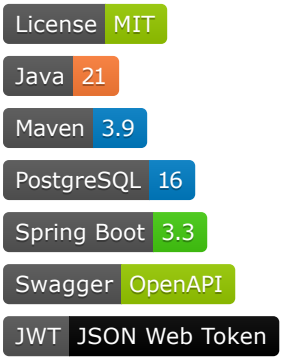
Escuela Colombiana de Ingeniería Julio Garavito

Arquitectura de Software – ARSW



Estudiantes:

- Alexandra Moreno
- Alison Valderrama
- Jeisson Sánchez
- Valentina Gutierrez

Requisitos

- **JDK 21**
- **Maven 3.9+**
- **Git**

Ejecución del proyecto

1. Clonar o descomprimir el proyecto:

   ```
   git clone https://github.com/arsw-la-ley-del-corazon/Lab5-
   BluePrintsApi_with_JWT.git
   ```

   ó si el profesor entrega el `.zip`, descomprimirlo y entrar en la carpeta.

2. Ejecutar con Maven:

   ```
   mvn spring-boot:run
   ```

3. Verificar que la aplicación levante en `http://localhost:8080`.

Actividades propuestas

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

```java
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http
        .csrf(csrf -> csrf.disable())
        .authorizeHttpRequests(auth -> auth
            .requestMatchers("/actuator/health", "/auth/login").permitAll()
            .requestMatchers("/v3/api-docs/**", "/swagger-ui/**", "/swagger-ui.html").permitAll()
            .requestMatchers("/api/**").hasAnyAuthority("SCOPE_blueprints.read", "SCOPE_blueprints.write")
            .anyRequest().authenticated()
        )
        .oauth2ResourceServer(oauth2 -> oauth2.jwt(Customizer.withDefaults()));
    return http.build();
}
```

- **Publico**

"/actuator/health", "/auth/login", /v3/api-docs/**, "/swagger-ui/**", "/swagger-ui.html".

La etiqueta `permitAll()` nos permite declarar excepciones de seguridad.

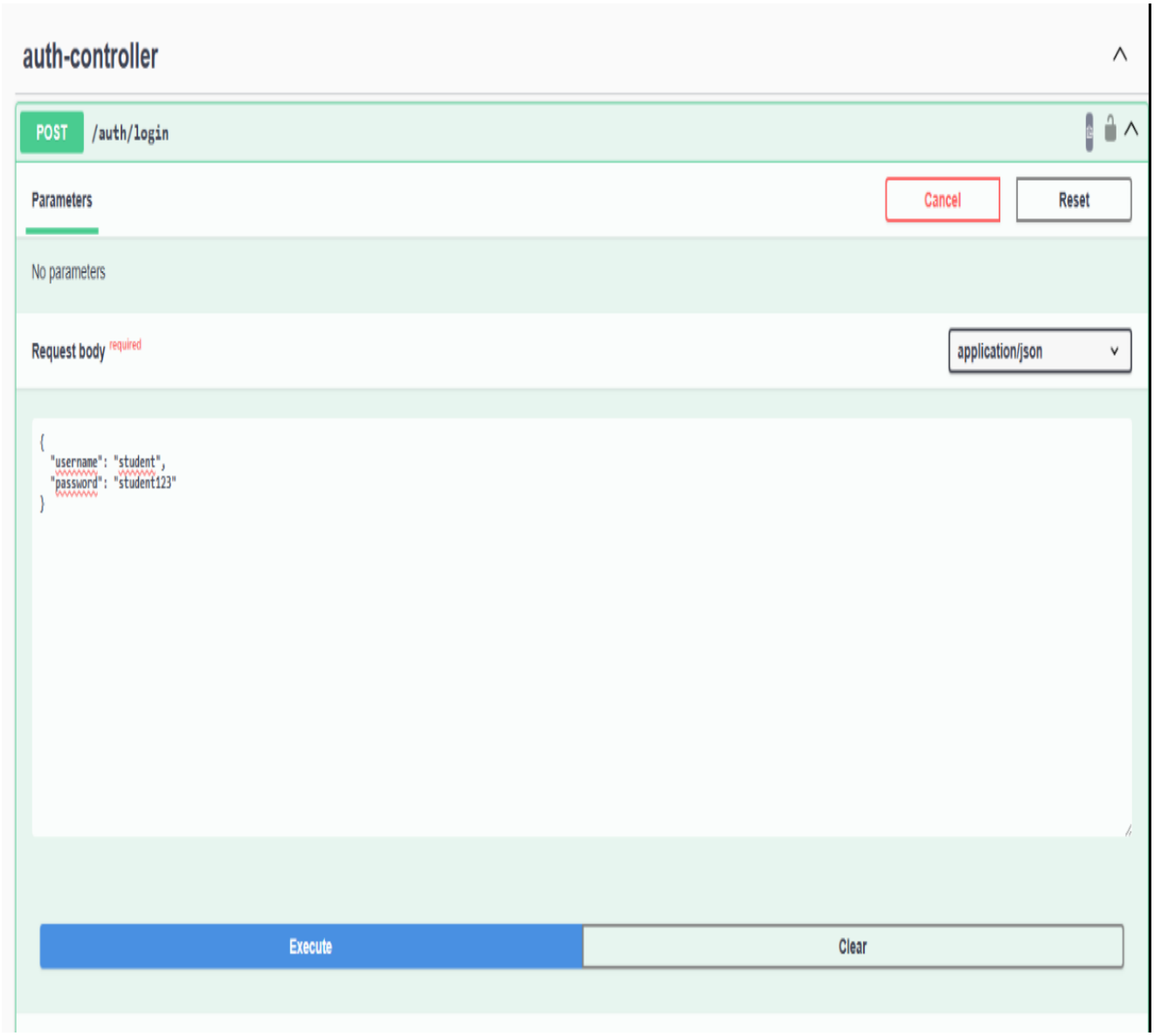
- **Protegido**

/api/**.hasAnyAuthority("SCOPE_blueprints.read", "SCOPE_blueprints.write"): Esto nos indica que esta /api/** esta protegido y requiere permisos, ya que obliga a que el token JWT contenga al menos una de esas authorites.

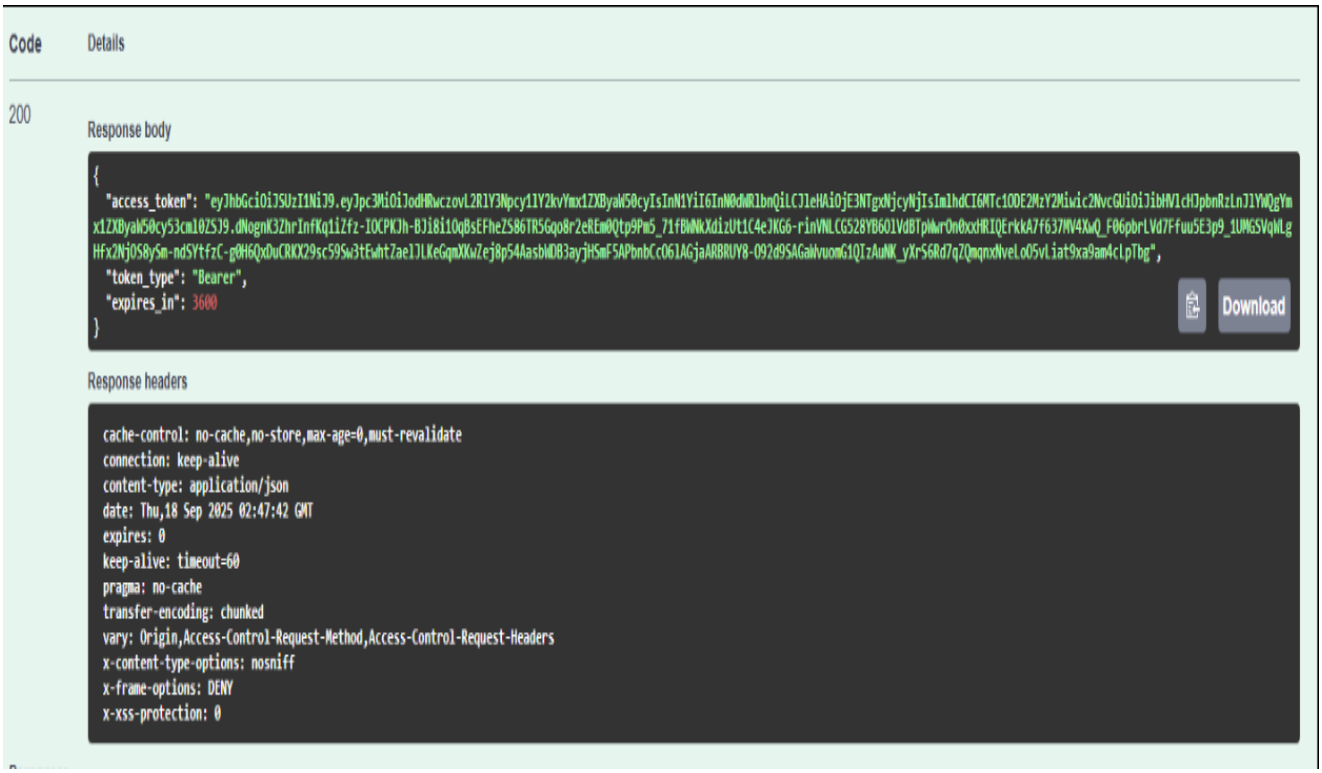
El `anyRequest().authenticated()` nos obliga a que haya un usuario autenticado, es decir, que el request traiga un token JWT valido.

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

En Swagger UI, ejecutamos el endpoint publico POST /auth/login con las credenciales de ejemplo:
Username : student
Password : student123

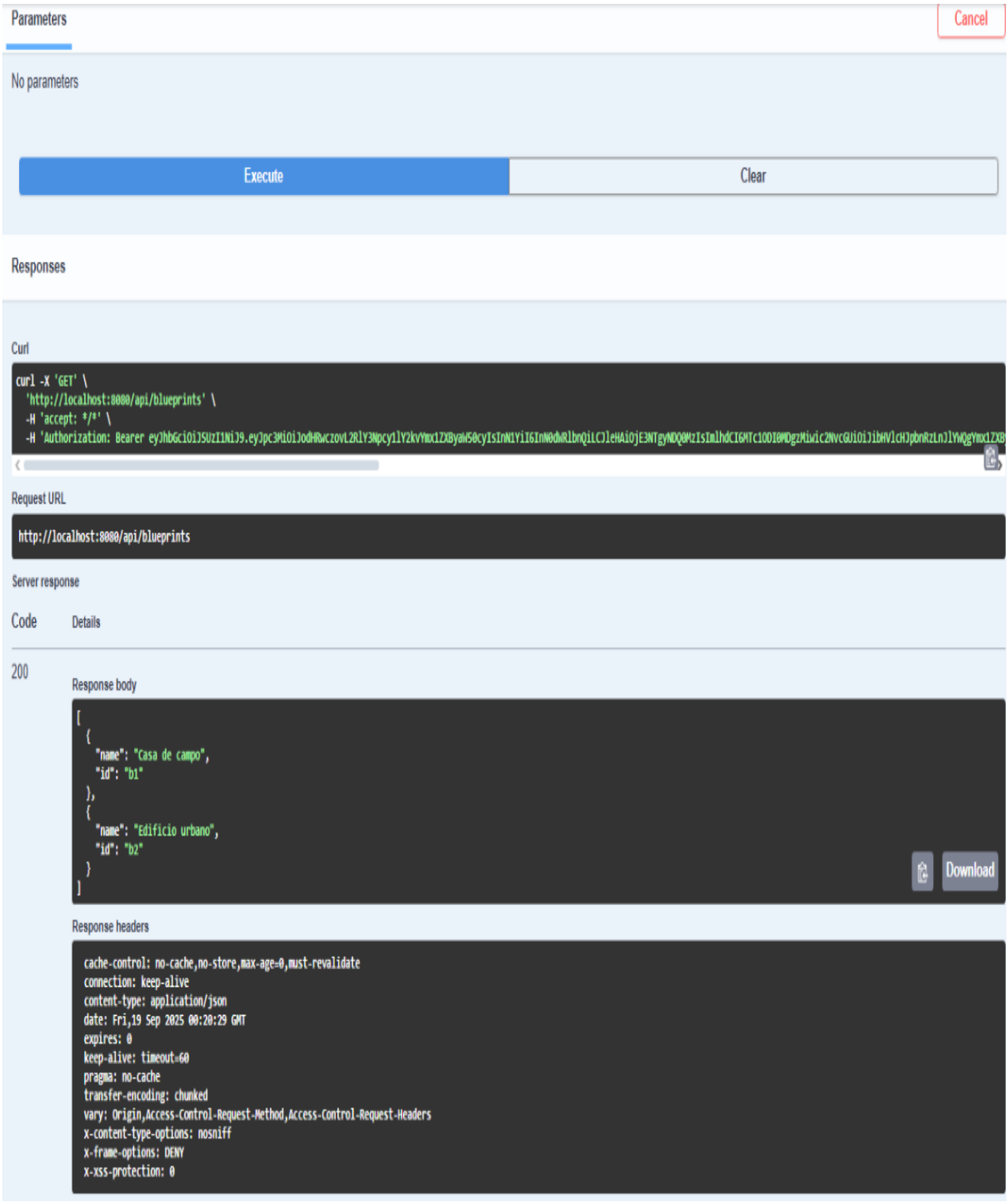


Esto nos genera un objeto JSON con el token JWT:

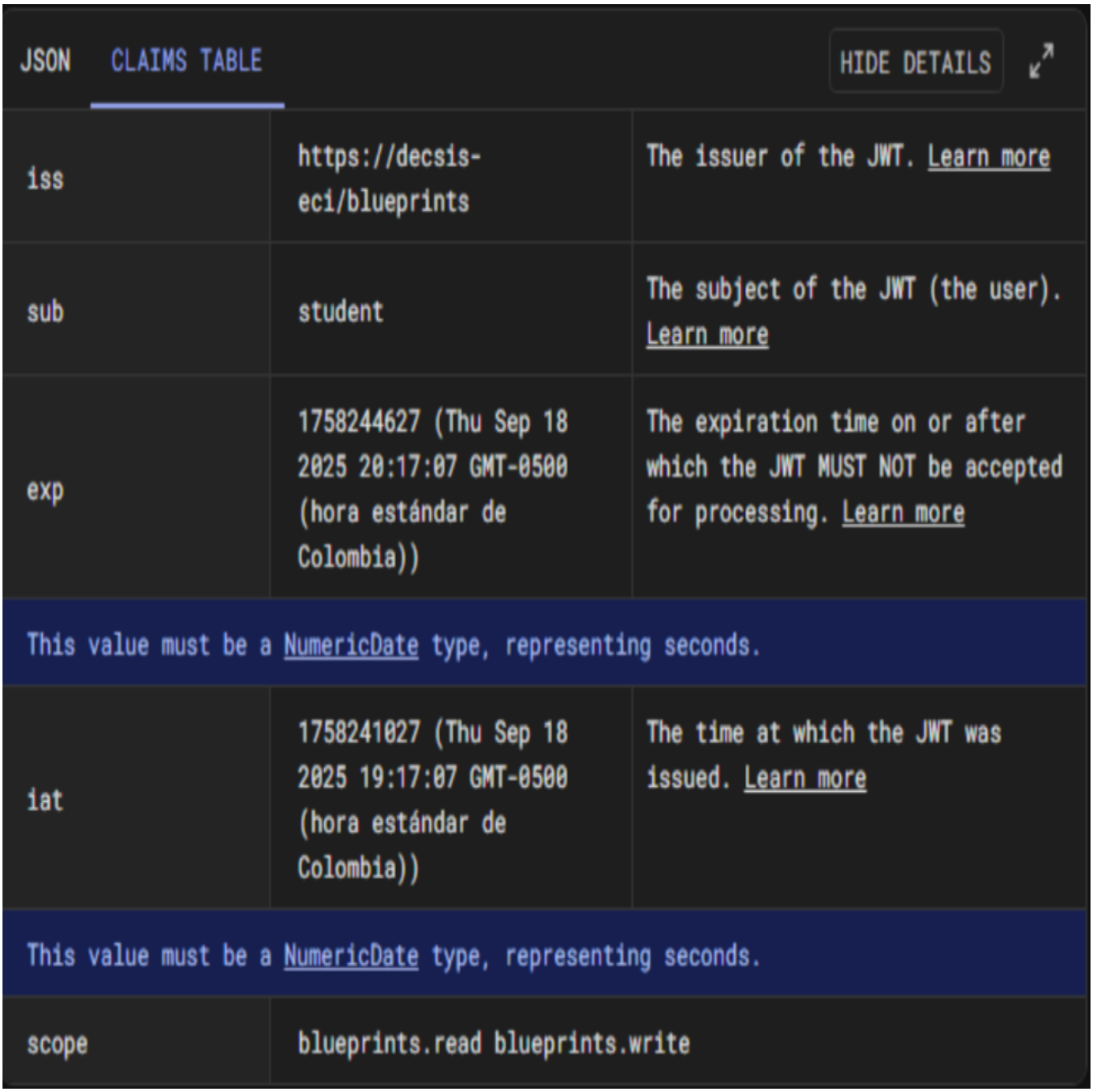


Copiamos el valor de access_token en la opción Authorize de Swagger

Una vez logrado, podemos invocar los endpoints protegidos por la API.



Para analizar los claims del JWT emitido, hacemos uso de jwt.io, el cual descodifica el access_token
Nos otorga las siguientes claims:



- **Iss**: Identifica el emisor del token, en este caso nuestra API https://decsis-eci/blueprints.
- **Sub**: Identifica el usuario autenticado que solicito el token, en este caso student.
- **Exp**: Es el tiempo de expiracion del token, en este caso Thu Sep 18 2025 20:17:07 GMT-0500.
- **Iat**: Se refiere al momento exacto en el que el token fue emitido: Thu Sep 18 2025 19:17:07 GMT-0500.
- **Scope**: Define los permisos del usuario dentro del sistema, en este caso SCOPE_blueprints.read y SCOPE_blueprints.write.

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

Para este punto importamos el proyecto trabajado en el lab P1 y le aplicamos la seguridad a los endpoints, para eso usamos `@PreAuthorize("hasAuthority('SCOPE_xxx')")`, esto nos dice que el token JWT solo funciona si incluye el scope, ya sea read o write dependiendo de la solicitud del endpoint.

```java
// GET /blueprints/{author}
@GetMapping("/{author}")
@PreAuthorize("hasAuthority('SCOPE_blueprints.read')")
@Operation(
    summary = "Listar blueprints por autor",
    description = "Obtiene todos los blueprints creados por un autor específico. Requiere permisos de lectura.",
    security = @SecurityRequirement(name = "bearer-jwt"),
    responses = {
        @io.swagger.v3.oas.annotations.responses.ApiResponse(responseCode = "200", description = "Consulta exitosa",
            content = @Content(schema = @Schema(implementation = ApiResponse.class))),
        @io.swagger.v3.oas.annotations.responses.ApiResponse(responseCode = "404", description = "Autor sin blueprints",
            content = @Content(schema = @Schema(implementation = ApiResponse.class))),
        @io.swagger.v3.oas.annotations.responses.ApiResponse(responseCode = "401", description = "No autorizado - Token JWT requerido",
            content = @Content(schema = @Schema(implementation = ApiResponse.class))),
        @io.swagger.v3.oas.annotations.responses.ApiResponse(responseCode = "403", description = "Prohibido - Permisos insuficientes",
            content = @Content(schema = @Schema(implementation = ApiResponse.class)))
    })
```

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

RTA

En el codigo en el archivo `AuthController.java` se modifico la linea 37 de:

```
    Instant exp = now.plusSeconds(60);
```

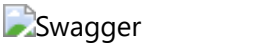
a:

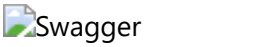
```
    Instant exp = now.plusSeconds(ttl);
```

Para poder comprobar como se comporta el sistema al expirar el token, no se validaba si el token expiraba, asi que se desarollo las debidas clases y metodos que permitieran validar el token en cada peticion, para esto se creo la clase `JwtValidationFilter.java` y se configuro en el archivo `SecurityConfig.java` para que se ejecutara antes de cada peticion.:


Test Jwt Validations

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


Swagger


Swagger

Se puede evidenciar todo los endpoint documentados con autentificación:
Swagger UI

Haz clic en el badge para abrir la documentación.
