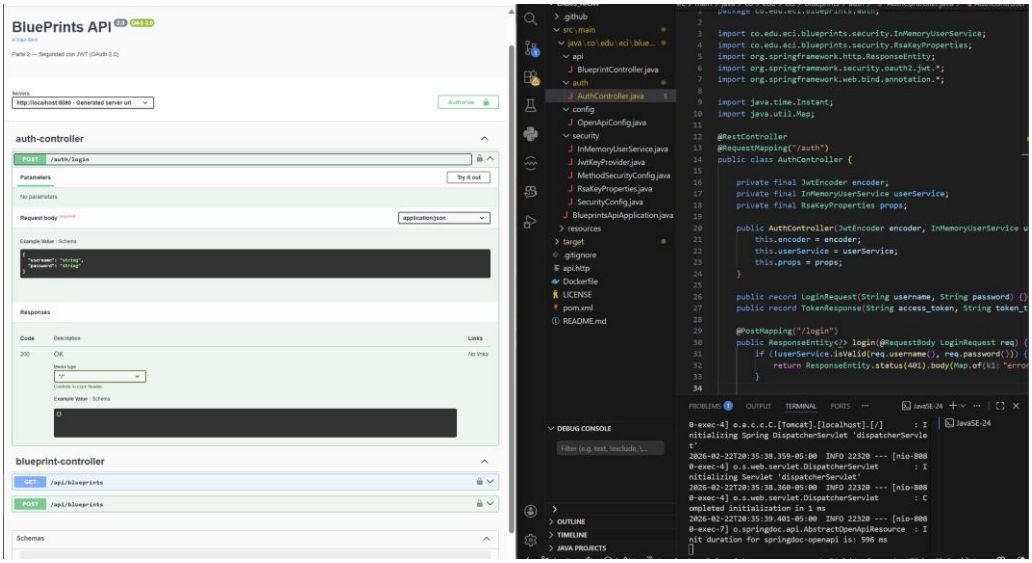
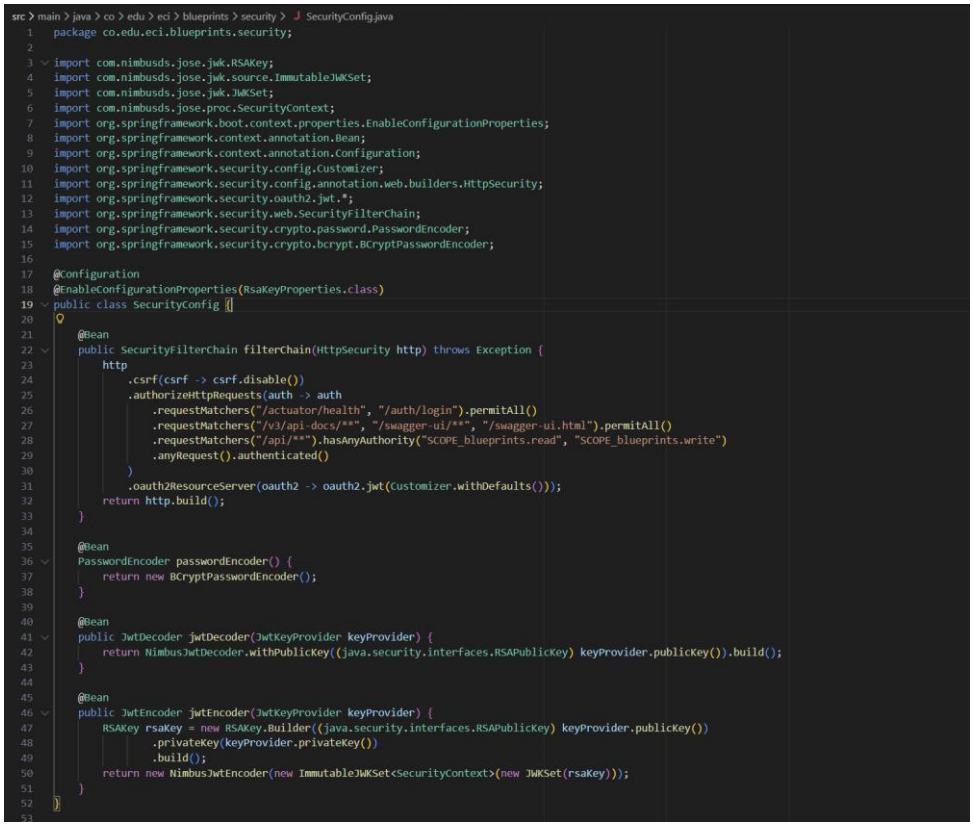
- Verificamos que la aplicación levante en localhost



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



```java
package co.edu.eci.blueprints.security;

import com.nimbusds.jose.jwk.RSAKey;
import com.nimbusds.jose.jwk.source.ImmutableJWKSet;
import com.nimbusds.jose.jwk.JWKSet;
import com.nimbusds.jose.proc.SecurityContext;
import org.springframework.boot.context.properties.EnableConfigurationProperties;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.Customizer;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.oauth2.jwt.*;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

@Configuration
@EnableConfigurationProperties(RsaKeyProperties.class)
public class SecurityConfig {

    @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();
    }

    @Bean
    PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @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)));
    }
}
```

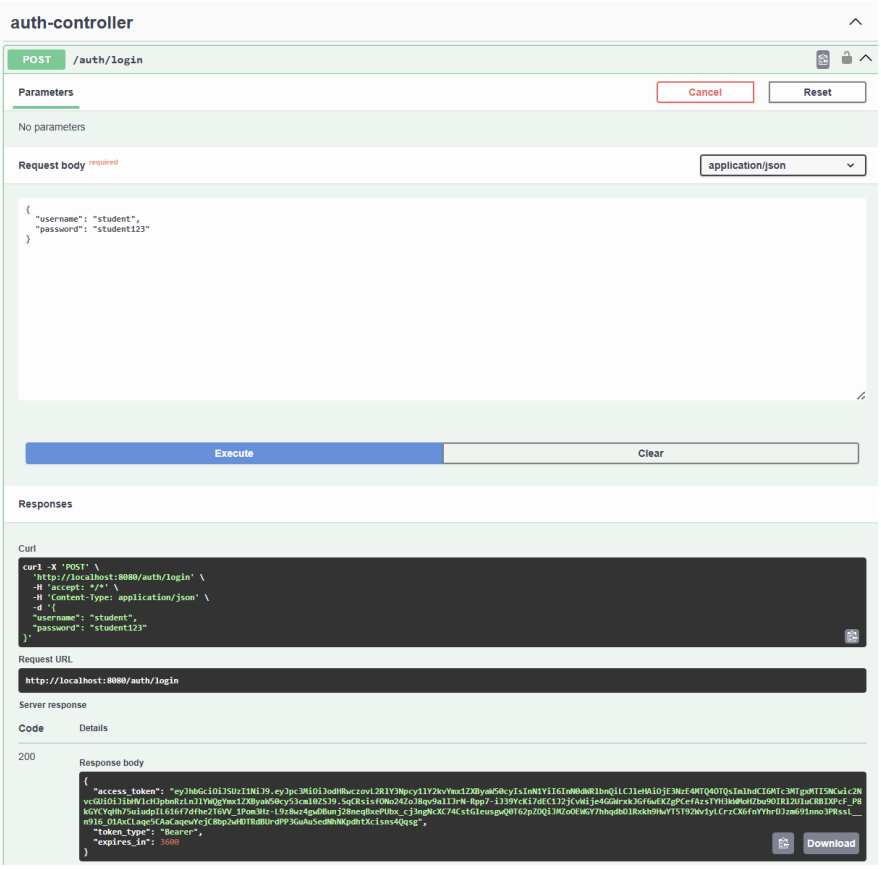
SecurityConfig es la clase de configuración de spring sercurity en donde se define como se esta clase convierte la API en un OAuth2 Resource Server.
El endpoint /auth/login es público y emite un JWT firmado con RS256 usando una llave privada RSA.
Los endpoints /api/** están protegidos y solo se pueden 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.

```java
package co.edu.eci.blueprints.security;

import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import java.util.Map;

@Service
public class InMemoryUserService {
    private final Map<String, String> users; // username -> hash
    private final PasswordEncoder encoder;

    public InMemoryUserService(PasswordEncoder encoder) {
        this.encoder = encoder;
        this.users = Map.of(
            k1: "student", encoder.encode("student123"),
            k2: "assistant", encoder.encode("assistant123")
        );
    }

    public boolean isValid(String username, String rawPassword) {
        String hash = users.get(username);
        return hash != null && encoder.matches(rawPassword, hash);
    }
}
```

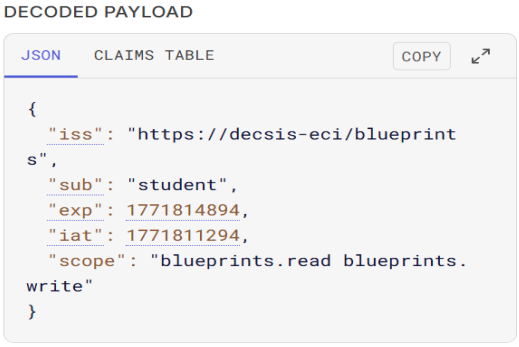
1) Hacemos la petición de login



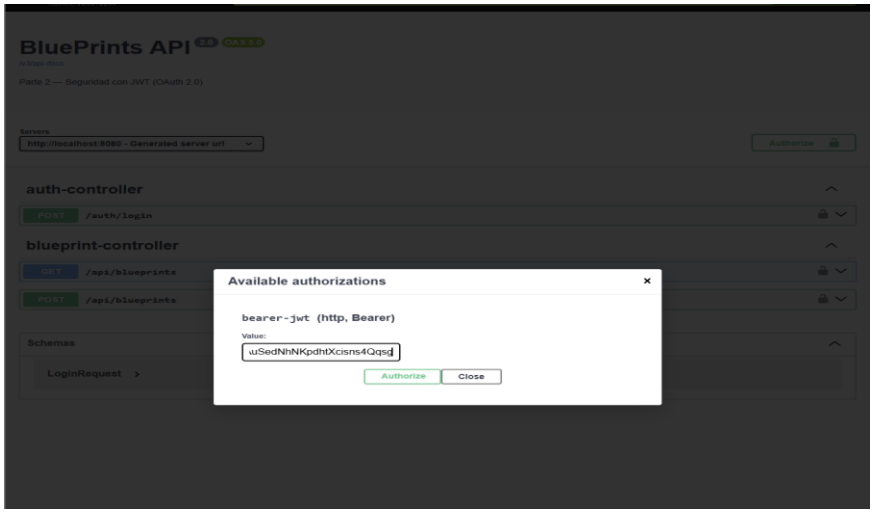
2) Una vez nos hallamos “autenticado” verificamos que nos devuelva el jwt eso representa la identidad y permisos del usuario que va a tener dentro de la aplicación, el token tiene 3 partes el HEADER – PLAYLOAD – SIGNATURE.

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

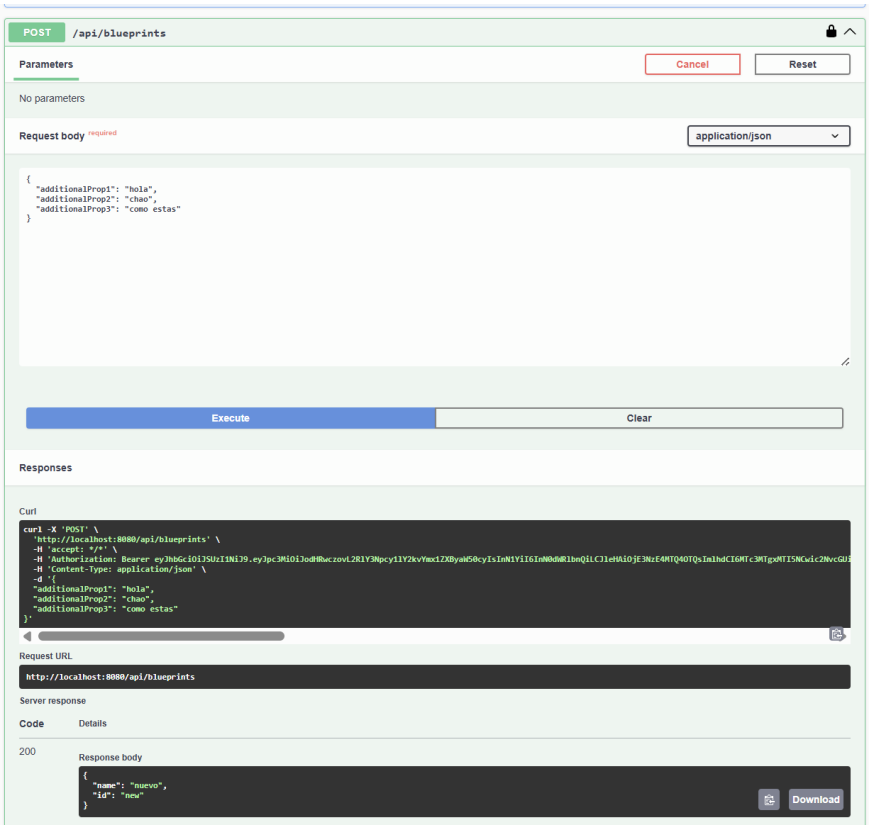
- sub: identifica al usuario autenticado.
- scope: define los permisos del usuario, como blueprints.read y blueprints.write.
- iat: indica el momento en que el token fue emitido.
- exp: define la fecha de expiración del token.

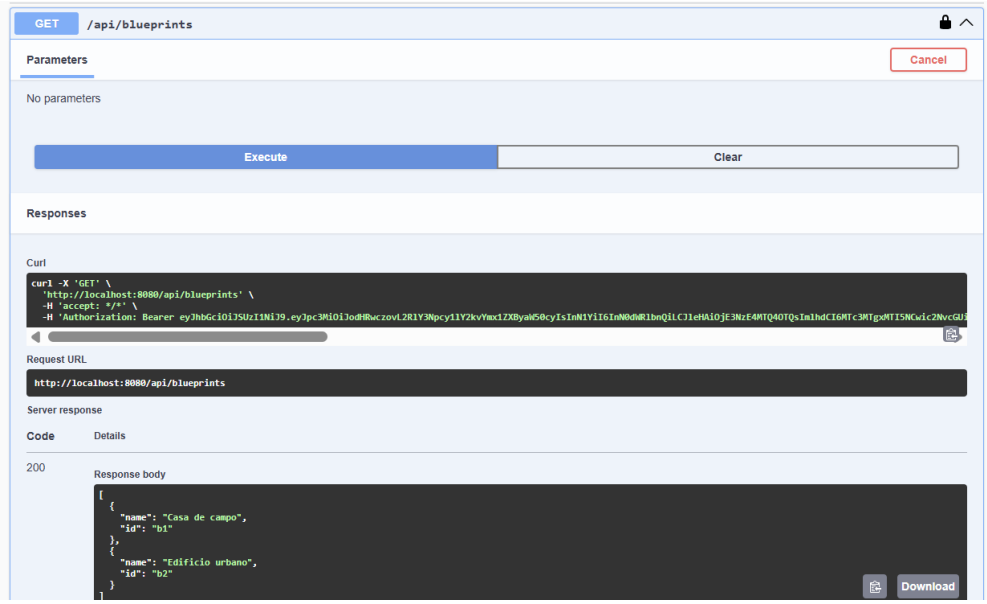


3) Ahora procedemos a verificar que el token le permita ingresar a ese usuario y navegar en la aplicación como por ejemplo haciendo consultas.

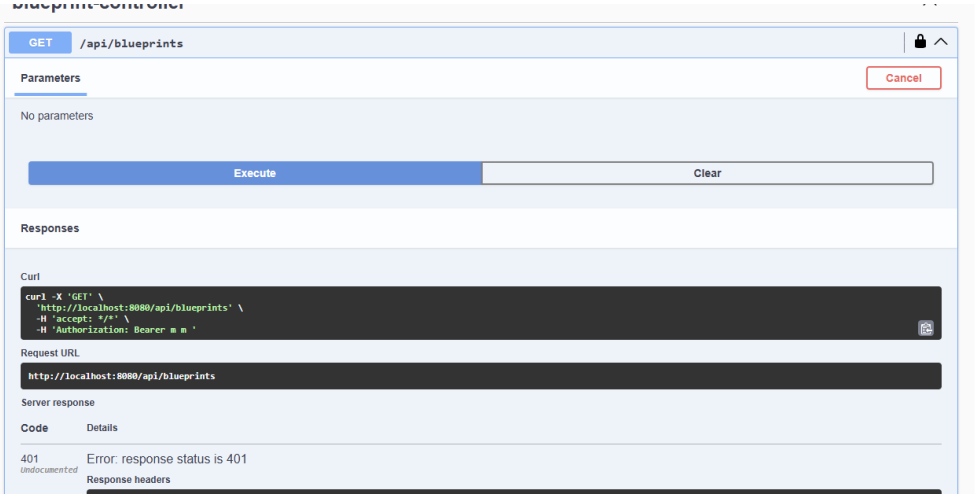


4) Se verifica que el token sea valido y permite crear un nuevo blueprint y consultar.





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.



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.

```java
@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(k1: "error", e.getMessage()));
    }
}
```