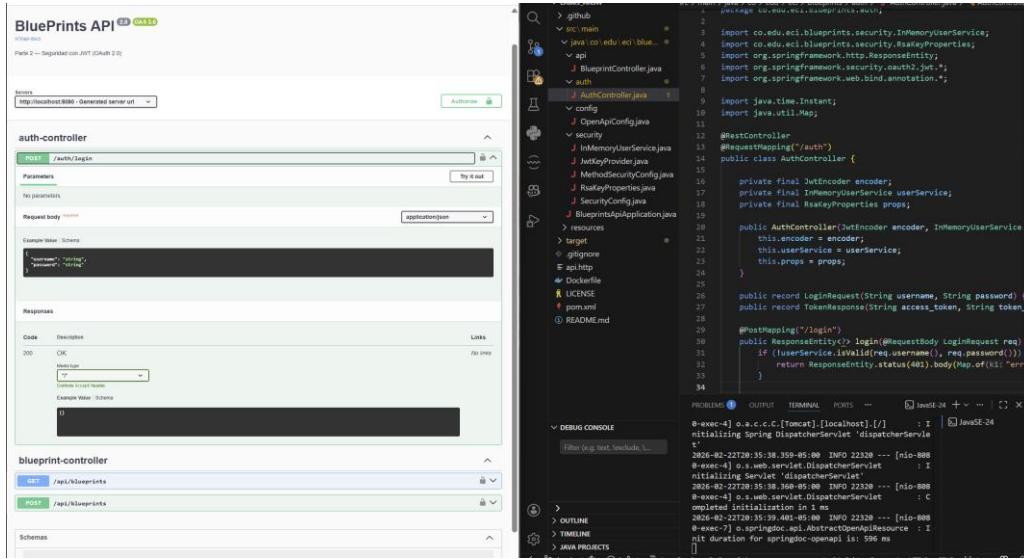


- Verificamos que la aplicación levante en localhost



## 1. Revisar el código de configuración de seguridad (SecurityConfig) e identificar cómo se definen los endpoints públicos y protegidos.

```

src > main > java > co > edu > eci > blueprints > security > SecurityConfig.java
  1 package co.edu.eci.blueprints.security;
  ...
  3 import com.nimbusds.jose.jwk.RSAKey;
  4 import com.nimbusds.jose.jwk.source.ImmutableJWKSet;
  5 import com.nimbusds.jose.jwk.JWKSet;
  6 import com.nimbusds.jose.proc.SecurityContext;
  7 import org.springframework.boot.context.properties.EnableConfigurationProperties;
  8 import org.springframework.context.annotation.Bean;
  9 import org.springframework.context.annotation.Configuration;
 10 import org.springframework.security.config.Customizer;
 11 import org.springframework.security.config.annotation.web.builders.HttpSecurity;
 12 import org.springframework.security.core.userdetails.UserDetailsService;
 13 import org.springframework.security.web.SecurityFilterChain;
 14 import org.springframework.security.crypto.password.PasswordEncoder;
 15 import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
 16 ...
 17 @Configuration
 18 @EnableConfigurationProperties(RsaKeyProperties.class)
 19 public class SecurityConfig {
 20     ...
 21     @Bean
 22     public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
 23         ...
 24         .csrf(csrf -> csrf.disable())
 25         .authorizeHttpRequests(auth -> auth
 26             .requestMatchers("/actuator/health", "/auth/login").permitAll()
 27             .requestMatchers("/v3/api-docs/**", "/swagger-ui/**", "/swagger-ui.html").permitAll()
 28             .requestMatchers("/api/**").hasAnyAuthority("SCOPE_blueprints.read", "SCOPE_blueprints.write")
 29             .anyRequest().authenticated()
 30         )
 31         .oauth2ResourceServer(oauth2 -> oauth2.jwt(customizer.withDefaults()));
 32         return http.build();
 33     }
 34 ...
 35     @Bean
 36     PasswordEncoder passwordEncoder() {
 37         return new BCryptPasswordEncoder();
 38     }
 39 ...
 40     @Bean
 41     public JwtDecoder jwtDecoder(JwtKeyProvider keyProvider) {
 42         return NimbusJwtDecoder.withPublicKey((java.security.interfaces.RSAPublicKey) keyProvider.publicKey()).build();
 43     }
 44 ...
 45     @Bean
 46     public JwtEncoder jwtEncoder(JwtKeyProvider keyProvider) {
 47         RSAKey rsaKey = new RSAKey.Builder((java.security.interfaces.RSAPublicKey) keyProvider.publicKey())
 48             .privateKey(keyProvider.privateKey())
 49             .build();
 50         return new NimbusJwtEncoder(new ImmutableJWKSet<SecurityContext>(new JWKSet(rsaKey)));
 51     }
 52 }
  
```

**SecurityConfig** es la clase de configuración de spring security en donde se define como se esta clase convierte la API en un OAuth2 Resource Server.

El endpoint /auth/login es público y emite un JWT firmado con RS256 usando una llave privada RSA.

Los endpoints /api/\*\* están protegidos y solo se pueden acceder si el token contiene los scopes blueprints.read o blueprints.write.

En cada request, Spring valida automáticamente la firma del token con la llave pública, verifica su expiración y traduce los scopes a autoridades.

## 2. Explorar el flujo de login y analizar las claims del JWT emitido.

**Respuesta:** Para empezar a probar debemos logearnos con los usuarios que están definidos.

```
1 package co.edu.eci.blueprints.security;
2
3 import org.springframework.security.crypto.password.PasswordEncoder;
4 import org.springframework.stereotype.Service;
5 import java.util.Map;
6
7 @Service
8 public class InMemoryUserService {
9     private final Map<String, String> users; // username -> hash
10    private final PasswordEncoder encoder;
11
12    public InMemoryUserService(PasswordEncoder encoder) {
13        this.encoder = encoder;
14        this.users = Map.of(
15            k1: "student", encoder.encode("student123"),
16            k2: "assistant", encoder.encode("assistant123")
17        );
18    }
19
20    public boolean isValid(String username, String rawPassword) {
21        String hash = users.get(username);
22        return hash != null && encoder.matches(rawPassword, hash);
23    }
24 }
```

1) Hacemos la petición de login

2) Una vez nos hallamos “autenticado” verificamos que nos devuelva el jwt eso representa la identidad y permisos del usuario que va a tener dentro de la aplicación, el token tiene 3 partes el HEADER – PLAYLOAD – SIGNATURE.

En el Payload del token se encuentran las claims, se uso la página jwt.io para el payload.

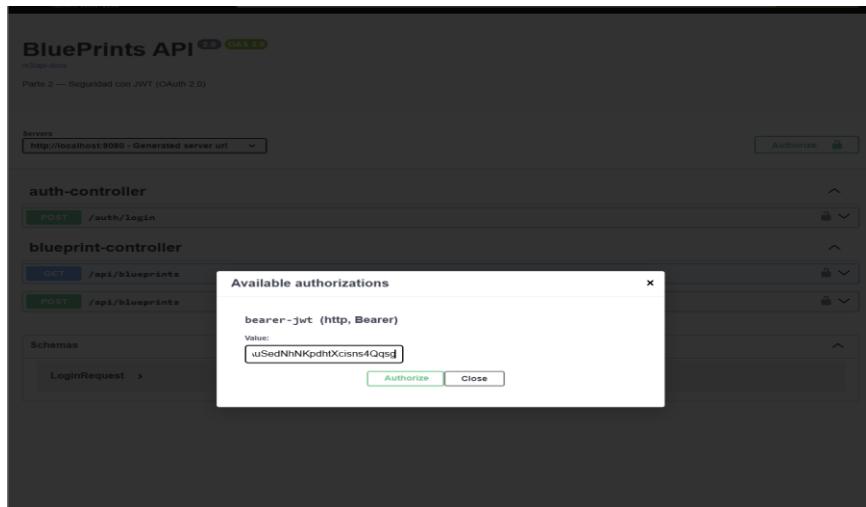
- sub: identifica al usuario autenticado.
  - scope: define los permisos del usuario, como blueprints.read y blueprints.write.
  - iat: indica el momento en que el token fue emitido.
  - exp: define la fecha de expiración del token.

### DECODED PAYLOAD

JSON	CLAIMS TABLE												
<pre>{     "iss": "https://decsis-eci/blueprint s",     "sub": "student",     "exp": 1771814894,     "iat": 1771811294,     "scope": "blueprints.read blueprints. write" }</pre>	<table border="1"><thead><tr><th>Claim</th><th>Value</th></tr></thead><tbody><tr><td>iss</td><td>https://decsis-eci/blueprint</td></tr><tr><td>sub</td><td>student</td></tr><tr><td>exp</td><td>1771814894</td></tr><tr><td>iat</td><td>1771811294</td></tr><tr><td>scope</td><td>blueprints.read blueprints.write</td></tr></tbody></table>	Claim	Value	iss	https://decsis-eci/blueprint	sub	student	exp	1771814894	iat	1771811294	scope	blueprints.read blueprints.write
Claim	Value												
iss	https://decsis-eci/blueprint												
sub	student												
exp	1771814894												
iat	1771811294												
scope	blueprints.read blueprints.write												

**COPY** ↵

- 3) Ahora procedemos a verificar que el token le permita ingresar a ese usuario y navegar en la aplicación como por ejemplo haciendo consultas.



- 4) Se verifica que el token sea valido y permite crear un nuevo blueprint y consultar.

**POST** /api/blueprints

**Parameters**

No parameters

**Request body** required

application/json

```
{  
  "additionalProp1": "hola",  
  "additionalProp2": "chao",  
  "additionalProp3": "como estas"  
}
```

**Execute** **Clear**

**Responses**

Curl

```
curl -X POST '\  
http://localhost:8080/api/blueprints'\  
-H 'accept: */*' \  
-H 'Authorization: Bearer eyJhbGciOiJSUzI1NiJ9.eyJpc3MiOiJodHRwczovL2RlY3Rpcy1yY2kvYmexZXByaih5OcyIsInN1YiI6Imh0dWlbeQilC3leHAIOjE3NzE4MjQ4OTQsImhlhdCI6MTc3Mjg5MTI5Ngic2NwciG1  
-H 'Content-Type: application/json' \  
-d '{  
  "additionalProp1": "hola",  
  "additionalProp2": "chao",  
  "additionalProp3": "como estas"  
}'
```

◀ **Request URL** ▶

<http://localhost:8080/api/blueprints>

**Server response**

**Code** **Details**

200 Response body

```
[  
  {"name": "nuevo",  
   "id": "new"  
}]
```

**Download**

The screenshot shows a REST API documentation interface for a `GET /api/blueprints` endpoint. The "Responses" section displays a successful response (HTTP 200) with a JSON payload containing two blueprint objects:

```
[{"name": "Casa de campo", "id": "b1"}, {"name": "Edificio urbano", "id": "b2"}]
```

- 5) Que pasa si el token no es válido, si no es válido no le permite aun usuario crear o consultar algo dentro de la aplicación.

The screenshot shows a REST API documentation interface for a `GET /api/blueprints` endpoint. The "Responses" section displays an unauthorized response (HTTP 401) with the message "Error: response status is 401".

Así que genera un código http 401 de no autorizado y no deja hacer la consulta, manteniendo la seguridad de la aplicación.

### 3. Extender los scopes (blueprints.read, blueprints.write) para controlar otros endpoints de la API, del laboratorio P1 trabajado.

**Respuesta:** El laboratorio P1 se extendió con la seguridad basada en JWT.

Se definieron los scopes blueprints.read para operaciones de consulta (GET) y blueprints.write para operaciones de modificación (POST, PUT).

Los scopes se implementaron mediante las anotaciones `@PreAuthorize`, las cuales validan las authorities derivadas de las claims del JWT.

```
@Operation(summary = "Obtener todos los blueprints")
@ApiResponses({
    @ApiResponse(responseCode = "200", description = "Lista de blueprints retornada")
})
@GetMapping
@PreAuthorize("hasAuthority('SCOPE_blueprints.read')")
public ResponseEntity<Set<Blueprint>> getAll() {
    return ResponseEntity.ok(services.getAllBlueprints());
}

@Operation(summary = "Obtener blueprints por autor")
@ApiResponses({
    @ApiResponse(responseCode = "200", description = "Blueprints del autor encontrados"),
    @ApiResponse(responseCode = "404", description = "Autor no encontrado")
})
@GetMapping("/{author}")
@PreAuthorize("hasAuthority('SCOPE_blueprints.read')")
public ResponseEntity<?> byAuthor(@PathVariable String author) {
    try {
        return ResponseEntity.ok(services.getBlueprintsByAuthor(author));
    } catch (BlueprintNotFoundException e) {
        return ResponseEntity.status(HttpStatus.NOT_FOUND)
            .body(Map.of("error", e.getMessage()));
    }
}
```

The screenshot shows a REST API testing interface. The URL is `/api/blueprints/{author}` with the description "Obtener blueprints por autor". A parameter `author` is set to `2`. The "Responses" section shows a Curl command and a Request URL (`http://localhost:8080/api/blueprints/2`). The Server response indicates a `401` error with the message "Error: response status is 401".