**Introducción a Spring Security**

**Spring Security** es un **marco de seguridad** potente y altamente personalizable para aplicaciones **Java**.

Está diseñado para proporcionar autenticación y autorización a nivel de aplicación para aplicaciones basadas en **Spring**.

Imagen que contiene Icono

El contenido generado por IA puede ser incorrecto.

Con **Spring Security**, puedes proteger tus aplicaciones contra amenazas comunes de seguridad mediante la administración de la autenticación de usuarios, la gestión de roles y permisos, la prevención de ataques de denegación de servicio, la protección contra la falsificación de solicitudes entre sitios (CSRF), entre otros.

El marco proporciona una amplia gama de características, incluyendo la autenticación basada en formularios, la autenticación basada en tokens, la integración con proveedores de autenticación externos (como **OAuth** y **OpenID**), y mucho más.

Una de las principales características de Spring Security es su capacidad para integrarse estrechamente con otros proyectos de Spring, como **Spring MVC**, **Spring Boot** o **Spring Cloud**, lo que facilita su configuración y uso en aplicaciones Spring existentes o nuevas.

**¿Qué conceptos se utilizan en Spring Security?**

**Filtros de seguridad**

Son componentes que implementan lainterfaz **javax.servlet.Filter** (actualmente **jakarta.servlet.Filter**) que interceptan las **peticiones HTTP** para realizar tareas de autenticación, autorización y seguridad.

*Spring Security permite crear filtros personalizados o, en defecto, utilizar alguno de sus 12 filtros predeterminados. Éstos últimos son:*

1. **ChannelProcessingFilter**: Este filtro se encarga de redirigir las solicitudes a un protocolo seguro (HTTPS) si es necesario.
2. **SecurityContextPersistenceFilter**: Mantiene el contexto de seguridad de la aplicación entre las solicitudes del usuario.
3. **ConcurrentSessionFilter**: Controla las sesiones concurrentes de los usuarios, permitiendo configurar cuántas sesiones activas puede tener un usuario.
4. **LogoutFilter**: Gestiona las solicitudes de cierre de sesión de los usuarios.
5. **UsernamePasswordAuthenticationFilter**: Procesa las solicitudes de autenticación basadas en nombre de usuario y contraseña.
6. **DefaultLoginPageGeneratingFilter**: Genera automáticamente una página de inicio de sesión predeterminada si no se proporciona una personalizada.
7. **DefaultLogoutPageGeneratingFilter**: Genera automáticamente una página de cierre de sesión predeterminada si no se proporciona una personalizada.
8. **BasicAuthenticationFilter**: Procesa las solicitudes de autenticación básica.
9. **RequestCacheAwareFilter**: Permite que las solicitudes sean almacenadas en caché para su uso posterior después de la autenticación.
10. **SecurityContextHolderAwareRequestFilter**: Permite que los objetos de solicitud sean conscientes del contexto de seguridad.
11. **JaasApiIntegrationFilter**: Integra la autenticación basada en JAAS (Java Authentication and AuthorizationService).
12. **RememberMeAuthenticationFilter**: Procesa las solicitudes para la autenticación basada en recordar al usuario.

**Proveedores de Autenticación**

Son componentes responsables de autenticar las credenciales. Trabajan en conjunto con otros componentes, como ser los filtros de seguridad y proveedores de detalles de usuario, para verificar las credenciales y establecer la autenticación del usuario. Algunos ejemplos:

* **DaoAuthenticationProvider:**Este proveedor de autenticación utiliza un objeto de acceso a datos (DAO) para autenticar las credenciales del usuario. Utiliza un objeto UserDetailsService para recuperar los detalles del usuario (como nombre de usuario, contraseña y roles) de una fuente de datos, como una base de datos, y luego compara las credenciales proporcionadas por el usuario con las almacenadas en la fuente de datos.
* **LdapAuthenticationProvider:**Este proveedor de autenticación se utiliza cuando las credenciales de usuario están almacenadas en un directorio LDAP (Protocolo Ligero de Acceso a Directorios). Autentica las credenciales del usuario consultando un servidor LDAP y comparando las credenciales proporcionadas con las almacenadas en el directorio LDAP.
* **JaasAuthenticationProvider:**Este proveedor de autenticación utiliza el servicio de autenticación de Java (JAAS) para autenticar las credenciales del usuario. Se utiliza principalmente en entornos donde JAAS es la solución de autenticación estándar.
* **OpenIDAuthenticationProvider y OAuth2AuthenticationProvider:**Estos proveedores de autenticación se utilizan para autenticar a los usuarios utilizando los protocolos OpenIDConnect y OAuth 2.0, respectivamente. Permiten que los usuarios inicien sesión utilizando sus credenciales de proveedores de identidad externos, como Google, Facebook o GitHub.

**Proveedores de Detalles de Usuarios**

El **Proveedor de Detalles de Usuarios** es un componente fundamental en **Spring Security** que se encarga de proporcionar la información de autenticación de los usuarios al sistema de seguridad. Básicamente, este proveedor es responsable de recuperar los detalles de los usuarios, como sus nombres de usuario, contraseñas y roles, para que **Spring Security** pueda realizar la autenticación de manera adecuada.

La interfaz principal que define este comportamiento es **UserDetailsService**. Esta interfaz tiene un único método, **loadUserByUsername()**, que recibe como parámetro el nombre de usuario y devuelve un objeto **UserDetails**. Este objeto contiene la información necesaria sobre el usuario, como su nombre de usuario, contraseña (ya encriptada), roles y cualquier otra información adicional relevante para la autenticación.

La implementación concreta de **UserDetailsService** puede variar según las necesidades del proyecto. Puede obtener los detalles de los usuarios de una base de datos, un servicio web, un repositorio LDAP, o cualquier otro origen de datos.

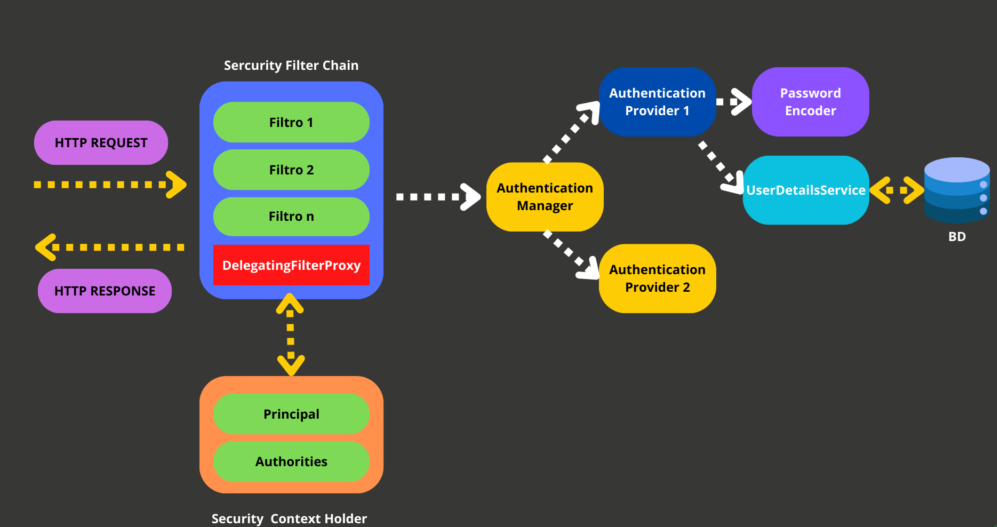
**Arquitectura de Spring Security**

**Arquitectura Principal**

**Spring Security**puede adaptarse para distintas arquitecturas, sin embargo, existen arquitecturas predefinidas que son mayormente utilizadas que otras.

Entre las arquitecturas más utilizadas encontramos aquellas que incluyen una cadena de filtros, un administrador de autenticación, uno (o varios) proveedores de autenticación y un contexto de seguridad donde almacenar los datos  de autenticación necesarios.

Un ejemplo de este tipo de arquitectura lo podemos ver en la imagen a continuación:



**Camino de autenticación**

En base a esta **arquitectura**planteada en la imagen, el proceso de autenticación mediante **Spring Security** comienza con una solicitud HTTP que se envía al servidor. La solicitud es interceptada por una **cadena de filtros de seguridad**. El primer filtro en la cadena casi siempre el filtro de autenticación, que se encarga de verificar las credenciales del usuario, como el nombre de usuario, la contraseña o los medios que se utilicen. Si las credenciales son válidas, el filtro de autenticación recupera la información del usuario de un proveedor de autenticación, como un servicio de directorio o una base de datos.

Luego de que se produzca esta primera parte, la información del usuario se pasa al siguiente filtro en la cadena (si existiese), que puede ser un filtro de autorización o un filtro de seguridad adicional, etc. Spring Security (en este caso) posee 12 filtros de seguridad por defecto y la posibilidad de crear filtros personalizados mediante el **DelegatingFilterProxy**.

Una vez que se cumple el proceso de autenticación inicial, el **Security ContextHolder** se encarga de almacenar la información de autenticación del usuario. El contexto de seguridad es un objeto de tipo **SecurityContextHolder** que contiene información sobre el usuario actual, como su nombre de usuario, roles, permisos, etc.

La principal característica del contexto de seguridad es el hecho de que está disponible para todos los filtros y componentes de la aplicación web.

**Partes del circuito/camino de autenticación**

La **arquitectura** del diagrama que analizamos anteriormente incluye los siguientes componentes en su circuito de autenticación que veremos en mayor detalle:

* **Solicitud HTTP:** La solicitud HTTP es una petición que el cliente envía al servidor para obtener un recurso. Contiene información sobre el recurso que se solicita, así como información sobre el cliente, como el navegador web y el sistema operativo.
* **Cadena de filtros de seguridad:** La cadena de filtros de seguridad es una lista de filtros que se ejecutan secuencialmente cuando se procesa una solicitud HTTP. Cada filtro puede realizar una tarea específica, como verificar la autenticación del usuario, autorizar el acceso a un recurso o registrar la actividad del usuario.
* **Proveedor de autenticación:** El proveedor de autenticación es un componente que proporciona información sobre los usuarios. Puede ser un servicio de directorio, una base de datos o cualquier otro componente que pueda proporcionar información sobre los usuarios.
* **Codificador:** El codificador es un componente que se utiliza para codificar/encriptar la contraseña del usuario antes de compararla con la contraseña almacenada en la base de datos. La codificación/encriptación de la contraseña ayuda a proteger la misma principalmente frente a ataques de fuerza bruta.
* **Filtro 1, Filtro 2, …, Filtro n:** Estos filtros pueden realizar cualquier tarea específica, como verificar la autorización del usuario, registrar la actividad del mismo o validar los datos de entrada. Cada uno de ellos pueden ser personalizados o parte de los estándares de Spring Security.
* **Base de datos:** La base de datos se utiliza para almacenar la información como ser el nombre de usuario, la contraseña y los roles o permisos.
* **DelegatingFilter Proxy:** El DelegatingFilter Proxy es un componente que se utiliza para delegar la ejecución de los filtros en la cadena de filtros de seguridad. El DelegatingFilter Proxy puede seleccionar el filtro adecuado para ejecutar en función de la solicitud HTTP.

**Security ContextHolder:** El Security ContextHolder es un componente que se utiliza para almacenar el contexto de seguridad.