

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(401).body(Map.of(k1: "error", v1: "invalid_cred"));
    }

    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

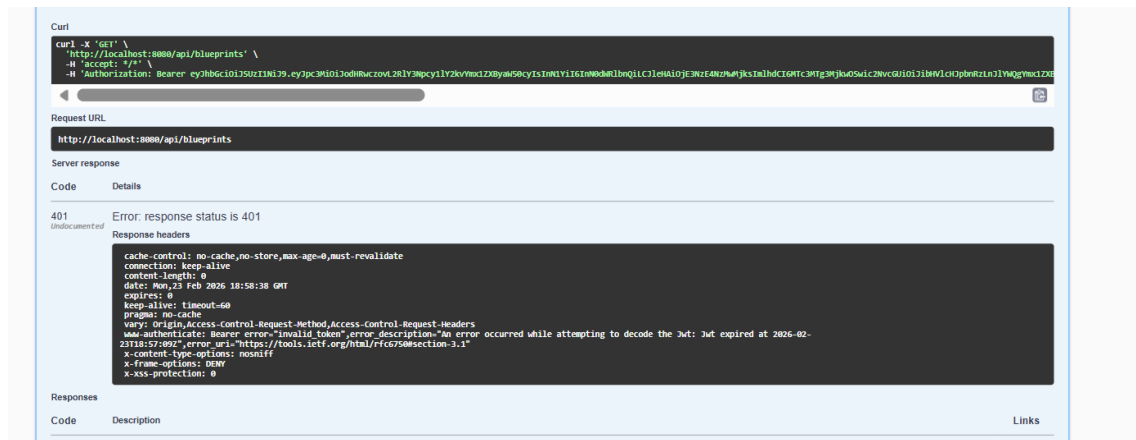
spring:
  security:
    oauth2:
      resourceserver:
        jwt:
          jwk-set-uri: https://auth.example.com/.well-known/jwks.json
  main:
    allow-bean-definition-overriding: true

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

3. 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:8800/auth/login | | |
| | Server response | | |
| | Code | Details | |
| 200 | | <div><div>Response body</div><div>{ "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXZWxkeS0yLWVudCkge3QwMjYzMmcyllY2kvbmclZXByYW5lS0YyLSlnWTlyTGI6ImNkbWJlbmcqClElaHoJEHnZrEWNubWtjcjslnIdldGVGTfCMgY2hlbnDScic2NecSUoiOjBibWltctDpbndmZmlnWWQUMmF1eXBkaWNoODY5cmIozTS9y-OUhaqdSdbydQMJSERONWhEAVqp-RhczePvMJ3fOfmq1SGoF6GJCRAEX_6VKyGSUZSXowei2WHIfvgKAcF4Cc0bdSnobYS1BSlfqmQrbaxdzrczSUEdi8BeWvciz2sq4neq_6JLP5faal89Z3uk1Shg5NmVIKgp-b2SqztSMouLRVBvhMuSUMAZ_ZUhzhAS7ti7GmmutIQVG8IP3sz_z3q89PC7LVGH_O_jTA-kdbG9kuodDe4MG7JT3ifgo2sbCHsYgz-SwlQuFLyxISprUYNX_wkhphGi_JQQrlrYAKBatij16GevtMLVRUj7caUGtmZPFOPCL8E4hv8eid", "token_type": "Bearer", "expires_in": 120 }</div><div><div>Download</div></div></div> | |
| | Response headers | | |
| | <div>cache-control: no-cache,no-store,max-age=0,must-revalidate connection: keep-alive content-type: application/json date: Mon,23 Feb 2026 18:55:09 GMT expires: 0 keep-alive: timeout=60 pragma: no-cache transfer-encoding: chunked vary: Origin,Access-Control-Request-Method,Access-Control-Request-Headers x-content-type-options: nosniff x-frame-options: DENY x-xss-protection: 0</div> | | |
| | Responses | | |
| | Code | Description | Links |
| 200 | OK | Middle | No links |

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



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": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXLTJ5bnVzIiwiaWF0IjoxNjUzOTUzOTUzLCJ0eXciOiJkaXIiLCJyb2N0IjoiYm9keiJ9",
                  "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(401).body(Map.of("error", "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 displays the Swagger UI interface. At the top, the 'Autenticación' section is highlighted with a red border. It contains a single endpoint: a POST request to /auth/login with the summary 'Iniciar sesión'. Below this, the 'blueprint-controller' section is visible, listing several endpoints: a PUT request to /api/blueprints/{author}/{bpname}/points (summary: 'Agregar un punto a un blueprint'), a GET request to /api/blueprints (summary: 'Obtener todos los blueprints'), a POST request to /api/blueprints (summary: 'Crear un nuevo blueprint'), a GET request to /api/blueprints/{author} (summary: 'Obtener blueprints por autor'), and a GET request to /api/blueprints/{author}/{bpname} (summary: 'Obtener blueprint por autor y nombre'). Each endpoint entry includes its HTTP method, URL, summary, and a lock icon indicating security requirements.

- Responses

| Responses | | |
|-----------|---|----------|
| Code | Description | Links |
| 200 | <div>Token emitido correctamente</div> <div>Media type application/json</div> <div>Controls Accept header.</div> <div>Example Value Schema</div> <div><pre>{ "access_token": "eyJhbGciOiJSUzI1NiJ9...", "token_type": "Bearer", "expires_in": 120}</pre></div> | No links |
| 401 | <div>Credenciales inválidas</div> <div>Media type application/json</div> <div>Example Value </div> <div><pre>{ "error": "invalid_credentials"}</pre></div> | No links |

