Índice de contenido

# CONFIGURACIÓN

* 1. Creación del proyecto:
     1. Creación del proyecto mediante la web de spring boot '<https://start.spring.io/>'.
     2. Creación del proyecto mediante ecripse: new/proyect/New Spring Starter Proyect
  2. Configuración

|  |
| --- |
|  |

* + 1. Group: Ruta inicial de las clases del proyecto.
    2. Artifact: Nombre del fichero de compilación del proyecto.
    3. Dependencies: Podemos introducir las dependencias que queremos introducir en el proyecto.
* Fichero pom.xml

|  |
| --- |
|  |

* Fichero de configuración del proyecto 'properties'.

|  |
| --- |
|  |

* Estructura de ficheros.

|  |
| --- |
|  |

* + Controller:
  + Converter:
  + Entity:
  + Model:
  + Repository:
  + Service:

# ENTIDADES Y MODELOS.

* Entity.

|  |
| --- |
|  |

* Permite mapear la Clase con una tabla de la base de datos por medio de anotaciones de JPA.
  + @Entity: Definimos que la clase es una entidad.
  + @Table: Definimos que la clase mapea una tabla, identificamos la tabla que mapea la clase por medio del nombre.
  + @Id: Define que la propiedad anotada representa al campo identificador de la tabla.
  + @GeneratedValue: Define como se va ha generar el valor de la propiedad identificadora de la tabla
  + @Column: Define que la propiedad anotada representa un campo de la tabla, pudiendo definir el nombre de la tabla, el tipo de dato que va a guardar, si es un campo que no se van a repetir los valores (unique), etc.
* Si utilizamos la anotación @XmlRootElement antes de la definición de la clase indicamos que vamos a utilizar xml en vez e json para convertir los datos a la clase entidad.
* Se va a trabajar con el por medio de los repositorios.
* Model.

|  |
| --- |
|  |

* + Representa la entidad como el modelo de dato que vamos a devolver al usuario en la vista.
  + Debamos crear un constructor que construya el modelo a partir de un objeto de la entidad que representa.
  + Se va a trabajar con el por medio de los controladores.

# REPOSITORIOS Y CONVERTIDORES.

* Repositorio.

|  |
| --- |
|  |

* + Se encarga de hacer peticiones a la base de datos y actualizar los datos mediante entidades relacionadas con las tablas a consultar o actualizar.
  + Se crea mediante una interfaz que extiende de JpaRepository, debemos definir la entidad que vamos a utilizar.
  + @Repository: Mediante esta anotación definimos que la interfaz es un repositorio y el nombre del bean para inyectar el repositorio en el servicio.
  + Podemos definir métodos abstractos que van a ser los encargados de realizar las consultas a la base de datos y devolvernos una entidad o una lista de entidades rellenas con el resultado de la consulta.
    - NombreEntidad findByNombreCampo: Devuelve una entidad que el valor del campo en la tabla que tiene el nombre del método coincida con el parámetro que le pasamos al método.
    - List<NombreEntidad> findByNombreCampo: Devuelve una lista de entidades en las que el valor del campo en la tabla que tiene el nombre del método coincida con el parámetro que le pasamos al método.
    - NombreEntidad findByNombreCampo1AndNombreCampo2: Devuelve una entidad realizando una consulta en la que coinciden los valores de los parámetros en los campos definidos en el nombre del método.
* Convertidor

|  |
| --- |
|  |

* + @Componet: Mediante esta anotación definimos que la clase es un componente y el nombre del bean para inyectar el componente en el servicio.
* Por medio del método convertirLista convierte una lista de tipo de entidad a un tipo de modelo para devolvérselo al Usuario en la vista.

# SERVICIO

|  |
| --- |
|  |

* El Servicio define una serie de métodos para interactuar con los repositorios de la aplicación, pudiendo consultar y actualizar datos en las tablas de la base de datos, etc.
* @Service: La anotación indica que la clase es un servicio y define el nombre del bean para inyectar el servicio en el controlador.
* @Autowired: La anotación indicamos a spring que vamos a inyectar un bean
* @Qualifier: Con la anotación indicamos el nombre del bean a inyectar.

# CONTROLADOR

## Creación de controlador.

|  |
| --- |
|  |

* @RestController: Esta anotación indica que esta clase en la controladora de una Api Rest.
* @RequestMapping: Esta anotación mapea el controlador, también indicamos la versión del Api Rest.
* @Autowired y @Qualifier: Por medio de estas anotaciones inyectamos las dependencias que necesitemos de los servicios.

## Método crear.

|  |
| --- |
|  |

* @PutMapping: Mediante esta anotación indicamos que el método atenderá las peticiones put que nos lleguen por medio de la url /nuevo.
* @RequestBody: Esta anotación nos permite capturar los datos contenidos en el cuerpo de la petición.
* @Valid: Esta anotación permite convertir en json que nos llega en el cuerpo de la petición a una clase que indicamos como parámetro del método.

## Método actualizar.

|  |
| --- |
|  |

* @PostMapping: Mediante esta anotación indicamos que el método atenderá las peticiones post que nos lleguen por medio de la url /actualizar.
* @RequestBody: Esta anotación nos permite capturar los datos contenidos en el cuerpo de la petición.
* @Valid: Esta anotación permite convertir en json que nos llega en el cuerpo de la petición a una clase que indicamos como parámetro del método.

## Método eliminar.

|  |
| --- |
|  |

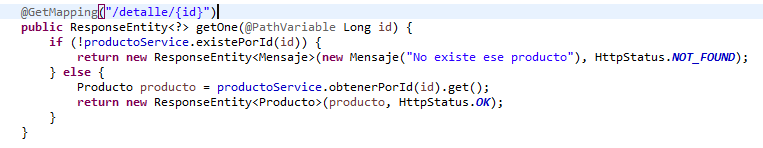
* @DeleteMapping: Mediante esta anotación indicamos que el método atenderá las peticiones delete que nos lleguen por medio de la url /borrar. En la url le indicamos los parámetros que va a recibir en la petición entre /{ nombreParametro}
* @PathVariable: Mediante esta anotación recuperamos un valor de un parámetro indicando el nombre del parámetro a recuperar.

## Método consultar una lista

|  |
| --- |
|  |

* @GetMapping: Mediante esta anotación indicamos que el método atenderá las peticiones get que nos lleguen por medio de la url /notas y devolverá una lista del modelo de datos.

## Método consulta de una entidad



* @GetMapping: Mediante esta anotación indicamos que el método atenderá las peticiones get que nos lleguen por medio de la url /detalle y devolverá una entidad por su id
* @PathVariable: Mediante esta anotación recuperamos un valor de un parámetro indicando el nombre del parámetro a recuperar.

## HttpStatus.

|  |  |
| --- | --- |
| NOT\_FOUND | No existe la entidad que solicitamos en el sistema mediante la petición. |
| BAD\_REQUEST | Hay un error de validación de la petición que hacemos al sistema. |
| OK | La petición se ha realizado correctamente. |
| CREATED | La petición de crear / modificar entidad se ha realizado correctamente. |

## CrossOrigin.

* Para habilitar la política cors en el controller del REST. Al inicio de la clase Controller ponemos la anotación de @CrossOrigin y la ruta origen con el puerto.

|  |
| --- |
| @CrossOrigin(origins = "[Ruta\_Servidor]")  Ejemplo: @CrossOrigin(origins = "http://localhost:4200") |

# GESTIÓN DE LOGS

|  |
| --- |
|  |

* Por medio de la clase Log podemos escribir en el log de la aplicación, para inicializar la clase utilizamos LogFactory.getLog(Nobre\_Clase.class)
* Tipos de logs:
  + info: Muestra información en el log.
  + error: Muestra un error en el log.
  + warn: Muestra una alerta en el log.

# PAGINACIÓN

* Configuración en fichero de propiedades ‘.properties’.

|  |
| --- |
|  |

* + Propiedad page-param-name: Indica el nombre del parámetro que se va a utilizar para indicar en que página estamos actualmente.
  + Propiedad limit-param-name: Indica el nombre del parámetro que define el límite de valores que se van a devolver en la consulta.
  + Propiedad sort-param-name:
  + Propiedad default-page-size: Indica el tamaño por defecto de la paginación.
  + Propiedad max-page-size: Indica la cantidad máxima de registros que se van a enviar.
* Repositorio.

|  |
| --- |
|  |

* La interfaz debe extender también a PagingAndSortingRepository, hay que indicar la entidad con al que va a trabajar.
* El método abstracto de devolución de la lista devuelve un objeto de tipo Page en donde le indicamos la entidad con la que va a trabajar y recibe como parámetro un objeto de tipo Pageable que contiene toda la información referente a la paginación.
* Servicio.

|  |
| --- |
|  |

* + El método que devuelve la lista recibe un parámetro de tipo Pageable y retorna la lista que devuelve el método del repositorio pasando como parámetro el parámetro de tipo Pageable que ha recibido.
  + El método del servicio devuelve un objeto de tipo Page y llamando al método getContent() se puede recuperar la lista.
* Controlador

|  |
| --- |
|  |

* El método que mapea la dirección ‘/notas-paginacion’ recibe como parámetro un objeto de tipo Pageable que después pasa como parámetro al método del servicio que devuelve la lista.
* Ejemplo de url de petición <http://localhost:8080/v1/notas-paginacion?page=1&size=8>.
  + El parámetro page le indica la página actual donde estamos.
  + El parámetro size le indica el número máximo de registros por página que vamos a recibir del controlador.
  + Si no le pasamos los parámetros devuelve todos los registros de la lista.

# SEGURIDAD JWT TOKEN.

|  |
| --- |
|  |

## Usuario y Rol

* Enum con los diferentes tipos de roles de la aplicación

|  |
| --- |
| *package com.inezpre5.angularjwt.enums;*  *public enum RolNombre {*  *ROLE\_ADMIN,*  *ROLE\_USER*  *}* |

* Clase Rol guarda los diferentes tipos que rol que existen en la aplicación

|  |
| --- |
| *package com.inezpre5.angularjwt.entity;*  *import com.inezpre5.angularjwt.enums.RolNombre;*  *import javax.persistence.\*;*  *import javax.validation.constraints.NotNull;*  *@Entity*  *public class Rol {*  *@Id*  *@GeneratedValue(strategy = GenerationType.IDENTITY)*  *private Long id;*  *@Enumerated(EnumType.STRING)*  *@NotNull*  *private RolNombre rolNombre;*  *public Rol() {*  *}*  *public Rol(@NotNull RolNombre rolNombre) {*  *this.rolNombre = rolNombre;*  *}*  *public Long getId() {*  *return id;*  *}*  *public void setId(Long id) {*  *this.id = id;*  *}*  *public RolNombre getRolNombre() {*  *return rolNombre;*  *}*  *public void setRolNombre(RolNombre rolNombre) {*  *this.rolNombre = rolNombre;*  *}*  *}* |

* + La anotación @Enumerated indica que es un campo que se corresponde con un enum tipo String, RolNombre
* La Clase usuario guarda los datos referente a un usuario,

|  |
| --- |
| *@Entity*  *public class Usuario {*  *@Id*  *@GeneratedValue(strategy = GenerationType.IDENTITY)*  *private Long id;*  *@NotNull*  *private String nombre;*  *@NotNull*  *@Column(unique = true)*  *private String nombreUsuario;*  *@NotNull*  *@Column(unique = true)*  *private String email;*  *@NotNull*  *private String password;*  *@NotNull*  *@ManyToMany*  *@JoinTable(name = "usuario\_rol", joinColumns = @JoinColumn(name = "usuario\_id"), inverseJoinColumns = @JoinColumn(name = "rol\_id"))*  *private Set<Rol> roles = new HashSet<>();*  *public Usuario() {*  *}*  *public Usuario(@NotNull String nombre, @NotNull String nombreUsuario, @NotNull String email, @NotNull String password) {*  *this.nombre = nombre;*  *this.nombreUsuario = nombreUsuario;*  *this.email = email;*  *this.password = password;*  *}*  */\*GET Y SET\*/* |

* + En el constructor no se debe incluir los campos id y roles.

## Class de Spring Security y JWT (UserDetails y UserDetailsServiceImpl)

## Json Web Token.

## WebSecurity

# TEST JUNIT

|  |
| --- |
|  |

* Debemos anotar la clase de test con las anotaciones:
  + @RunWith(SpringRunner.class) que indica que es una clase ejecutable de Spring.
  + @SpringBootTest que indica que es una clase de test de Spring Boot