

DEMO 11

ApiRest | Spring Security 6 | Auth JWT

Introducción

Este **README** proporciona una descripción general de una aplicación de ejemplo creada con **Spring Security 6** y **JWT**. La aplicación utiliza **JWT** para autenticar y autorizar a los usuarios. Usaremos un **RestController** de **Alumnos** para los ejemplos de los endpoints con autenticación y comprobaremos por la consola del IDE la respuesta de nuestras peticiones.


Ha diferencia de la demo anterior, esta tiene como objetivo comprobar el autoconsumo de una **APIREST** desde nuestro propio **backend**. La organización del proyecto y las clases de autorización son iguales a la demo anterior, solo hay modificaciones en la clase **MainController** que es el controlador principal de la app y donde esta toda la lógica de autoconsumo.

Tecnologías utilizadas

- Spring Boot
- Spring Security 6
- JWT (JSON Web Token)
- Maven

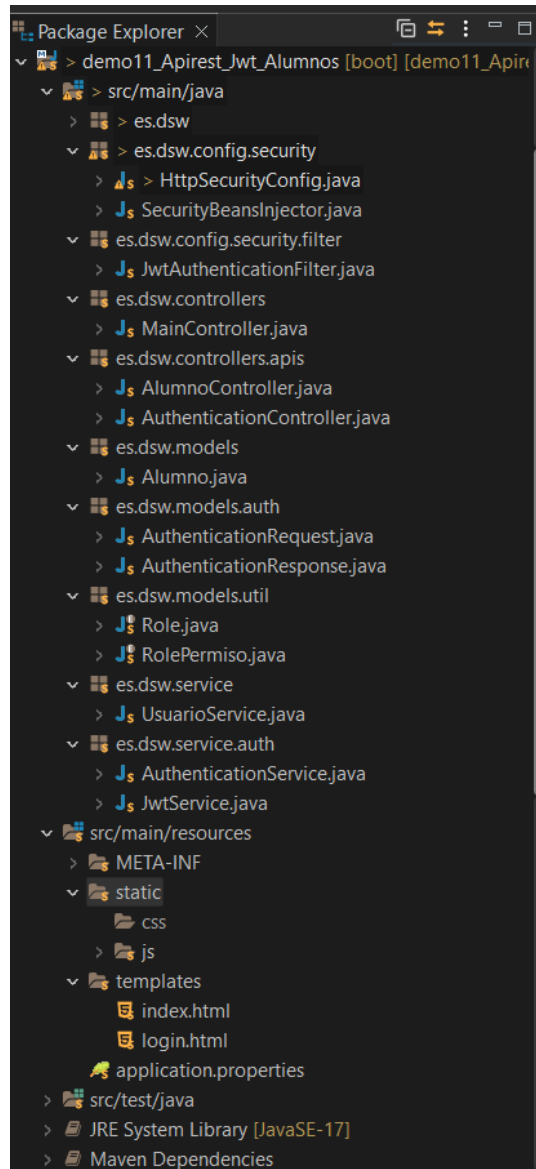
Estructura del proyecto

El proyecto está estructurado de la siguiente manera:

 **src/main/java/es.dsw:** Contiene las clases de Java de la aplicación usando la estructura MVC.

- **src/main/java/es.dsw/config.security:**
Clases → { **HttpSecurityConfig** || **SecurityBeansInjector** }
- **src/main/java/es.dsw/config.security.filter:**
Clases → { **JwtAuthenticationFilter** }
- **src/main/java/es.dsw/controllers.api:**
Clases → { **AlumnoController** || **AuthenticationController** }
- **src/main/java/es.dsw/models:**
Clases → { **Alumno** }
- **src/main/java/es.dsw/models.auth:**
Clases → { **AuthenticationRequest** || **AuthenticationResponse** }
- **src/main/java/es.dsw/models.util:**

- Clases → { **Role** || **RolePermiso** }
- **src/main/java/es.dsw/service:**
Clases → { **UsuarioService** }
 - **src/main/java/es.dsw/service.auth:**
Clases → { **AuthenticationService** || **JwtService** }



- ✚ **src/main/resources:** Contiene los archivos de configuración de la aplicación.
- ✚ **src/test/java:** Contiene las pruebas unitarias de la aplicación.

Configuración de Spring Security y Maven

La configuración de **Spring Security** se define en el archivo ubicado en **src/main/resources/application.properties**. Las propiedades siguientes se utilizan para configurar la autenticación JWT:

application.properties

```
spring.application.name=Java-SpringBoot-Thymeleaf-DEM011-APIREST-JWT-ALUMNOS

server.servlet.context-path=/api/demo11

security.jwt.secret-key =
bWkgY2xhdmUgc2VjcmlV0YSBzdXB1ciBkaWZpY2lsIDEyMzQ=
security.jwt.expiration-in-minutes = 30

logging.level.org.springframework.security=DEBUG
```

La configuración de **Maven** se define en el archivo ubicado en la raíz del proyecto **demo11_ApiRest_Jwt_Alumnos** llamdo **pom.xml**. Las dependencias siguientes son necesarias para la utilización de JWT:

pom.xml

La version se puede encapsula en una variable y se idica de la siguiente manera:

```
<properties>
    <java.version>17</java.version>
    <jjwt.version>0.12.5</jjwt.version>
</properties>
```

Podemos cargar la variable de la versión de la siguiente manera:

```
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-impl</artifactId>
    <version>${jjwt.version}</version>
</dependency>
```

Dependencias necesarias:

```
<!--  
https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-databind -->  
    <dependency>  
        <groupId>com.fasterxml.jackson.core</groupId>  
        <artifactId>jackson-databind</artifactId>  
    </dependency>  
  
    <!-- dependencias de JWT -->  
  
    <!--  
https://mvnrepository.com/artifact/io.jsonwebtoken/jjwt-api -->  
    <dependency>  
        <groupId>io.jsonwebtoken</groupId>  
        <artifactId>jjwt-api</artifactId>  
        <version>${jjwt.version}</version>  
    </dependency>  
  
    <!--  
https://mvnrepository.com/artifact/io.jsonwebtoken/jjwt-impl -->  
    <dependency>  
        <groupId>io.jsonwebtoken</groupId>  
        <artifactId>jjwt-impl</artifactId>  
        <version>${jjwt.version}</version>  
    </dependency>  
  
    <!--  
https://mvnrepository.com/artifact/io.jsonwebtoken/jjwt-jackson -->  
    <dependency>  
        <groupId>io.jsonwebtoken</groupId>  
        <artifactId>jjwt-jackson</artifactId>  
        <version>${jjwt.version}</version>  
    </dependency>  
  
    <!--  
https://mvnrepository.com/artifact/io.jsonwebtoken/jjwt -->  
    <dependency>  
        <groupId>io.jsonwebtoken</groupId>  
        <artifactId>jjwt</artifactId>  
        <version>${jjwt.version}</version>  
    </dependency>
```

Generación de tokens JWT

Los tokens **JWT** se generan utilizando el controlador **AuthenticationController**. El controlador expone endpoint con metodo **POST “/authenticate”** que acepta un nombre de usuario y una contraseña enviadas por el body con la clase **AuthenticationRequest**, devolviendo un **ResponseEntity** con el **JWT** de la clase **AuthenticatationResponse** la cual obtiene su valor de un método **login** de la clase **AuthenticationService** que recibe por parametro un **AuthenticationRequest**. Si las credenciales son válidas, el controlador devuelve un token JWT.

Validación de tokens JWT

Los tokens JWT se validan utilizando la clase filtro **JwtAuthenticationFilter**. El filtro intercepta todas las solicitudes y valida el token **JWT** presente en la cabecera **Authorization**. Si el token es válido, el filtro establece un objeto Principal en el contexto de seguridad de la clase **SecurityContextHolder** con un **UserDetails**.

En esta demo y las siguientes, el usuario se carga con el objeto **InMemoryUserDetailsManager** de Spring en la clase **UsuarioService**, dado que no estamos usando entidades ni base de datos para el login o para la gestion de usuarios.

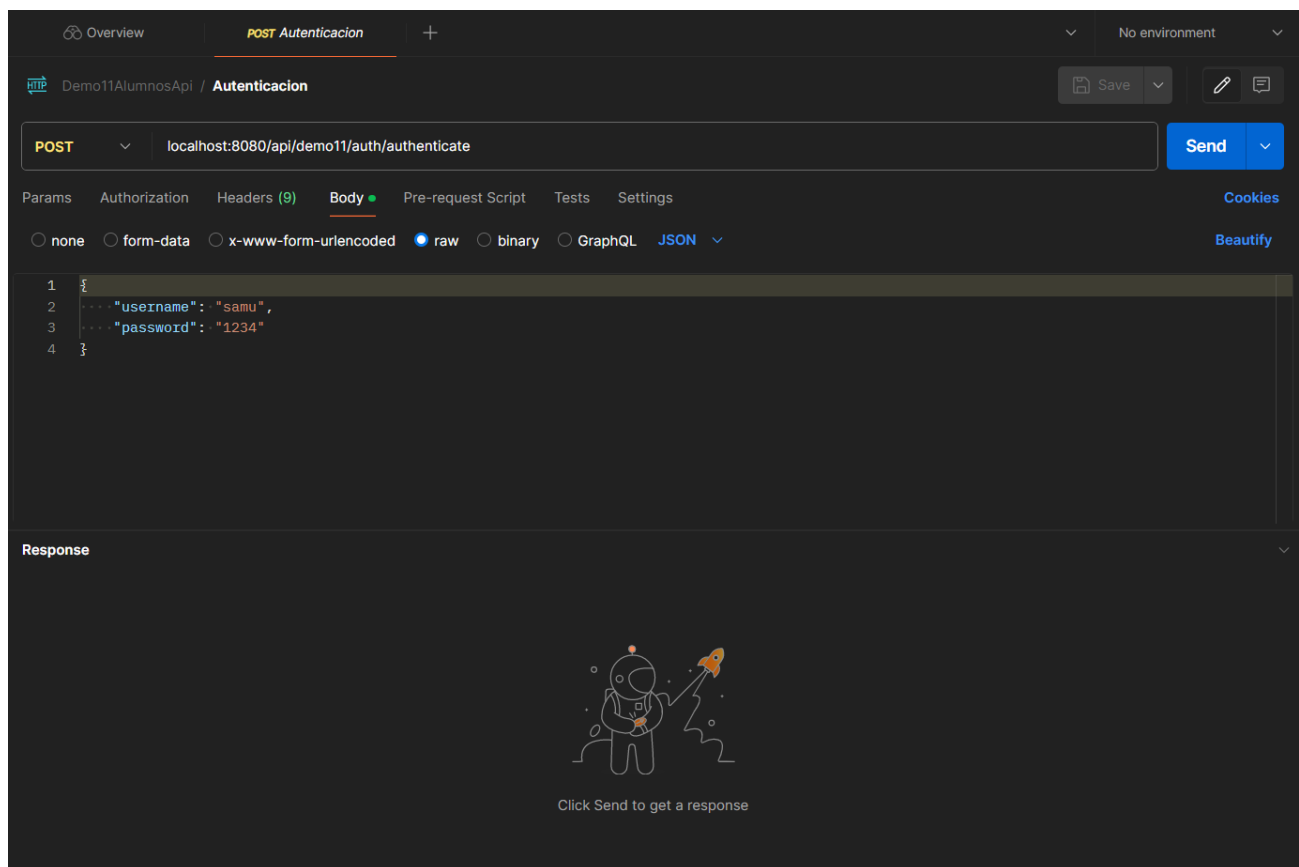
Autorización

La autorización se basa en roles. Los roles se asignan a los usuarios con el objeto **InMemoryUserDetailsManager** de Spring en la clase **UsuarioService**. La clase filtro **JwtAuthenticationFilter** establece los roles del usuario en el objeto **UserDetails**.

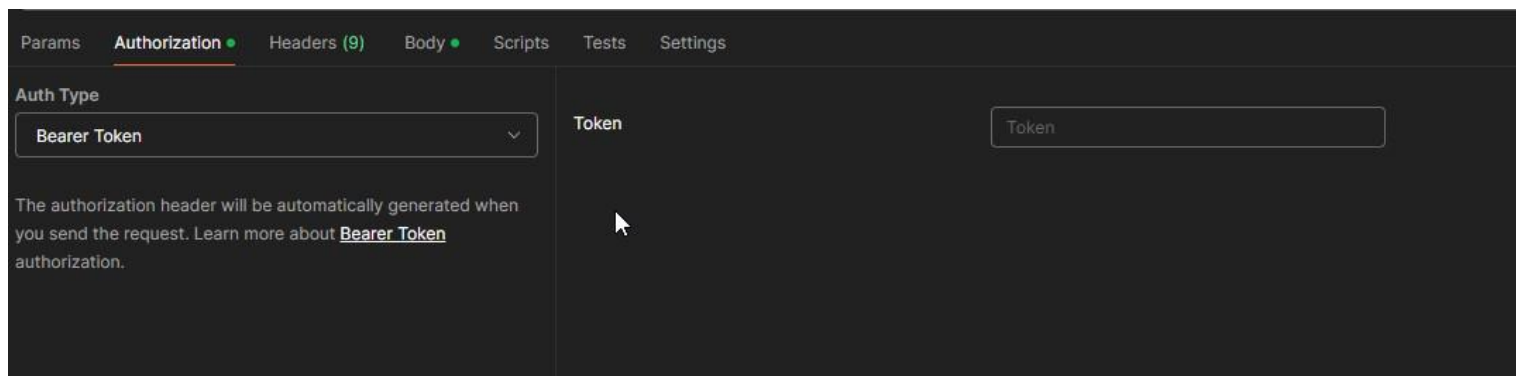
Ejemplo de uso

Para usar la aplicación usaremos una aplicación de peticiones como **Postman**.

1. Iniciamos el proyecto.
2. Se debe generar un token **JWT**, enviando una solicitud **POST** al endpoint **localhost:8080/api/demo11/auth/authenticate**, al tratarse de **JWT (JSON WEB TOKEN)** debemos pasar en formato **JSON** un nombre de usuario y una contraseña válidos, en Postman seleccionamos pasar los datos por el Body elegimos **raw** y luego **JSON**. La respuesta contendrá un token **JWT**.



3. Para acceder a un recurso protegido, debe incluir el token **JWT** en la cabecera **Authorization** de la solicitud, en Postman seleccionamos **Authorization** y elegimos el tipo **Bearer Token**, pegamos el token generado y le damos a Send para ver la respuesta.



Implementación

La implementación de la aplicación se puede encontrar en el código fuente del proyecto.

El código se puede descargar desde el repositorio de GitHub de la aplicación.

También se adjunta un video de muestra.

Próximos pasos

Se pueden realizar las siguientes mejoras en la aplicación:

- Implementar la autenticación de dos factores.
- Integrar la aplicación con un proveedor de identidad externo.
- Generar tokens JWT con diferentes duraciones.
- Implementar la actualización de tokens JWT.

Recursos adicionales

- [Spring Security](#)
- [JWT](#)