Backend de Productos y Servicios

Sprint deadline	30/10/2023
Tarjeta	SCRUM-25
Responsable	Malleret, Luciano Joaquín

Tabla 1 Detalle de la tarjeta correspondiente en Jira

1. Objetivos. Contenido de la tarjeta:

- Importación de Spring security.
- Crear beans de configuración de seguridad (SecurityFilterChain, Authentication manager, encoder, etc).
- Configurar el detailService.
- Configuración JWT.

2. Dependencias

Fue necesario solamente la implementación del repositorio con un proyecto base. Luego la tarea fue realizada en paralelo con las demás tarjetas, ya que consistió en la adición de una capa de seguridad para la aplicación.

3. Procedimientos

3.1. Importación de Spring Security

Se agregó la dependencia pertinente en el archivo pom.xml para poder acceder a las funcionalidades ofrecidas por el framework. Spring Security por defecto viene configurado para ofrecer una capa de autenticación dentro del mismo servidor, que puede servir si se utilizase una web estática desde el lado del servidor. Pero como el objetivo es diseñar una aplicación web que corra en otro servidor es necesario implementar otras formas de seguridad. En este caso hemos optado por utilizar un sistema basado en JWT (Jason Web Tokens).

3.2. Importación de H2

Se incorporó el uso de la base de datos de prueba H2 para ayudar y agilizar el proceso de desarrollo y también para alivianar la utilización de recursos del sistema en contraposición a docker. Se incluyó la dependencia en el pom.xml y se configuró en el application.properties. La utilización de esta base de datos de prueba o de la base de datos deployada en docker puede escogerse desde application.properties, comentando y descomentando sus bloques de código respectivamente.

3.3. Configuración de la SecurityFilterChain

Se creó una nueva clase para configurar los filtros de seguridad. Dichos filtros son la forma en la que Spring Security maneja las acciones que se deben tomar antes de recibir o enviar una petición hasta o desde el servidor. De esta manera es posible interceptar la petición para autenticar el token de seguridad o para validar que la persona que intenta acceder a una ruta determinada tenga permisos para hacerlo, entre otras cosas.
En esta clase también se definió el encriptador a utilizar a la hora de persistir contraseñas en la base de datos.

3.4. Creación de JwtGenerator

Se creó una clase para englobar la lógica correspondiente a la creación y validación de los tokens, así como ciertas utilidades correspondientes al mismo, como obtener el nombre de usuario correspondiente al token.

3.5. Creación de SecurityConstants

Una clase creada específicamente para contener valores de referencia que se deben utilizar por toda la aplicación, como el tiempo de vida de los tokens, o la clave de encriptación para los mismos.

3.6. Creación del JWTAuthenticationFilter

Una clase que sirve de filtro personalizado para aplicar en el SecurityFilterChain. En esta clase se chequea el encabezado de la petición en busca del Bearer Token, si no lo encuentra o no resulta ser válido no autentica la petición, pero si lo valida autentica la petición, devuelve un nuevo token con un tiempo de expiración renovado (para que un usuario no pierda autorización mientras está usando la app) y manda el viejo token a una lista negra, para que nadie pueda usar un token viejo que aún no haya caducado.

3.7. Creación de TokenBlacklist

Un componente que vive dentro del contexto de ejecución de la aplicación que guarda en memoria una lista de tokens prohibidos. Podría haberse implementado como entidad en la base de datos, pero siendo que el tiempo de caducidad es muy reducido (media hora) se creyó conveniente esta variante. El almacenamiento se realiza a través de un ConcurrentHashMap que permite la lectura y edición multi hilos de la colección para evitar problemas de concurrencia de datos.

Un token es añadido a la colección cuando es reemplazado por un nuevo token (como se explicó en el punto anterior) y es eliminado de la misma cuando su tiempo de expiración ha caducado.

3.8. Creación de AuthController y AuthService

Se creó un controlador y una capa de servicios para encargarse específicamente del apartado de seguridad y así aislar la lógica de negocios correspondiente y segregar las responsabilidades.

4. Resultados

La aplicación ahora cuenta con un sistema complejo de seguridad basado en tokens, totalmente escalable y con la capacidad de manejar roles y autorizaciones granulares.

4.1. Especificaciones de uso

El acceso a los endpoints queda totalmente aislado del anonimato, a excepción del Login, al cual debe accederse desde :

http://localhost:8080/auth/login

En el cuerpo de la petición deben viajar los datos de la siguiente manera :

```
{
        “username” : “nombre-de-usuario”,
        “password” : “contraseña-de-usuario”
}
```

Como la aplicación no posee un registro de usuarios, se provee un super usuario de **username: admin** y **password: admin** para poder comenzar a utilizar la página y agregar usuarios nuevos.

De ser exitoso el logueo, la respuesta del servidor proveerá en el encabezado **"Authorization"** un token de identificación con el formato "Bearer *texto-encriptado-del-token*". Ese token debe viajar en el encabezado **"Authorization"** en las sucesivas peticiones que se hagan al backend.
Al momento de realizar cualquier otra petición, en respuesta se recibirá un nuevo token con tiempo de vida extendido para que la sesión no caduque si es que el usuario está activo. Es trabajo del frontend el actualizar y almacenar debidamente el token, ya que los tokens viejos serán insertados en una lista negra hasta que caduquen, de esta forma se evita que un token viejo que caiga en manos maliciosas pueda acceder al sistema antes de caducar.

La otra ruta disponible en el controlador es el logout a la cual puede accederse desde :

http://localhost:8080/auth/logout

Para acceder debe enviarse el token correspondiente como en cualquier otra petición.
Si el logout ocurre exitosamente el usuario será eliminado del contexto de seguridad y su token será enviado a la lista negra, por lo que ya no podrá realizar ninguna otra petición sin volver a loguearse.

4.2. Conexión a la base de datos

Se dejó implementada una base de datos de prueba, la cual se reinicia cada vez que el servidor termina de ejecutar la aplicación. Tiene sus ventajas y desventajas comparado a la utilización de la base de datos dockerizada, queda a discreción del desarrollador utilizar la que más le convenga dentro del ambiente de desarrollo. Para producción es necesario utilizar la base de datos dockerizada.
El cambio se realiza desde **application.properties** comentando y descomentando los bloques de código pertinentes.

5. Conclusión

La aplicación quedó virtualmente segura con la nueva capa de autenticación y autorización. Se eligió este enfoque ya que resultó el más convincente de entre las opciones

disponibles para el Bootcamp, donde no era necesaria tanta complejidad, pero de esta manera queda implementada la seguridad de forma mucho más profesional.