Manual Json Web Token

David Doichita

1. Pre-requisitos

1.1 Entidades

Tener una entidad de usuario y una tabla de roles parecida al ejemplo siguiente en la base de datos y tanto mapeados como con su DTO respectivo en spring boot.

```
+--------+-------------+------+-----+---------+----------------+
| Field  | Type        | Null | Key | Default | Extra          |
+--------+-------------+------+-----+---------+----------------+
| id     | int(11)     | NO   | PRI | NULL    | auto_increment |
| nombre | varchar(60) | YES  |     | NULL    |                |
+--------+-------------+------+-----+---------+----------------+
```

y para relación usuario con roles:

```
+------------+---------+------+-----+---------+----------------+
| Field      | Type    | Null | Key | Default | Extra          |
+------------+---------+------+-----+---------+----------------+
| id         | int(11) | NO   | PRI | NULL    | auto_increment |
| idprofesor | int(11) | YES  | MUL | NULL    |                |
| idrol      | int(11) | YES  | MUL | NULL    |                |
+------------+---------+------+-----+---------+----------------+
```

En mi caso todo esta hecho con profesor, cambiarlo a lo que necesitéis.

1.2 Dependecias del proyecto

Estas son mis dependencias:

```
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-security</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-thymeleaf</artifactId>
        </dependency>
        <dependency>
            <groupId>me.paulschwarz</groupId>
```

```
            <artifactId>spring-dotenv</artifactId>
            <version>2.5.4</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.thymeleaf.extras</groupId>
            <artifactId>thymeleaf-extras-springsecurity6</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <scope>runtime</scope>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>com.mysql</groupId>
            <artifactId>mysql-connector-j</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-validation</artifactId>
        </dependency>
        <dependency>
            <groupId>io.jsonwebtoken</groupId>
            <artifactId>jjwt-api</artifactId>
            <version>0.11.5</version>
        </dependency>
        <dependency>
            <groupId>io.jsonwebtoken</groupId>
            <artifactId>jjwt-impl</artifactId>
            <version>0.11.5</version>
```

```
        </dependency>
        <dependency>
            <groupId>io.jsonwebtoken</groupId>
            <artifactId>jjwt-jackson</artifactId>
            <version>0.11.5</version>
        </dependency>
```

Comparadlas con las vuestras

1.3 application.properties

Necesitais añadir estas dos propiedas al application.properties de vuestro proyecto.

security.jwt.secret=
security.jwt.expiration=86400000

La propiedad security.jwt.secret debe ser un string de 32 bytes en formato base64, convenientemente el git bash puede ejecutar este comando:

openssl rand -base64 32

Que nos da un string de 32 bytes en formato base64

La propiedad security.jwt.expiration es el tiempo en el que expirara el token, yo lo he puesto a 24 horas.

1.4 Modificar el UsuarioDTO para que contega UserDetails

Implementamos UserDetails en el UsuarioDTO y añadimos los métodos requeridos

```
public class ProfesorDTO implements UserDetails {

    private Long id;
    [...]
    private String contrasenya;
    private String email;
    private Set<RolDTO> roles;

    [...]

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return roles.stream().map(
            rol -> new SimpleGrantedAuthority("ROLE_" +
                 rol.getNombre().toUpperCase())
        ).collect(Collectors.toList());
    }

    @Override
```

```
    public String getPassword() {
        return this.contrasenya;
    }

    @Override
    public String getUsername() {
        return this.email;
    }
}
```

2. Configuración de la aplicacion

2.1 ApplicationConfiguration

Creamos la siguiente clase:

```
@Configuration
public class ApplicationConfiguration {

    @Autowired
    private ProfesorRepository profesorRepository;

    @Bean
    public UserDetailsService userDetailsService() {
        return username ->
profesorRepository.findByEmail(username).map(ProfesorDTO::convertToDTO)
                .orElseThrow(() -> new IllegalArgumentException("Usuario
no encontrado: " + username));
    }

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

    @Bean
    public AuthenticationManager
authenticationManager(AuthenticationConfiguration config) throws Exception
{
        return config.getAuthenticationManager();
    }

    @Bean
    public AuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider provider = new
DaoAuthenticationProvider();
        provider.setUserDetailsService(this.userDetailsService());
```

```
        provider.setPasswordEncoder(this.passwordEncoder());
        return provider;
    }
}
```

2.2 JwtAuthenticationFilter

Creamos la siguiente clase:

```
@Component
@AllArgsConstructor
public class JwtAuthenticationFilter extends OncePerRequestFilter {

    private final HandlerExceptionResolver handlerExceptionResolver;
    private final JwtService jwtService;
    private final UserDetailsService userDetailsService;

    @Override
    protected void doFilterInternal(HttpServletRequest request,
HttpServletResponse response, FilterChain filterChain)
            throws ServletException, IOException {
        final String authHeader = request.getHeader("Authorization");

        if (authHeader == null || !authHeader.startsWith("Bearer ")) {
            filterChain.doFilter(request, response);
            return;
        }

        try {
            final String jwt = authHeader.substring(7);
            final String userEmail = jwtService.extractUsername(jwt);

            Authentication authentication =
SecurityContextHolder.getContext().getAuthentication();

            if (userEmail != null && authentication == null) {
                UserDetails userDetails =
this.userDetailsService.loadUserByUsername(userEmail);

                if (jwtService.isTokenValid(jwt, userDetails)) {
                    UsernamePasswordAuthenticationToken authToken = new
UsernamePasswordAuthenticationToken(userDetails,
                            null, userDetails.getAuthorities());

                    authToken.setDetails(new
WebAuthenticationDetailsSource().buildDetails(request));

SecurityContextHolder.getContext().setAuthentication(authToken);
```

```
                }
            }

            filterChain.doFilter(request, response);
        } catch (Exception e) {
            handlerExceptionResolver.resolveException(request, response,
null, e);
        }
    }
}
```

2.3 MethodSecurityConfig

Creamos la siguiente clase, aunque no tenga nada solo activa la verificación de roles para los métodos:

```
@Configuration
@EnableMethodSecurity(prePostEnabled = true)
public class MethodSecurityConfig {}
```

2.4 SecurityConfiguration

Creamos la siguiente clase, donde en el metodo securityFilterChain podeis ver que tengo algunas rutas con la autorizacion desabilitada, como es el login:

```
@Configuration
@EnableWebSecurity
@AllArgsConstructor
public class SecurityConfiguration {

    @Autowired
    private AuthenticationProvider authenticationProvider;
    @Autowired
    private JwtAuthenticationFilter jwtAuthenticationFilter;

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http)
throws Exception {
        http
                .csrf(csrf -> csrf.disable())
                .authorizeHttpRequests(authorize -> authorize
                        .requestMatchers("/api/auth/**").permitAll()
                        .requestMatchers("/api/intervalos/**").permitAll()
                        .requestMatchers("/api/dias/**").permitAll()

.requestMatchers("/api/cuadrantes/currentWeek").permitAll()
```

```
.requestMatchers("/api/cuadrantes/today").permitAll()
                        .anyRequest().authenticated())
                .sessionManagement(session ->
session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
                .authenticationProvider(authenticationProvider);

        http.addFilterBefore(jwtAuthenticationFilter,
UsernamePasswordAuthenticationFilter.class);

        http.headers(headers -> headers.frameOptions(f -> f.disable()));

        http.cors(cors ->
cors.configurationSource(corsConfigurationSource()));

        return http.build();
    }

    @Bean
    public CorsConfigurationSource corsConfigurationSource() {
        CorsConfiguration configuration = new CorsConfiguration();

configuration.setAllowedOrigins(List.of("https://localhost:4200"));
        configuration.setAllowedMethods(List.of("GET", "POST", "PUT",
"DELETE", "OPTIONS"));
        configuration.setAllowedHeaders(List.of("*"));

        UrlBasedCorsConfigurationSource source = new
UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration("/**", configuration);

        return source;
    }
}
```

3. Creacion DTOs para la comunicacion con el cliente

3.1 LoginUsuarioDTO

Creamos el siguiente DTO para la entrada de los datos del login del usuario:

```
public class LoginUsuarioDTO implements Serializable {

    private String email;
    private String password;
}
```

No es estrictamente necesario, lo podéis hacer en el RestController también con @RequestParam

3.2 LoginResponseDTO

Creamos el siguiente DTO con el que le mandaremos el token y la información que queráis al usuario cuando el login sea correcto:

```
public class LoginResponseDTO implements Serializable {

    private String token;
    private Long expiraEn;
}
```

4. Configuracion repository usuario

Tened un metodo para buscar al usuario por sus credenciales, por ejemplo:

```
    @Query("SELECT p FROM Profesor p WHERE p.email = ?1")
    Optional<Profesor> findByEmail(String email);
```

5. Servicios de autenticacion

5.1 JwtService

Creamos el servicio que nos va a proporcionar la codificion y la decodificacion del token del usuario

```
@Service
public class JwtService {

    private static final Logger log =
LoggerFactory.getLogger(JwtService.class);

    @Value("${security.jwt.secret}")
    private String secretKey;

    @Value("${security.jwt.expiration}")
    private long jwtExpiration;

    /**
     * Extrae el nombre de usuario del token
     *
     * @param token token de donde extaer el usuario
     * @return nombre de usuario
     */
```

```java
    public String extractUsername(String token) {
        return extractClaim(token, Claims::getSubject);
    }

    /**
     * Extrae una Claim específica del token usando un resolutor
     *
     * @param <T>            tipo de la Claim
     * @param token          token de donde extraer la Claim
     * @param claimsResolver resolutor de la Claim
     * @return Claim extraida
     */
    public <T> T extractClaim(String token, Function<Claims, T>
claimsResolver) {
        final Claims claims = extractAllClaims(token);
        return claimsResolver.apply(claims);
    }

    /**
     * Genera un token para un usuario
     *
     * @param userDetails usuario para el que se genera el token
     * @return token generado
     */
    public String generateToken(UserDetails userDetails) {
        return generateToken(new HashMap<>(), userDetails);
    }

    /**
     * Genera un token para un usuario con Claims adicionales
     *
     * @param extraClaims Claims adicionales
     * @param userDetails usuario para el que se genera el token
     * @return token generado
     */
    public String generateToken(Map<String, Object> extraClaims,
UserDetails userDetails) {
        return buildToken(extraClaims, userDetails, jwtExpiration);
    }

    /**
     * Devuelve el tiempo de expiracion del token
     *
     * @return tiempo de expiracion del token
     */
    public long getExpirationTime() {
        return jwtExpiration;
    }

    /**
```

```
     * Construye el token JWT
     *
     * @param extraClaims Claims adicionales
     * @param userDetails usuario para el que se genera el token
     * @param expiration  tiempo de expiracion del token
     * @return token generado
     */
    private String buildToken(Map<String, Object> extraClaims, UserDetails
userDetails, long expiration) {
        return Jwts.builder()
                .setClaims(extraClaims)
                .setSubject(userDetails.getUsername())
                .setIssuedAt(new Date(System.currentTimeMillis()))
                .setExpiration(new Date(System.currentTimeMillis() +
expiration))
                .signWith(getSignInKey(), SignatureAlgorithm.HS256)
                .compact();
    }

    /**
     * Comprueba si un token es valido para un usuario
     *
     * @param token       token a validar
     * @param userDetails usuario para el que se valida el token
     * @return true si el token es valido, false en caso contrario
     */
    public boolean isTokenValid(String token, UserDetails userDetails) {
        final String username = extractUsername(token);
        boolean isValid = username.equals(userDetails.getUsername()) &&
!isTokenExpired(token);

        log.info(this.getClass().getSimpleName() + " isTokenValid: token
validado para el usuario {}: {}", username,
                isValid);
        return isValid;
    }

    /**
     * Comprueba si un token a expirado
     *
     * @param token token a comprobar
     * @return true si el token a expirado, false en caso contrario
     */
    private boolean isTokenExpired(String token) {
        return extractExpiration(token).before(new Date());
    }

    /**
     * Extrae la fecha de expiracion del token
     *
```

```
     * @param token token de donde extraer la fecha de expiracion
     * @return fecha de expiracion
     */
    private Date extractExpiration(String token) {
        return extractClaim(token, Claims::getExpiration);
    }

    /**
     * Extrae todas las Claims del token
     *
     * @param token token de donde extraer las Claims
     * @return Claims extraidas
     */
    private Claims extractAllClaims(String token) {
        try {
            return Jwts.parserBuilder()
                    .setSigningKey(getSignInKey())
                    .build()
                    .parseClaimsJws(token)
                    .getBody();
        } catch (Exception e) {
            log.error(" extractAllClaims: fallo al extraer todas las
Claims del token: {}", e.getMessage());
            return null;
        }
    }

    /**
     * Devuelve la clave secreta para firmar el token
     *
     * @return clave secreta
     */
    private Key getSignInKey() {
        return Keys.hmacShaKeyFor(secretKey.getBytes());
    }
}
```

5.2 AuthenticationService

Creamos el servicio que nos va a proporcionar la logica de inicio de sesion:

```
@Service
public class AuthenticationService {

    @Autowired
    private ProfesorRepository profesorRepository;
    @Autowired
    private PasswordEncoder passwordEncoder;
    @Autowired
```

```
    private AuthenticationManager authenticationManager;

    public ProfesorDTO login(LoginUsuarioDTO input) {
        ProfesorDTO profesor =
this.profesorRepository.findByEmail(input.getEmail())
                .map(ProfesorDTO::convertToDTO)
                .orElseThrow(() -> new IllegalArgumentException("Email no
encontrado: " + input.getEmail()));

        if (!passwordEncoder.matches(input.getPassword(),
profesor.getContrasenya())) {
            throw new IllegalArgumentException("Contraseña incorrecta para
el email: " + input.getEmail());
        }

        authenticationManager
                .authenticate(new
UsernamePasswordAuthenticationToken(input.getEmail(),
input.getPassword()));

        return profesor;
    }
}
```

6. Controlador de inicio de sesion

Creamos el rest controller que nos proporcionara el inicio de sesion:

```
@RequestMapping("/api/auth")
@RestController
public class AuthenticationRestController {

    @Autowired
    private AuthenticationService authenticationService;
    @Autowired
    private JwtService jwtService;

    @PostMapping("/login")
    public ResponseEntity<LoginResponseDTO> login(@RequestBody
LoginUsuarioDTO loginUsuarioDTO) {
        ProfesorDTO profesor =
authenticationService.login(loginUsuarioDTO);

        String jwtToken = jwtService.generateToken(profesor);
        LoginResponseDTO loginResponseDTO = new LoginResponseDTO(jwtToken,
jwtService.getExpirationTime());

        return ResponseEntity.ok(loginResponseDTO);
```

```
    }

    @ExceptionHandler(MethodArgumentNotValidException.class)
    public ResponseEntity<Map<String, String>>
handleValidationExceptions(MethodArgumentNotValidException e) {
        Map<String, String> errors = new HashMap<>();
        e.getBindingResult()
                .getAllErrors()
                .forEach(err -> {
                    String name = err.getObjectName();
                    String msg = err.getDefaultMessage();
                    errors.put(name, msg);
                });
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(errors);
    }
}
```

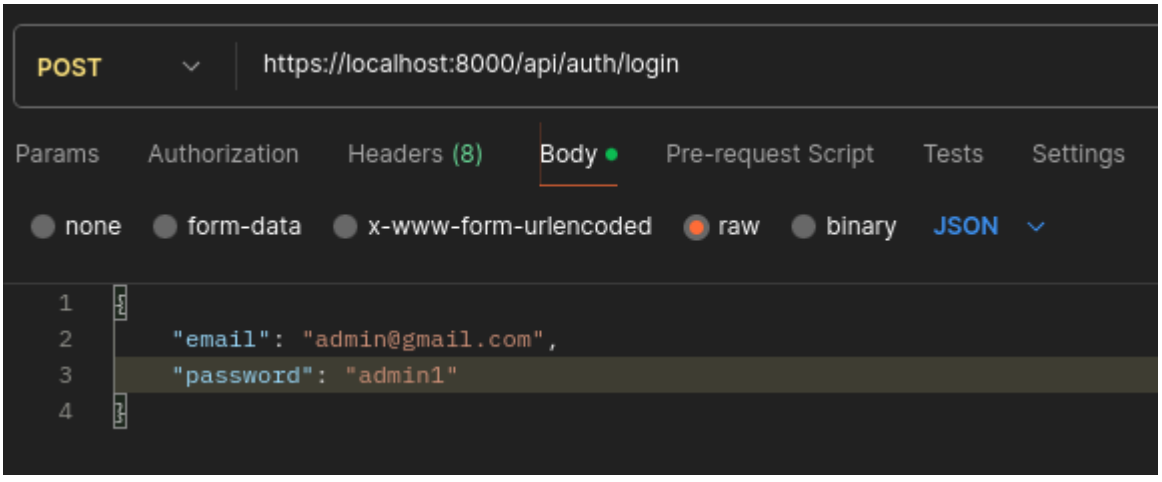
7. Filtro de autenticacion por rol

Mi rol de admin se llama en la base de datos direccion así que añado esta propiedad a los métodos de admin o a los controladores de admin

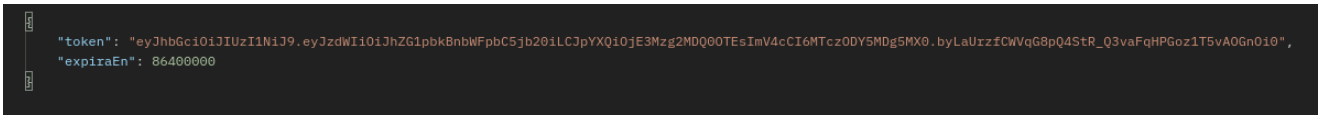
```
@PreAuthorize("hasRole('DIRECCION')")
```

8. Probar el inicio de sesión

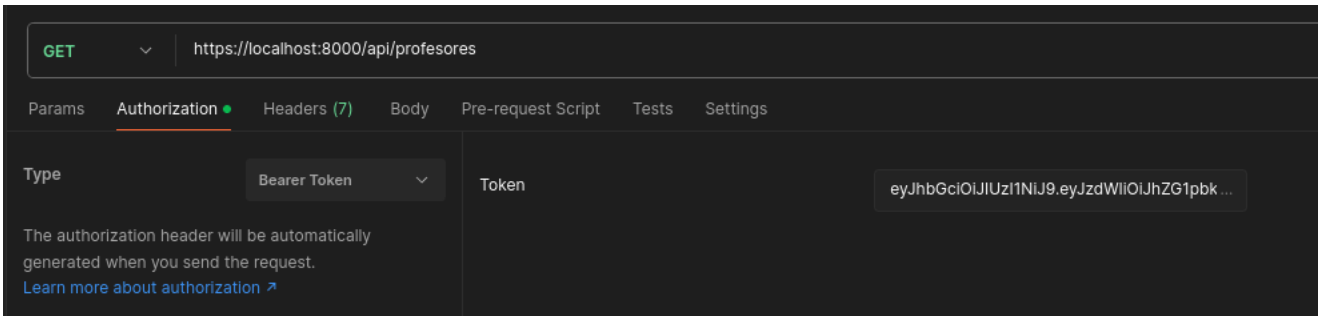
Primero teneis que tener todas las contraseñas de la base de datos encriptadas con bcrypt, despues de eso, la request seria algo asi (si no habeis cambiado nada):



Y la respuesta lo siguiente:



Ahora para hacer una request autenticada ponemos el token recibido en esta opción:



Y desde angular para crear el header necesario para las request con el HttpClient haced algun metodo al logear guarde el token y luego otro que devuelva algo así:

```
return new HttpHeaders({
    Authorization: "Bearer " + token,
});
```

También deberéis comprobar si el token a caducado y volver a pedir que el usuario inicie sesión si esta caducado