



UNIVERSIDAD DE LAS FUERZAS ARMADAS

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN

APLICACIONES DISSTRIBUIDAS

PARCIAL 3

TEMA: OAuth 2.0

INTEGRANTES:

- Simbaña Adrian
- Socasi Moises
- Torres Marlon

Sangolquí

14 .de febrero del 2025

implementación de OAuth 2.0

Introducción

Introducción

En el desarrollo de arquitecturas basadas en microservicios, la gestión de la autenticación es un aspecto fundamental para garantizar la seguridad de los sistemas. Spring Boot, en conjunto con Spring Security, proporciona un conjunto de herramientas robustas y flexibles para implementar autenticación y autorización de manera eficiente en entornos distribuidos.

A diferencia de las aplicaciones monolíticas, en un ecosistema de microservicios es necesario manejar la autenticación de forma descentralizada y escalable, asegurando que cada servicio pueda validar las credenciales de los usuarios sin comprometer el rendimiento o la seguridad del sistema. Para ello, Spring Boot permite la integración con múltiples mecanismos de autenticación, tales como:

- JWT (JSON Web Token): Un estándar ligero y seguro que permite la transmisión de información de autenticación entre servicios sin necesidad de mantener sesiones en el servidor.
- OAuth 2.0 y OpenID Connect: Protocolos ampliamente utilizados para la autenticación y autorización basados en tokens, que permiten la delegación de credenciales entre servicios.
- Autenticación basada en API Gateway: Uso de un Gateway para centralizar la validación de tokens y gestionar el acceso a los microservicios internos.
- Spring Security con bases de datos y LDAP: Uso de mecanismos tradicionales como bases de datos relacionales o directorios LDAP para la validación de credenciales.

La implementación de Spring Security en microservicios con Spring Boot permite definir estrategias avanzadas de autenticación.

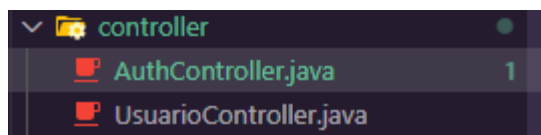
Desarrollo

Dentro del microservicio de usuarios se añade las dependencias necesarias para el uso de **OAuth 2.0**

OAuth 2.0 y Security

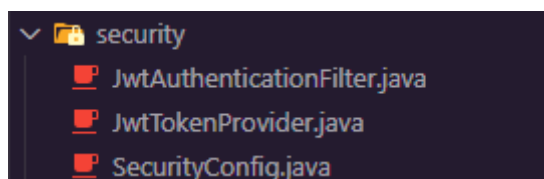
```
<!-- Spring Security -->
<dependency>
|   <groupId>org.springframework.boot</groupId>
|   <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<!-- OAuth2 -->
<dependency>
|   <groupId>org.springframework.security</groupId>
|   <artifactId>spring-security-oauth2-resource-server</artifactId>
</dependency>
```

Agregación del controlador AuthController



En este código se define un controlador de autenticación en Spring Boot para un microservicio de gestión de usuarios. Implementa inicio de sesión (login) y registro (register) utilizando Spring Security y JWT.

Se agrega la capa de servicios



JwtAuthenticationFiltre:

- Se ejecuta en cada solicitud para validar el token JWT.
 - Si el token es válido, extrae el usuario y roles.
 - Autentica al usuario y lo coloca en el contexto de seguridad de Spring.
- Filtro de Autenticación

Código clave para realizar el filtro de seguridad:

```
@Override
protected void doFilterInternal(HttpServletRequest request,
                                HttpServletResponse response,
                                FilterChain filterChain) throws ServletException, IOException {
    try {
        String jwt = getJwtFromRequest(request);

        if (StringUtils.hasText(jwt) && tokenProvider.validateToken(jwt)) {
            String username = tokenProvider.getUsernameFromJWT(jwt);
            String roles = tokenProvider.getRolesFromJWT(jwt);

            List<SimpleGrantedAuthority> authorities = Arrays.stream(roles.split(","))
                .map(SimpleGrantedAuthority::new)
                .collect(Collectors.toList());

            UserDetails userDetails = userDetailsService.loadUserByUsername(username);
            UsernamePasswordAuthentication authentication = new UsernamePasswordAuthenticationToken(
                userDetails, null, authorities);
            authentication.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));

            SecurityContextHolder.getContext().setAuthentication(authentication);
        }
    } catch (Exception ex) {
        logger.error("Could not set user authentication in security context", ex);
    }

    filterChain.doFilter(request, response);
}
```

JwtTokenProvider:

- Genera un token JWT cuando el usuario inicia sesión.
- Extrae el usuario y roles de un token existente.

- Valida tokens para verificar autenticidad.

Código clave para generar un token:

```
public String generateToken(Authentication authentication) {
    String username = authentication.getName();
    String roles = authentication.getAuthorities().stream()
        .map(GrantedAuthority::getAuthority)
        .collect(Collectors.joining(","));

    Date currentDate = new Date();
    Date expireDate = new Date(currentDate.getTime() + jwtExpirationInMs);

    return Jwts.builder()
        .setSubject(username)
        .claim("roles", roles)
        .setIssuedAt(new Date())
        .setExpiration(expireDate)
        .signWith(key, SignatureAlgorithm.HS512)
        .compact();
}
```

SecurityConfig:

- Configura las reglas de seguridad del microservicio.
- Define qué endpoints requieren autenticación y cuáles no.
- Desactiva CSRF y configura la sesión como stateless (sin estado).
- Añade el filtro JwtAuthenticationFilter antes del filtro de autenticación de Spring.

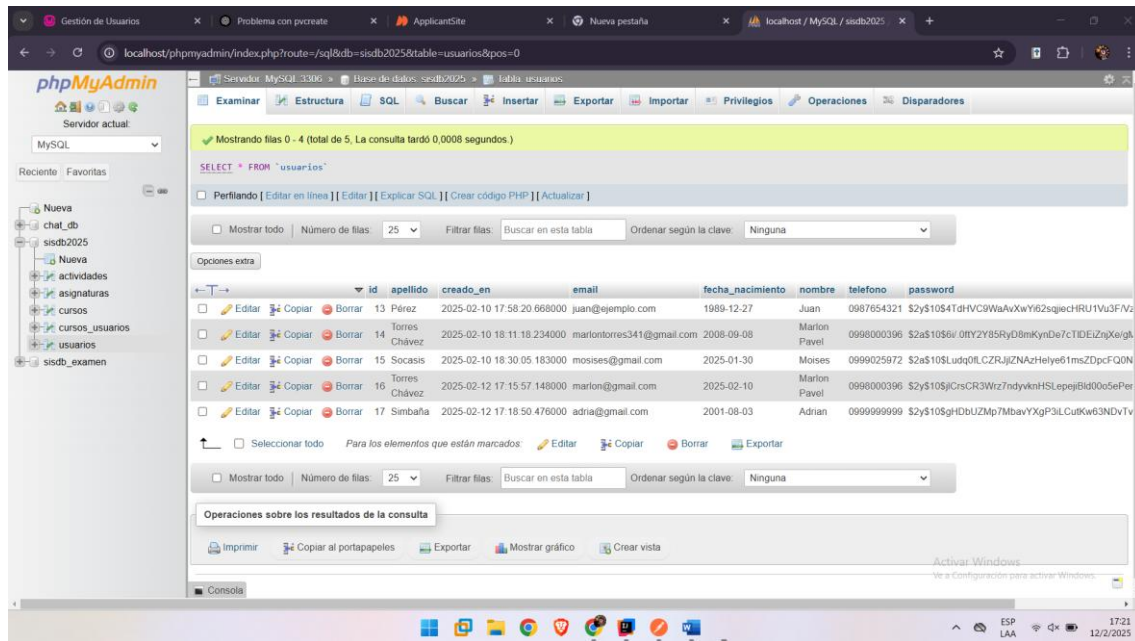
Código clave para definir accesos:

```
@Bean
SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    http
        .csrf(csrf -> csrf.disable())
        .sessionManagement(session -> session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
        .authorizeHttpRequests(authorize ->
            authorize
                .requestMatchers("/api/auth/**").permitAll()
                .requestMatchers("/api/usuarios/public/**").permitAll()
                .requestMatchers(HttpMethod.GET, "/api/usuarios/**").hasAnyRole("ADMIN", "USER")
                .requestMatchers("/api/usuarios/**").hasRole("ADMIN")
                .anyRequest().authenticated()
            )
        .addFilterBefore(jwtAuthenticationFilter, UsernamePasswordAuthenticationFilter.class);

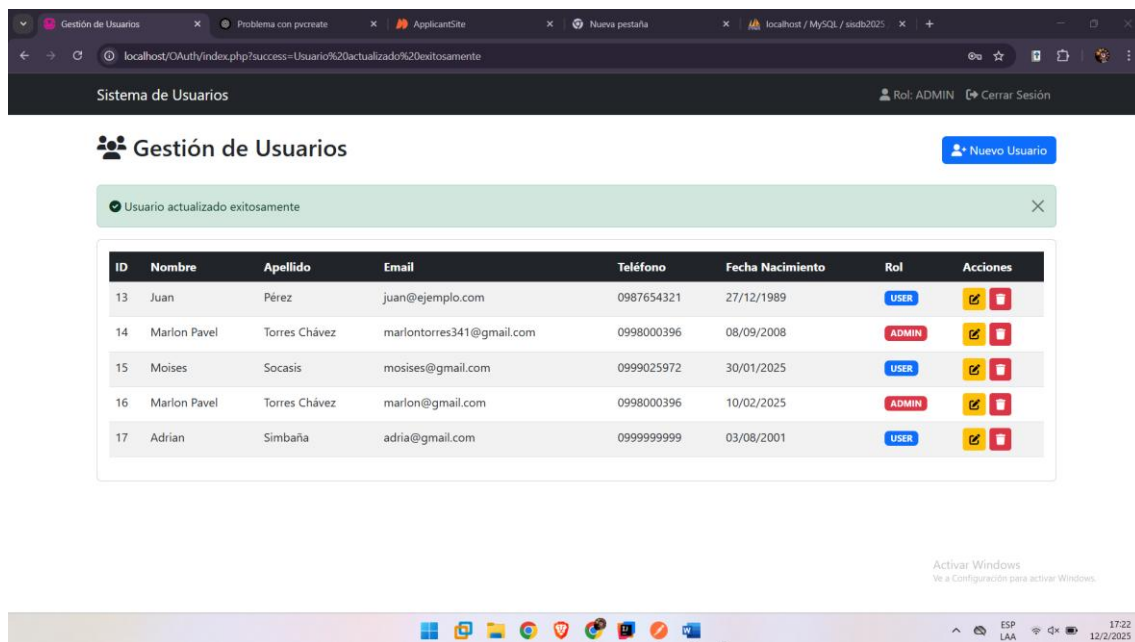
    return http.build();
}
```

Como resultado se implementa autenticación y autorización basada en JWT en **Spring Boot**, asegurando que solo usuarios autenticados accedan a los recursos adecuados en el microservicio.

Front para manejo de usuarios



Visualización del usuario en el Font



Edicion de datos de usuario existente mediante el front

localhost/OAuth/editar.php?id=13

Editar Usuario

Nombre: Juan

Apellido: Pérez

Email: juan@ejemplo.com

Contraseña (dejar en blanco para mantener la actual):

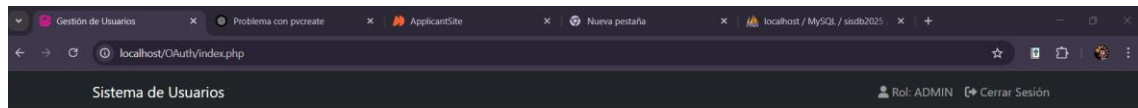
Teléfono: 0987654321

Fecha de Nacimiento: 27/12/1989

Rol: Usuario

[Actualizar](#) [Cancelar](#)

Activar Windows. Ve a Configuración para activar Windows.



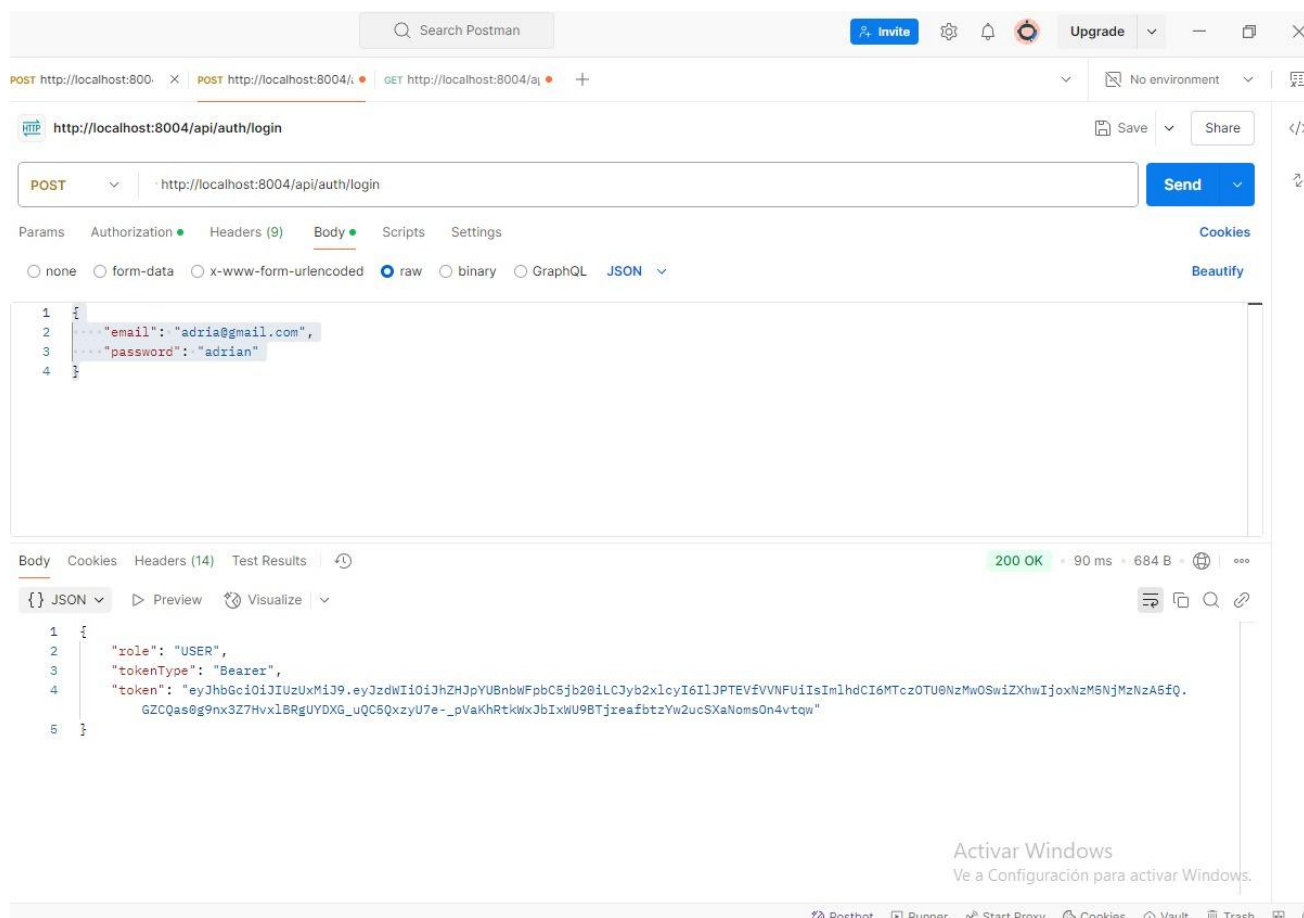
Gestión de Usuarios

[Nuevo Usuario](#)

ID	Nombre	Apellido	Email	Teléfono	Fecha Nacimiento	Rol	Acciones
13	Juan	Pérez	juan@ejemplo.com	0987654321	27/12/1989	USER	Editar Eliminar
14	Marlon Pavel	Torres Chávez	marlontorres341@gmail.com	0998000396	08/09/2008	ADMIN	Editar Eliminar
15	Moises	Socasis	mosises@gmail.com	0999025972	30/01/2025	USER	Editar Eliminar
16	Marlon Pavel	Torres Chávez	marlon@gmail.com	0998000396	10/02/2025	ADMIN	Editar Eliminar
17	Adrian	Simbaña	adria@gmail.com	0999999999	03/08/2001	USER	Editar Eliminar

Activar Windows. Ve a Configuración para activar Windows.


Para poder realizar en postman
1. Primero debemos iniciar sesion



2. Nos da un token y vamos al apartado de Authorization y legimos la opción de OAuth 2.0 y Ponemos el token que nos genero y podemos hacer el CRUD

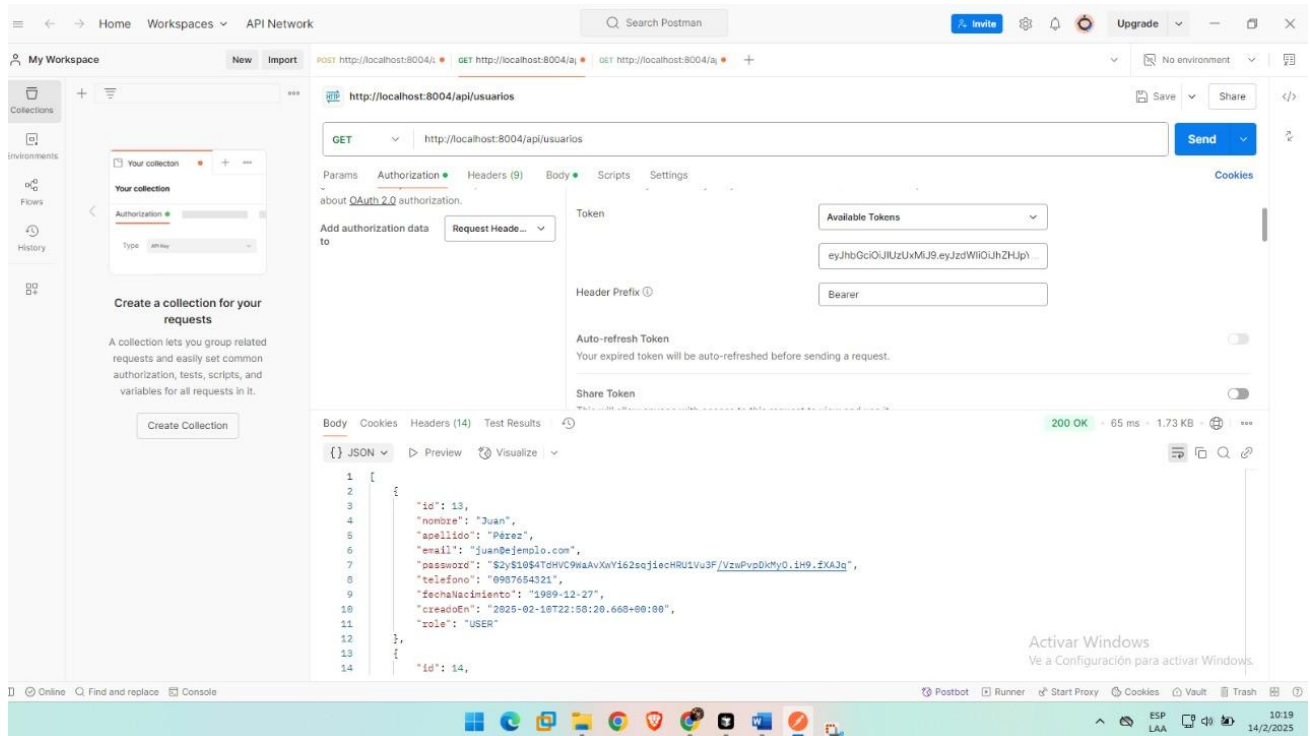
03

to let collaborators on this request use it.

Available Tokens 

eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhZHZhZjY...
Bearer

Estudiantes de la ESPE Ingeniería en Tecnologías de la Información



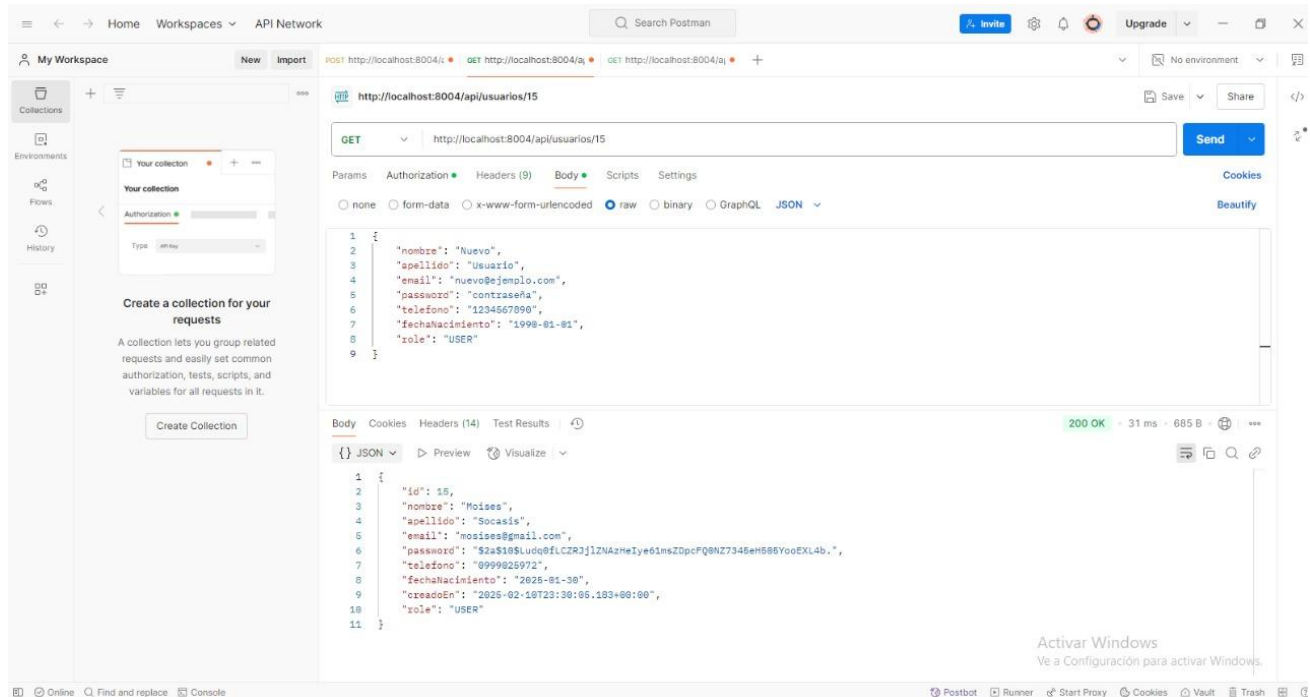
Postman interface showing a GET request to `http://localhost:8004/api/usuarios`. The response is a JSON array with two user objects.

```

1 [
2   {
3     "id": 13,
4     "nombre": "Juan",
5     "apellido": "Pérez",
6     "email": "juan@ejemplo.com",
7     "password": "$2y$10$47dHVC9wAvXvYis2sejleCHRU1Vu3F/VzaPvp0kMy0.iH9.fXA3q",
8     "telefono": "0987654321",
9     "fechaNacimiento": "1989-12-27",
10    "creadoEn": "2025-02-18T22:50:20.668+00:00",
11    "role": "USER"
12  },
13  {
14    "id": 14,

```

Consulta mediante id

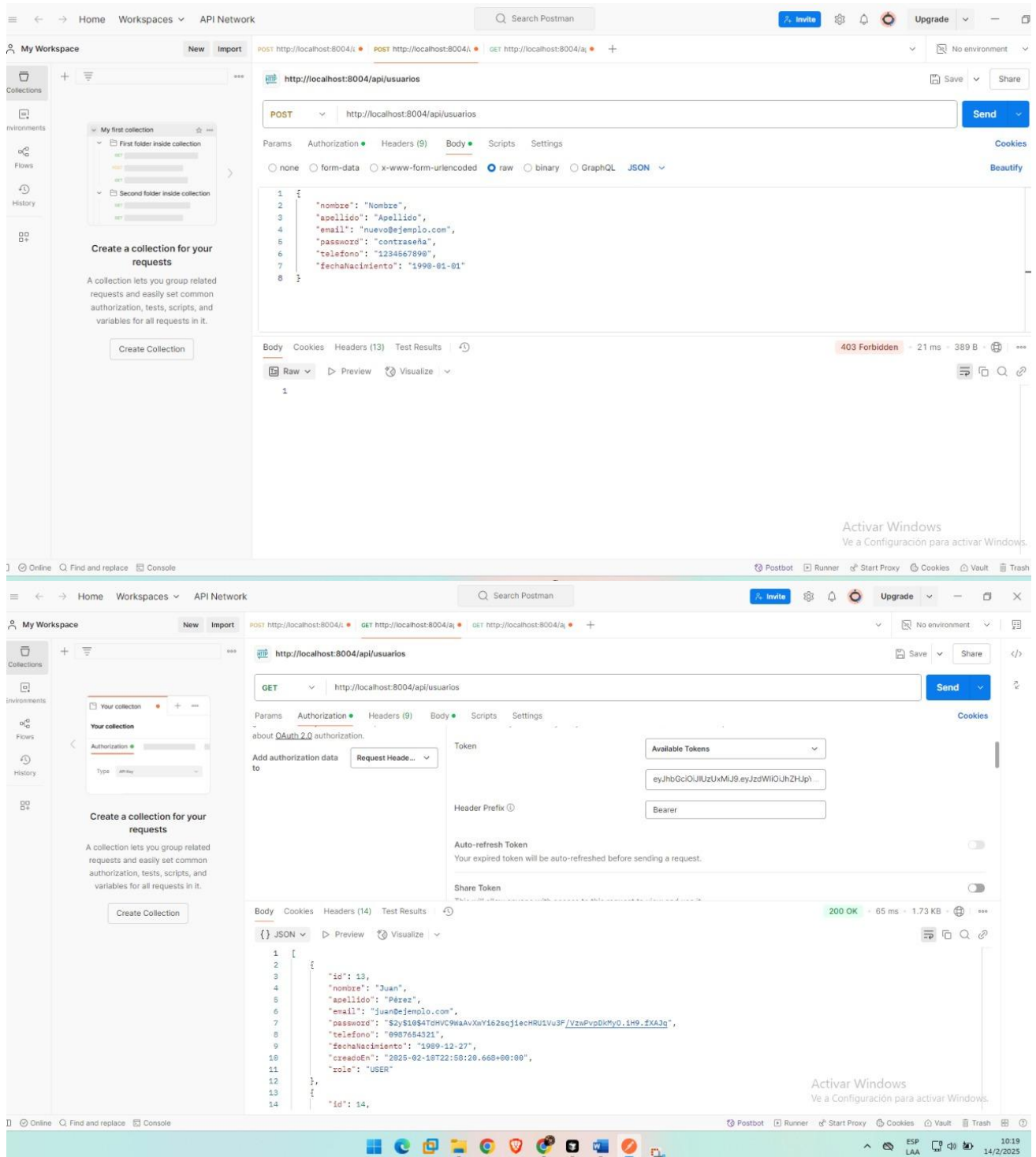


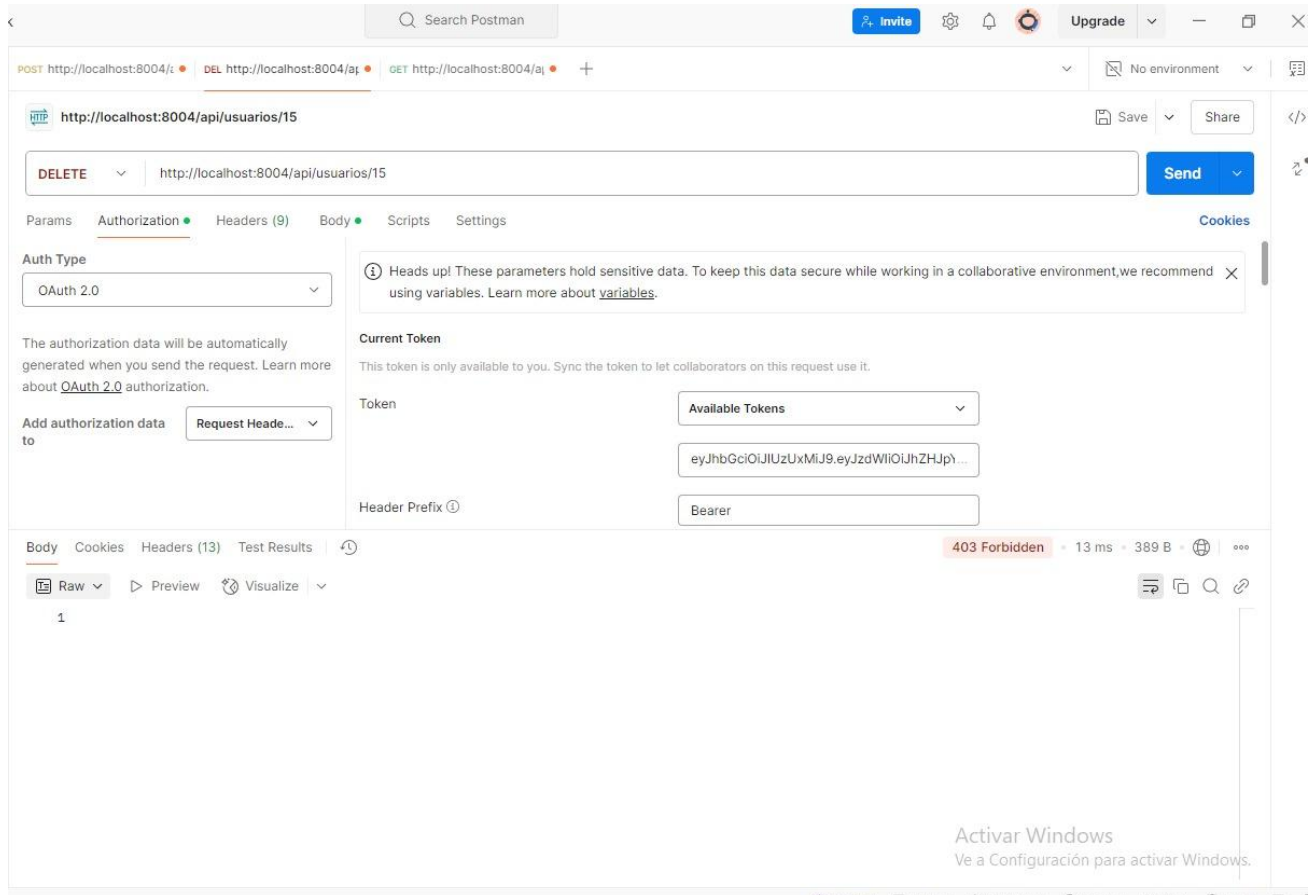
Postman interface showing a GET request to `http://localhost:8004/api/usuarios/15`. The response is a JSON object for a user with id 15.

```

1 {
2   "id": 15,
3   "nombre": "Moises",
4   "apellido": "Socasis",
5   "email": "mosises@gmail.com",
6   "password": "$2a$10$Ludo8fLCR3j1ZNAzMeIye61msZ0pcFQ8NZ7348em86YooEXL4b.",
7   "telefono": "0999825972",
8   "fechaNacimiento": "2025-05-30",
9   "creadoEn": "2025-02-18T23:30:06.183+00:00",
10  "role": "USER"
11 }

```





Conclusión

En conclusión, la arquitectura implementada cumple con las mejores prácticas de seguridad en microservicios, facilitando una gestión eficiente de usuarios y roles dentro del sistema. Esta solución no solo mejora la seguridad y control de acceso, sino que también permite una fácil integración con otros microservicios, habilitando un ecosistema escalable y seguro.