

#### 4. Modificar el tiempo de expiración del token y observar el efecto.

1. Usamos RsaKeyProperties con esto ya podemos cambiar el tiempo de expiración de manera dinámica.

```
import java.time.Instant;
import java.util.Map;

@RestController
@RequestMapping("/auth")
public class AuthController {

    private final JwtEncoder encoder;
    private final InMemoryUserService userService;
    private final RsaKeyProperties props;
```

```
@PostMapping("/login")
public ResponseEntity<?> login(@RequestBody LoginRequest req) {
    if (!userService.isValid(req.username(), req.password())) {
        return ResponseEntity.status(status: 401).body(Map.of(k1: "error", v1: "invalid_credentials"));
    }

    Instant now = Instant.now();
    long ttl = props.tokenTtlSeconds() != null ? props.tokenTtlSeconds() : 3600;
    Instant exp = now.plusSeconds(ttl);

    String scope = "blueprints.read blueprints.write";

    JwtClaimsSet claims = JwtClaimsSet.builder()
        .issuer(props.issuer())
        .issuedAt(now)
        .expiresAt(exp)
        .subject(req.username())
        .claim("scope", scope)
        .build();

    JwsHeader jws = JwsHeader.with(() -> "RS256").build();
    String token = this.encoder.encode(JwtEncoderParameters.from(jws, claims)).getTokenValue();

    return ResponseEntity.ok(new TokenResponse(token, token_type: "Bearer", ttl));
}
```

2. Luego en *src/main/resources/application.yml* lo ponemos en 120 segundos para que el efecto sea visible.

```

server:
|   port: 8080

spring:
|   security:
|     oauth2:
|       resourceserver:
|         jwt:
|           jwk-set-uri: https://auth.example.com/.well-known/jwks.json
main:
|   allow-bean-definition-overriding: true

blueprints:
|   security:
|     issuer: "https://decsis-eci/blueprints"
|     token-ttl-seconds: 120
|   
```

- Luego volvemos a ejecutar la aplicación y nos dirigimos al navegador a <http://localhost:8080/swagger-ui/index.html#/auth-controller/login> y hacemos un login para obtener el token.

<http://localhost:8080/auth/login>

Server response

Code	Details						
200	<p>Response body</p> <pre>{   "access_token": "eyJhbGciOiJSUzI1NiJ9.eyJpc3MiIiJodHRwczovL2R1Y3NpcyI1Y2kvYmx1ZXByah50cyIsInMTYiI0InN0dR1bmQiLC1leHAiOjE3NzE4NzAwMjksImIhdCIGMTc3Mtg3MjkuOSwiC2NvcGU1OjIi4hV1chOpbnRzLnJ1YnQgIw x1Zxyahr50cy53cm10Z39.CUhdkpsG8ey4QmE5hGNhEawgo-ReCzePwM0F0fWq1sGoF6GJCRARX_6Vky6ShZsxo12M11FvGt4CF4JC00dd5no8y51B51FrQmJ0qBx0zzcs5UEdR8eM6vCJz5qIneq_GJ05fiaaB92C3uk15Hg59wViKgP-b2SqtSbOUUat VB9hem5bW0x_J0ZhZAS/t1/xommtQYG8IP365z_3q090P7tVGH0_jTA-wdb99kuod04h67J131fcgo2ShxIHSYgZ-Sw/1qaFLYX18pR0YJNx_wkphonLjQK1r7akdhj163gevdWlW0Uj/06tm2PM-0pCL8t4HuV8wzg",   "token_type": "Bearer",   "expires_in": 120 }</pre> <p>Response headers</p> <pre> cache-control: no-cache,no-store,max-age=0,must-revalidate connection: keep-alive content-type: application/json date: Mon, 07 Feb 2026 18:55:09 GMT expires: 0 keep-alive: timeout=60 pragma: no-cache transfer-encoding: chunked vary: Accept,Access-Control-Request-Method,Access-Control-Request-Headers x-content-type-options: nosniff x-frame-options: DENY x-xss-protection: 0 </pre> <p>Responses</p> <table border="1"> <thead> <tr> <th>Code</th> <th>Description</th> <th>Links</th> </tr> </thead> <tbody> <tr> <td>200</td> <td>OK</td> <td>No links</td> </tr> </tbody> </table> <p>Media type</p>	Code	Description	Links	200	OK	No links
Code	Description	Links					
200	OK	No links					

- Luego de obtenerlo le damos a Authorize y allí ponemos el token.

#### 4. Probamos obteniendo el token rápido con GET <http://localhost:8080/api/blueprints>

- En jwt.io nos muestra exp: 1771873029 que representa el timestamp de expiración del token

- #### 5. Luego esperamos que pase los 120 segundos y efectivamente nos sale el HTTP response 401 Unauthorized

```

Curl
curl -X 'GET' \
  'http://localhost:8080/api/blueprints' \
  -H 'accept: */*' \
  -H 'Authorization: Bearer eyJhbGciOiJSUzI1NiJ9.eyJpc3MiOiJodHRwczovL2RlY3Mpcy1yZk9mZXByaW50cyIsInN1YiI6In00dRlbmQiLCJleHAiOjE3NzE4NzPhMjksInIhdC16HrC3Mtg3MjkuOSwic2hvcGUiOiJ1bWVlc3pbeRzn21YQgYmx1ZD'
Request URL
http://localhost:8080/api/blueprints
Server response
Code Details
401 Error: response status is 401
  undocumented
Response headers
cache-control: no-cache,no-store,max-age=0,must-revalidate
connection: keep-alive
content-length: 0
date: Mon, 23 Feb 2026 18:58:38 GMT
expires: 0
keep-alive: timeout=60
proxy-connection: keep-alive
vary: Origin;Access-Control-Request-Method,Access-Control-Request-Headers
www-authenticate: Bearer error="invalid_token",error_description="An error occurred while attempting to decode the Jwt: Jwt expired at 2026-02-23T18:57:09Z",error_uri="https://tools.ietf.org/html/rfc6750#section-3.1"
x-content-type-options: nosniff
x-frame-options: DENY
x-xss-protection: 0
Responses
Code Description
Links

```

## Conclusión:

```

@Bean
public JwtDecoder jwtDecoder(JwtKeyProvider keyProvider) {
    return NimbusJwtDecoder.withPublicKey((java.security.interfaces.RSAPublicKey) keyProvider.publicKey()).build();
}

@Bean
public JwtEncoder jwtEncoder(JwtKeyProvider keyProvider) {
    RSAKey rsaKey = new RSAKey.Builder((java.security.interfaces.RSAPublicKey) keyProvider.publicKey())
        .privateKey(keyProvider.privateKey())
        .build();
    return new NimbusJwtEncoder(new ImmutableJWKSet<SecurityContext>(new JWKSet(rsaKey)));
}

```

Cuando llega una petición protegida, por ejemplo, GET /api/blueprints, Spring Security extrae automáticamente el token del encabezado Authorization mediante el filtro BearerTokenAuthenticationFilter.

Luego, el framework invoca jwtDecoder.decode(token), el cual valida la firma digital utilizando la llave pública RSA configurada en SecurityConfig.

El token fue previamente firmado con la llave privada mediante el JwtEncoder, garantizando que no haya sido alterado.

Además de validar la firma, Spring Security verifica automáticamente la claim exp usando un validador interno JwtTimestampValidator.

Si el tiempo actual es mayor que el valor de exp, el framework rechaza la petición retornando 401 Unauthorized, bloqueando el acceso al recurso protegido.

## 5. Documentar en Swagger los endpoints de autenticación y de negocio.

- Código de documentación del endpoint de login

```
public record LoginRequest(String username, String password) {}
public record TokenResponse(String access_token, String token_type, long expires_in) {}

@Operation(
    summary = "Iniciar sesión",
    description = "Autentica al usuario y retorna un token JWT Bearer firmado con RS256."
)
@ApiResponses({
    @ApiResponse(
        responseCode = "200",
        description = "Token emitido correctamente",
        content = @Content(
            mediaType = "application/json",
            schema = @Schema(implementation = TokenResponse.class),
            examples = @ExampleObject(value = """
                {
                    "access_token": "eyJhbGciOiJSUzI1NiJ9...",
                    "token_type": "Bearer",
                    "expires_in": 120
                }
            """)
        ),
        @ApiResponse(
            responseCode = "401",
            description = "Credenciales inválidas",
            content = @Content(
                mediaType = "application/json",
                examples = @ExampleObject(value = "{ \"error\": \"invalid_credentials\" }")
            )
        )
    )
})
@PostMapping("/login")
public ResponseEntity<?> login(@RequestBody LoginRequest req) {
    if (!userService.isValid(req.username(), req.password())) {
        return ResponseEntity.status(status: 401).body(Map.of(k1: "error", v1: "invalid_credentials"));
    }
}
```

- Añadimos un Tag en BlueprintController

```
@RestController
@RequestMapping("/api/blueprints")
@Tag(name = "Blueprints", description = "CRUD de blueprints, requiere token Bearer")
public class BlueprintController {

    private final BlueprintsServices services;

    public BlueprintController(BlueprintsServices services) {
        this.services = services;
    }
}
```

### 1. Endpoints de autenticación documentados

The screenshot shows the Swagger UI interface. At the top, there is a red box highlighting the 'Autenticación' section, which contains the '/auth/login' endpoint. Below this, there is a large red box highlighting the 'blueprint-controller' section, which lists several endpoints for managing blueprints:

- PUT /api/blueprints/{author}/{bpname}/points: Agregar un punto a un blueprint
- GET /api/blueprints: Obtener todos los blueprints
- POST /api/blueprints: Crear un nuevo blueprint
- GET /api/blueprints/{author}: Obtener blueprints por autor
- GET /api/blueprints/{author}/{bpname}: Obtener blueprint por autor y nombre

## - Responses

Responses		Links
Code	Description	
200	Token emitido correctamente	No links
401	Credenciales inválidas	No links

