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.

|  |
| --- |
|  |

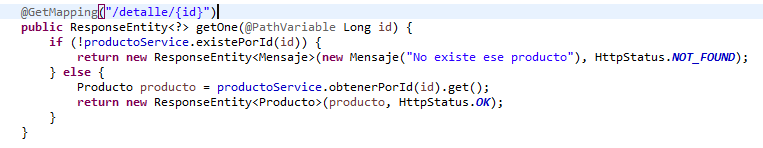
* @DeleteMapping: Mediante esta anotación indicamos que el método atenderá las peticiones delete que nos lleguen por medio de la url /borrar. En la url le indicamos los parámetros que va a recibir en la petición entre /{ nombreParametro}
* @PathVariable: Mediante esta anotación recuperamos un valor de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

* Sitios web de referencia:
  + <https://spring.io/guides/gs/securing-web/>
  + <https://jwt.io/introduction/>

La clase de configuración de seguridad debe extender de la clase WebMvcConfigurer. A través de esta clase añadimos los controladores que queremos que estén registrados para que tengan seguridad.

## Dependencias en el fichero pom.xml.

|  |
| --- |
|  |

* spring-boot.starter-security: Librería para control de seguridad y autentificación de la aplicación.
* io.jsonwebtoken: Librería que construye el token de seguridad en formato json.

## Clases de configuración.

* Paquete configuration: Se guardan todas las clases de configuración de seguridad de la aplicación.
* Clase WebSecurity.

|  |
| --- |
| **package** com.notas.core.configuration;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.beans.factory.annotation.Qualifier;  **import** org.springframework.context.annotation.Configuration;  **import** org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;  **import** org.springframework.security.config.annotation.web.builders.HttpSecurity;  **import** org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;  **import** org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;  **import** com.notas.core.service.UsuarioService;  @Configuration  @EnableWebSecurity  **public** **class** WebSecurity **extends** WebSecurityConfigurerAdapter {  @Autowired  @Qualifier("usuarioService")  **private** UsuarioService usuarioService;  @Override  **protected** **void** configure(AuthenticationManagerBuilder auth) **throws** Exception {  auth.userDetailsService(usuarioService);  } |

* + La clase de configuración de spring security debe extender de la clase WebSecurityConfigurerAdapter.
  + La clase tiene que ser anotada con:
    - @Configuration que indica que esta es una clase de configuración de Spring.
    - @EnabledWebSecurity activa la seguridad de spring.
  + El método sobrescrito configure(AuthenticationManagerBuilder auth) por medio del método del auth.userDetailsService le pasamos el servicio web que devuelve el usuario autentificar en la base de datos. Por cada autentificación que Spring vamos a realizar se busca el usuario en la base de datos.

|  |
| --- |
| @Override  **protected** **void** configure(HttpSecurity http) **throws** Exception {  http.csrf().disable().authorizeRequests()  .antMatchers("/login").permitAll()  .anyRequest().authenticated()  .and()  .addFilterBefore(**new** LoginFilter("/login", authenticationManager()),  UsernamePasswordAuthenticationFilter.**class**)  .addFilterBefore(**new** JwtFilter(),  UsernamePasswordAuthenticationFilter.**class**);  } |

* + En el método configure(HttpSecurity http) configuramos las rutas http en las que queremos autorización para acceder y cuales están libre de la autorización.
  + .antMatchers("/login").permitAll() indica que el acceso a la petición de /login no requiera autentificación previa.
  + .anyRequest().authenticated() cualquier otra petición requiere autentificación.
  + .addFilterBefore(new LoginFilter("/login", authenticationManager()), UsernamePasswordAuthenticationFilter.class) indica que todas las peticiones /login debe pasar por el filtro LoginFilter.
  + .addFilterBefore(new JwtFilter(), UsernamePasswordAuthenticationFilter.class) indica que las demás peticiones pasarán por este filtro para validar el token.
* Clase usuario.

|  |
| --- |
|  |

* + Esta clase guardará los datos de identificación de nos llegará desde la parte del cliente en formato Json.

## Gestión de usuario.

* Entidad usuario en la base de datos

|  |
| --- |
| **package** com.notas.core.entity;  **import** java.io.Serializable;  **import** java.util.Collection;  **import** javax.persistence.Column;  **import** javax.persistence.Entity;  **import** javax.persistence.GeneratedValue;  **import** javax.persistence.Id;  **import** javax.persistence.Table;  **import** org.springframework.security.core.GrantedAuthority;  **import** org.springframework.security.core.userdetails.UserDetails;  @Entity  @Table  **public** **class** Usuario **implements** Serializable, UserDetails {    **private** **static** **final** **long** ***serialVersionUID*** = -7722887051788610309L;  @GeneratedValue  @Id  @Column(name="ID")  **private** **long** id;    @Column(name="USUARIO", unique = **true**)  **private** String usuario;    @Column(name="PASSWORD")  **private** String password;    @Column(name="ROL")  **private** **byte** rol;    @Column(name="ACTIVO")  **private** **boolean** activo;  **public** **long** getId() {  **return** id;  }  **public** **void** setId(**long** id) {w  **this**.id = id;  }  **public** String getUsuario() {  **return** usuario;  }  **public** **void** setUsuario(String usuario) {  **this**.usuario = usuario;  }  **public** String getPassword() {  **return** password;  }  **public** **void** setPassword(String password) {  **this**.password = password;  }  **public** **byte** getRol() {  **return** rol;  }  **public** **void** setRol(**byte** rol) {  **this**.rol = rol;  }  **public** **boolean** isActivo() {  **return** activo;  }  **public** **void** setActivo(**boolean** activo) {  **this**.activo = activo;  }  @Override  **public** Collection<? **extends** GrantedAuthority> getAuthorities() {  // **TODO** Auto-generated method stub  **return** **null**;  }  @Override  **public** String getUsername() {  // **TODO** Auto-generated method stub  **return** **null**;  }  @Override  **public** **boolean** isAccountNonExpired() {  // **TODO** Auto-generated method stub  **return** **false**;  }  @Override  **public** **boolean** isAccountNonLocked() {  // **TODO** Auto-generated method stub  **return** **false**;  }  @Override  **public** **boolean** isCredentialsNonExpired() {  // **TODO** Auto-generated method stub  **return** **false**;  }  @Override  **public** **boolean** isEnabled() {  **return** isActivo();  }    } |

* + Debe implementar UserDetails.
  + Implementa un método GratedAuthority que devuelve una colección de roles de aplicación que tiene el usuario.
  + Implementa un método getUsername que devuelve el nombre del usuario para identificarse en la aplicación
  + Implementa un método isAccountNotExpired que devuelve si la cuenta a caducado.
  + Implementa un método isAccountNonLocked que devuelve si la cuenta de usuario esta caducada.
  + Implementa un método isCredentialsNosExpired que devuelve si las credenciales han caducado desde el ultimo logueo.
  + Implementa un método isEnabled que devuelve si el usuario está habilitada o deshabilitado.
  + Esta clase debe ser una entidad de una tabla
* Repositorio del Usuario.

|  |
| --- |
| **package** com.notas.core.repository;  **import** java.io.Serializable;  **import** org.springframework.data.jpa.repository.JpaRepository;  **import** org.springframework.stereotype.Repository;  **import** com.notas.core.entity.Usuario;  @Repository("usuarioRepository")  **public** **interface** UsuarioRepository **extends** JpaRepository<Usuario, Serializable>{    **public** **abstract** Usuario findByUsuario(String usuario);    } |

* Debe tener un método que devuelva un objeto usuario pasándole como parámetro del nombre de usuario.

## Servicio de autentificación.

|  |
| --- |
| **package** com.notas.core.service;  **import** java.util.ArrayList;  **import** java.util.List;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.beans.factory.annotation.Qualifier;  **import** org.springframework.security.core.GrantedAuthority;  **import** org.springframework.security.core.authority.SimpleGrantedAuthority;  **import** org.springframework.security.core.userdetails.User;  **import** org.springframework.security.core.userdetails.UserDetails;  **import** org.springframework.security.core.userdetails.UserDetailsService;  **import** org.springframework.security.core.userdetails.UsernameNotFoundException;  **import** org.springframework.stereotype.Service;  **import** com.notas.core.entity.Usuario;  **import** com.notas.core.repository.UsuarioRepository;  @Service("usuarioService")  **public** **class** UsuarioService **implements** UserDetailsService{    @Autowired  @Qualifier("usuarioRepository")  **private** UsuarioRepository usuarioRepository;  @Override  **public** UserDetails loadUserByUsername(String username)  **throws** UsernameNotFoundException {  Usuario usuario = **this**.usuarioRepository.findByUsuario(username);  **return** **new** User(usuario.getUsuario(), usuario.getPassword(),  usuario.isActivo(), usuario.isActivo(), usuario.isActivo(),  usuario.isActivo(), buildgranted(usuario.getRol())  );  }    **public** List<GrantedAuthority> buildgranted(**byte** rol){  String[] roles = {"LECTOR","USUARIO","ADMINISTRADOR"};  List<GrantedAuthority> auths = **new** ArrayList<>();    **for**(**int** i=0; i<roles.length; i++) {  auths.add(**new** SimpleGrantedAuthority(roles[i]));  }    **return** auths;  }    } |
|  |
|  |

* El servicio debe implementar a UserDetailService y al hacerlo implementamos el método loadUserByUsername al que se le pasa como parámetro el nombre del usurio y debe devolver un objeto de tipo UserDetails.

# 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