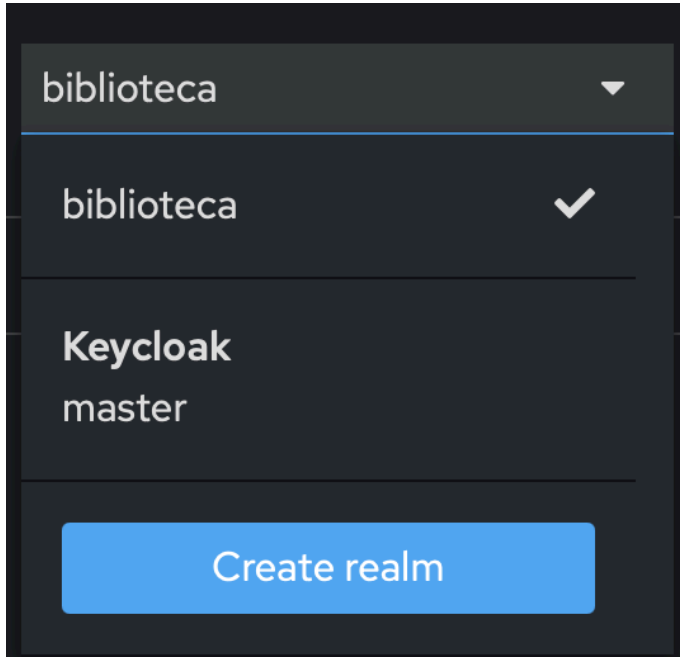
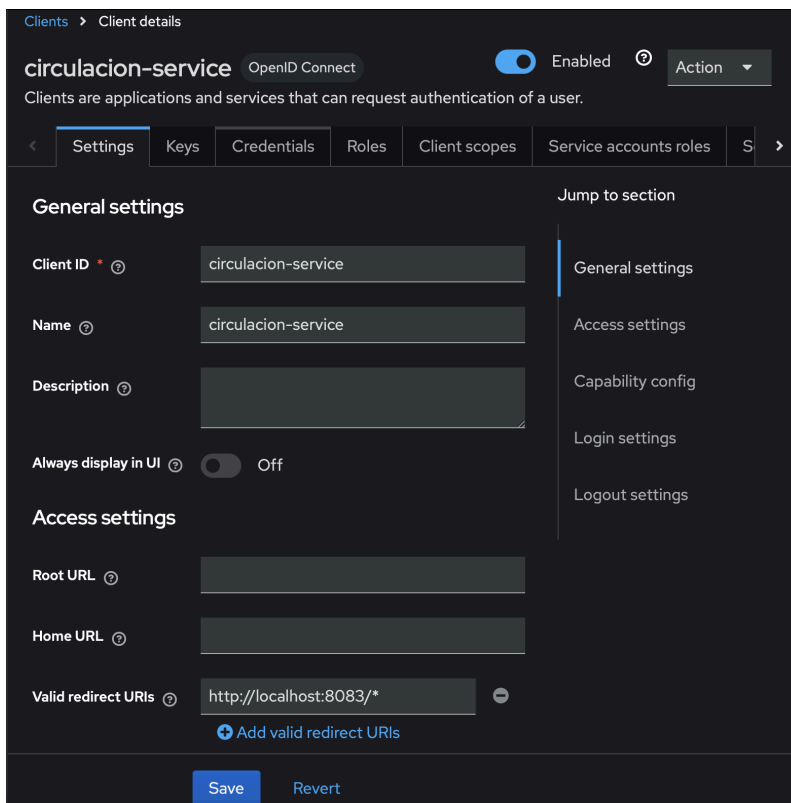


Configuración de Keycloak

Realm de biblioteca



circulacion-service



DwHcyiodvHOARIEhCQEiolokJW5Csa6g

Roles:

The screenshot shows the 'Roles' tab in the Keycloak administration console. At the top, the client 'circulacion-service' is selected, and the 'OpenID Connect' protocol is enabled. Below the client name, it states 'Clients are applications and services that can request authentication of a user.' The navigation bar includes tabs for Settings, Keys, Credentials, Roles (selected), Client scopes, Service accounts roles, and S. The Roles tab contains a search bar 'Search role by name', a 'Create role' button, a 'Refresh' button, and a pagination control showing '1 - 2'. The main content is a table with columns: Role name, Composite, and Description. Two roles are listed: 'ROLE_LIBRARIAN' and 'ROLE_USER', both with a 'Composite' value of 'False' and a 'Description' of '-'. Each role has a three-dot menu icon to its right.

Role name	Composite	Description
ROLE_LIBRARIAN	False	-
ROLE_USER	False	-

Usuarios de prueba y asignación de roles: Cada uno con su contraseña y rol asignado.

librarian1 -> ROLE_LIBRARIAN

user1 -> ROLE_USER

The screenshot shows the 'Users' tab in the Keycloak administration console. At the top, it says 'Users are the users in the current realm.' with a 'Learn more' link. The 'User list' tab is selected. Below the tab, there is a search bar 'Search user', an 'Add user' button, and a 'Delete user' button. A 'Refresh' button and a pagination control showing '1 - 2' are also present. The main content is a table with columns: Username, Email, Last name, and First name. Two users are listed: 'librarian1' and 'user1'. Both users have a red exclamation mark icon next to their email field, indicating an error or warning. Each user has a three-dot menu icon to its right.

Username	Email	Last name	First name
librarian1	-	-	-
user1	-	-	-

Implementación de Seguridad

Para integrar JWT y Keycloak con Spring Security

Se agregan las dependencias en pom.xml en cada microservicio.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-oauth2-resource-server</artifactId>
</dependency>
```

Configurar el application.properties

```
server.port=8083
spring.security.oauth2.resourceserver.jwt.issuer-uri=http://localhost:8080/realms/biblioteca
spring.security.oauth2.resourceserver.jwt.jwk-set-uri=${spring.security.oauth2.resourceserver.jwt.issuer-uri}\
/protocol/openid-connect/certs
```

Luego se crea el SecurityConfig para proteger los endpoints

```
1  package co.analisis.biblioteca.config;
2  > import ...
13
14  @Configuration
15  @EnableWebSecurity
16  @EnableMethodSecurity
17  public class SecurityConfig {
18      @Bean
19      public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
20          http
21              .authorizeHttpRequests((AuthorizationManagerRequestMat... authz -> authz
22                  .requestMatchers("/circulacion/public/**").permitAll()
23                  .anyRequest().authenticated())
24              .oauth2ResourceServer((OAuth2ResourceServerConfigurer<HttpSecurity> oauth2 -> oauth2
25                  .jwt((JwtConfigurer jwt ->
26                      jwt.jwtAuthenticationConverter(jwtAuthenticationConverter())));
27          return http.build();
28      }
29  @
30      private Converter<Jwt, ? extends AbstractAuthenticationToken> jwtAuthenticationConverter() { 1 usage
31          JwtAuthenticationConverter jwtConverter = new JwtAuthenticationConverter();
32          jwtConverter.setJwtGrantedAuthoritiesConverter(new KeycloakRealmRoleConverter());
33          return jwtConverter;
34      }
```

Luego se aplican las restricciones en los controladores

```
14
15 @RestController
16 @RequestMapping("/circulacion")
17 public class CirculacionController {
18     @Autowired
19     private CirculacionService circulacionService;
20
21     @PostMapping("/prestar")
22     @PreAuthorize("hasRole('ROLE_LIBRARIAN')")
23     public void prestarLibro(@RequestParam String usuarioId, @RequestParam String libroId) {
24         circulacionService.prestarLibro(new UsuarioId(usuarioId), new LibroId(libroId));
25     }
26
27     @PostMapping("/devolver")
28     @PreAuthorize("hasRole('ROLE_LIBRARIAN')")
29     public void devolverLibro(@RequestParam String prestamoId) {
30         circulacionService.devolverLibro(new PrestamoId(prestamoId));
31     }
32     @Operation(
33         summary = "Consultar todos los préstamos",
34         description = "Este endpoint permite obtener una lista de todos los préstamos registrados. Si no se encuentra el préstamo, se devuelve un mensaje de error. De lo contrario no podrá acceder a esta información."
35     )
36
37
38     @GetMapping("/prestamos")
39     public List<Prestamo> obtenerTodosPrestamos() {
40         return circulacionService.obtenerTodosPrestamos();
41     }
42
43     @GetMapping("/public/status")
44     public String getPublicStatus() {
45         return "El servicio de circulación está funcionando correctamente";
46     }
47 }
48
```

Documentación de APIs con Swagger

Se agrega la dependencia en pom.xml

```
<dependency>
    <groupId>org.springdoc</groupId>
    <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
    <version>2.5.0</version>
</dependency>
```

Configurar Swagger en application.properties

```
springdoc.api-docs.path=/v3/api-docs
springdoc.swagger-ui.path=/swagger-ui.html
```

Documentar los endpoints con anotaciones

```
@RestController
@RequestMapping("/circulacion")
public class CirculacionController {
    @Autowired
    private CirculacionService circulacionService;

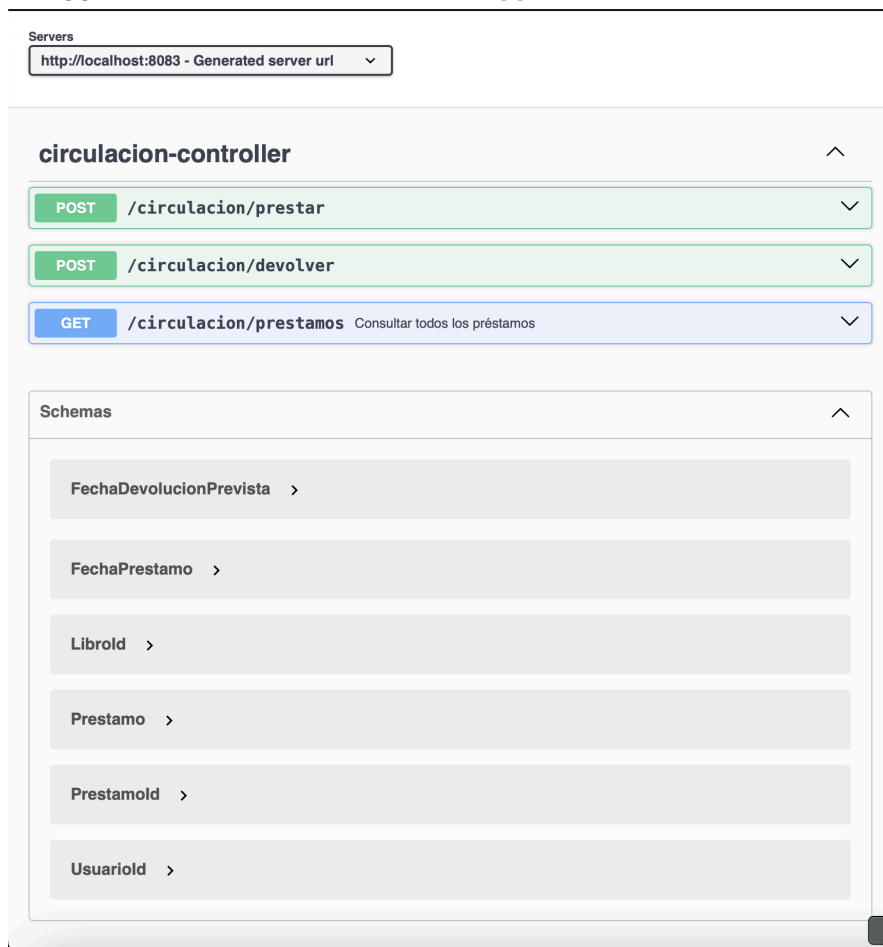
    @Operation(
        summary = "Prestar un libro",
        description = "Este endpoint permite prestar un libro a un usuario registrado en la base de datos. " + "Es importante que el cliente esté registrado previamente en la base de datos. " +
        "de lo contrario no podrá acceder a esta información."
    )
    @PostMapping("/prestar")
    @PreAuthorize("hasRole('ROLE_LIBRARIAN')")
    public void prestarLibro(@RequestParam String usuarioId, @RequestParam String libroid) {
        circulacionService.prestarLibro(new UsuarioId(usuarioId), new Libroid(libroid));
    }

    @Operation(
        summary = "Devolver un libro",
        description = "Este endpoint permite devolver un libro que ha sido prestado previamente. " + "Es importante que el cliente esté registrado previamente en la base de datos. " +
        "de lo contrario no podrá acceder a esta información."
    )
    @PostMapping("/devolver")
    @PreAuthorize("hasRole('ROLE_LIBRARIAN')")
    public void devolverLibro(@RequestParam String prestamoid) {
        circulacionService.devolverLibro(new Prestamoid(prestamoid));
    }

    @Operation(
        summary = "Consultar todos los préstamos",
        description = "Este endpoint permite obtener una lista de todos los préstamos registrados en el sistema. " + "Es importante que el cliente esté registrado previamente en la base de datos. " +
        "de lo contrario no podrá acceder a esta información."
    )
    @GetMapping("/prestamos")
    public List<Prestamo> obtenerTodosPrestamos() {
        return circulacionService.obtenerTodosPrestamos();
    }
}
```

Luego se accede a la documentación

Swagger UI: <http://localhost:8083/swagger-ui/index.html>



JSON OpenAPI: <http://localhost:8083/v3/api-docs>

```

"openapi": "3.0.1", "info": { "title": "OpenAPI definition", "version": "v0"}, "servers":
[{"url": "http://localhost:8083", "description": "Generated server url"}], "paths": {""/circulacion/prestar":
{"post": {"tags": ["circulacion-controller"], "operationId": "prestarLibro", "parameters":
[{"name": "usuarioId", "in": "query", "required": true, "schema": {"type": "string"}},
{"name": "libroId", "in": "query", "required": true, "schema": {"type": "string"}}], "responses": {"200":
{"description": "OK"}}, "/circulacion/devolver": {"post": {"tags": ["circulacion-
controller"], "operationId": "devolverLibro", "parameters":
[{"name": "prestamoId", "in": "query", "required": true, "schema": {"type": "string"}}], "responses": {"200":
{"description": "OK"}}, "/circulacion/prestamos": {"get": {"tags": ["circulacion-
controller"], "summary": "Consultar todos los prÃ©stamos", "description": "Este endpoint permite obtener una
lista de todos los prÃ©stamos registrados en el sistema. Es importante que el cliente estÃ© registrado
previamente en la base de datos, de lo contrario no podrÃ¡ acceder a esta
informaciÃ³n.", "operationId": "obtenerTodosPrestamos", "responses": {"200": {"description": "OK", "content":
{"/*": {"schema": {"type": "array", "items": {"$ref": "#/components/schemas/Prestamo"}}}}}, "components":
{"schemas": {"FechaDevolucionPrevista": {"type": "object", "properties": {"fechaDevolucionprevista value":
{"type": "string", "format": "date"}}, "FechaPrestamo": {"type": "object", "properties": {"fechaPrestamo value":
{"type": "string", "format": "date"}}, "LibroId": {"type": "object", "properties": {"libroid value":
{"type": "string"}}, "Prestamo": {"type": "object", "properties": {"id":
{"$ref": "#/components/schemas/PrestamoId"}, "usuarioId": {"$ref": "#/components/schemas/UsuarioId"}, "libroId":
{"$ref": "#/components/schemas/LibroId"}, "fechaPrestamo":
{"$ref": "#/components/schemas/FechaPrestamo"}, "fechaDevolucionPrevista":
{"$ref": "#/components/schemas/FechaDevolucionPrevista"}, "estado": {"type": "string", "enum":
["ACTIVO", "DEVUELTO", "VENCIDO"]}}, "PrestamoId": {"type": "object", "properties": {"prestamoid value":
{"type": "string"}}, "UsuarioId": {"type": "object", "properties": {"usuarioid value": {"type": "string"}}}}}

```

Ya que el resto de endpoints están protegidos, Swagger no los va a mostrar.

Pruebas de seguridad

Obtener un token JWT válido con Postman

New Collection / http://localhost:8080/reals/biblioteca/protocol/openid-connect/token

POST http://localhost:8080/reals/biblioteca/protocol/openid-connect/token

Params Authorization Headers (10) Body Scripts Settings Cookies

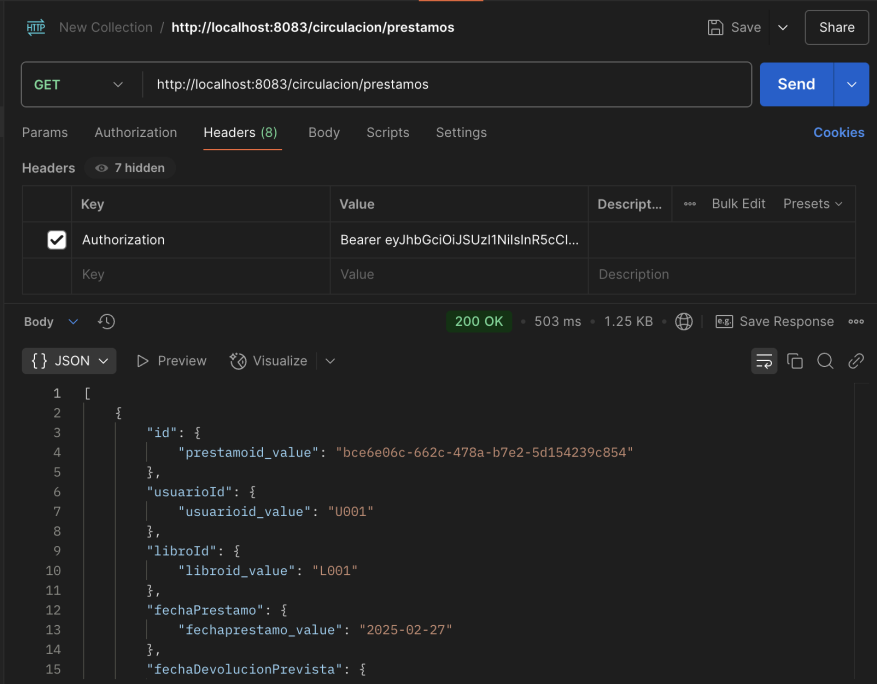
☐ none ☐ form-data ☒ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

	Key	Value	Description	Bulk Edit
<input checked="" type="checkbox"/>	grant_type	password		
<input checked="" type="checkbox"/>	client_id	circulacion-service		
<input checked="" type="checkbox"/>	client_secret	DwHcyIodvHOARIEHQEIoIokJW5Cs..		
<input checked="" type="checkbox"/>	username	librarian1		
<input checked="" type="checkbox"/>	password	Librarian@2024		

Body JSON Preview Visualize Save Response

```
{
  "access_token": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpzZW50LmV4IiwiaWF0IjoiYXZlcjZlMzIxR1eEJydkF6Z3VyblNkXHNtSWNrIn0.",
  "expires_in": 300,
  "refresh_expires_in": 1800,
  "refresh_token": "
```

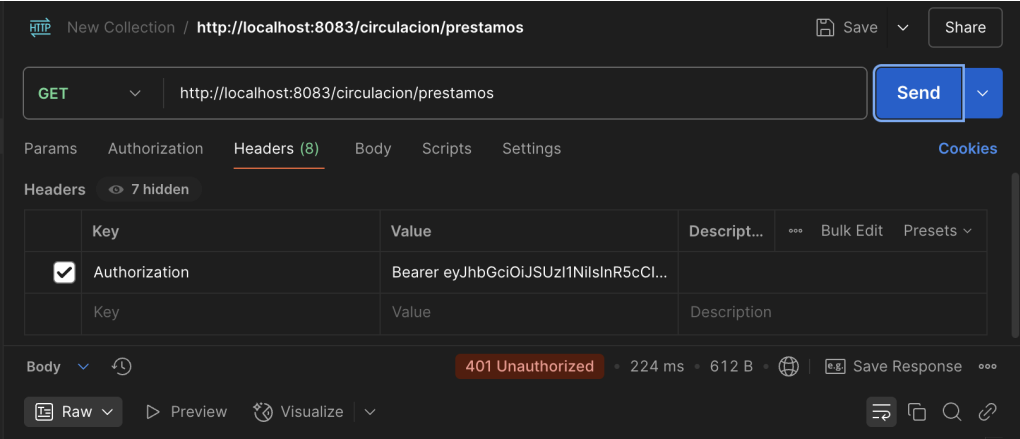
Probar el acceso con un token válido



Postman interface showing a successful GET request to `http://localhost:8083/circulacion/prestamos` with a valid Bearer token. The response is `200 OK` with a status of `503 ms` and a body size of `1.25 KB`. The response body is a JSON object:

```
1 [
2   {
3     "id": {
4       "prestamoid_value": "bce6e06c-662c-478a-b7e2-5d154239c854"
5     },
6     "usuarioId": {
7       "usuarioid_value": "U001"
8     },
9     "libroId": {
10      "libroid_value": "L001"
11    },
12    "fechaPrestamo": {
13      "fechaprestamo_value": "2025-02-27"
14    },
15    "fechaDevolucionPrevista": {
```

Probar acceso con token inválido o expirado



Postman interface showing a failed GET request to `http://localhost:8083/circulacion/prestamos` with an invalid Bearer token. The response is `401 Unauthorized` with a status of `224 ms` and a body size of `612 B`. The response body is raw text:

```
Raw
```

Esto se repitió para todos los microservicios

<https://github.com/AndresPin0/microservicios-biblioteca.git>