Desarrollo avanzado de aplicaciones II

Tema Nº7:Spring Security.

Indicador de logro Nº7:

Utiliza las funciones de seguridad en Spring basados en el módulo de Spring Security.

**TEMA 01 Teoría de los**

Imagen que contiene Icono

Descripción generada automáticamente

**TEMA Nº7:**

**Spring Security.**

**Subtema 7.1:**

¿Qué es Spring Security?

Spring Security es una librería que forma parte del paraguas del proyecto Spring. Spring tiene más de 25 sub proyectos o módulos que aportan funcionalidad que las aplicaciones pueden utilizar si lo creen conveniente. En este caso Spring Security trata de agrupar todas las funcionalidades de control de acceso de usuarios sobre proyectos Spring.

El control de acceso permite limitar las opciones que pueden ejecutar un determinado conjunto de usuarios o roles sobre la aplicación. En esta dirección, Spring Security controla las invocaciones a la lógica de negocios o limita el acceso de peticiones HTTP a determinadas URLs.

Para ello, el programador debe realizar una configuración sobre la aplicación indicando a Spring Security cómo debe comportarse la capa de seguridad. Y aquí es una de las grandes ventajas de Spring Security ya que permite realizar toda una serie de parametrizaciones y ajustes para un gran abanico de posibilidades, permitiendo que el módulo se adapte bien a casi cualquier escenario de aplicaciones realizadas con Spring IoC o Spring Boot.

Si la implementación de Spring Security no fuera suficiente, aun así, proporciona toda una serie de interfaces Java que el programador puede implementar para cambiar el modo de comportarse de una determinada funcionalidad, como el ir a buscar los usuarios a una fuente de datos muy particulares.

Siempre que se pueda, conviene usar las implementaciones provenientes en la librería de Spring Security, ya que es código muy maduro ampliamente testado y además se mantiene actualizado por la comunidad cuando se publica un aviso de seguridad o ante cualquier publicación un nuevo estándar como pasó al liberarse el protocolo HTTP 2.

En definitiva, Spring Security es el método más conveniente para incorporar una capa de seguridad en donde se desea que sólo algunos usuarios tengan acceso a métodos y controladores de una aplicación Spring en base al uso de roles de usuario.

Además, al ser un proyecto maduro y ampliamente utilizado, es posible encontrar implementaciones incluidas de serie para los principales sistemas de seguridad de usuarios más desplegados en el mundo del desarrollo de software o en la administración de sistemas de TI, como LDAP o Kerberos entre otros.

**Subtema 7.2:**

Implementar y configurar Spring Security.

Añadir Spring Security en cualquier proyecto Spring de Java es muy sencillo, sobre todo si se ha utilizado la capa de auto-configuración de Spring MVC y Spring IoC denominada Spring Boot. Si deseas incorporar Spring Security en una aplicación Java o Spring MVC conviene que consideres la migración a Spring Boot antes de proceder con los siguientes pasos, ya que el uso de Spring Boot simplifica enormemente las acciones de configuración que debe realizar el desarrollador para poner en marcha cualquier aplicación Spring.

Suponiendo que tienes una aplicación Spring Boot en marcha gestionado por Maven, en donde el fichero pom.xml tiene como parent el artefacto spring-boot-starter-parent, lo primero que se debe hacer es incorporar el conjunto de dependencias provenientes de spring-boot-starter-security. Ésta traerá de forma transitiva el resto de librerías JARs que Spring necesitará para aplicar los mecanismos de seguridad requeridos.

Por sólo incluir la dependencia, si el programador no realiza ninguna configuración adicional, Spring Boot de manera predeterminada protegerá todo el acceso a la aplicación, impidiendo que ningún usuario no identificado pueda invocar a cualquier controlador. Este mecanismo tan restrictivo suele ser suficiente para pequeñas aplicaciones donde sólo se necesita restringir el acceso de forma general, pero queda algo reducido en aplicaciones donde hay secciones accesibles y otras protegidas, dependiendo de si el usuario está identificado o de si éste tiene un rol determinado.

Como el resto de aspectos configurables de Spring Boot, en el fichero de configuración de la aplicación, application.properties o application.yml, hay varias propiedades que pueden ajustarse para controlar el comportamiento base de Spring Security.

Las propiedades más importantes de Spring Security son:

Texto

Descripción generada automáticamente

De esta manera, sólo por incluir la dependencia, en el fichero de propiedades de la aplicación puede indicarse un usuario y clave por defecto que podrá usarse para tener acceso a los controladores, y como consecuencia a las diferentes pantallas HTML de la aplicación. Cuando el usuario intente acceder a cualquier URL de la aplicación, Spring Boot y el Security Filter de HTTP redirigirá al usuario al formulario de identificación, donde solicitará al usuario a insertar el nombre y password para proceder. En ese formulario se debe indicar los valores fijados en spring.security.user.name y spring.security.user.password. Si el usuario ha introducido los valores correctos, se considera al usuario autenticado, y sin tener en cuenta el rol, dejará continuar al usuario con una navegación normal.

Se debe tener en cuenta que la configuración por defecto no incluye el auto registro de usuarios. En el caso de necesitar que los usuarios puedan registrarse automáticamente, por ejemplo con un email, se deberá programar el mecanismo por el cual se confía en la información proporcionada por el usuario procediendo a la creación de éste en la tabla de datos o fuente de usuarios confiables usada por la aplicación.

En el archivo pom.xml se deben agregar las siguientes dependencias.

Una captura de pantalla de un celular con texto e imagen

Descripción generada automáticamente con confianza media

**Subtema 7.3:**

Autenticación de usuario y obtención de roles.

**Ejemplos:**

Tener un sólo usuario en la aplicación puede ser útil en ciertas ocasiones, aunque desde luego es un escenario algo limitado para aplicaciones abiertas en Internet. Por norma general se usarán varios usuarios, presumiblemente con roles de acceso diferentes para limitar las acciones que pueden realizar cada tipo de usuario.

Para ello necesitamos añadir una fuente de usuarios administrador por la aplicación y que serán usados en el proceso de autenticación por Spring Security. De esta manera, nuestra aplicación puede incorporar un mecanismo de registro de usuarios que puedan autenticarse.

Para que Spring pueda obtener la información de un usuario durante el proceso de login, se solicitará al conjunto de clases Java registradas en el contexto de Spring, de alguna que implemente la interfaz UserDetailsService:

Texto

Descripción generada automáticamente

Esta interfaz tiene un único método de obligada implementación que es aquel que devuelve los detalles de un usuario UserDetails dado un nombre de usuario, el que trata de ingresar en la aplicación. Por tanto, será obligatorio implementar un bean con esa interfaz dando cuerpo a ese método, por ejemplo como el siguiente:

Texto

Descripción generada automáticamente Texto

Descripción generada automáticamente

La implementación de este bean dependerá de dónde tenemos registrados los usuarios. Normalmente éstos estarán en una tabla de la base de datos, así que ya sea con SQL nativo o usando JPA con EntityManager o a través de Spring Data Repositories deberemos consultar la fila asociada según el nombre de usuario. Una vez localizado sólo restará convertir o mapear ese objeto a uno esperado por Spring, del tipo UserDetails.

Por último, en la clase de configuración de Spring Security registraremos nuestra clase recién creada:

Texto

Descripción generada automáticamente

¡Y listos! Con esta simple configuración ya se tiene una aplicación con gestión de usuarios capaces de identificarse en Spring Security y por tanto usar todos los mecanismos que esta capa de seguridad ofrece.

Se debe tener en cuenta que en el ejemplo se ha optado por usar un PasswordEncoder que permite almacenar las contraseñas de los usuarios de forma encriptada en la base de datos. Esta manera es una gran aliado para evitar fugas de información importantes si algún usuario malintencionado pudiera tener en algún momento acceso a la tabla de usuarios, evitando que se haga con las credenciales de los usuarios.

Cuando tenga que implementarlo en su aplicación no debe olvidar proporcionar una implementación válida como UserRepository para que pueda obtenerse el usuario solicitado. Recuerde que este información puede proceder de una base de datos o también de algún archivo, o incluso de algún servicio remoto para la centralización de usuarios como LDAP.

**Subtema 7.4:**

Personalizar accesos a URLs por ROLES.

**Ejemplos:**

Si la protección de forma global a toda la aplicación no se adapta a tus necesidades, probablemente será porque hay secciones que deseas que sean accesibles mientras que otras estén protegidas para un determinado número de usuarios. Si este es el caso, al paso anterior de incluir la dependencia spring-boot-starter-security se debe acompañar con la creación de una clase de configuración de Spring anotada con @Configuration, en donde se podrá especificar qué secciones estarán habilitadas y cuáles no en función de los patrones de URL. Por comodidad, conviene que esta clase a crear extienda a WebSecurityConfigurerAdapter ya que simplifica la implementación.

Por ejemplo, supóngase que la aplicación web que se está desarrollando tiene una parte pública compuesta por páginas HTML, otra parte protegida y accesible para usuarios autenticados (sin importar su rol) y por último otra sección accesible únicamente para el o los usuarios con el rol de administrador. Para poder llevar a cabo esta configuración se debe segmentar los accesos por URLs de tal manera que permita realizar la siguiente distribución:

* /admin/\* Patrón de URL que agrupa las funcionalidades del administrador
* /user/\* Grupo de funcionalidades de los usuarios autenticados
* /\* Resto de accesos públicos a la aplicación

Para que el mecanismo de control de acceso por URL funcione correctamente conviene hacer esta división por espacios de URL e incluir las secciones más restrictivas primero, dejando al final las más generales. Así se simplifica el número de configuraciones necesarias a realizar.

La configuración de Spring Security necesaria para este supuesto quedaría de la siguiente manera:

Texto

Descripción generada automáticamente

**Subtema 7.5:**

Agregar seguridad en el controlador usando anotaciones @Secured o @PreAuthorize.

**Ejemplos:**

La anotación @Secured se usa para especificar una lista de roles en un método. Por lo tanto, un usuario solo puede acceder a ese método si tiene al menos uno de los roles especificados.

Texto

Descripción generada automáticamente

Aquí, la anotación @Secured (“ROLE\_VIEWER”) define que solo los usuarios que tienen el rol ROLE\_VIEWER pueden ejecutar el método getUsername.

Además, podemos definir una lista de roles en una anotación @Secured:

Texto

Descripción generada automáticamente

Las anotaciones @PreAuthorize y @PostAuthorize proporcionan control de acceso basado en expresiones. Por lo tanto, los predicados se pueden escribir usando SpEL (Spring Expression Language).

La anotación @PreAuthorize verifica la expresión dada antes de ingresar al método, mientras que la anotación @PostAuthorize la verifica después de la ejecución del método y podría alterar el resultado.

Ahora, declaremos un método getUsernameInUpperCase como se muestra a continuación:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

**Actividad:**

a) CUESTIONARIO TÉCNICO

* ¿Qué dependencias necesita nuestro proyecto para añadir Spring Security?
* ¿Qué ventajas tiene trabajar con Spring Security?
* ¿Para qué sirve la anotación @Secured?
* ¿Para qué sirve la anotación @PreAuthorize?
* Cree un nuevo proyecto con Spring Boot y realice una autenticación utilizando Spring Security con base de datos MySQL.

b) CONCLUSIONES DE LA EXPERIENCIA

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_