**Implementación de Autenticación Basada en JWT para un Sistema de Creación y Obtención de Usuarios con Backend en Spring Boot y Frontend en React**

Muñoz Aguilar Andrés Felipe – 2210087

Ricardo Svensson Jaimes Estupiñán – 2202007

Sergio Hernando Barón Rivera – 2201885

# Introducción

En el presente proyecto, se desarrolla una aplicación web que integra un backend implementado en Spring Boot con un frontend construido en React, HTML y JavaScript. El objetivo principal es demostrar la implementación de autenticación y autorización mediante Spring Security y JSON Web Tokens (JWT), garantizando que el acceso a determinados recursos solo se permita a usuarios autenticados. Para ello, se habilita un endpoint dummy, el cual proporciona información sencilla únicamente a aquellos usuarios que han iniciado sesión tras registrarse en el sistema. A través de esta implementación, se ejemplifica el uso de JWT para la autenticación de peticiones, asegurando la protección de los datos y restringiendo el acceso a la información según los permisos otorgados.

Para esto, se construyeron los siguientes componentes:

# Frontend

Se agregan dependencias, bootstrap para organizar y mostrar rápidamente componentes Ahora se crea el primer componente para visualizar, un encabezado y un mensaje de bienvenida.

Se solicita el endpoint del backend y se muestra en otro componente. Para ello, se necesita axios. Se crea una función auxiliar y se usa con los componentes necesarios. Ahora, se crea el componente para mostrar el contenido del backend. En este caso, se llama Auth, ya que más adelante protegerá el endpoint anterior. Por lo tanto, este será el contenido protegido que requiere autenticación.

Este es el error más común al ejecutar un backend con un frontend separado: CORS (intercambio de recursos entre orígenes). Esto significa que el backend no confía en las solicitudes que provienen del frontend. Por defecto, el backend solo acepta solicitudes que provienen de sí mismo. Por lo tanto, se debe configurar el backend para que acepte solicitudes del frontend.

Se agregan más clases y CSS. Se empieza a crear el inicio de sesión en el frontend. El componente de inicio de sesión estará compuesto por dos formularios: el formulario de inicio de sesión para usuarios conocidos y el formulario de registro para nuevos usuarios. Se exporta el comportamiento de inicio de sesión y registro al componente principal. De esta manera, el componente principal puede ocultar el formulario de inicio de sesión una vez que el usuario envía las credenciales. Con este método, puedo almacenar en el estado los valores actualizados de los campos.

Ahora se modifica AppContent para mostrar el formulario de inicio de sesión, el mensaje de bienvenida o el mensaje protegido. Se crean dos botones. Por defecto, se mostrará el mensaje de bienvenida. Al hacer clic en el botón de inicio de sesión, se mostrará el formulario. Una vez registrado, se muestra el mensaje protegido. También se añadirá un botón de cierre de sesión para volver a mostrar el mensaje de bienvenida.

Una vez que el usuario inicia sesión, el JWT es devuelto al frontend. Es necesario almacenar este token y utilizarlo en las solicitudes posteriores. Para ello, se actualizará el helper de Axios, de modo que pueda guardar y recuperar el JWT desde el localStorage y, cuando esté disponible, incluirlo automáticamente en todas las peticiones al backend.

Después de completar el proceso de inicio de sesión o registro, el JWT se almacena en el navegador. A continuación, se realizará una prueba completa del proceso de inicio de sesión.

En el frontend, se utilizarán las herramientas de inspección del navegador para visualizar todas las solicitudes enviadas al backend. A través de la interfaz, se podrá alternar entre la vista del formulario de inicio de sesión y el mensaje de bienvenida utilizando dos botones.

El primer paso en el proceso es el registro de un usuario. Se envía una solicitud al backend con los datos del usuario y, como respuesta, se recibe un token JWT, que se puede utilizar para autenticación en futuras solicitudes.

Después del registro, se realiza una solicitud a la ruta de mensajes protegida, la cual requiere autenticación. Como el usuario ya está registrado y autenticado, la solicitud devuelve el contenido protegido correctamente.

A continuación, se procede a cerrar sesión. Luego, se realiza una nueva prueba, esta vez iniciando sesión con un usuario ya existente. Se envía la solicitud de inicio de sesión al backend, y este responde con un JWT válido. Posteriormente, se realiza nuevamente la solicitud a la ruta de mensajes protegida, que devuelve el contenido esperado.

En la cabecera de la solicitud a la ruta de mensajes, se puede observar que el JWT está incluido en el Authorization Header, lo que permite la autenticación del usuario. Con esto, se ha completado la implementación de una funcionalidad fullstack que abarca desde el backend hasta el frontend, incluyendo pruebas del flujo completo.

# Backend

El frontend solicita un endpoint simple para mostrar los resultados en otro componente. Se usa Spring Initializer para crear el nuevo proyecto Spring Boot y se agregan algunas dependencias que no están en el propio Initializer. Agregamos un controlador para devolver datos ficticios.

Se crea un archivo de configuración, se especifica todo lo que viene del frontend. El frontend enviará las credenciales.

Se coloca un bean en la posición más baja para usarlo antes de cualquier filtro de Spring Security. Se añade la configuración de seguridad para proteger todos los endpoints. Como se mencionó al principio, se agrega un filtro para las solicitudes JWT. Se añade primero un controlador de excepciones. De esta forma, se devuelve un mensaje personalizado cuando se produce un problema de seguridad.

Se añade el filtro JWT antes de cualquier filtro de autenticación de Spring Security. Se deshabilita el csrf para evitar complejidad. Configuramos Spring para que sepa que es una aplicación sin estado. De esta manera, Spring no creará ninguna sesión ni cookie.

En el controlador de excepciones, se devuelve un código HTTP no autorizado con un mensaje. Ahora con la parte del filtro, si las credenciales son válidas, se añade el bean de autenticación en el contexto de seguridad. Si algo sale mal, se borra el contexto de seguridad y lanza el error.

Al finalizar el filtro, es fundamental llamar al método doFilter para garantizar la correcta ejecución de la cadena de filtros. La adición del bean de autenticación al contexto de seguridad permite utilizar la anotación @AuthenticationPrincipal como parámetro en cualquier controlador. Esta anotación inyecta automáticamente el usuario autenticado, permitiendo que el filtro proporcione al controlador un objeto representando al usuario autenticado.

A continuación, se creará el proveedor utilizado anteriormente para devolver el bean de autenticación. Para ello, se incorporará una biblioteca que permitirá la generación y verificación de tokens JWT. La generación y lectura de JWT requiere una clave secreta, la cual puede configurarse en el archivo YAML de la aplicación e inyectarse en el código. Alternativamente, se puede definir un valor predeterminado. Para evitar almacenar la clave secreta en texto plano dentro de la JVM, se codificará en Base64.

El JWT generado tendrá una validez de una hora. Para verificar su autenticidad, primero se procederá a su decodificación. En caso de que la fecha de expiración haya sido superada, se lanzará una excepción. Además, se validará que el usuario correspondiente al token exista en la base de datos.

Continuando con la implementación del servicio de usuario, se encargará de gestionar el inicio de sesión y el registro de usuarios. Para garantizar la seguridad de las credenciales, se utiliza un codificador de contraseñas (password encoder), evitando así el almacenamiento de contraseñas en texto plano. En su lugar, las contraseñas se almacenan de forma cifrada mediante un algoritmo de hash, lo que las hace ilegibles. La implementación de este mecanismo es fundamental en cualquier aplicación que maneje autenticación de usuarios.

El password encoder permite seleccionar el algoritmo de codificación de las contraseñas. Además, se ha desarrollado una excepción personalizada para devolver códigos HTTP específicos en función del tipo de error que ocurra. Para manejar estas excepciones, se implementará un ExceptionHandler, el cual permitirá devolver mensajes personalizados junto con el código HTTP correspondiente.

Finalmente, se creará un aspecto (Aspect) que interceptará todas las solicitudes a los controladores. Este método solo se ejecutará en caso de que se lance una excepción del tipo AppException.

Se ha creado un MapStruct mapper en el cual se ignora el campo de contraseña, ya que no tiene el mismo formato en todas las etapas del proceso. También se ha definido la entidad de usuario, la cual está mapeada a una tabla en la base de datos. No se ha nombrado esta tabla como user, dado que PostgreSQL ya contiene una tabla reservada con ese nombre.

Por último, se procederá a la implementación de los endpoints de autenticación: login y registro. Una vez que el usuario ha iniciado sesión, el sistema devuelve un JWT recién generado. Lo mismo ocurre con el proceso de registro, donde se genera y retorna un JWT nuevo.

Cuando se crea una entidad en el sistema, es una buena práctica devolver un código HTTP 201 (Created) junto con la URL donde se puede acceder a la nueva entidad.