Spring Boot Security + JWT Hello World Ejemplo

Rameez Shaikh
13 de junio · 9 min de lectura ★

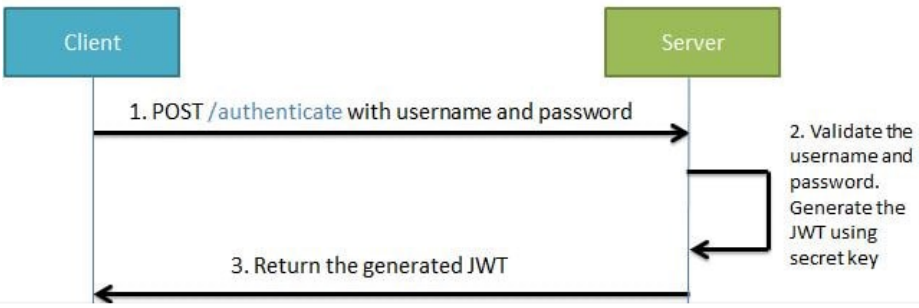
En este tutorial, desarrollaremos una aplicación Spring Boot que utiliza la autenticación JWT para asegurar una API REST expuesta. En este ejemplo, haremos uso de valores de usuario codificados para la autenticación del usuario. En el próximo tutorial, implementaremos Spring Boot + JWT + MYSQL JPA para almacenar y obtener credenciales de usuario. Cualquier usuario podrá consumir esta API solo si tiene un JSON Web Token (JWT) válido. En un tutorial anterior, aprendimos qué es JWT y cuándo y cómo usarlo.

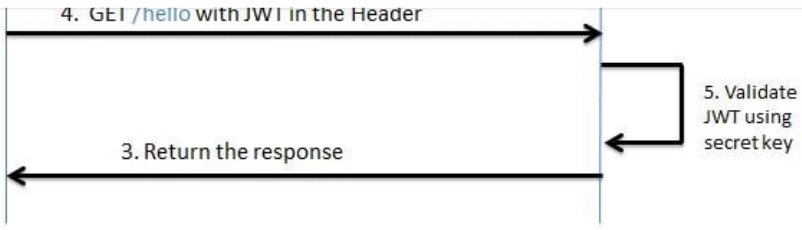
Este tutorial se explica en el siguiente video:

Para una mejor comprensión, desarrollaremos el proyecto por etapas:

Desarrolle una aplicación Spring Boot que exponga una API REST GET simple con mapeo / hola.

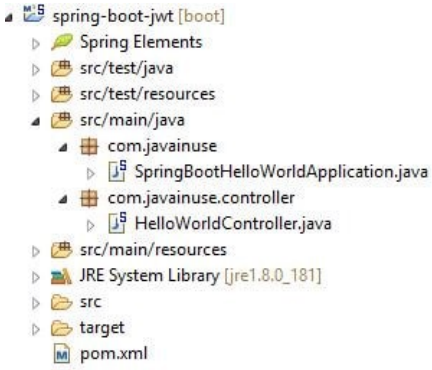
Configure Spring Security para JWT. Exponga REST POST API con mapeo / autenticación usando qué usuario obtendrá un token web JSON válido. Y luego, permita al usuario acceder a la API / hola solo si tiene un token válido





Desarrolle una aplicación Spring Boot que exponga una API GET REST

El proyecto Maven tendrá el siguiente aspecto:



El pom.xml es el siguiente:

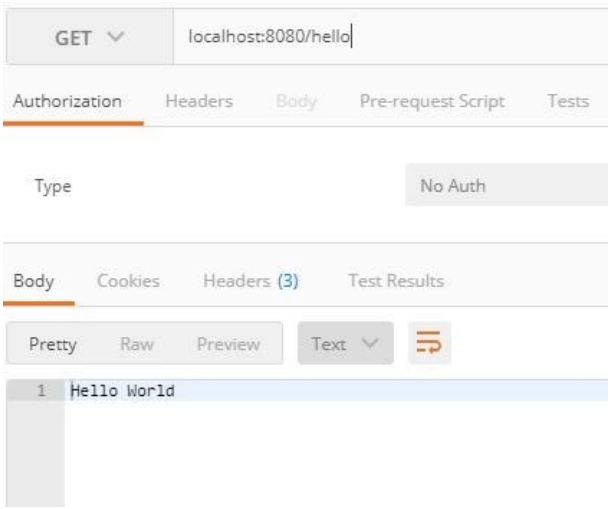
```
<project xmlns = "http://maven.apache.org/POM/4.0.0" xmlns: xsi =
"http://www.w3.org/2001/XMLSchema-instance"
        xsi: schemaLocation = "http: / /maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd ">
        <modelVersion> 4.0.0 </modelVersion>
        <groupId> com.javainuse < / groupId>
        <artifactId> spring-boot-jwt </artifactId>
        <version> 0.0.1-SNAPSHOT </version>

        <parent>
                <groupId> org.springframework.boot </groupId>
                <artifactId> spring-boot-starter -parent
</artifactId>
                <version> 2.1.1.RELEASE </version>
                <relativePath /> <! - buscar padre desde el
repositorio ->
        </parent>

        <properties>
                <project.build.sourceEncoding> UTF-8
</project.build.sourceEncoding>
                <project.reporting.outputEncoding> UTF-8
</project.reporting.outputEncoding>
                <java.version> 1.8 </java.version>
        </properties>

        <dependencies>
                <dependency>
                        <groupId> org.springframework.boot </
groupId>
                        <artifactId> spring-boot-starter-web
</artifactId>
                </dependency>
        </dependencies>

</project>
```

```
paquete com.javainuse.controller;

import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HelloWorldController {

        @RequestMapping ({"/ hello"})
        public String firstPage () {
                return "Hello World";
        }

}
```

Cree la clase bootstrap con Spring Boot Annotation

```
paquete com.javainuse;

importar org.springframework.boot.SpringApplication;
importar
org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
clase pública SpringBootHelloWorldApplication {

        public static void main (String [] args) {
                SpringApplication.run
(SpringBootHelloWorldApplication.class, args);
        }
}
```

Compile y ejecute SpringBootHelloWorldApplication.java como una aplicación Java.
Ir a **localhost: 8080 / hola**

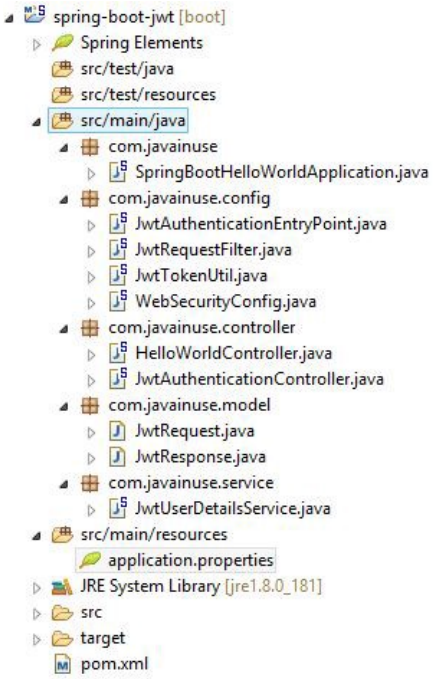


Configuración de Spring Security y JWT

Configuraremos Spring Security y JWT para realizar 2 operaciones:

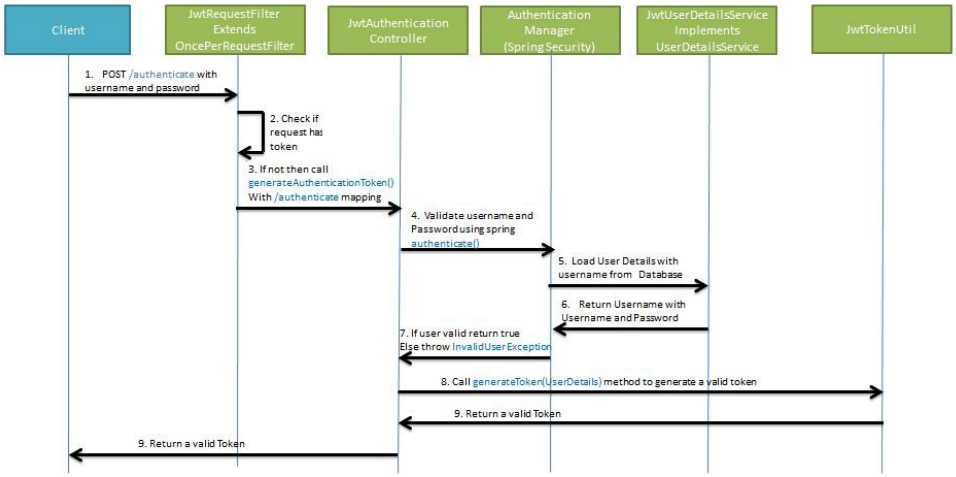
- **Generando JWT** : exponga una API POST con mapeo / **autenticación** . Al pasar el nombre de usuario y la contraseña correctos, generará un token web JSON (JWT)

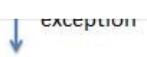
El Proyecto Maven será como sigue:



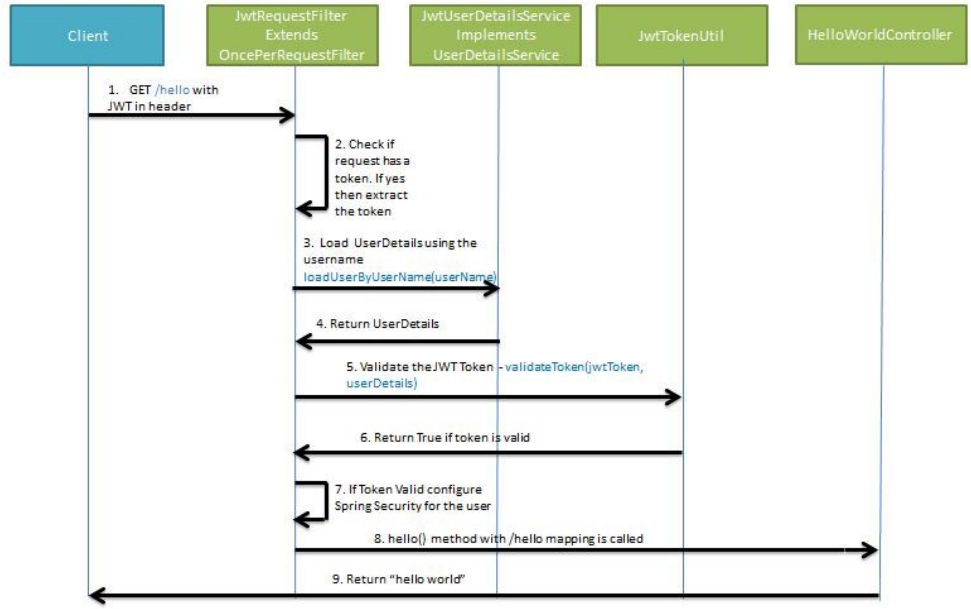
El flujo de secuencia para estas operaciones será el siguiente:

Generando JWT





Validando JWT



Agregue las dependencias de Spring Security y JWT

```
<project xmlns = "http://maven.apache.org/POM/4.0.0" xmlns: xsi =
"http://www.w3.org/2001/XMLSchema-instance"
        xsi: schemaLocation = "http: / /maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd ">
        <modelVersion> 4.0.0 </modelVersion>
        <groupId> com.javainuse < / groupId>
        <artifactId> spring-boot-jwt </artifactId>
        <version> 0.0.1-SNAPSHOT </version>

        <parent>
                <groupId> org.springframework.boot </groupId>
                <artifactId> spring-boot-starter -parent
</artifactId>
                <version> 2.1.1.RELEASE </version>
                <relativePath /> <! - buscar padre desde el
repositorio ->
        </parent>

        <properties>
                <project.build.sourceEncoding> UTF-8
</project.build.sourceEncoding>
                <project.reporting.outputEncoding> UTF-8
</project.reporting.outputEncoding>
                <java.version> 1.8 </java.version>
        </properties>

        <dependencies>
                <dependency>
                        <groupId> org.springframework.boot </
groupId>
                        <artifactId> spring-boot-starter-web
</artifactId>
                </dependency>
                <dependency>
                        <groupId> org.springframework.boot
</groupId>
                        <artifactId> spring-boot-starter-security
</artifactId>
                < / dependency>
```

```
                        <version> 0.9.1 </version>
                    <
/ dependency>    </dependencies>

</project>
```

Defina la application.properties. Como se vio en el tutorial anterior de JWT, especificamos la clave secreta que usaremos para el algoritmo de hash. La clave secreta se combina con el encabezado y la carga útil para crear un hash único. Solo podemos verificar este hash si tiene la clave secreta.

```
jwt.secret = javainuse
```

JwtTokenUtil

El JwtTokenUtil es responsable de realizar operaciones JWT como la creación y validación. Utiliza el io.jsonwebtoken.Jwts para lograr esto.

```
paquete com.javainuse.config;

import java.io.Serializable;
import java.util.Date;
importar java.util.HashMap;
import java.util.Map;
import java.util.function.Function;

import org.springframework.beans.factory.annotation.Value;
importar org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Component;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;

@Component
public class JwtTokenUtil implementa Serializable {

        private static final long serialVersionUID =
-25501851656260074888L;

        público estático final largo JWT_TOKEN_VALIDITY = 5 * 60 *
60;

        @Value ("$ {jwt.secret}")
        secreto de cadena privado;

// recupera el nombre de usuario de jwt token

        public String getUsernameFromToken (String token) {

                return getClaimFromToken (token, Claims ::
getSubject);

        }

// recupera la fecha de vencimiento del jwt token

        public Date getExpirationDateFromToken (String token) {

                return getClaimFromToken (token, Claims ::
getExpiration);

        }
```

```
                final Claims Claim = getAllClaimsFromToken (token);

                return reclamosResolver.apply (reclamos);

        }

// para recuperar cualquier información del token, necesitaremos la clave secreta

        Private Claims getAllClaimsFromToken (String token) {









                return Jwts.parser (). setSigningKey (secreto) .parseClaimsJws (token) .getBody ();
        }

// comprueba si el token ha caducado

        privado Boolean isTokenExpired (String token) {

                fecha final de caducidad = getExpirationDateFromToken (token);

                expiración de retorno. antes (nueva Fecha ());

        }

// generar token para el usuario

        public String generateToken (UserDetails userDetails) {

                Map <String, Object>

Claim

= new HashMap <> ();

                return doGenerateToken (reclamaciones, userDetails.getUsername ());

        }

// al crear el token -

        // 1.

Defina los reclamos del token, como Emisor, Caducidad, Asunto y el ID

        // 2.

Firme el JWT usando el algoritmo HS512 y la clave secreta.
```

```
        // 3. De acuerdo con la serialización compacta JWS
(https://tools.ietf.org/html/draft-ietf-jose-json-web-signature-
41#section-3.1)
        // compactación del JWT a una cadena
        privada segura para URL doGenerateToken (Map <String,
Object> reclamos, String asunto) {

                return Jwts.builder (). setClaims (reclamos)
.setSubject (subject) .setIssuedAt (new Date
(System.currentTimeMillis ()))
                                .setExpiration (new Date (System
.currentTimeMillis () + JWT_TOKEN_VALIDITY * 1000))
                                .signWith
(SignatureAlgorithm.HS512, secret) .compact ();
        }

// validar token

        public Boolean validateToken (token de cadena, UserDetails
userDetails) {

                final String username = getUsernameFromToken
(token);




                return (username.equals (userDetails.getUsername ())
&&! isTokenExpired (token));
        }
}
```

JWTUserDetailsService

JWTUserDetailsService implementa la interfaz Spring Security UserDetailsService. Anula el loadUserByUsername para obtener detalles de usuario de la base de datos utilizando el nombre de usuario. Spring Security Authentication Manager llama a este método para obtener los detalles del usuario de la base de datos al autenticar los detalles del usuario proporcionados por el usuario. Aquí estamos obteniendo los **detalles del usuario de una lista de usuarios codificada** . En el próximo tutorial agregaremos la implementación DAO para obtener los detalles del usuario de la base de datos. Además, la contraseña de un usuario se almacena en formato cifrado con BCrypt. Anteriormente hemos visto Spring Boot Security - Codificación de contraseña usando Bcrypt. Aquí, utilizando el generador de Bcrypt en línea, puede generar el Bcrypt para una contraseña.

```
paquete com.javainuse.service;

import java.util.ArrayList;

import org.springframework.security.core.userdetails.User;
importar org.springframework.security.core.userdetails.UserDetails;
importar
org.springframework.security.core.userdetails.UserDetailsService;
import
org.springframework.security.core.userdetails.UsernameNotFoundExcepti
on;
import org.springframework.stereotype.Service;

@Service
clase pública implementa JwtUserDetailsService UserDetailsService {

        @ Override
        DetallesUsuario loadUserByUsername pública (String nombre de
usuario) lanza UsernameNotFoundException {
```

```
                                                nueva ArrayList <> ());
                } else {
                        lanzar nuevo UsernameNotFoundException
("Usuario no encontrado con nombre de usuario:" + nombre de usuario);
                }
        }
}
```

JwtAuthenticationController

Exponga una API POST / autenticación utilizando el JwtAuthenticationController. La API POST obtiene el nombre de usuario y la contraseña en el cuerpo: con Spring Authentication Manager autenticamos el nombre de usuario y la contraseña. Si las credenciales son válidas, se crea un token JWT utilizando JWTTokenUtil y se proporciona al cliente.

```
paquete com.javainuse.controller;

import java.util.Objects;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
importar
org.springframework.security.authentication.AuthenticationManager;
importar
org.springframework.security.authentication.BadCredentialsException;
importar
org.springframework.security.authentication.DisabledException;
import
org.springframework.security.authentication.UsernamePasswordAuthentic
ationToken;
importar org.springframework.security.core.userdetails.UserDetails;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;
import com.javainuse.service.JwtUserDetailsService;

import com.javainuse.config.JwtTokenUtil;
import com.javainuse.model.JwtRequest;
import com.javainuse.model.JwtResponse;

@RestController
@CrossOrigin
public class JwtAuthenticationController {

        @Autowired
        private AuthenticationManager autenticaciónManager;

        @Autowired
        private JwtTokenUtil jwtTokenUtil;

        @Autowired
        private JwtUserDetailsService userDetailsService;

        @RequestMapping (value = "/ authenticate", method =
RequestMethod.POST)
        public ResponseEntity <?> CreateAuthenticationToken
(@RequestBody JwtRequest autenticaciónRequest) arroja la excepción {

                authenticate (autenticaciónRequest.getUsername (),
autenticaciónRequest.getPassword ());

                UserDetails finales userDetails = userDetailsService
                                .loadUserByUsername
(autenticaciónRequest.getUsername ());

                token de cadena final = jwtTokenUtil.generateToken
(userDetails);
```

```
        autenticación privada vacía (nombre de usuario de cadena,
contraseña de cadena) arroja una excepción {
                try {
                        autenticaciónManager.authenticate (nuevo
nombre de usuario ContraseñaAuthenticationToken (nombre de usuario,
contraseña));
                } catch (DisabledException e) {
                        lanzar una nueva excepción ("USER_DISABLED",
e);
                } catch (BadCredentialsException e) {
                        lanzar una nueva excepción
("INVALID_CREDENTIALS", e);
                }
        }
}
```

JwtRequest

Esta clase es necesaria para almacenar el nombre de usuario y la contraseña que recibimos del cliente.

```
paquete com.javainuse.model;

import java.io.Serializable;

public class JwtRequest implementa Serializable {

        private static final long serialVersionUID =
5926468583005150707L;

        nombre de usuario de cadena privada;
        contraseña de cadena privada;

        // necesita un constructor predeterminado para JSON
        Analizar public JwtRequest ()
        {

        }

        public JwtRequest (String username, String password) {
                this.setUsername (username);
                this.setPassword (contraseña);
        }

        public String getUsername () {
                return this.username;
        }

        public void setUsername (String username) {
                this.username = username;
        }

        public String getPassword () {
                devuelve this.password;
        }

        public void setPassword (contraseña de cadena) {
                this.password = contraseña;
        }
}
```

JwtResponse

Esta clase es necesaria para crear una respuesta que contenga el JWT que se devolverá al usuario.

```
import java.io.Serializable;

la clase pública JwtResponse implementa Serializable {

        private static final long serialVersionUID =
-8091879091924046844L;
        Cadena final privada jwttoken;

        public JwtResponse (String jwttoken) {
                this.jwttoken = jwttoken;
        }

        public String getToken () {
                return this.jwttoken;
        }
}
```

JwtRequestFilter

JwtRequestFilter extiende la clase Spring Web Filter OncePerRequestFilter. Para cualquier solicitud entrante, esta clase de filtro se ejecuta. Comprueba si la solicitud tiene un token JWT válido. Si tiene un token JWT válido, establece la autenticación en el contexto, para especificar que el usuario actual está autenticado.

```
paquete com.javainuse.config;

import java.io.IOException;

import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.security.authentication.UsernamePasswordAuthentic
ationToken;
import
org.springframework.security.core.context.SecurityContextHolder;
importar org.springframework.security.core.userdetails.UserDetails;
import
org.springframework.security.web.authentication.WebAuthenticationDeta
ilsSource;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;

import com.javainuse.service.JwtUserDetailsService;

import io.jsonwebtoken.ExpiredJwtException;

@Component
public class JwtRequestFilter extiende OncePerRequestFilter {

        @Autowired
        private JwtUserDetailsService jwtUserDetailsService;

        @Autowired
        private JwtTokenUtil jwtTokenUtil;

        @Override
        protected void doFilterInternal (solicitud
HttpServletRequest, respuesta HttpServletResponse, cadena
FilterChain)
                        arroja ServletException, IOException {

                final String requestTokenHeader = request.getHeader
("Autorización");
```

```
**Elimine la palabra Bearer y obtenga**
                **// solo el token**
                if (requestTokenHeader! = Null &&
requestTokenHeader.startsWith ("Bearer")) {
                        jwtToken = requestTokenHeader.substring (7);
                        pruebe {
                                nombre de usuario =
jwtTokenUtil.getUsernameFromToken (jwtToken);
                        } catch (IllegalArgumentException e) {
                                System.out.println ("No se puede
obtener el token JWT");
                        } catch (ExpiredJwtException e) {
                                System.out.println ("JWT Token ha
expirado");
                        }
                } else {
                        logger.warn ("El token JWT no comienza con
la cadena de portador");
                }

**// Una vez que obtengamos el token validarlo.**


                if (username! = null &&
SecurityContextHolder.getContext (). getAuthentication () == null) {

                        UserDetails userDetails =
this.jwtUserDetailsService.loadUserByUsername (username);

**// si el token es válido, configure Spring Security para establecer
manualmente**

                        **// autenticación**

                        if (jwtTokenUtil.validateToken (jwtToken,
userDetails)) {

                                UsernamePasswordAuthenticationToken
usernamePasswordAuthenticationToken = new
UsernamePasswordAuthenticationToken (

                                                userDetails, null,
user)

                                usernamePasswordAuthenticationToken

                                                .setDetails (nuevo
WebAuthenticationDetailsSource (). buildDetails (solicitud));

**// Después de establecer la autenticación en el contexto,
especificamos**

                                **// que el usuario actual está
autenticado. Por lo tanto, pasa las**
                                **// Configuraciones de seguridad de
Spring con éxito.**
                                SecurityContextHolder.getContext
(). SetAuthentication (usernamePasswordAuthenticationToken);
                        }
                }
                chain.doFilter (solicitud, respuesta);
        }

}
```

de su método. Rechaza todas las solicitudes no autenticadas y envía el código de error 401

```
paquete com.javainuse.config;

import java.io.IOException;
import java.io.Serializable;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.security.core.AuthenticationException;
importar org.springframework.security.web.AuthenticationEntryPoint;
import org.springframework.stereotype.Component;

@Component
public class JwtAuthenticationEntryPoint implementa
AuthenticationEntryPoint, serializable {

        privado estático final largo serialVersionUID =
-78586995589532438 75L;

        @Override
        public void commence (solicitud HttpServletRequest,
respuesta HttpServletResponse,
                              AuthenticationException authException)
arroja IOException {

                response.sendError
(HttpServletResponse.SC_UNAUTHORIZED, "No autorizado");
        }
}
```

WebSecurityConfig

Esta clase extiende WebSecurityConfigurerAdapter es una clase de conveniencia que permite la personalización de WebSecurity y HttpSecurity.

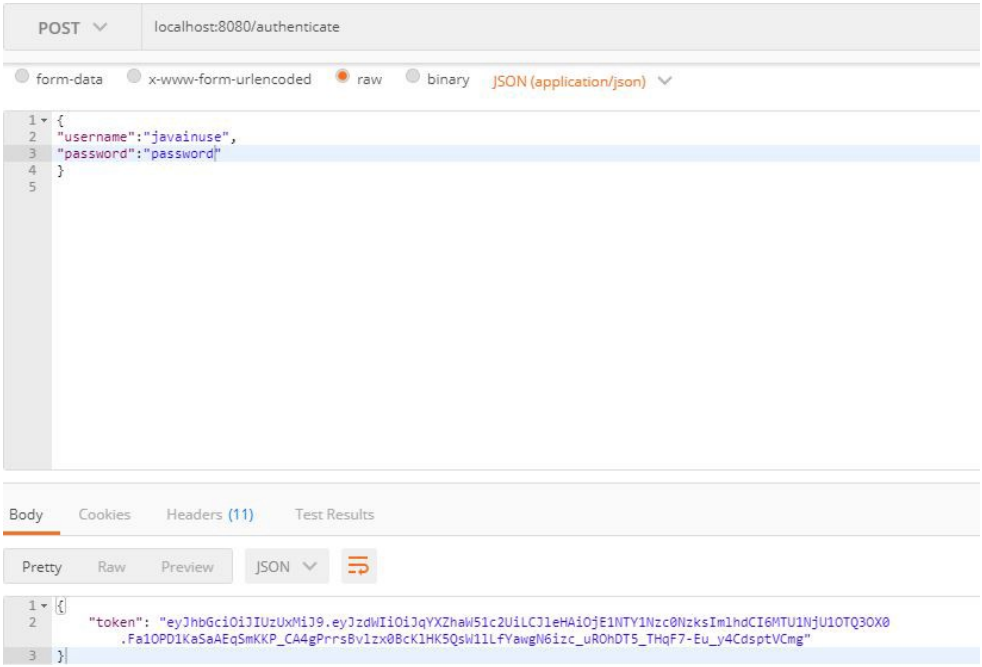
```
paquete com.javainuse.config;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
importar org.springframework.context.annotation.Configuration;
importar
org.springframework.security.authentication.AuthenticationManager;
importar
org.springframework.security.config.annotation.authentication.builder
s.AuthenticationManagerBuilder;
importar
org.springframework.security.config.annotation.method.configuration.E
nableGlobalMethodSecurity;
importar
org.springframework.security.config.annotation.web.builders.HttpSecur
ity;
importar
org.springframework.security.config.annotation.web.configuration.Enab
leWebSecurity;
importar
org.springframework.security.config.annotation.web.configuration.WebS
ecurityConfigurerAdapter;
import
org.springframework.security.config.http.SessionCreationPolicy;
importar
org.springframework.security.core.userdetails.UserDetailsService;
importar
org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
importar
org.springframework.security.crypto.password.PasswordEncoder;
import
```

```
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity (prePostEnabled = true)
clase pública WebSecurityConfig extiende WebSecurityConfigurerAdapter
{

        @Autowired
        private JwtAuthenticationEntryPoint
jwtAuthenticationEntryPoint;

        @Autowired
        private UserDetailsService jwtUserDetailsService;

        @Autowired
        private JwtRequestFilter jwtRequestFilter;

        @Autowired
        public void configureGlobal (AuthenticationManagerBuilder
auth) lanza Exception {
                // configure AuthenticationManager para que sepa
desde dónde cargar
                // usuario para credenciales coincidentes
                // Use BCryptPasswordEncoder
                auth.userDetailsService (jwtUserDetailsService)
.passworddercoder)
        }

        @Bean
        public PasswordEncoder passwordEncoder () {
                return new BCryptPasswordEncoder ();
        }

        @Bean
        @Override
        public AuthenticationManager autenticacionManagerBean ()
lanza Excepción {
                return super.authenticationManagerBean ();
        }

        @Override
        protected void configure (HttpSecurity httpSecurity) lanza
Exception {
                // No necesitamos CSRF para este ejemplo
                httpSecurity.csrf (). Disable ()
                                // no autentique esta solicitud en
particular
                                .authorizeRequests (). AntMatchers
("/ authenticate ") .permitAll ().
                                // todas las demás solicitudes
deben autenticarse
                                anyRequest (). authenticated (). y
().
                                // asegúrese de que usemos sesión
sin estado; La sesión no se utilizará para
                                // almacenar el estado del usuario.
                                exceptionHandling ().
autenticaciónEntryPoint (jwtAuthenticationEntryPoint) .and ().
sessionManagement ()
                                .sessionCreationPolicy
(SessionCreationPolicy.STATELESS);

                // Agregue un filtro para validar los tokens con
cada solicitud
                httpSecurity.addFilterBefore (jwtRequestFilter,
UsernamePasswordAuthenticationFilter.class);
        }
}
```

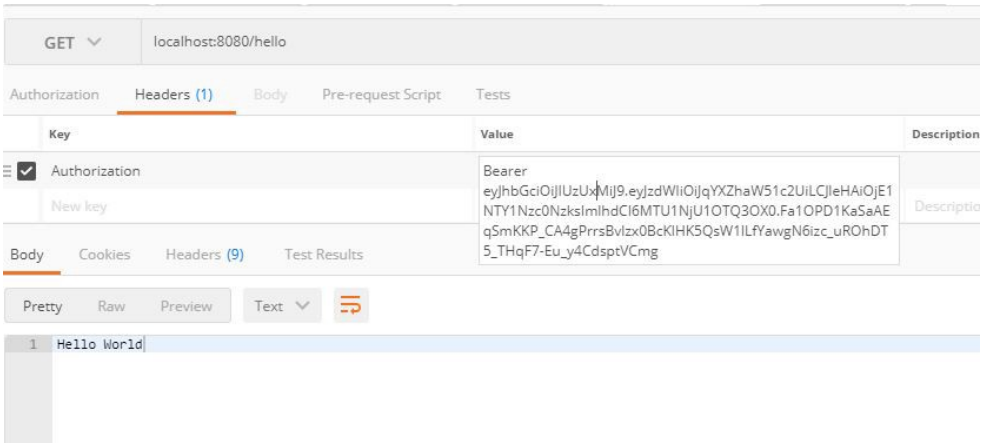
Inicie la aplicación Spring Boot

- Generar un token web JSON:

javainuse y la contraseña es contraseña.



- Validar el token web JSON
- - Intente acceder a la url localhost: 8080 / hello utilizando el token generado anteriormente en el encabezado de la siguiente manera



Spring Boot Jwt Seguridad de primavera Oauth Oauth2