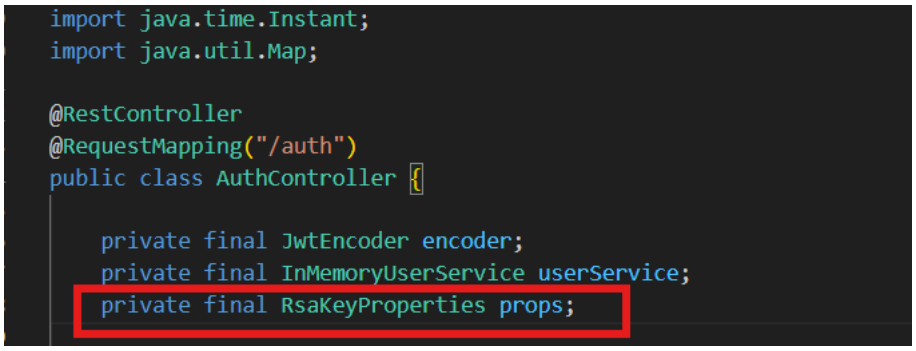
4. Modificar el tiempo de expiración del token y observar el efecto.

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



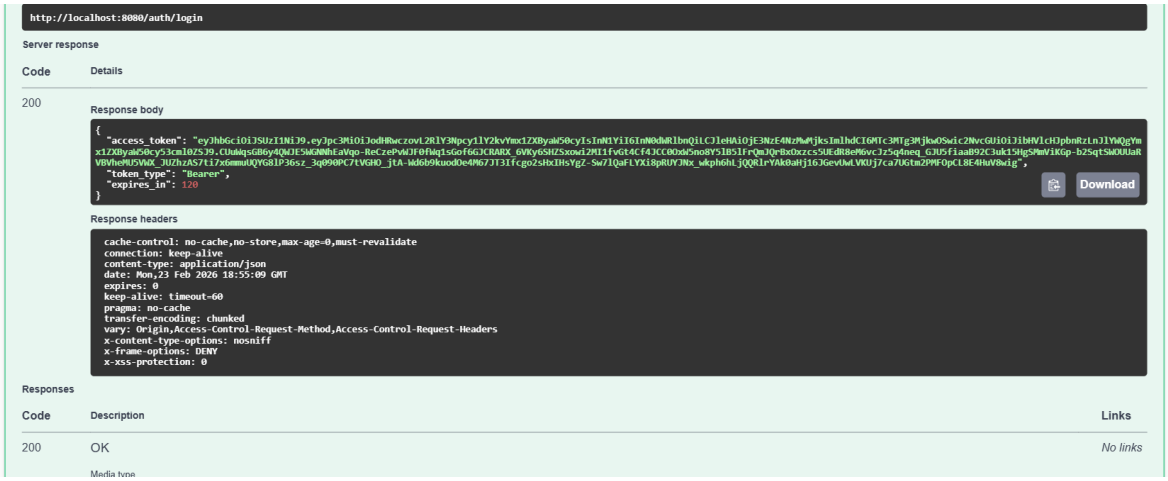
```java
import java.time.Instant;
import java.util.Map;

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

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

```java
@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_cre
    }

    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)).getTokenVal

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

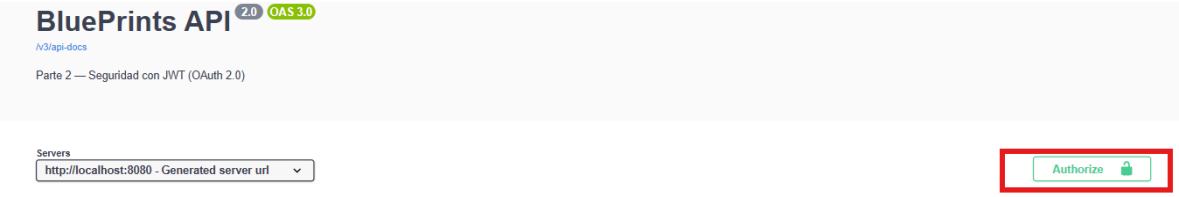
2. Luego en $src/main/resources/application.yml$ lo ponemos en 120 segundos para que el efecto sea visible.

```yaml
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
```

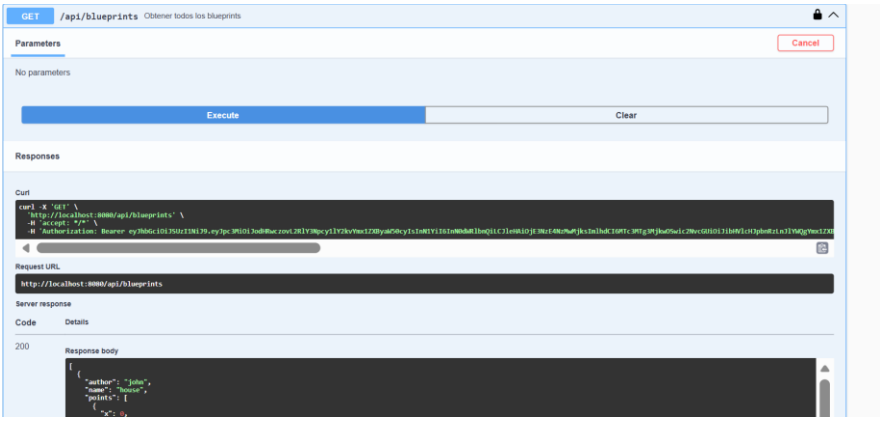
3. Luego volvemos a ejecutar la aplicación y nos dirigimos al navegador a http://localhost:8080/swagger-ui/index.html#/auth-controller/login y hacemos un login para obtener el token.



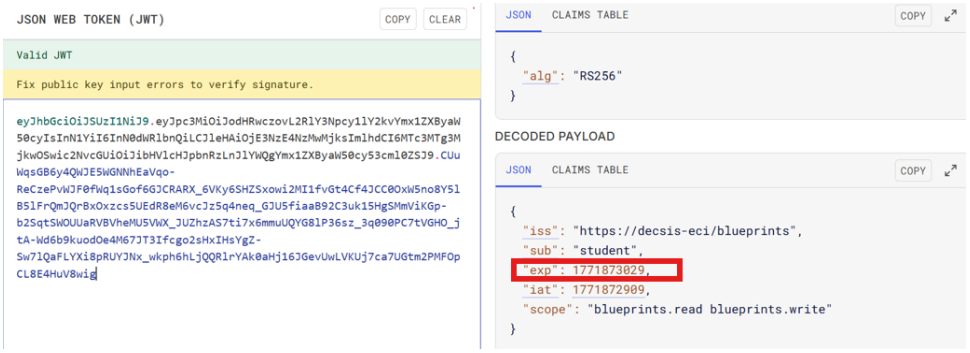
- Luego de obtenerlo le damos a Authorize y allí ponemos el token.



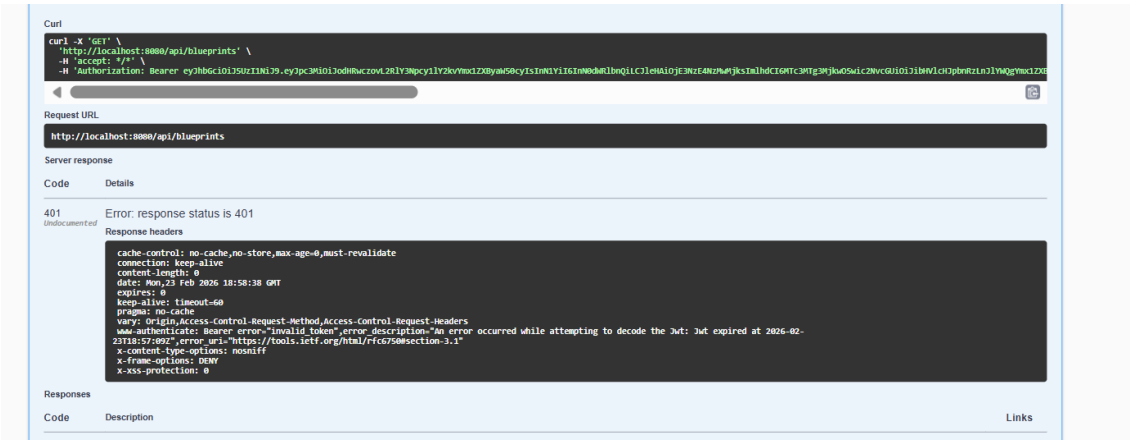
4. Probamos obteniendo el token rápido con GET http://localhost:8080/api/blueprints



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



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



Conclusión:

```java
@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.

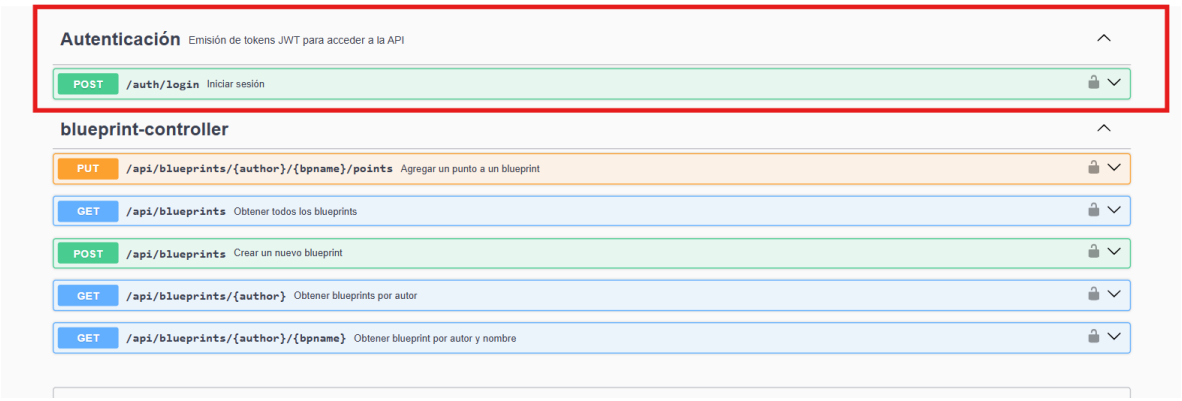
- Codigo de documentación del endpoint de login

```java
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 u Tag en BlueprintController

```java
@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



- Responses


Responses
Code
Description
Links
200
Token emitido correctamente
No links
Media type
application/json
Controls Accept header.
Example Value | Schema
{
"access_token": "eyJhbGciOiJSUzI1Ni39...",
"token_type": "Bearer",
"expires_in": 120
}
401
Credenciales inválidas
No links
Media type
application/json
Example Value |
{
"error": "invalid_credentials"
}