



# Sistemas Distribuidos e Internet

## Sesión, Roles, Consultas, Búsqueda y Paginación

### Sesión - 4

Curso 2022/2023



## Contenido

1	Sesión.....	3
1.1	Volver a la versión anterior.....	6
2	Roles.....	8
2.1	Actualización de entidades.....	8
2.2	RolesService, servicio para la gestión de roles.....	8
2.3	InsertSampleDataService generación de datos de prueba.....	9
2.4	Actualización de controlador UserController.....	9
2.5	Actualización de la vista /user/add.....	10
2.6	Carga del Rol en la autenticación.....	11
3	Control de acceso por roles.....	12
3.1.1	Configuración: autorización y control de acceso.....	12
3.1.2	Vistas dependientes de roles.....	14
3.1.3	Accion asociada al perfil de un alumno.....	20
3.1.4	Actualización de la tabla como fragmento.....	23
4	Consultas.....	26
4.1	Consultar notas.....	26
4.1.1	Actualizar MarksRepository.....	26
4.1.2	Actualizar MarksService.....	26
4.1.3	Modificar MarksController.....	27
5	Búsqueda.....	29
5.1	Actualizar MarksRepository.....	29
5.2	Actualizar MarksService.....	29
5.3	Actualizar MarksController.....	30
5.4	Actualizar la vista mark/list.....	30
5.5	Buscar por cadena contenida.....	31
6	Paginación.....	32
6.1	Actualizar MarksRepository.....	32
6.2	Actualizar MarksService.....	33
6.3	Modificar MarksController.....	34
6.4	Actualizar la Configuración de la aplicación.....	35
6.5	Actualización de las vistas.....	37



6.6	Corrigiendo errores y advertencias (warnings) .....	42
6.1	Etiquetar proyecto en GitHub .....	43
6.2	Resultado esperado en el repositorio de GitHub .....	44

## 1 Sesión

Al igual que en la mayor parte de tecnologías de desarrollo en el servidor, disponemos del objeto **sesión**. Este objeto se utiliza para almacenar y recuperar datos correspondientes a la sesión de un usuario y permanecen en el servidor hasta la expiración de la sesión.

Algunos usos comunes de la sesión son:

- Identificar al usuario autenticado (en nuestro caso, Spring Security emplea por debajo objetos HttpSession de la API de Servlets).
- Guardar datos temporales como los productos que metemos en el carrito en una tienda.

Para ver cómo funciona el concepto de sesión, vamos a modificar la aplicación para guardar en sesión una lista de las últimas notas consultadas (últimas notas vistas en detalles). Para llevar a cabo esta modificación llevamos a cabo las siguientes modificaciones:

### *Actualizar el servicio MarkService*

Accedemos a **MarkService** e inyectamos la sesión, objeto **HttpSession**.

```
@Service
public class MarksService {

    /* Inyección de dependencias basada en atributos */
    @Autowired
    private MarksRepository marksRepository;

    /* Inyección de dependencias basada en constructor */
    private final HttpSession httpSession;

    @Autowired
    public MarksService(HttpSession httpSession) {
        this.httpSession = httpSession;
    }
}
```

Cada vez que visitamos los detalles de una nota **/getMark(Long id)** vamos a almacenar la información de la nota en sesión (también podríamos almacenar únicamente el identificador de la nota).

Los atributos del objeto **HttpSession** se consultan con el método **getAttribute(<clave del atributo>)**.



Obtenemos el objeto con clave **consultedList** y, como es la primera vez que obtenemos la lista de notas consultadas, **esta puede ser null**.

Después agregamos la nota a la lista de notas consultadas y la volvemos a guardar en sesión con el método **setAttribute(<clave del atributo>, valor);**

```
public Mark getMark(Long id){
    Set<Mark> consultedList = (Set<Mark>) httpSession.getAttribute("consultedList");
    if (consultedList == null) {
        consultedList = new HashSet<Mark>();
    }
    Mark obtainedMark = marksRepository.findById(id).get();
    consultedList.add(obtainedMark);
    httpSession.setAttribute("consultedList", consultedList);
    return obtainedMark;
}
```

Al utilizar una estructura de datos basada en el Set, con el fin de evitar insertar notas repetidas, debemos **redefinir el método equals de la clase Mark**:

```
@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    Mark mark = (Mark) o;
    return Objects.equals(id, mark.id);
}
```

En este caso, consideramos que **dos notas son iguales si tienen la misma id**.

### *Actualizar el controlador MarksController*

Modificamos el controlador **MarksController**, inyectando el bean HttpSession.

```
@Controller
public class MarksController {
    @Autowired
    private HttpSession httpSession;
```

En el método que responde a **/mark/list** obtenemos el atributo de la sesión que contiene la lista de notas consultadas (debemos tener en cuenta que el atributo puede ser null), enviamos la lista a la vista bajo el nombre **consultedList**.

```
@RequestMapping("/mark/list")
public String getList(Model model) {
    Set<Mark> consultedList = (Set<Mark>) httpSession.getAttribute("consultedList");
    if (consultedList == null) {
        consultedList = new HashSet<Mark>();
    }
    model.addAttribute("consultedList", consultedList);
    model.addAttribute("markList", marksService.getMarks());
    return "mark/list";
}
```



### Actualizar la vista `mark/list.html`

En la vista `mark/list` creamos una nueva tabla al inicio del contenedor principal que mostrará las notas consultadas recientemente y que están almacenadas en `consultedList`.

```
<div class="container" id="main-container">
  <h2>Notas</h2>
  <!-- A partir de aquí es el código Nuevo -->
  <p>Notas consultadas recientemente por el usuario:</p>
  <div class="table-responsive">
    <table class="table table-hover" th:fragment="tableMarks" id="consultedTableMarks">
      <thead>
        <tr>
          <th scope="col">id</th>
          <th scope="col">Descripción</th>
          <th scope="col">Puntuación</th>
          <th scope="col"></th>
          <th scope="col"></th>
          <th scope="col"></th>
        </tr>
      </thead>
      <tbody>
        <tr th:each="mark : ${consultedList}">
          <td th:text="${mark.id}"> 1</td>
          <td th:text="${mark.description}"> Ejercicio 1</td>
          <td th:text="${mark.score}"> 10</td>
          <td><a th:href="'${'/mark/details/' + mark.id}">detalles</a></td>
          <td><a th:href="'${'/mark/edit/' + mark.id}">modificar</a></td>
          <td><a th:href="'${'/mark/delete/' + mark.id}">eliminar</a></td>
        </tr>
      </tbody>
    </table>
  </div>
  <!-- Aquí acaba el código nuevo -->
  <p>Las notas que actualmente figuran en el sistema son las siguientes:</p>
```

Si nos identificamos con un usuario (999999990A contraseña: 123456) y **tras navegar por los detalles de varias notas**, al acceder veremos las ultimas notas desde la vista `/mark/list`

id	Descripción	Puntuación			
3	Nota A1	10.0	detalles	modificar	eliminar
4	Nota A4	6.5	detalles	modificar	eliminar

Las notas que actualmente figuran en el sistema son las siguientes:

Actualizar

id	Descripción	Puntuación			
2	Nota A2	9.0	detalles	modificar	eliminar
3	Nota A1	10.0	detalles	modificar	eliminar
4	Nota A4	6.5	detalles	modificar	eliminar



También podemos **gestionar la sesión desde la propia plantilla**, utilizando el objeto **session**.

```
${session.isEmpty()} // Comprobar si la sesión está vacía  
${session.containsKey('consultedList')} // Comprobar si la sesión contiene un atributo  
${session.consultedList} // Acceder al atributo
```

**Nota: Incluir el siguiente Commit Message ->**

**“SDI-IDGIT-4.1-SpringBoot-Sesiones.”**

**OJO: sustituir IDGIT por tu número asignado (p.e. 2223-101):**

**“SDI-2223-101-4.1-SpringBoot-Sesiones”**

*(No olvides incluir los guiones y NO incluyas BLANCOS al principio o al final de la oración)*

## 1.1 Volver a la versión anterior

**Importante:** Una vez comprobada la funcionalidad del manejo de sesión y para simplificar la aplicación vamos a eliminar de la vista `/mark/list` la tabla de “notas consultadas recientemente por el usuario”.

```
<div class="container" id="main-container">  
  <h2>Notas</h2>  
  <p>Notas consultadas recientemente por el usuario:</p>  
  <div class="table-responsive">  
    <table class="table table-hover" th:fragment="tableMarks" id="tableMarks">  
      <thead>  
        <tr>  
          <th scope="col">id</th>  
          <th scope="col">Descripción</th>  
          <th scope="col">Puntuación</th>  
          <th scope="col"></th>  
          <th scope="col"></th>  
          <th scope="col"></th>  
        </tr>  
      </thead>  
      <tbody>  
        <tr th:each="mark : ${consultedList}">  
          <td scope="row" th:text="${mark.id}"> 1</td>  
          <td th:text="${mark.description}"> Ejercicio 1</td>  
          <td th:text="${mark.score}"> 10</td>  
          <td><a th:href="'${'/mark/details/' + mark.id}">detalles</a></td>  
          <td><a th:href="'${'/mark/edit/' + mark.id}">modificar</a></td>  
          <td><a th:href="'${'/mark/delete/' + mark.id}">eliminar</a></td>  
        </tr>  
      </tbody>  
    </table>  
  </div>  
</div>
```



```
<p>Las notas que actualmente figuran en el sistema son las siguientes:</p>
<button type="button" id="updateButton" class="btn btn-default">Actualizar</button>
<script>
    $( "#updateButton" ).click(function() {
        $("#tableMarks").load('/mark/list/update');
    });
</script>
```

Accedemos al controlador **MarksController**, eliminamos el uso de la sesión y volvemos a probar la aplicación para comprobar que todo funciona correctamente

```
@RequestMapping("/mark/list")
public String getList(Model model) {
    Set<Mark> consultedList = (Set<Mark>) httpSession.getAttribute("consultedList");
    if (consultedList == null) {
        consultedList = new HashSet<Mark>();
    }
    model.addAttribute("consultedList", consultedList);
    model.addAttribute("markList", marksService.getMarks());
    return "mark/list";
}
```

Para finalizar, eliminamos también la funcionalidad del **MarksService**:

```
public Mark getMark(Long id) {
    Set<Mark> consultedList = (Set<Mark>) httpSession.getAttribute("consultedList");
    if (consultedList == null) {
        consultedList = new HashSet<Mark>();
    }
    Mark obtainedMark = marksRepository.findById(id).get();
    consultedList.add(obtainedMark);
    httpSession.setAttribute("consultedList", consultedList);
    return obtainedMark;
    return marksRepository.findById(id).get();
}
```



## 2 Roles

Vamos a permitir tres tipos de roles en la aplicación:

- `ROLE_STUDENT`: estudiantes.
- `ROLE_PROFESSOR`: profesores.
- `ROLE_ADMIN`: administrador.

Según el rol del usuario, tendrá acceso a unas u otras funcionalidades.

`/user/add` agregar usuarios con cualquier rol nos permitirá elegir el rol.

`/login` identificar usuario (mismo funcionamiento que el actual).

`/signup` registrarse (solo usuario con `ROLE_STUDENT`).

### 2.1 Actualización de entidades

Si no está definido, agregamos el atributo **role** a la entidad **User**.

```
private String role;
```

Agregamos los métodos get y set para **role**.

```
public String getRole() {  
    return role;  
}  
  
public void setRole(String role) {  
    this.role = role;  
}
```

### 2.2 RolesService, servicio para la gestión de roles

Creamos la clase **RolesService** en el paquete **com.uniovi.notaneitor.services**, implementamos la función **getRoles()** que retorna la lista con los roles. **Vamos a incluir los roles en una lista, pero lo ideal sería incluir una entidad Role en la base de datos y relacionarla con la entidad user.**

```
@Service  
public class RolesService {  
    String[] roles = {"ROLE_STUDENT", "ROLE_PROFESSOR", "ROLE_ADMIN"};  
  
    public String[] getRoles() {  
        return roles;  
    }  
}
```





## 2.3 InsertSampleDataService generación de datos de prueba

Modificamos nuestro servicio de pruebas **InsertSampleDataService** para incluir un rol a los usuarios. Inyectamos el servicio **RolesService** para obtener el nombre de los roles y evitar confundirnos al escribir el String.

```
@Service
public class InsertSampleDataService {

    @Autowired
    private UsersService usersService;

    @Autowired
    private RolesService rolesService;

    @PostConstruct
    public void init() {
        User user1 = new User("99999990A", "Pedro", "Díaz");
        user1.setPassword("123456");
        user1.setRole(rolesService.getRoles()[0]);
        User user2 = new User("99999991B", "Lucas", "Núñez");
        user2.setPassword("123456");
        user2.setRole(rolesService.getRoles()[0]);
        User user3 = new User("99999992C", "María", "Rodríguez");
        user3.setPassword("123456");
        user3.setRole(rolesService.getRoles()[0]);
        User user4 = new User("99999993D", "Marta", "Almonte");
        user4.setPassword("123456");
        user4.setRole(rolesService.getRoles()[1]);
        User user5 = new User("99999977E", "Pelayo", "Valdes");
        user5.setPassword("123456");
        user5.setRole(rolesService.getRoles()[1]);
        User user6 = new User("99999988F", "Edward", "Núñez");
        user6.setPassword("123456");
        user6.setRole(rolesService.getRoles()[2]);
    }
}
```

## 2.4 Actualización de controlador UserController

Modificamos el controlador para usar **GET /user/add** para que obtenga la lista de roles, la envíe a la vista y se muestren los roles disponibles.

Primero inyectamos el servicio **RolesService** en el controlador de usuarios.

```
@Controller
public class UsersController {

    @Autowired
    private RolesService rolesService;
```



En el método **GET /user/add** solicitamos la lista de roles y se la enviamos a la vista bajo el atributo **rolesList**. De esta forma la vista nos permitirá seleccionar un rol.

```
@RequestMapping(value = "/user/add")
public String getUser(Model model) {
    model.addAttribute("rolesList", rolesService.getRoles());
    return "user/add";
}
```

El **POST /user/add** no es necesario modificarlo, ya que el objeto usuario se construirá automáticamente con los datos recibidos desde el formulario. Basta con que reciba un dato con clave "role".

Sí que es necesario modificar el **POST /signup** porque ahora los usuarios deben tener un **rol** y el formulario de signup no nos permite seleccionarlo. Vamos a asignar a todos esos usuarios el **ROLE\_STUDENT**.

```
@RequestMapping(value = "/signup", method = RequestMethod.POST)
public String signup(@Validated User user, BindingResult result) {
    signUpFormValidator.validate(user, result);
    if (result.hasErrors()) {
        return "signup";
    }
    user.setRole(rolesService.getRoles()[0]);
    userService.addUser(user);
    securityService.autoLogin(user.getDni(), user.getPasswordConfirm());
    return "redirect:home";
}
```

## 2.5 Actualización de la vista /user/add

La vista **/user/add** debe mostrar los roles de la aplicación para que se puedan crear usuarios con diferentes roles.

Creemos un nuevo campo en el formulario con *name* **role**. Recuerda que, para que el objeto se construya automáticamente al enviar el formulario, el *name* del campo tiene que coincidir con el nombre del atributo en la clase **User**.

Creemos un *select* y recorremos la lista con los roles (**rolesList**) y para cada rol incluimos un elemento *option* (**rolesList** es simplemente una lista de cadenas de texto).

```
<div class="form-group">
  <label class="control-label col-sm-2" for="role">Rol:</label>
  <div class="col-sm-10">
    <select id="role" class="form-control" name="role">
      <option th:each="role : ${rolesList}"
        th:value="${role}"
        th:text="${role}">
      </option>
    </select>
  </div>
</div>
```



También sería necesario especificar un **password** para los usuarios que se añaden desde **/user/add**. Añadimos el campo con nombre **password**.

```
<div class="form-group">
  <label class="control-label col-sm-2" for="password">Password:</label>
  <div class="col-sm-10">
    <input type="password" id="password" class="form-control" name="password"
      placeholder="Introduzca el Password" />
  </div>
</div>
```

## 2.6 Carga del Rol en la autenticación

Modificamos el servicio **UserDetailsServiceImp** y en lugar de utilizar un rol fijo, cargamos el rol asociado al usuario.

```
@Override
public UserDetails loadUserByUsername(String dni) throws UsernameNotFoundException {
    User user = usersRepository.findByDni(dni);
    if (user == null) {
        throw new UsernameNotFoundException(dni);
    }
    Set<GrantedAuthority> grantedAuthorities = new HashSet<>();
    //grantedAuthorities.add(new SimpleGrantedAuthority("ROLE_ESTUDIANTE"));
    grantedAuthorities.add(new SimpleGrantedAuthority(user.getRole()));
    return new org.springframework.security.core.userdetails.User(
        user.getDni(), user.getPassword(), grantedAuthorities);
}
```

Reiniciamos la aplicación y desde el menú de gestión de usuarios, probamos dar de alta a un nuevo usuario e iniciamos sesión con el nuevo usuario.

**Nota:** Faltaría añadir un campo de confirmación del password (passwordConfirm) y realizar todas las validaciones correspondientes.

**Nota:** Incluir el siguiente Commit Message ->

**“SDI-IDGIT-4.2-SpringBoot-Roles”**

**OJO:** sustituir IDGIT por tu número asignado (p.e. 2223-101):

**“SDI-2223-101-4.2-SpringBoot-Roles”**

*(No olvides incluir los guiones y NO incluyas BLANCOS al principio o al final de la oración)*



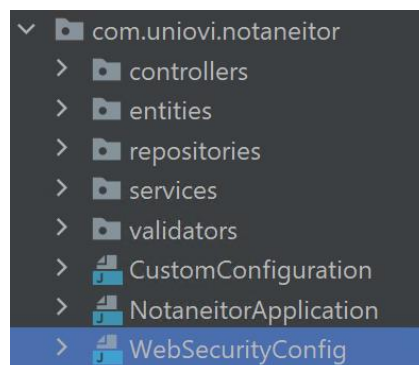
### 3 Control de acceso por roles

Para definir un sistema de control de acceso y autorización con Spring Web Security, debemos configurar el servicio de seguridad utilizando un conjunto de métodos. Con estos métodos permitimos o denegamos el acceso a usuarios. Estas acciones pueden especificarse en función de roles específicos (si es que la aplicación tiene roles, ya que algunas aplicaciones muy simples no los tienen).

Algunos métodos comunmente usados son: **hasRole**, **hasAnyRole**, **hasAuthority**, **hasAnyAuthority**, **permitAll**, **denyAll**, **isAnonymous**, **isRememberMe**, **isAuthenticated**<sup>1</sup>, **isFullyAuthenticated**<sup>2</sup>, **authentication**, etc.

#### 3.1.1 Configuración: autorización y control de acceso

Para definir el control de acceso tenemos que modificar el método **configure** la configuración de **WebSecurityConfig** e indicar las URLs a las que tienen acceso los usuarios autenticados.



A continuación, vamos a implementar la configuración relativa a las **notas**

- Los usuarios con cualquier rol pueden consultar la lista de notas, esto implica a las URLs **/mark/list**, **/mark/update** y **/mark/details/\***.
- Solo un usuario con **ROLE\_PROFESSOR** puede añadir **/mark/add**, eliminar **/mark/delete/\*** y modificar **/mark/edit/\*** notas.

Para indicar varios roles, debemos usar el metodo **hasAnyAuthority()**. Para indicar un único rol, podemos usar el método **hasAuthority()**.

**URLs:** las URLs de **detalles**, **eliminar** y **modificar**, incluyen parámetros y son de tipo **/mark/edit/34**. Para especificar URLs utilizamos expresiones regulares con **\*\*** **/mark/edit/\*\***. Esto nos servirá para especificar cualquier URL que comience por **/mark/edit/**

<sup>1</sup> Devuelve True si el usuario principal no es anónimo (Está autenticado)

<sup>2</sup> Devuelve True si el usuario principal se ha autenticado o es un usuario remember-me (credenciales guardada en el navegador)



**De específico a genérico:** cuando especificamos los permisos debemos empezar por los permisos más específicos (por ejemplo, los relativos a ROLE\_PROFESSOR, puede añadir, modificar y borrar)

Acabamos por los permisos más **genéricos**, como el resto de URLs relativas a mark son varias (/mark/list, /mark/update, /mark/details/\*) las podemos encapsular en la expresión regular **/mark/\*\***

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http
        .csrf().disable()
        .authorizeRequests()
            .antMatchers("/css/**", "/images/**", "/script/**", "/", "/signup", "/login/**").permitAll()
            .antMatchers("/mark/add").hasAuthority("ROLE_PROFESSOR")
            .antMatchers("/mark/edit/**").hasAuthority("ROLE_PROFESSOR")
            .antMatchers("/mark/delete/**").hasAuthority("ROLE_PROFESSOR")
            .antMatchers("/mark/**").hasAnyAuthority("ROLE_STUDENT", "ROLE_PROFESSOR", "ROLE_ADMIN")
        .anyRequest().authenticated()
    .and()
        .formLogin()
            .loginPage("/login")
            .permitAll()
            .defaultSuccessUrl("/home")
        .and()
        .logout()
            .permitAll();
}
```

**Nota:** El orden de declaración es muy importante. Si hubiésemos añadido en primer lugar:

```
.antMatchers("/mark/**").hasAnyAuthority("ROLE_STUDENT", "ROLE_PROFESSOR", "ROLE_ADMIN")
```

Estos perfiles tendrían acceso a todas las URLs que empiezan con /mark/ incluyendo las /mark/edit/\*, etc.

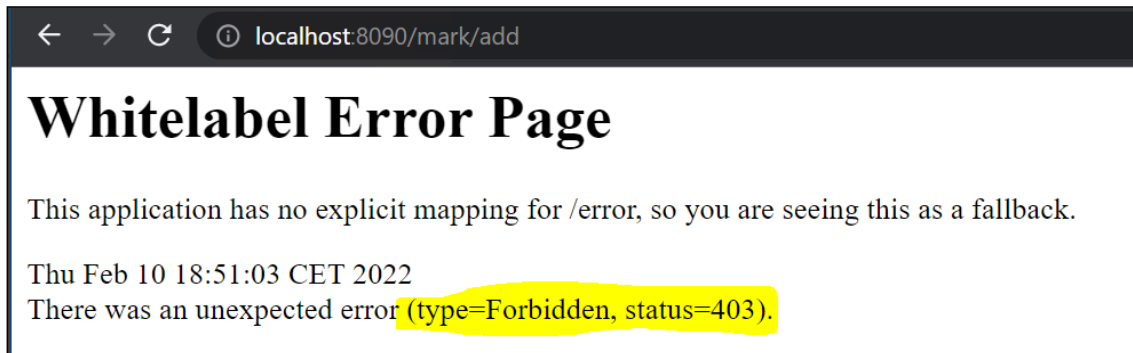
Implementamos ahora la configuración relativa a usuarios

- Solo un ROLE\_ADMIN puede acceder a todo lo relativo a la gestión de usuarios **/user/\*\***

```
.authorizeRequests()
    .antMatchers("/css/**", "/images/**", "/script/**", "/", "/signup", "/login/**").permitAll()
    .antMatchers("/mark/add").hasAuthority("ROLE_PROFESSOR")
    .antMatchers("/mark/edit/**").hasAuthority("ROLE_PROFESSOR")
    .antMatchers("/mark/delete/**").hasAuthority("ROLE_PROFESSOR")
    .antMatchers("/mark/**").hasAnyAuthority("ROLE_STUDENT", "ROLE_PROFESSOR", "ROLE_ADMIN")
    .antMatchers("/user/**").hasAnyRole("ADMIN")
    .anyRequest().authenticated()
```



Ejecutamos la aplicación y nos autenticamos con DNI: 99999990A password: 123456 (tiene ROLE\_STUDENT) y vamos a probar a añadir una nota. La aplicación nos muestra un error de acceso: **“Forbidden”** o **“Access is Denied”**.



**Nota:** *Roles y authorities* son similares en Spring. La diferencia es que, a partir de Spring Security 4, **Roles** tiene un significado más semántico. Los roles tienen que comenzar con el prefijo ROLE\_. Si usamos el método `hasRole()` este lo añade automáticamente. Así que, *hasAuthority (“ROLE\_PROFESSOR”)* es similar a *hasRole (“PROFESSOR”)*.

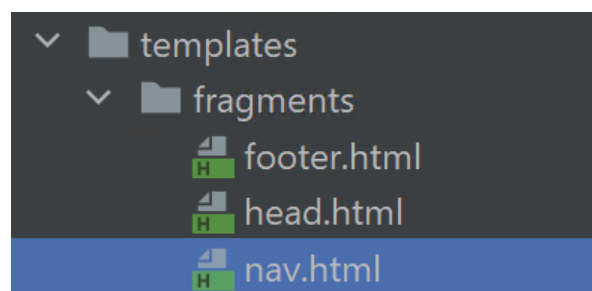
### 3.1.2 Vistas dependientes de roles

Vamos a **obtener el rol del usuario y mostrar unos elementos u otros en la vista en función del mismo**. Por ejemplo, es muy común que las opciones del menú de navegación sean diferentes para cada rol.

Como comentamos anteriormente, *Thymeleaf* proporciona muy buena integración con Spring Security (gracias a la dependencia *thymeleaf-extras-springsecurity4* que añadimos anteriormente).

Comenzamos modificando el `/fragments/nav.html` en función de la autorización del usuario se muestre un contenido u otro.

- `sec:authorize="isAuthenticated()"`: nos permite saber si el usuario está autenticado.
- `sec:authorize="hasRole('ROLE_ADMIN')"`: nos permite saber si el usuario autorizado tiene un rol específico.





- El contenido de nav se muestra solamente si el usuario está autenticado  
***sec:authorize="isAuthenticated()"***
- En el menú desplegable de “gestión de notas” la opción de “Agregar Nota” solo se muestra si el usuario autenticado tiene ROLE\_PROFESSOR  
***sec:authorize="hasRole('ROLE\_PROFESSOR')"***
- El menú desplegable de “gestión de usuarios” se muestra solo si el usuario autenticado tiene ROLE\_ADMIN,  
***sec:authorize="hasRole('ROLE\_ADMIN')"***.
- Añadimos el espacio de nombre de thymeleaf-extras-springsecurity para evitar errores de compilación en el HTML.

```
<!-- nav.html -->
<nav class="navbar navbar-expand-lg navbar-dark bg-primary" xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity4">
  <a class="navbar-brand" href="#"></a>
  
  <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#myNavbar"
    aria-controls="navbarColor02" aria-expanded="false" aria-label="Toggle navigation">
    <span class="navbar-toggler-icon"></span>
  </button>
  <div class="collapse navbar-collapse" id="myNavbar">
    <ul class="navbar-nav mr-auto" sec:authorize="isAuthenticated()">
      <li class="nav-item">
        <a class="nav-link" href="/home">Home<span class="sr-only">{current}</span></a>
      </li>
      <li class="nav-item dropdown active">
        <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button" data-toggle="dropdown"
          aria-haspopup="true" aria-expanded="false">Gestión de notas
        </a>
        <div class="dropdown-menu" aria-labelledby="navbarDropdown">
          <a class="dropdown-item" href="/mark/list">Ver Notas</a>
          <a class="dropdown-item" href="/mark/add" sec:authorize="hasRole('ROLE_PROFESSOR')">Agregar Nota</a>
        </div>
      </li>
      <li class="nav-item dropdown" sec:authorize="hasRole('ROLE_ADMIN')">
        <a class="nav-link dropdown-toggle" href="#" id="userDropdown" role="button" data-toggle="dropdown"
          aria-haspopup="true" aria-expanded="false">Gestión de usuarios
        </a>
        <div class="dropdown-menu" aria-labelledby="navbarDropdown">
          <a class="dropdown-item" href="/user/add">Agregar usuario</a>
          <a class="dropdown-item" href="/user/list">Ver usuarios</a>
        </div>
      </li>
      <li class="nav-item dropdown" sec:authorize="hasRole('ROLE_ADMIN')">
        <a class="nav-link dropdown-toggle" href="#" id="profesorDropdown" role="button" data-toggle="dropdown"
          aria-haspopup="true" aria-expanded="false">Gestión de profesores
        </a>
        <div class="dropdown-menu" aria-labelledby="navbarDropdown">
          <a class="dropdown-item" href="/profesor/list">Ver profesores</a>
        </div>
      </li>
    </ul>
    <ul class="navbar-nav justify-content-end">
      <li class="nav-item dropdown">
        <a class="nav-link dropdown-toggle" href="#" id="btnLanguage" role="button" data-toggle="dropdown"
          aria-haspopup="true" aria-expanded="false">
          <span th:text="#{language.change}"></span>
        </a>
      </li>
    </ul>
  </div>
</nav>
```





```
</a>
<div id="languageDropDownMenuButton" class="dropdown-menu" aria-labelledby="navbarDropDown">
  <a class="dropdown-item" id="btnEnglish" value="EN">
    
    <span th:text="#{{language.en}}">Inglés</span>
  </a>
  <a class="dropdown-item" id="btnSpanish" value="ES">
    
    <span th:text="#{{language.es}}">Español</span>
  </a>
</div>
</li>
```

Mostramos la opción de **Identificarse** y **Registrarse** solo si no hay un usuario autenticado. Si hay un usuario autenticado mostramos la opción de **Desconectar**.

```
<li sec:authorize="!isAuthenticated()" class="nav-item">
  <a class="nav-link" href="/signup" th:text="#{signup.message}">
    <i class="fas fa-user-alt" style="font-size:16px;"></i>
    <span>Registrarse</span>
  </a>
</li>
<li sec:authorize="!isAuthenticated()" class="nav-item">
  <a class="nav-link" href="/login" th:text="#{login.message}">
    <i class="fas fa-sign-in-alt" style="font-size:16px;"></i>
    <span>Identificarse</span>
  </a>
</li>
<li sec:authorize="isAuthenticated()" class="nav-item">
  <a class="nav-link" href="/logout">
    <i class="fas fa-sign-out-alt" style="font-size:16px;"></i>
    <span>Desconectar</span>
  </a>
</li>
</ul>
</div>
</nav>
```

**Nota:** No hace falta implementar `/logout` ya que se trata de la URL estándar en Spring Security. En **WebSecurityConfig** especificamos que se permitía el logout.

```
.formLogin()
  .loginPage("/login")
  .permitAll()
  .defaultSuccessUrl("/home")
  .and()
  .logout()
  .permitAll();
```

Hay que agregar mediante un Bean el dialecto *SpringSecurityDialect* para que con el motor de plantillas se puedan usar los atributos **sec:** \* y los objetos de utilidad de Thymeleaf en las vistas. Añadimos el Bean en la clase **WebSecurityConfig**.

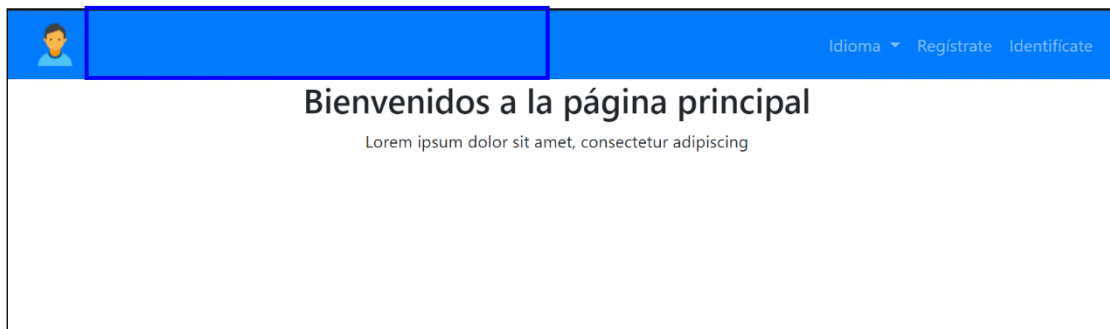
```
@Bean
public SpringSecurityDialect securityDialect() {
  return new SpringSecurityDialect();
}
```





**Nota:** En algunas versiones de Spring Boot no es necesario definir este Bean, ya que la dependencia lo configura y lo inyecta automáticamente.

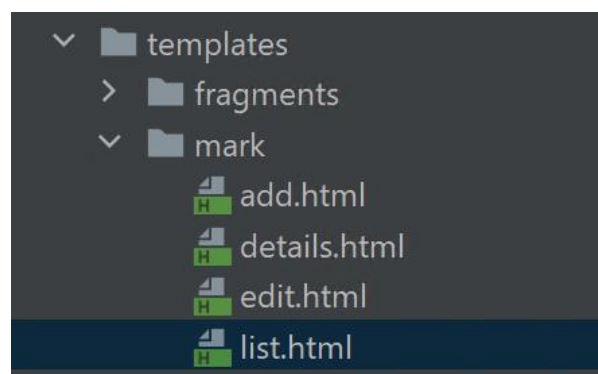
Deplegamos la aplicación y probamos los cambios realizados en la aplicación.



Nos identificamos con el usuario DNI: 99999990A y password: 123456 que tiene ROLE\_STUDENT.



Ahora vamos a modificar la vista `/mark/list` para que los enlaces de **editar** y **borrar** solo estén visibles para usuarios con rol de **PROFESSOR**. Se puede observar que se pueden usar las dos nomenclaturas: `hasRole("ROLE_PROFESSOR")` y `hasRole("PROFESSOR")`





```
<div class="container" id="main-container">
  <h2>Notas</h2>
  <p>Las notas que actualmente figuran en el sistema son las siguientes:</p>
  <button type="button" id="updateButton" class="btn btn-primary">Actualizar</button>
  <script>
    $('#updateButton').click(function () {
      $('#tableMarks').load('/mark/list/update');
    });
  </script>
  <div class="table-responsive">
    <table class="table table-hover" th:fragment="tableMarks" id="tableMarks">
      <thead>
        <tr>
          <th scope="col">id</th>
          <th scope="col">Descripción</th>
          <th scope="col">Puntuación</th>
          <th scope="col"></th>
          <th scope="col"></th>
          <th scope="col"></th>
        </tr>
      </thead>
      <tbody>
        <tr th:each="mark : ${markList}">
          <td scope="row" th:text="${mark.id}"> 1</td>
          <td th:text="${mark.description}"> Ejercicio 1</td>
          <td th:text="${mark.score}">10</td>
          <td><a th:href="'${'/mark/details/' + mark.id}">detalles</a></td>
          <td><a sec:authorize="hasRole('ROLE_PROFESSOR')" th:href="'${'/mark/edit/' +
mark.id}">modificar</a></td>
          <td><a sec:authorize="hasRole('PROFESSOR')" th:href="'${'/mark/delete/' +
mark.id}">eliminar</a>
          </td>
        </tr>
      </tbody>
    </table>
    <div th:if="${#lists.isEmpty(markList)}"> No marks </div>
  </div>
</div>
```

**Nota:** Añadimos el espacio de nombre de thymeleaf-extras-springsecurity para evitar errores de compilación en el HTML.

```
<html lang="en" xmlns="http://www.w3.org/1999/xhtml"
  xmlns:th="http://thymeleaf.org"
  xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity4">
```



Ejecutamos la aplicación y comprobamos desde el menú **Gestión de notas** → **Ver notas** que a los usuarios con `ROLE_STUDENT` no se les muestran los botones de **editar** y **eliminar**.

id	Descripción	Puntuación	
2	Nota A3	7.0	<a href="#">detalles</a>
3	Nota A4	6.5	<a href="#">detalles</a>
4	Nota A1	10.0	<a href="#">detalles</a>
5	Nota A2	9.0	<a href="#">detalles</a>

**Nota:** Esta no es la única vista de la aplicación en la que estamos mostrando los enlaces de editar y eliminar nota. Deberíamos hacerlo condicional al rol en todas las partes (`/home`).

Finalmente, podemos eliminar el uso de la etiqueta `[[${#httpServletRequest.remoteUser}]]` empleada en `home.html`, cambiándola por la etiqueta `sec:authentication="principal.username"` (ambas obtienen el mismo parámetro)

```
<div class="container" style="text-align: center">
  <h2 th:text="#{welcome.message}"></h2>
  <h3>Esta es una zona privada la web</h3>
  <p>Usuario Autenticado como :
    <del>[[${#httpServletRequest.remoteUser}]]</del>
    <b th:inline="text" sec:authentication="principal.username"></b>
  </p>
```



Bienvenidos a la página principal

Esta es una zona privada la web

Usuario Autenticado como : 99999990A

Notas del usuario

id	Descripción	Puntuación			
2	Nota A3	7.0	<a href="#">detalles</a>	<a href="#">modificar</a>	<a href="#">eliminar</a>



### 3.1.3 Accion asociada al perfil de un alumno

Vamos a permitir que los alumnos puedan modificar una propiedad **reenvío** de sus notas. Utilizarán esta propiedad para simular que quieren reenviar el trabajo evaluado.

#### 3.1.3.1 Actualizar entidad Mark

Incluimos la nueva propiedad en la entidad, por defecto tendrá el valor false.

```
@Entity
public class Mark {
    @Id
    @GeneratedValue
    private Long id;
    private String description;
    private Double score;
    private Boolean resend = false;
```

Implementamos los métodos get y set.

```
public Boolean getResend() {
    return resend;
}

public void setResend(Boolean resend) {
    this.resend = resend;
}
```

#### 3.1.3.2 Actualizar MarkRepository

Implementamos un nuevo metodo que actualice la propiedad **resend** de la nota. Los métodos que modifican registros deben incluir la anotación **@Modifying**. Además, la operación debería realizarse de forma transaccional **@Transactional**.

```
...
import org.springframework.data.jpa.repository.Modifying;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.data.jpa.repository.Query;

public interface MarksRepository extends CrudRepository<Mark, Long> {
    @Modifying
    @Transactional
    @Query("UPDATE Mark SET resend = ?1 WHERE id = ?2")
    void updateResend(Boolean resend, Long id);
}
```

#### 3.1.3.3 Actualizar MarkService

En el servicio incluimos un nuevo método que nos permita modificar la propiedad **revised**. Este servicio hará uso del repositorio implementado anteriormente.

```
public void setMarkResend(boolean revised, Long id) {
    marksRepository.updateResend(revised, id);
}
```



### 3.1.3.4 Controlador MarksController

Incluimos respuesta a la URL `/mark/{id}/resend` para poner a **true** el atributo `resend` de una nota y otra URL `/mark/{id}/noresend` para ponerlo a **false**

```
@RequestMapping(value = "/mark/{id}/resend", method = RequestMethod.GET)
public String setResendTrue(@PathVariable Long id) {
    marksService.setMarkResend(true, id);
    return "redirect:/mark/list";
}

@RequestMapping(value = "/mark/{id}/noresend", method = RequestMethod.GET)
public String setResendFalse(@PathVariable Long id) {
    marksService.setMarkResend(false, id);
    return "redirect:/mark/list";
}
```

### 3.1.3.5 Actualizar vista /mark/list

Si el usuario está autenticado como `ROLE_STUDENT` mostramos el atributo **resend** y la opción con los dos enlaces para poder modificarlo. Incluimos los elementos resaltados dentro del elemento `<td>` donde se encuentra el enlace modificar, en el fichero `mark/list.html`.

```
<td><a sec:authorize="hasRole('ROLE_PROFESSOR')" th:href="'${'/mark/edit/' + mark.id}">modificar</a>
<div sec:authorize="hasRole('ROLE_STUDENT')">
    <div th:if="${mark.resend}">
        <a th:href="'${'/mark/' + mark.id + '/noresend'}">Reenviar</a>
    </div>
    <div th:unless="${mark.resend}">
        <a th:href="'${'/mark/' + mark.id + '/resend'}">No reenviar</a>
    </div>
</div>
</td>
```

Ejecutamos la aplicación, nos identificamos con DNI: 99999990A y password: 123456, accedemos a la lista de notas y comprobamos el funcionamiento. Hacemos clic sobre el enlace para cambiar de estado.



Home

Gestión de notas

Idioma

Desconectar

# Notas

Las notas que actualmente figuran en el sistema son las siguientes:

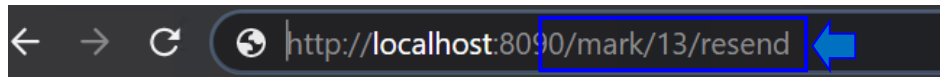
Actualizar

id	Descripción	Puntuación	
2	Nota A2	9.0	<a href="#">detalles</a> <a href="#">Reenviar</a>
3	Nota A3	7.0	<a href="#">detalles</a> <a href="#">No reenviar</a>
4	Nota A4	6.5	<a href="#">detalles</a> <a href="#">Reenviar</a>
5	Nota A1	10.0	<a href="#">detalles</a> <a href="#">No reenviar</a>



### 3.1.3.6 Comprobación de seguridad: propietario del recurso

**Es importante validar que el usuario que está modificando la propiedad reenvío es el propietario de la nota.** De lo contrario, cualquier usuario que conozca la id de la nota podría cambiar el valor de la propiedad.



**Nota:** podemos comprobar que desde el enlace podemos modificar el estado del reenvío en una nota que no es del usuario en sesión.

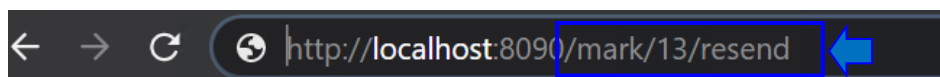
Debemos hacer comprobaciones adicionales en el servicio **MarksService**, concretamente en el método **setMarkResend()**.

Utilizamos el **SecurityContextHolder** para obtener los datos del usuario autenticado, el atributo name del usuario autenticado se corresponde con el **dni**. Obtenemos la nota que se está intentando modificar y comprobamos si el **dni** de su usuario coincide con el **dni** del usuario autenticado. Solamente en ese caso realizamos la modificación.

```
public void setMarkResend(boolean revised, Long id) {  
    Authentication auth = SecurityContextHolder.getContext().getAuthentication();  
    String dni = auth.getName();  
  
    Mark mark = marksRepository.findById(id).get();  
  
    if(mark.getUser().getDni().equals(dni) ){  
        marksRepository.updateResend(revised, id);  
    }  
}
```

Importamos los paquetes correspondientes, luego ejecutamos la aplicación y comprobamos que no podemos modificar notas que no pertenecen al usuario autenticado.

```
import org.springframework.security.core.Authentication;  
import org.springframework.security.core.context.SecurityContextHolder;
```



**Nota:** Ahora podemos comprobar que desde el enlace **NO** podemos modificar el estado del reenvío en una nota que no es de este usuario.



### 3.1.4 Actualización de la tabla como fragmento

La tabla en la que se muestran las notas en la vista `/mark/list` está identificada como un fragmento.

```
<div class="table-responsive">
  <table class="table table-hover" th:fragment="tableMarks" id="tableMarks">
    <thead>
      <tr>
        <th scope="col">id</th>
        <th scope="col">Descripción</th>
        <th scope="col">Puntuación</th>
      </tr>
    </thead>
  </table>
</div>
```

En prácticas anteriores habíamos visto un ejemplo sobre cómo actualizar únicamente un fragmento de una vista. Esta funcionalidad se encontraba en el botón `update` de esta misma vista. La URL `/mark/list/update` devuelve únicamente el fragmento de la tabla actualizado y con la función `load` de JavaScript insertábamos el retorno de esa URL en un componente HTML de la página cuya id fuese `tableMarks`

```
<p>Las notas que actualmente figuran en el sistema son las siguientes:</p>
<button type="button" id="updateButton" class="btn btn-primary">Actualizar</button>
<script>
  $("#updateButton").click(function () {
    $("#tableMarks").load('/mark/list/update');
  });
</script>
```

Vamos a hacer que los botones de **reenvío** y **no reenvío** actualicen solamente el fragmento de la tabla, en lugar de toda la vista.

#### Puntos que debemos tener en cuenta:

- **Botones.** Debemos crear los botones de **reenvío** y **no reenvío** para cada nota. Por lo tanto, cada botón debe tener una **id** diferente. Podemos utilizar el **mark.id** para crear los id de los botones. La propiedad **th:id** nos permite asignar ids combinando texto y propiedades, por ejemplo: **th:id="'resendButton' + mark.id"**
- **Fragmentos de script declaración.** Necesitamos crear scripts que realicen las peticiones a las URLs: **mark/{id}/resend** y **/mark/{id}/noresend**. Los scripts que utilizan elementos de Thymeleaf deben ser declarados de una forma muy específica, incluyendo la etiqueta **<script th:inline="javascript">** y encapsulando todo el script en un **/\*<![CDATA[\*]**
- **Fragmentos de script y atributos de Thymeleaf.** Para insertar el valor de un atributo de Thymeleaf en un script utilizamos la sintaxis **[[\${<nombre\_atributo>}]]**. Por ejemplo: **[[\${mark.id}]]**
- **Fragmentos de script, peticiones:** para realizar una petición `get` en jQuery usamos la función `$.get(URL, función de callback)`. La función de callback se ejecuta cuando la petición finaliza, una vez finalizada la petición, podemos volver a cargar el fragmento de la tabla.



```
<tbody>
<tr th:each="mark : ${markList}">
  <td scope="row" th:text="${mark.id}"> 1</td>
  <td th:text="${mark.description}"> Ejercicio 1</td>
  <td th:text="${mark.score}">10</td>
  <td><a th:href="'${mark.id}' + mark.id">detalles</a></td>
  <td><a sec:authorize="hasRole('ROLE_PROFESSOR')" th:href="'${mark.id}' + mark.id">modificar</a>
    <div sec:authorize="hasRole('ROLE_STUDENT')">
      <div th:if="${mark.resend}">
        <!-- <a th:href="'${mark.id}' + mark.id + '/noresend'">Reenviar</a> -->
        <button type="button" th:id="'${mark.id}' + mark.id"
          class="btn btn-success">Reenviar</button>
        <script th:inline="javascript">
          /*<![CDATA[*/
          $( "#resendButton" + "[[${mark.id}]]").click(function() {
            $.get( "/mark/[[${mark.id}]]/noresend", function( data ) {
              $("#tableMarks").load('/mark/list/update');
            });
          });
          /*]]>*/
        </script>
      </div>
      <div th:unless="${mark.resend}">
        <!-- <a th:href="'${mark.id}' + mark.id + '/resend'">No reenviar</a> -->
        <button type="button" th:id="'${mark.id}' + mark.id"
          class="btn btn-info">No reenviar</button>
        <script th:inline="javascript">
          /*<![CDATA[*/
          $( "#noresendButton" + "[[${mark.id}]]").click(function() {
            $.get( "/mark/[[${mark.id}]]/resend", function( data ) {
              $("#tableMarks").load('/mark/list/update');
            });
          });
          /*]]>*/
        </script>
      </div>
    </div>
  </td>
  <td><a sec:authorize="hasRole('ROLE_PROFESSOR')" th:href="'${mark.id}' + mark.id">eliminar</a>
  </td>
</tr>
</tbody>
```

Ejecutamos la aplicación y comprobamos el resultado. Actualmente aparecen todas las notas de todos los usuarios. Sin embargo, solamente se actualizarán los reenvíos de aquellas notas que pertenezcan al usuario en sesión.





## Notas

Las notas que actualmente figuran en el sistema son las siguientes:

Actualizar

id	Descripción	Puntuación		
2	Nota A2	9.0	<a href="#">detalles</a>	<button>Reenviar</button>
3	Nota A3	7.0	<a href="#">detalles</a>	<button>No reenviar</button>
4	Nota A4	6.5	<a href="#">detalles</a>	<button>No reenviar</button>

**Nota: Incluir el siguiente Commit Message ->**

**“SDI-IDGIT-4.3-SpringBoot-Control de acceso por roles”**

**OJO: sustituir IDGIT por tu número asignado (p.e. 2223-101):**

**“SDI-2223-101-4.3-SpringBoot-Control de acceso por roles”**

*(No olvides incluir los guiones y NO incluyas BLANCOS al principio o al final de la oración)*



## 4 Consultas

En esta sección veremos cómo podemos realizar consultas a la base de datos en función de unos criterios y mostrar únicamente los resultados que coincidan con ese criterio.

Los repositorios JPA admiten la definición de consultas de forma sencilla. Una opción, aunque no la única, para realizar estas consultas es utilizar la anotación **@Query**.

### 4.1 Consultar notas

Actualmente la aplicación muestra todas las notas en **/mark/list**. Vamos a modificar la aplicación para:

- **Mostrar solo las notas del usuario autenticado**, si este tiene **ROLE\_STUDENT**
- **Mostrar todas las notas de la aplicación**, si el usuario autenticado tiene **ROLE\_PROFESSOR**

#### 4.1.1 Actualizar MarksRepository

Accedemos a **MarksRepository** y añadimos el método **findAllByUser(User user)**. Al tratarse de una función que usa el patrón de nombrado **findAllBy<parámetro>** no es necesario especificar la consulta SQL con el atributo **@Query**.

```
import java.util.List;

public interface MarksRepository extends CrudRepository<Mark, Long> {
    List<Mark> findAllByUser(User user);
}
```

**Nota:** Sí que sería necesario utilizar el atributo **@Query** si quieramos hacer una consulta más específica que no sea simplemente un select de un parámetro. Por **ejemplo**, solo las notas con un valor de 5 o más. Es decir, aplicar filtros con WHERE:

```
@Query("SELECT r FROM Mark r WHERE r.user = ?1 AND r.score >= 5 ")
List<Mark> findAllPassedByUser(User user);
```

#### 4.1.2 Actualizar MarksService

Accedemos a **MarksService** y añadimos un método que reciba un **usuario**. En función del rol del usuario, devolverá todas las notas de la aplicación (**ROLE\_PROFESSOR**) o solo las notas relativas al usuario (**ROLE\_STUDENT**).

```
public List<Mark> getMarksForUser(User user) {
    List<Mark> marks = new ArrayList<>();
    if (user.getRole().equals("ROLE_STUDENT")) {
        marks = marksRepository.findAllByUser(user);
    }
    if (user.getRole().equals("ROLE_PROFESSOR")) {
        marks = getMarks();
    }
    return marks;
}
```



### 4.1.3 Modificar MarksController

Accedemos a **MarksController** y modificamos la respuesta a la URL **/mark/list**. Para obtener el usuario autenticado podemos hacerlo incluyendo el objeto de tipo **Principal** como parámetro del método. La variable principal tiene el nombre de la autenticación (que coincide con el dni del usuario). Obtenemos el usuario al que pertenece el DNI y llamamos al servicio **getMarksForUser (user)**;

```
@RequestMapping("/mark/list")
public String getList(Model model, Principal principal){
    String dni = principal.getName(); // DNI es el name de la autenticación
    User user = userService.getUserByDni(dni);
    model.addAttribute("markList", marksService.getMarksForUser(user));
    return "mark/list";
}
```

Modificamos de igual modo la respuesta a la URL **/mark/list/update** la cual se encarga de actualizar de forma dinámica la lista de notas.

```
@RequestMapping("/mark/list/update")
public String updateList(Model model, Principal principal){
    String dni = principal.getName(); // DNI es el name de la autenticación
    User user = userService.getUserByDni(dni);
    model.addAttribute("markList", marksService.getMarksForUser(user));
    return "mark/list :: tableMarks";
}
```

Nota: hay que importar los paquetes:

```
import java.security.Principal;
import com.uniovi.notaneitor.entities.User;
```

**Nota:** Anteriormente habíamos visto otra forma de obtener el usuario autenticado, a través del **SecurityContextHolder**, ambas son válidas. No obstante, el **SecurityContextHolder** se puede utilizar desde cualquier punto de la aplicación, no solo desde los controladores.

```
@RequestMapping("/mark/list")
public String getList(Model model){
    Authentication auth = SecurityContextHolder.getContext().getAuthentication();
    String dni = auth.getName();
    User user = userService.getUserByDni(dni);
    model.addAttribute("markList", marksService.getMarksForUser(user));
    return "mark/list";
}
```

Guardamos los cambios y accedemos a la aplicación con los siguientes perfiles:

- DNI: 99999990A password: 123456 -> ROLE\_STUDENT
- DNI: 99999977E password: 123456 -> ROLE\_PROFESSOR



Si accedemos a **/mark/list** se deberían mostrar solo las notas propias en el caso del estudiante y todas las notas en el caso del profesor (que no podrá reenviar).

Actualizar				
id	Descripción	Puntuación		
2	Nota A4	6.5	<a href="#">detalles</a>	No reenviar
3	Nota A3	7.0	<a href="#">detalles</a>	No reenviar
4	Nota A2	9.0	<a href="#">detalles</a>	No reenviar
5	Nota A1	10.0	<a href="#">detalles</a>	No reenviar

La aplicación funciona correctamente, pero observamos un comportamiento inadecuado cuando pulsamos el botón de **Reenviar**: se cambia el orden de los registros (JPA nos devuelve los registros por orden de modificación. Cuando un registro se modifica se devuelve primero, alterando el orden de las notas).

id	Descripción	Puntuación		
3	Nota A3	7.0	<a href="#">detalles</a>	No reenviar
4	Nota A2	9.0	<a href="#">detalles</a>	No reenviar
5	Nota A1	10.0	<a href="#">detalles</a>	No reenviar
2	Nota A4	6.5	<a href="#">detalles</a>	Reenviar

Podemos solucionar este problema especificando la **@Query** en el repositorio **MarksRepository**, haciendo uso de ORDER BY:

```
public interface MarksRepository extends CrudRepository<Mark, Long> {  
  
    @Query("SELECT r FROM Mark r WHERE r.user = ?1 ORDER BY r.id ASC")  
    List<Mark> findAllByUser(User user);  
}
```

**Nota: Incluir el siguiente Commit Message ->**

**“SDI-IDGIT-4.4-SpringBoot-Consulta”**

**OJO: sustituir IDGIT por tu número asignado (p.e. 2223-101):**

**“SDI-2223-101-4.4-SpringBoot-Consulta”**

*(No olvides incluir los guiones y NO incluyas BLANCOS al principio o al final de la oración)*



## 5 Búsqueda

Vamos a incluir un formulario de búsqueda para poder encontrar **notas** en base a unos criterios. Realizaremos una consulta en el repositorio y mostraremos las notas que cumplan cierto criterio, permitiendo realizar búsquedas por: **nombre**, **DNI** o **descripción**. Las búsquedas que vamos a realizar en esta funcionalidad van a ser parciales. El contenido buscado no va a tener que coincidir exactamente con el contenido del campo.

### 5.1 Actualizar MarksRepository

Modificamos el repositorio **MarksRepository** añadiendo los siguientes métodos:

- **searchByDescriptionAndName** (*String searchText*), devuelve notas de toda la aplicación cuando el texto buscado coincide con el nombre del usuario o la descripción de la nota.
- **searchByDescriptionNameAndUser** (*String searchText, User user*), devuelve notas relacionadas con el usuario enviado como parámetro, cuando el texto buscado coincide con el nombre del usuario o la descripción de la nota.

Hemos pasado todos los textos a minúsculas con la función **LOWER** para que la aplicación no sea sensible a mayúsculas/minúsculas.

```
@Query('Select r FROM Mark r WHERE (LOWER(r.description) LIKE LOWER(?) OR LOWER(r.user.name) LIKE LOWER(?))')
List<Mark> searchByDescriptionAndName(String searchText);

@Query("SELECT r FROM Mark r WHERE (LOWER(r.description) LIKE LOWER(?) OR LOWER(r.user.name) LIKE LOWER(?)) AND r.user = ?2")
List<Mark> searchByDescriptionNameAndUser(String searchText, User user);
```

### 5.2 Actualizar MarksService

Actualizamos el servicio **MarksService**, añadimos el método **searchMarksByDescriptionAndNameForUser** (*String searchText, User user*) se encargará de realizar una búsqueda en las notas:

- Las notas del propio usuario si el usuario autenticado es **ROLE\_STUDENT**
- Las notas de todos los usuarios si el usuario autenticado es **ROLE\_PROFESSOR**.

```
public List<Mark> searchMarksByDescriptionAndNameForUser(String searchText, User user) {
    List<Mark> marks = new ArrayList<>();
    if (user.getRole().equals("ROLE_STUDENT")) {
        marks = marksRepository.searchByDescriptionNameAndUser(searchText, user);
    }
    if (user.getRole().equals("ROLE_PROFESSOR")) {
        marks = marksRepository.searchByDescriptionAndName(searchText);
    }
    return marks;
}
```



### 5.3 Actualizar MarksController

Accedemos a **MarksController** y actualizamos el método que responde a la URL **/mark/list**. Vamos a permitir que esta URL reciba un parámetro con clave **searchText** que será opcional.

- Si recibimos el parámetro **searchText** utilizamos el servicio **searchMarksByDescriptionAndNameForUser**
- Si **no** recibimos ningún parámetro **searchText**, utilizamos el servicio que devuelve todas las notas: **getMarksForUser**

```
@RequestMapping("/mark/list")
public String getList(Model model, Principal principal,
    @RequestParam(value = "", required = false) String searchText) {

    String dni = principal.getName(); // DNI es el name de la autenticación
    User user = userService.getUserByDni(dni);
    if (searchText != null && !searchText.isEmpty()) {
        model.addAttribute("markList",
            marksService.searchMarksByDescriptionAndNameForUser(searchText, user));
    } else {
        model.addAttribute("markList", marksService.getMarksForUser(user));
    }

    return "mark/list";
}
```

### 5.4 Actualizar la vista mark/list

Ahora modificamos el fichero **mark/list** para incluir un formulario de búsqueda. Se basará en un único campo con nombre **searchText** que enviará una petición GET contra **/mark/list**

```
<form class="form-inline" action="/mark/list">
  <div class="form-group">
    <input name="searchText" type="text" class="form-control" size="50"
      placeholder="Buscar por descripción o nombre del alumno">
  </div>
  <button type="submit" class="btn btn-primary">Buscar</button>
</form>
```

Guardamos los cambios y accedemos a la aplicación con los siguientes perfiles:

- DNI: 99999990A password: 123456 -> ROLE\_STUDENT
- DNI: 99999977E password: 123456 -> ROLE\_PROFESSOR

Nota: Las búsquedas deben realizarse con cadenas exactas.



Home Gestión de notas ▾ Idioma ▾ Desconectar

## Notas

Las notas que actualmente figuran en el sistema son las siguientes:

Actualizar

id	Descripción	Puntuación	
3	Nota A1	10.0	<a href="#">detalles</a> <input type="button" value="No reenviar"/>

Home Gestión de notas ▾ Idioma ▾ Desconectar

## Notas

Las notas que actualmente figuran en el sistema son las siguientes:

Actualizar

id	Descripción	Puntuación	
2	Nota A2	9.0	<a href="#">detalles</a> <a href="#">modificar</a> <a href="#">eliminar</a>
3	Nota A1	10.0	<a href="#">detalles</a> <a href="#">modificar</a> <a href="#">eliminar</a>
4	Nota A4	6.5	<a href="#">detalles</a> <a href="#">modificar</a> <a href="#">eliminar</a>
5	Nota A3	7.0	<a href="#">detalles</a> <a href="#">modificar</a> <a href="#">eliminar</a>

## 5.5 Buscar por cadena contenida

Si quisiésemos que la búsqueda se base únicamente en si la cadena introducida se encuentra en la descripción o el nombre del usuario (sin necesidad de que sea una coincidencia exacta) podríamos utilizar los comodines de SQL `%searchText%`. Para lograrlo, vamos a **MarkService** e incluimos los comodines en el método `searchMarksByDescriptionAndNameForUser`.

```
public List<Mark> searchMarksByDescriptionAndNameForUser(String searchText, User user) {  
    List<Mark> marks = new ArrayList<Mark>();  
    searchText = "%" + searchText + "%";  
    if (user.getRole().equals("ROLE_STUDENT")) {  
        marks = marksRepository.searchByDescriptionNameAndUser(searchText, user);  
    }  
    if (user.getRole().equals("ROLE_PROFESSOR")) {  
        marks = marksRepository.searchByDescriptionAndName(searchText);  
    }  
    return marks;  
}
```



id	Descripción	Puntuación
3	Nota A1	10.0

Nota: Incluir el siguiente Commit Message ->

“SDI-IDGIT-4.5-SpringBoot-Búsqueda”

OJO: sustituir IDGIT por tu número asignado (p.e. 2223-101):

“SDI-2223-101-4.5-SpringBoot-Búsqueda”

*(No olvides incluir los guiones y NO incluyas BLANCOS al principio o al final de la oración)*

## 6 Paginación

Cuando una lista contiene muchos elementos, no suele ser aconsejable mostrarlos en una única página. Tampoco es eficiente solicitar al servidor/base de datos listas con cientos, miles o millones de recursos que realmente no van a ser utilizados. Por eso, es importante tener un sistema de paginación. Vamos a incluir un sistema de paginación para visualizar las notas.

### 6.1 Actualizar MarksRepository

Actualmente, los métodos definidos en *MarksRepository* devuelven listas de objetos de notas (marks). Como por ejemplo `List<Mark>` `findAllByUser(User user)`;

A partir de ahora, para devolver una lista de notas paginadas utilizaremos el paquete *org.springframework.data.domain.Page*. Solo tenemos que modificar el retorno de los métodos y utilizar `Page<Mark>`.

Todos los métodos van a recibir además un parámetro adicional `Pageable`

También tenemos que incluir el método `findAll()` (a pesar de que el *CrudRepository* ya nos lo da implementado) porque por defecto este método no retorna un `Page<Mark>`

Nota: Asegurarse de importar la clase `Pageable` correcta:  
`import org.springframework.data.domain.Pageable;`





```
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;

public interface MarksRepository extends CrudRepository<Mark, Long> {

    @Query("SELECT r FROM Mark r WHERE (LOWER(r.description) LIKE LOWER(?) OR LOWER(r.user.name) LIKE LOWER(?1))")
    Page<Mark> searchByDescriptionAndName(Pageable pageable, String searchText);

    @Query("SELECT r FROM Mark r WHERE (LOWER(r.description) LIKE LOWER(?) OR LOWER(r.user.name) LIKE LOWER(?1)) AND r.user = ?2 ")
    Page<Mark> searchByDescriptionNameAndUser(Pageable pageable, String searchText, User user);

    @Query("SELECT r FROM Mark r WHERE r.user = ?1 ORDER BY r.id ASC ")
    Page<Mark> findAllByUser(Pageable pageable, User user);

    Page<Mark> findAll(Pageable pageable);

    @Modifying
    @Transactional
    @Query("UPDATE Mark SET resend = ?1 WHERE id = ?2")
    void updateResend(Boolean resend, Long id);
}
```

## 6.2 Actualizar MarksService

Ahora tenemos que modificar el servicio **MarksService** encargado de gestionar las notas.

Modificamos los métodos que devolvían la lista de notas. El tipo de retorno pasa de ser **List<Mark>** a **Page<Mark>**. También deben recibir un parámetro de tipo **Pageable**.

**Nota:** Asegurarse de importar la clase **Pageable** correcta:  
**import org.springframework.data.domain.Pageable;**



```
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;

@Service
public class MarksService {

    @Autowired
    private HttpSession httpSession;

    @Autowired
    private MarksRepository marksRepository;

    public Page<Mark> getMarks(Pageable pageable) {
        Page<Mark> marks = marksRepository.findAll(pageable);
        return marks;
    }

    public Page<Mark> getMarksForUser(Pageable pageable, User user) {
        Page<Mark> marks = new PageImpl<Mark>(new LinkedList<Mark>());
        if (user.getRole().equals("ROLE_STUDENT")) {
            marks = marksRepository.findAllByUser(pageable, user);
        }
        if (user.getRole().equals("ROLE_PROFESSOR")) {
            marks = getMarks(pageable);
        }
        return marks;
    }

    public Page<Mark> searchMarksByDescriptionAndNameForUser(Pageable pageable,
        String searchText, User user) {
        Page<Mark> marks = new PageImpl<Mark>(new LinkedList<Mark>());
        searchText = "%" + searchText + "%";
        if (user.getRole().equals("ROLE_STUDENT")) {
            marks = marksRepository.
                searchByDescriptionNameAndUser(pageable, searchText, user);
        }
        if (user.getRole().equals("ROLE_PROFESSOR")) {
            marks = marksRepository.
                searchByDescriptionAndName(pageable, searchText);
        }
        return marks;
    }
}
```

### 6.3 Modificar MarksController

Debemos modificar en el controlador **MarksController**, para que todos los métodos que devuelven una **List<Mark>** devuelvan una lista paginada **Page<Mark>**. El objeto **Page<Mark>** contiene mucha información sobre el sistema de paginación (página actual, totales, etc.) y también la lista con las notas. Podemos acceder a ella utilizando el método



**getContent(); Utilizando este método obtenemos la lista de notas que las vistas están esperando.**

Los métodos afectados por la paginación también deberán recibir un parámetro **Pageable pageable**. Este parámetro sirve para gestionar en las URL los parámetros **page** y **size**, <http://localhost:8090/mark/list?page=1&size=5>. El parámetro **size** es opcional, si no lo incluimos usa uno por defecto.

**Nota:** Asegurarse de importar la clase **Pageable** correcta:  
**import org.springframework.data.domain.Pageable;**

```
@RequestMapping("/mark/list")
public String getList(Model model, Pageable pageable, Principal principal,
    @RequestParam(value = "", required = false) String searchText) {
    String dni = principal.getName(); // DNI es el name de la autenticación
    User user = userService.getUserByDni(dni);
    Page<Mark> marks = new PageImpl<Mark>(new LinkedList<Mark>());
    if (searchText != null && !searchText.isEmpty()) {
        marks = marksService.searchMarksByDescriptionAndNameForUser(pageable,
searchText, user);
    } else {
        marks = marksService.getMarksForUser(pageable, user);
    }

    model.addAttribute("markList", marks.getContent());

    return "mark/list";
}

@RequestMapping("/mark/list/update")
public String updateList(Model model, Pageable pageable, Principal principal) {
    String dni = principal.getName(); // DNI es el name de la autenticación
    User user = userService.getUserByDni(dni);
    Page<Mark> marks = marksService.getMarksForUser(pageable, user);
    model.addAttribute("markList", marks.getContent());
    return "mark/list :: tableMarks";
}
```

## 6.4 Actualizar la Configuración de la aplicación

Para que Spring sepa cómo convertir los parámetros GET **page** y **size** a objetos **Pageable**, debemos configurar un **HandlerMethodArgumentResolver**. Spring Data proporciona un **PageResearchResolver**, pero utiliza la antigua interfaz **ArgumentResolver** en lugar de la



nueva interfaz (Spring 3.1) **HandlerMethodArgumentResolver**.<sup>3</sup> Debemos añadir un nuevo método **addArgumentResolvers** a la clase **CustomConfiguration**,

Indicamos también los valores por defecto para Page y size que serán 0 y 5. (page=0 es la primera).

```
import org.springframework.data.web.PageableHandlerMethodArgumentResolver;
import org.springframework.web.method.support.HandlerMethodArgumentResolver;
import org.springframework.data.domain.PageRequest;

@Configuration
public class CustomConfiguration implements WebMvcConfigurer {

    @Override
    public void addArgumentResolvers(List<HandlerMethodArgumentResolver> argumentResolvers) {
        int page = 0;
        int size = 5;
        PageableHandlerMethodArgumentResolver resolver = new PageableHandlerMethodArgumentResolver();
        resolver.setFallbackPageable(PageRequest.of(page, size));
        argumentResolvers.add(resolver);
    }
}
```

En este punto, el sistema de paginación ya estaría activo (aunque todavía nos falta incluir los botones de acceso a las páginas en la vista). Podemos probarlo entrando con el perfil profesor (ya que puede visualizar muchas notas):

DNI: 99999977E password: 123456 -> ROLE\_PROFESSOR

Podemos probar varias URLs:

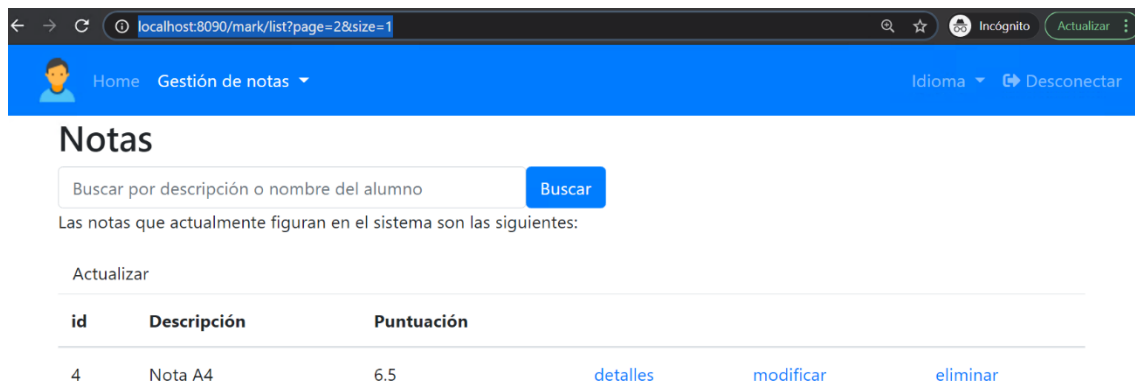
<http://localhost:8090/mark/list?page=2&size=1>

<http://localhost:8090/mark/list?page=2> (si no incluimos size usa uno por defecto: 5).

<http://localhost:8090/mark/list?page=3&size=2>

---

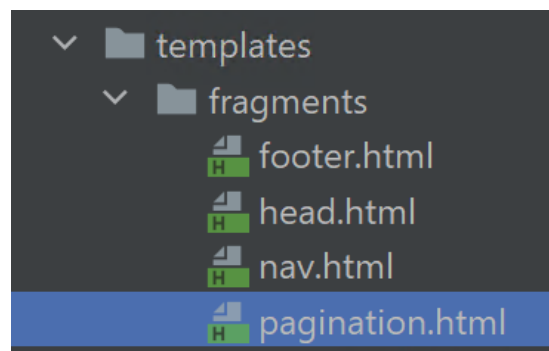
<sup>3</sup><https://www.javacodegeeks.com/2013/03/implement-bootstrap-pagination-with-spring-data-and-thymeleaf.html>



Existe un problema con el botón de actualizar que resolveremos al editar las vistas. Se debe a que las peticiones a `/mark/list/update` actualmente no reciben parámetros `page` y `size` (estábamos realizando estas peticiones desde JavaScript sin incluir los parámetros `page` y `size`). Sería necesario incluir los mismos que se recibieron en la petición actual)

## 6.5 Actualización de las vistas

Vamos a crear un nuevo fragmento ***pagination.html*** con el selector de las páginas por si más adelante nos interesa incluir paginación en otras partes de la aplicación.



Esta vista va a recibir como parámetro el objeto **Page<Mark>** bajo la clave **page**.

Este objeto tiene el método **getNumber()** para obtener la página actual y **getTotalPages()** para obtener el total de páginas. Si bien hay muchas formas de mostrar las opciones de paginación, nosotros vamos a optar por mostrar: la primera página, la última y las cercanas a la actual. Seguiremos la siguiente estrategia:

- Mostramos la primera página: `page=0`
- Mostramos la página anterior a la actual, solamente si existe.
- Mostramos la página actual.
- Mostramos la página siguiente a la actual, solamente si existe.
- Mostramos la página final `page=getTotalPages()`



Las páginas empiezan a contar en 0 aunque eso puede no resultar muy lógico para el usuario. Por ello, en las etiquetas de texto, vamos a mostrar un número más del que realmente corresponde al enlace.

El enlace 1 lleva a <http://localhost:8090/mark/list?page=0> , el enlace 2 lleva a la <http://localhost:8090/mark/list?page=1> y así sucesivamente.

```
<div class="text-center">
  <ul class="pagination justify-content-center">
    <!-- Primera -->
    <li class="page-item" >
      <a class="page-link" th:href="@{'?page=0'}">Primera</a>
    </li>

    <!-- Anterior (si la hay) -->
    <li class="page-item" th:if='${page.getNumber()-1 >= 0}'>
      <a class="page-link" th:href="@{'?page='+${page.getNumber()-1} }"
        th:text="${page.getNumber()}"></a>
    </li>

    <!-- Actual -->
    <li class="page-item active" >
      <a class="page-link" th:href="@{'?page='+${page.getNumber()}}"
        th:text="${page.getNumber()+1}"></a>
    </li>

    <!-- Siguiente (si la hay) -->
    <li class="page-item" th:if='${page.getNumber()+1 <= page.getTotalPages()-1}'>
      <a class="page-link" th:href="@{'?page='+${page.getNumber()+1} }"
        th:text="${page.getNumber()+2}"></a>
    </li>

    <!-- Última -->
    <li class="page-item" >
      <a class="page-link"
        th:href="@{'?page='+${page.getTotalPages()-1} }"> Última</a>
    </li>
  </ul>
</div>
```

Ahora incluimos el fragmento ***pagination.html*** en la vista ***/marks/list.html***. Lo hacemos en la parte final del contenedor principal. También incluimos el pie de página, si no lo tenemos incluido.

```
</table>
</div>
<footer th:replace="fragments/pagination"/>
</div>
<footer th:replace="fragments/footer"/>
</body>
```



Modificamos el **MarksController** para que en su respuesta a la petición **/mark/list** envíe el parámetro **page** que se necesita para mostrar el sistema de paginación de forma correcta.

```
@RequestMapping("/mark/list")
public String getList(Model model, Pageable pageable, Principal principal,
    @RequestParam(value = "", required = false) String searchText) {
    String dni = principal.getName(); // DNI es el name de la autenticación
    User user = userService.getUserByDni(dni);
    Page<Mark> marks = new PageImpl<Mark>(new LinkedList<Mark>());
    if (searchText != null && !searchText.isEmpty()) {
        marks = marksService.searchMarksByDescriptionAndNameForUser(pageable, searchText, user);
    } else {
        marks = marksService.getMarksForUser(pageable, user);
    }
    model.addAttribute("markList", marks.getContent());
    model.addAttribute("page", marks);
    return "mark/list";
}
```

Comprobamos que el sistema de paginación funciona de la forma esperada.

Accedemos a la aplicación utilizando el perfil:

- DNI: 99999977E password: 123456 -> ROLE\_PROFESSOR

### Notas


Buscar por descripción o nombre del alumno

Las notas que actualmente figuran en el sistema son las siguientes:

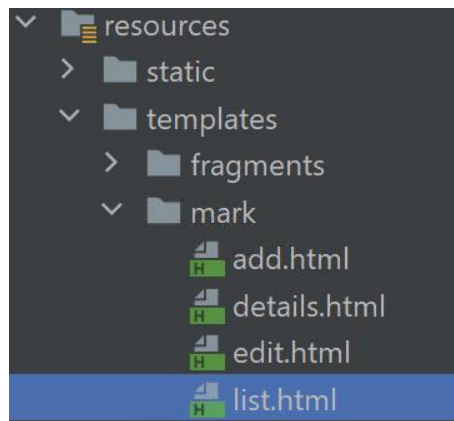
Actualizar

id	Descripción	Puntuación			
8	Nota B4	3.5	<a href="#">detalles</a>	<a href="#">modificar</a>	<a href="#">eliminar</a>
9	Nota B1	5.0	<a href="#">detalles</a>	<a href="#">modificar</a>	<a href="#">eliminar</a>
10	Nota B2	4.3	<a href="#">detalles</a>	<a href="#">modificar</a>	<a href="#">eliminar</a>
12	Nota C1	5.5	<a href="#">detalles</a>	<a href="#">modificar</a>	<a href="#">eliminar</a>
13	Nota C3	7.0	<a href="#">detalles</a>	<a href="#">modificar</a>	<a href="#">eliminar</a>

Primera 1 2 3 Última



Únicamente nos falta modificar las llamadas a **/mark/list/update** que realizamos desde el código JavaScript en dos puntos de la vista **/mark/list**



El primero es el botón de actualizar que habíamos colocado encima de la tabla que mostraba las notas.

Debemos preparar el script para incluir parámetros Thymeleaf (incluimos el `th:inline="javascript"` y el `CDATA`).

Dentro del script, para incluir a través de Thymeleaf los parámetros de la URL usamos `[[${param.<nombre del parámetro>}]]`. Esta sentencia nos devuelve un **null** si no hay parámetro o un **array** de una única posición si hay parámetro. Para acceder al valor del parámetro tenemos que consultar la posición `[0]` del array.

```
<p>Las notas que actualmente figuran en el sistema son las siguientes:</p>
<button type="button" id="updateButton" class="btn btn-primary">Actualizar</button>
<script th:inline="javascript">
  /*<![CDATA[*]
    $('#updateButton').click(function() {
      let numberPage = [[${param.page}]];
      let urlUpdate = '/mark/list/update';
      if (numberPage != null){
        urlUpdate += "?page="+numberPage[0];
      }
      $('#tableMarks').load(urlUpdate);
    });
  /*]]>*/
</script>
```

La segunda invocación de `/mark/list/update` se encuentra en los botones de **Reenviar / No reenviar** la actividad que se mostraba solo a los usuarios con `ROLE_STUDENT`.

El procedimiento a seguir va a ser el mismo que el caso anterior. Sería adecuado sacar el código común a una función

```
<td><a sec:authorize="hasRole('ROLE_PROFESSOR')" th:href="${'/mark/edit/' + mark.id}">modificar</a>
  <div sec:authorize="hasRole('ROLE_STUDENT')" >
    <div th:if="${mark.resend}">
      <!-- <a th:href="${'/mark/' + mark.id + '/noresend'}">Reenviar</a> -->
      <button type="button" th:id="${'resendButton' + mark.id}"
```





```
class="btn btn-success">Reenviar</button>
<script th:inline="javascript">
/*<![CDATA[* /
    $('#resendButton' + "[$ {mark.id}]]").click(function() {
        $.get( "/mark/[$ {mark.id}]/noresend", function( data ) {
            let numberPage = [[${param.page}]];
            let urlUpdate = '/mark/list/update';
            if ( numberPage != null ){
                urlUpdate += "?page="+numberPage[0];
            }
            $('#tableMarks').load(urlUpdate);
        });
    });
/*]]>*/
</script>

</div>
<div th:unless="${mark.resend}">
    <!--<a th:href="${'/mark/' + mark.id + '/resend'}">No reenviar</a-->
    <button type="button" th:id="${'noresendButton' + mark.id}"
        class="btn btn-info">No reenviar</button>
    <script th:inline="javascript">
/*<![CDATA[* /
    $('#noresendButton' + "[$ {mark.id}]]" ).click(function() {
        $.get( "/mark/[$ {mark.id}]/resend", function( data ) {
            let numberPage = [[${param.page}]];
            let urlUpdate = '/mark/list/update';
            if ( numberPage != null ){
                urlUpdate += "?page="+numberPage[0];
            }
            $('#tableMarks').load(urlUpdate);
        });
    });
/*]]>*/
</script>
</div>
</div>
```

Guardamos los cambios y probamos la aplicación. Para probar la paginación más fácilmente podemos modificar la configuración **CustomConfiguration** de forma que se muestren **solo dos registros** por página.

Comprobamos que el sistema de paginación funciona de la forma esperada.

Accedemos a la aplicación utilizando el perfil:

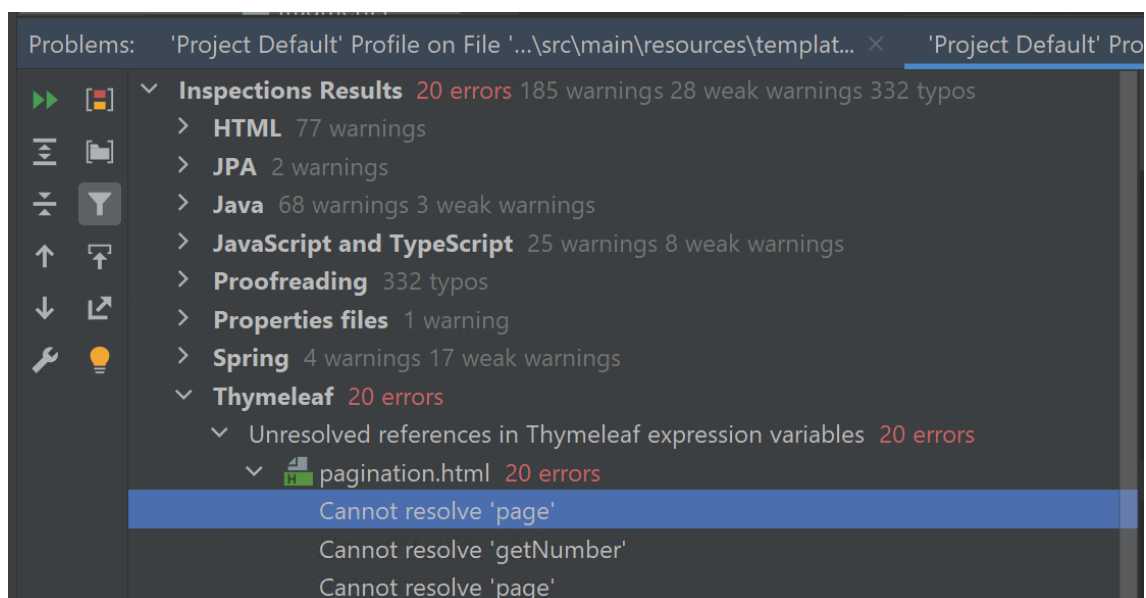
- DNI: 99999990A password: 123456 -> ROLE\_STUDENT



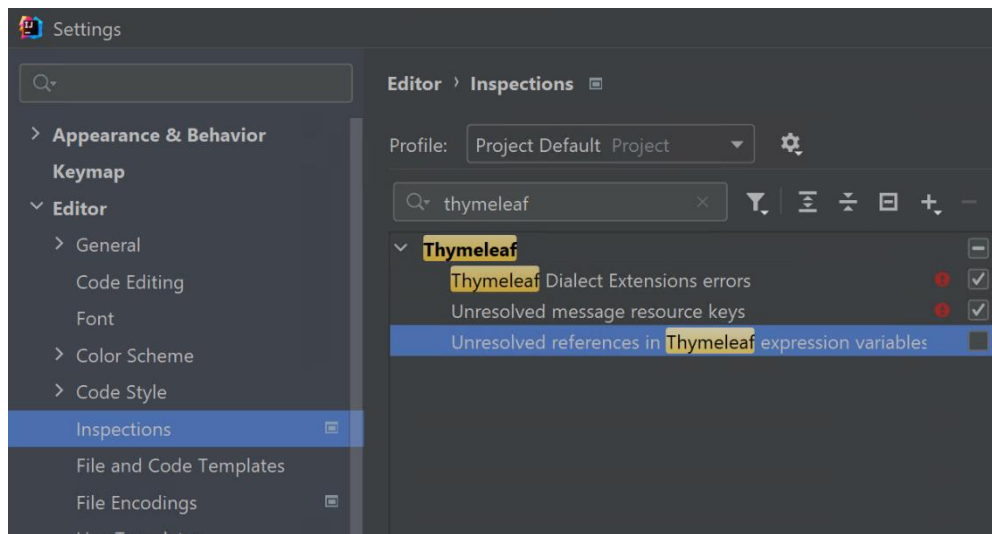
Notas				
<input type="text" value="Buscar por descripción o nombre del alumno"/>				<button>Buscar</button>
Las notas que actualmente figuran en el sistema son las siguientes:				
<button>Actualizar</button>				
id	Descripción	Puntuación		
4	Nota A4	6.5	<a href="#">detalles</a>	<button>No reenviar</button>
5	Nota A2	9.0	<a href="#">detalles</a>	<button>Reenviar</button>
<div><span>Primera</span> <span>1</span> <span>2</span> <span>Última</span></div>				

## 6.6 Corrigiendo errores y advertencias (warnings)

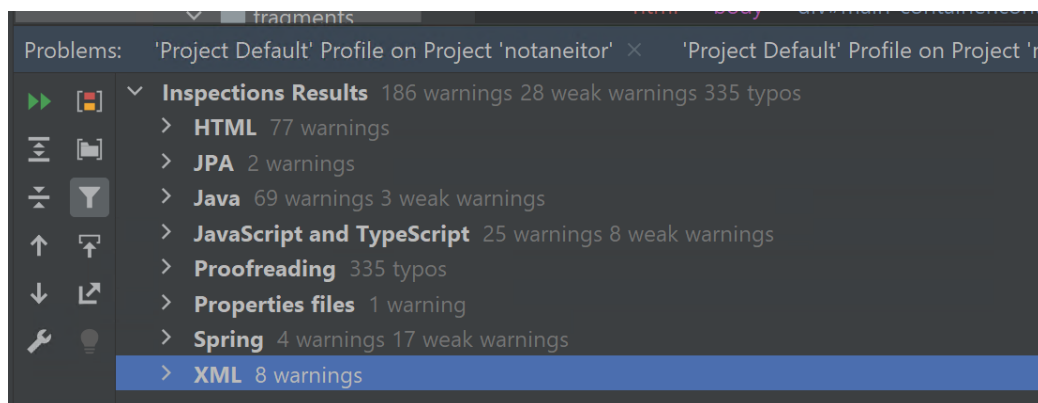
En este apartado hay que corregir las distintas advertencias (warnings) y algunos errores del proyecto. Si ejecutamos el inspector de código desde el menú **Code|Inspect Code**, nos mostrará todos los errores y warnings del proyecto. Vamos a solucionar los errores ocasionados por IntelliJ, porque algunas veces no reconoce variables y expresiones de Thymeleaf, tal como se muestra en esta imagen.



Esto no son errores de código, son errores de integración del IDE y el motor de plantillas. Para evitar que se muestren este tipo de errores en IntelliJ, vamos al menú **File|Setting|Editor|Inspections** y buscamos Thymeleaf y desmarcamos la opción “Unresolved references in Thymeleaf expression variable” y pulsamos el botón OK. Ver siguiente imagen:



Si ejecutamos el inspector de código de nuevo veremos que se han corregidos esos errores.



Muchas de estas advertencias son generadas por el propio IDE, o por el uso de algún método discontinuado, atributos o etiquetas HTML mal definidas o incompletas, etc.

**Nota:** Intenta analizar el código e intenta solucionar algunos de estos warnings (advertencias).

**Nota:** Incluir el siguiente Commit Message ->

**“SDI-IDGIT-4.6-SpringBoot-Paginación”**

**OJO:** sustituir IDGIT por tu número asignado (p.e. 2223-101):

**“SDI-2223-101-4.6-SpringBoot-Paginación.”**

*(No olvides incluir los guiones y NO incluyas BLANCOS al principio o al final de la oración)*

## 6.1 Etiquetar proyecto en GitHub

**Nota:** Etiquetar el proyecto en este punto con la siguiente etiqueta -> **sdi-springboot-p3**



Para crear una **Release** en un proyecto que está almacenado en un repositorio de control de versiones lo común es utilizar etiquetas (Tags), que nos permitirá marcar el avance de un proyecto en un punto dado (una funcionalidad, una versión del proyecto, etc).

Para crear la etiqueta el proyecto en guíate del ejemplo que realizamos en el guión 2.

## 6.2 Resultado esperado en el repositorio de GitHub

Al revisar el repositorio de código en GitHub, los resultados de los commits realizados deberían ser parecidos a estos.

- ⇒ SDI-2223-101-4.1-SpringBoot-Sesiones.
- ⇒ SDI-2223-101-4.2-SpringBoot-Roles.
- ⇒ SDI-2223-101-4.3-SpringBoot-Control de acceso por roles.
- ⇒ SDI-2223-101-4.4-SpringBoot-Consulta.
- ⇒ SDI-2223-101-4.6-SpringBoot-Paginación.