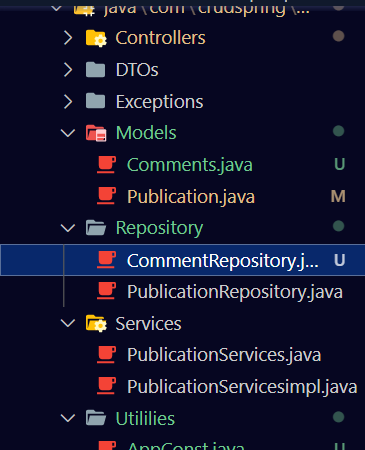
­­ **Relaciones entre tablas con Spring Boot**

@*OneToMany*(mappedBy = "publication", cascade = CascadeType.ALL, orphanRemoval = true)

    private Set<Comments> comments = new HashSet<>();

@*OneToMany* Esta anotación se utiliza para la relación uno a muchos, y el removal, para cuando se elimine un elemento de publicación, este elimine los registros relacionados a este y el set es una clase de java, está la utilizamos aquí porque vamos a retornar muchos comentarios de una sola publicación, es por esto que creamos este tipo de arreglo.



Creamos el Repositorio o interfaz que va conectada a la implementación jpa que realizara todas las funciones crud de esta entidad.

package com.crudspring.api.Repository;

import org.springframework.data.jpa.repository.JpaRepository;

import com.crudspring.api.Models.Comments;

public interface CommentRepository extends JpaRepository<Comments, Long> {

}

Esta interfaz es la encargada de hacer la implementación JPA

En el momento de crear la entidad creamos la relación entre las tablas en el caso de la tabla comentarios se relaciona de muchos a uno y para esto utilizamos lo siguiente

@*ManyToOne*(fetch = FetchType.LAZY)

    @*JoinColumn*(name = "publcation\_id", nullable = false)

Esta anotación crea un campo en la tabla de la base de datos el cual será el encargado de hacer la unión con la tabla seleccionada

Lo utilizamos para optimizar recursos y solo se haga la petición cuando el cliente lo solicite

**Estructura de la funcionalidad comments**

Esta estructura comienza en el controlador allí creamos la clase commentcontrollers a la cual le colocamos las anotaciones

@*RestController*

@RestController -> esta porque es una Incluye las anotaciones @Controller y @ResponseBody, y como resultado, simplifica la implementación del controlador

@*RequestMapping*("/api/")

public class CommentsController {

Nos da la ruta por donde se va a consumir el api del con controlador, para acceder a los métodos, a partir de esta ruta ya podemos utilizar los métodos rest para acceder a ellos.

 @*Autowired*

    private CommentsServices commentservices;

hacemos la inyección de dependencias a la implementación de los comentarios.

@*PostMapping*("/publication/{id}/comments")

Utilizamos el método post para la ruta del método del controlador

public ResponseEntity<CommentsDTO> saveComments(@*PathVariable*(value = "id") *long* *id*,

            @*RequestBody* CommentsDTO *commentsdto*) {

        return new ResponseEntity<>(commentservices.createComment(*id*, *commentsdto*), HttpStatus.CREATED);

Este es el metodo post el cual lo creamos ResponseEntity debido que este va a retornar una respuesta al servidor, el método también va a ser de tipo dto, este se le va a enviar dos parámetros el id y el dto que previamente es convertido de json a objeto por la anotación *RequestBody* .

public interface CommentsServices {

    public CommentsDTO createComment(*long* *id*, CommentsDTO *commentsdto*);

}

En la interface de la implementación de comentarios hacemos el llamado al método creado en la implementación, NOTA en la implementación debe estar nombrado igual para que se pueda utilizar.

@*Override*

    public CommentsDTO createComment(*long* *id*, CommentsDTO *commentsdto*) {

        Comments comments = mapEntity(*commentsdto*);

        Publication publication = publicationrepository.findById(*id*)

                .orElseThrow(() *->* new ResourceNotFountException("publication", "id", *id*));

El método recibe dos variables una la id de la publicación, y el objeto del comentario.

Convertimos el dto a entidad

        comments.setPublication(publication);

        Comments newComments = commentrepository.save(comments);

        return mapDTO(newComments);

Y por último guardamos el comentario con la información ya asignada

Asignamos lo que retorno la búsqueda de la publicación en la entidad del comentario

En la clase de la implementación aplicamos este método para realizar el guardado del comentario con la asignación de la publicación.