento independiente cuya relación con el Curso será simplemente a través de un courseId, de modo que ni el curso sabrá qué reviews tiene, ni las reviews sabrán a qué curso pertenecen, por lo que no necesitaremos levantar todo el curso de BD, sino solo el valor del identificador de ese curso (Relación entre clases por identificador)

Podemos ver en estas clases cómo estamos manteniendo dos Value Objects diferentes para el Rating, hemos decidido hacerlo de este modo porque podríamos querer que cada uno tuviera una lógica distinta, por ejemplo, computar el valor de `CourseRating` en base al valor de un `CourseReviewRating`



Al igual que con las reviews del curso, también extraeremos las lecciones como Aggregate `CourseLesson`, de modo que nuestro `Course` queda como una clase mucho más pequeña y 'tonta',

facilitándo así su mantenibilidad. Además, en términos transaccionabilidad, tener agregados más pequeños reducirá los posibles bloqueos en BD y hará que las consultas sean mucho más rápidas

El hecho de extraer estos agregados, podía generar algunos problemas, pero veremos que solución podemos darle a cada uno de ellos:

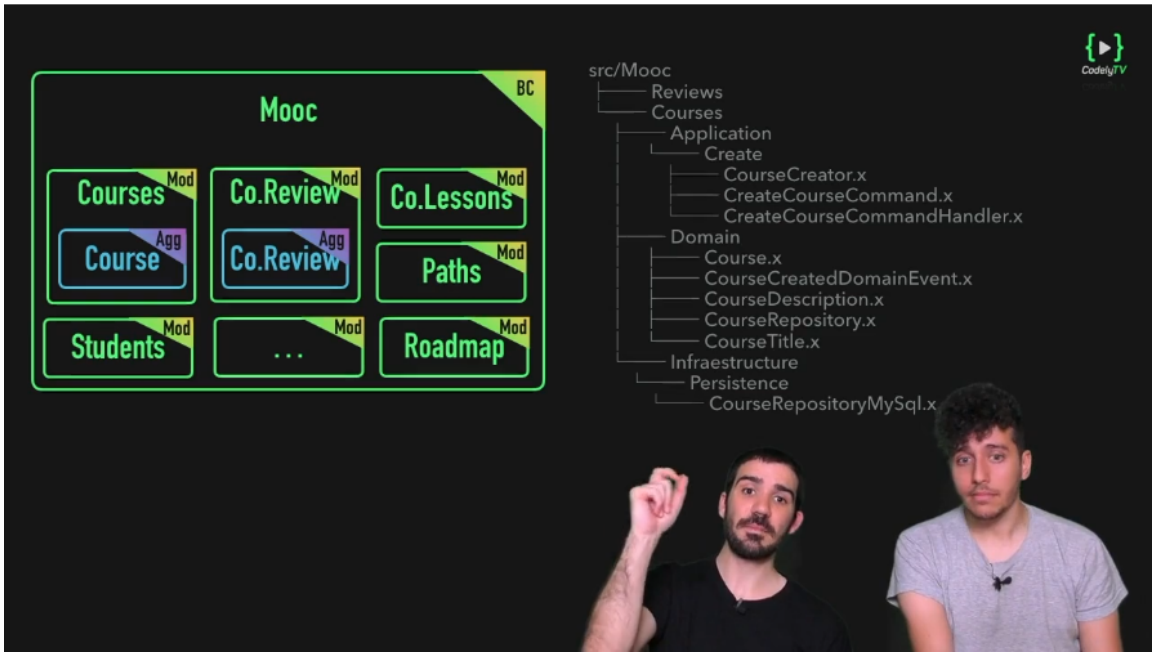
Asegurando consistencia en CourseLessons

- Validar orden entre lecciones: Aquí al ser una **validación que debemos ejercer previamente a ejecutar el caso de uso** ya que si no, éste no se debería poder llevar a cabo, haremos la consulta de forma previa encapsulándola en un Domain Service.

Asegurando consistencia en Course

- Incrementar total steps y videos: Aquí la estrategia será gestionar este tipo de tareas y cómputos de forma asíncrona por medio de Eventos de Dominio que nos permitan además una mayor performance a la hora de recuperar la información de BD

Estructura



Finalmente, estos agregados estarían contenidos en un módulo (normalmente la relación será de un módulo por cada agregado, con lo que respetaremos el **SRP** de SOLID). Dentro de cada módulo, a su vez, encontraremos los directorios de Application, Domain e Infraestructure siguiendo el diseño de **Arquitectura Hexagonal** (Podéis acceder al curso de Arquitectura Hexagonal desde este enlace)

¿Alguna Duda?

Esta es una propuesta de diseño que nos gusta, seguiremos viendo más en los siguientes video. Si tienes cualquier duda, sugerencias para esta lección o quieres compartir alguna otra propuesta que te guste no dudes en abrir una discusión nueva aquí abajo. Nos vemos en la siguiente Lección: Fallando miserablemente modelando Agregados 👼👇👇