Se deben crear los packages:

* Controllers
* Entities
* Repositories
* Services

En entities se crean las clases como Product con sus contructores getter y setters

* Se debe poner @entity
* Se debe poner el nombre de la tabla en @table
* Mapear las variables como la primary key como @id

En repositories poner las interfaces que hereden de crudrepository

* Se pone en los tag el objeto y el tipo de variable de la primary key

public interface ProductRepository extends *CrudRepository*<*Product*,*Long*> {

}

En controllers reciben los datos de la pantalla del usuario. Se debe llamar al services con autowired

@*Autowired*

    private *ProductService* service;

luego se organizan los add, delete, edit y view según lo hecho anteriormente

 // permite visualizar el product

    @*GetMapping*("/{id}")

    public *ResponseEntity*<*?*> view(@*PathVariable* *Long* *id*){ // la id se enviara atraves de la url de getmapping, usando PathVariable

*Optional*<*Product*> productOptional =  service.findById(id); // busca el product usandolo en el optional

        if(productOptional.isPresent()){ // si se encuentra el producto

            return ResponseEntity.ok(productOptional.orElseThrow()); // da la respuesta de ok y se pasa el productOptional

        }

        return ResponseEntity.notFound().build(); // sino entrega un mensaje de no encontrado

    }

    // crea el producto enviado desde el cliente

    @*PostMapping*

    public *ResponseEntity*<*Product*> create(@*RequestBody* *Product* *product*){

*Product* productNew = service.save(product); // guarda el product recibido por parametro

        return ResponseEntity.status( HttpStatus.CREATED ).body( productNew );

    }

    // actualiza el product

    @*PutMapping*("/{id}")

    public *ResponseEntity*<*Product*> update(@*PathVariable* *Long* *id*, @*RequestBody* *Product* *product*){

        return ResponseEntity.status( HttpStatus.CREATED ).body( service.save(product) );

    }

    // permite eliminar el product

    @*DeleteMapping*("/{id}")

    public *ResponseEntity*<*?*> delete(@*PathVariable* *Long* *id*){ // la id se enviara atraves de la url de getmapping, usando PathVariable

*Product* product = new Product();

        product.setId(id);

*Optional*<*Product*> productOptional =  service.delete(product); // busca el product usandolo en el optional

        if(productOptional.isPresent()){ // si se encuentra el producto

            return ResponseEntity.ok(productOptional.orElseThrow()); // da la respuesta de ok y se pasa el productOptional

        }

        return ResponseEntity.notFound().build(); // sino entrega un mensaje de no encontrado

    }

@valid: sirve para validar campos (nulos, cantidad de caracteres, entre otros)

Se debe poner en el pom.xml

<dependency>

            <groupId>org.springframework.boot</groupId>

            <artifactId>spring-boot-starter-validation</artifactId>

        </dependency>

Luego en los @requestbody se pone los @valid

En los objetos se usan los tag

@*NotEmpty* // para string debe ser notEmpty en vez de notNull

    @*Size*(min = 3, max = 20) // establece la cantidad de caracteres

BindingResult sirve para validar errores y mostrarlos al cliente

// el <?> indica que puede ser cualquier valor

    private *ResponseEntity*<*?*> validation(*BindingResult* *result*) {

*Map*<*String*, *String*> errors = new *HashMap*<>();

        // por cada campo que ocurra un error, se guada en err y se muestra el campo con su error correspondiente

        result.getFieldErrors().forEach(err *->* {

            errors.put(err.getField(), "El campo " + err.getField() + " " + err.getDefaultMessage());

        });

        return ResponseEntity.badRequest().body(errors);

    }

Se debe poner al costado derecho del campo a validar

 // actualiza el product

    @*PutMapping*("/{id}")

    public *ResponseEntity*<*?*> update(@*PathVariable* *Long* *id*, @*Valid* @*RequestBody* *Product* *product*, *BindingResult* *result*){ // BindingResult debe estar a la derecha (al lado) del objeto a validar (Product)