



FEIRNNR - Carrera de computación

UNIVERSIDAD NACIONAL DE LOJA

**Facultad De Industria, Energía y Recursos
Naturales No Renovables**

Carrera: Computación

Tema:

APE-003

Asignatura:

Desarrollo Basado en Plataformas

Docente:

Ing. Edison Coronel

Integrantes:

- Santiago Jimenez
- Leonardo Espinoza
- Isabel Morocho

Curso:

5to

LOJA – ECUADOR

2025

Actividades Práctico-Experimentales

Nro. 002

1. Datos Generales

Asignatura	Desarrollo Basado en Plataformas
Ciclo	5to "A"
Unidad	1
Resultado de aprendizaje de la unidad	Discute cómo los estándares Web influyen en el desarrollo de software, bajo los principios de solidaridad, transparencia, responsabilidad y honestidad.
Integrantes	<ul style="list-style-type: none">• Leonardo Espinoza• Santiago Jimenez• Isabel Morocho

2. Título:

Implementación del flujo de autenticación y autorización en el backend, aplicando mecanismos de seguridad (JWT u OAuth2), validaciones, CORS y principios OWASP Top 10, e incorporación de estos componentes al modelo C4.

3. Objetivo:

Configurar e implementar un mecanismo de autenticación y autorización seguro en el backend del proyecto.

Aplicar políticas CORS, validaciones y manejo de errores según buenas prácticas OWASP.

Documentar el proceso mediante pruebas Postman/Swagger y actualizar los diagramas C4 (Container y Component) reflejando los puntos de seguridad.

4. Procedimiento / Metodología

Inicio

- Presentación de los objetivos y repaso de los conceptos JWT/OAuth2.
- Conformación de equipos y revisión de la rama de desarrollo (*feature/security*).

Desarrollo

1. Configuración de entorno

- Instalar dependencias necesarias para seguridad (por ejemplo, *json web token*, *bcrypt*, *cors*).

Instalación de json web token bcrypt cors

```
PS C:\Users\HP\Desktop\Git_Repositorio\Sistema-FitZone\sistema-FitSIL> npm install jsonwebtoken bcrypt cors
added 21 packages in 13s
1 package is looking for funding
  run `npm fund` for details
PS C:\Users\HP\Desktop\Git_Repositorio\Sistema-FitZone\sistema-FitSIL>
```

```
PS C:\Users\HP\Desktop\Git_Repositorio\Sistema-FitZone\sistema-FitSIL> npm list jsonwebtoken bcrypt cors
sistema-FitSIL@ C:\Users\HP\Desktop\Git_Repositorio\Sistema-FitZone\sistema-FitSIL
├── bcrypt@6.0.0
├── cors@2.8.5
└── jsonwebtoken@9.0.2
PS C:\Users\HP\Desktop\Git_Repositorio\Sistema-FitZone\sistema-FitSIL>
```

- Crear archivo `.env` con claves secretas (no versionadas).

Spring Boot no necesita un `.env` porque ya tiene su propio mecanismo para manejar variables de entorno, llamado el Environment Abstraction

2. Implementación del flujo JWT / OAuth2

- Crear rutas `/auth/login` y `/auth/register`.

LOGIN

```
// Login
@PostMapping("/login")
public ResponseEntity<Object> login(@RequestBody Usuario usuario) {
    Optional<Usuario> logeado = usuarioService.login(usuario.getCorreo(), usuario.getContrasenia());
    if (logeado.isPresent()) {
        Usuario u = logeado.get();
        String token = jwtService.generarToken(u.getCorreo(), u.getRol().toString());
        Map<String, Object> respuesta = new HashMap<>();
        respuesta.put(key:"usuario", u);
        respuesta.put(key:"token", token);
        return ResponseEntity.ok(respuesta);
    } else {
        return ResponseEntity.status(401).body("Credenciales inválidas");
    }
}
```

REGISTER

```
@Autowired
private JwtService jwtService;

// Registro de usuario
@PostMapping("/registro")
public ResponseEntity<Usuario> registrar(
    @Valid @RequestBody Usuario usuario,
    BindingResult bindingResult) {

    if (bindingResult.hasErrors()) {
        String mensaje = bindingResult.getAllErrors()
            .get(0)
            .getDefaultMessage();
        return ResponseEntity.badRequest().body(null);
    }

    Usuario nuevo = usuarioService.registrarUsuario(usuario);
    return ResponseEntity.status(201).body(nuevo);
}
```

- **Generar token JWT con exp, iat y roles de usuario.**

Se implementó la generación de tokens JWT (JSON Web Token) para el proceso de autenticación, y cuando un usuario inicia sesión correctamente, el sistema crea un token firmado que incluye los (claims iat y exp). Los roles también se incluyen como claim "roles": "USUARIO".

```
Body Cookies Headers (15) Test Results [Refresh] 200 OK • 1.45 s • 910 B [Globe] [Save Response] ...  
[JSON] Preview Visualize  
7     "correo": "isa@example.com",  
8     "usuario": "isa2",  
9     "contrasenia": "$2a$10$PP0YFsRdl1mUUTZ2nke4A.P6Ipie6cNydhB40rZyy5dvvES8EJgEy",  
10    "rol": "USUARIO",  
11    "peso": 50.0,  
12    "altura": 1.65  
13 },  
14    "token": "eyJhbGciOiJIUzI1NiJ9.  
      eyJzdWIiOiJpc2FAZXhhbXBsZS5jb20iLCJyb2wiOiJVU1VBUCkPIiwiaWF0IjoxNzYxMTg5Mzk4LClleHAiOjE3NjExOTI  
      50Th9.1j6Q8csh-PHb5X-YplHLeIvvE5sS1xlt4tdruF77rRg"  
15 }
```

- Crear middleware de verificación de token y roles (RBAC- Control de acceso basado en roles).

RBAC es un técnica que se emplea en sistemas para la seguridad restringiendo el acceso a los diferentes tipos de recursos según el rol de usuario autenticado, para nuestro sistema cada usuario tiene si rol (ADMINISTRADOR Y USUARIO) y en el backend se valida el rol antes de acceder a los endpoints

RoleFilter

Este es el archivo middleware que intercepta las peticiones y valida el rol.

```
@Component
public class RoleFilter extends OncePerRequestFilter {

    private static final List<String> PUBLIC_PATHS = List.of(
        "/usuarios/registro",
        "/usuarios/login",
        "/auth/login"
    );

    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain filterChain)
        throws ServletException, IOException {

        String path = request.getServletPath();

        // Permitir acceso libre a rutas públicas
        if (PUBLIC_PATHS.stream().anyMatch(path::startsWith)) {
            filterChain.doFilter(request, response);
            return;
        }

        String authHeader = request.getHeader("Authorization");
        if (authHeader == null || !authHeader.startsWith("Bearer ")) {
            response.setStatus(HttpServletResponse.SC_FORBIDDEN);
            response.setContentType("application/json");
            response.getWriter().write("{\"error\": \"Acceso denegado: falta token JWT\"}");
            return;
        }
    }
}
```

SecurityConfig

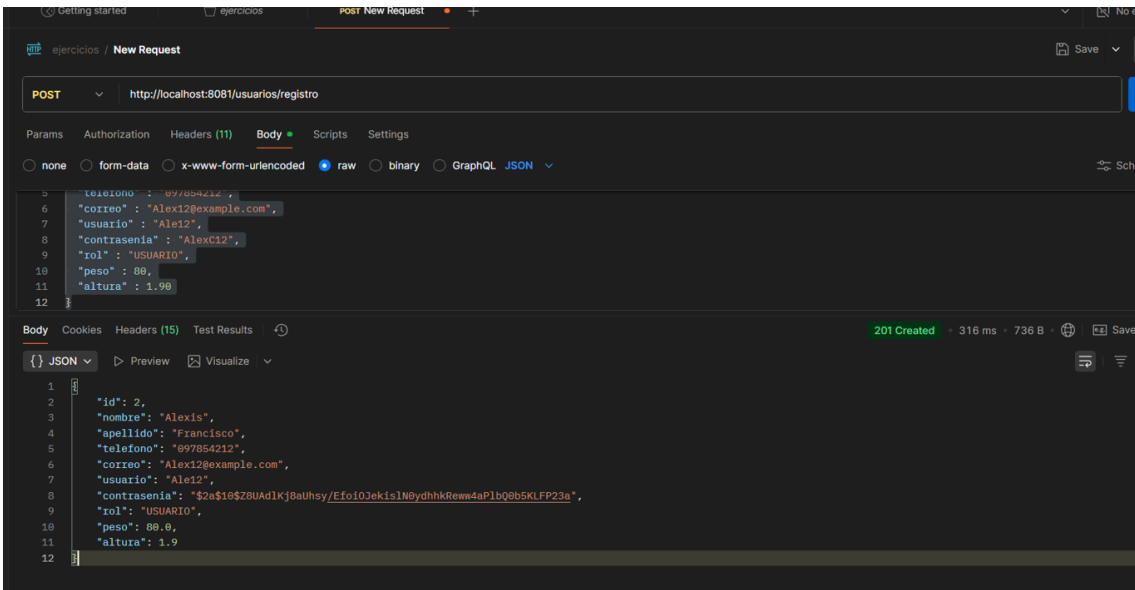
```
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {

    http
        .csrf(csrf -> csrf.disable()) // CSRF off porque usas JWT
        .cors(cors -> cors.configurationSource(corsConfigurationSource()))
        .authorizeHttpRequests(auth -> auth
            .requestMatchers("/usuarios/registro", "/usuarios/login", "/auth/**").permitAll()
            .anyRequest().authenticated()
        )
        .headers(headers -> headers
            .contentSecurityPolicy("default-src 'self'; script-src 'self'")
            .and()
            .httpStrictTransportSecurity(hsts -> hsts.includeSubDomains(true).maxAgeInSeconds(31536000))
            .frameOptions().sameOrigin()
        )
        .sessionManagement(session -> session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
        .addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class)
        .addFilterAfter(roleFilter, JwtAuthFilter.class);

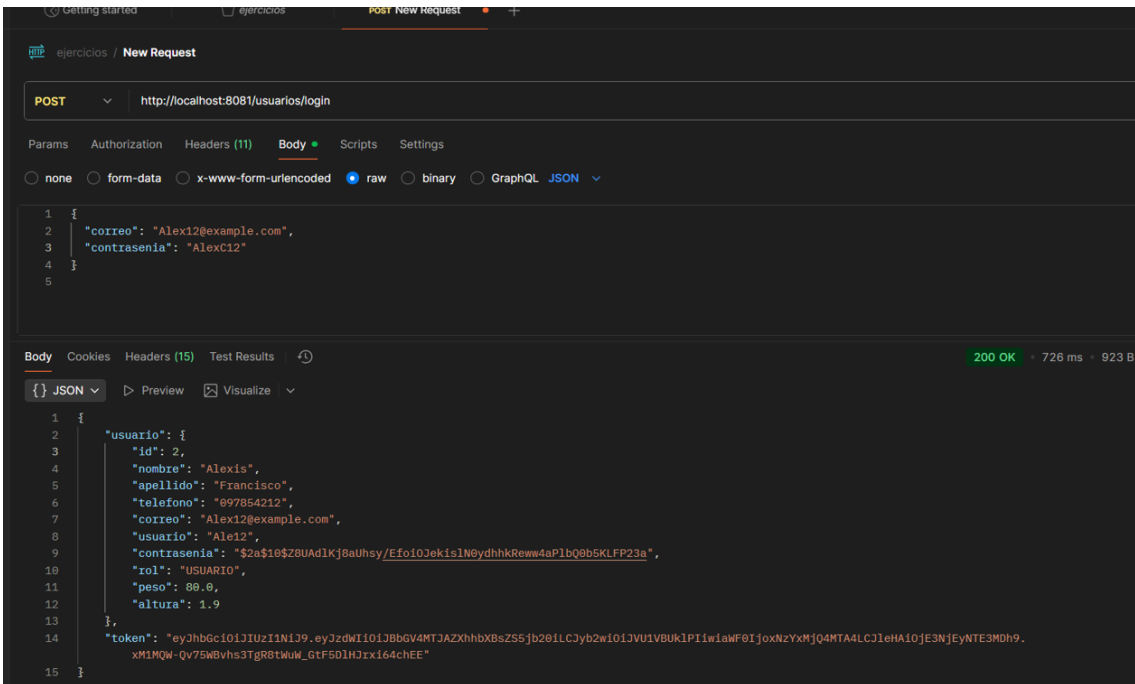
    return http.build();
}
```

Mientras que en el archivo SecurityConfig se añade el Rol Filter como middleware que se ejecuta luego de validar el Jwt.

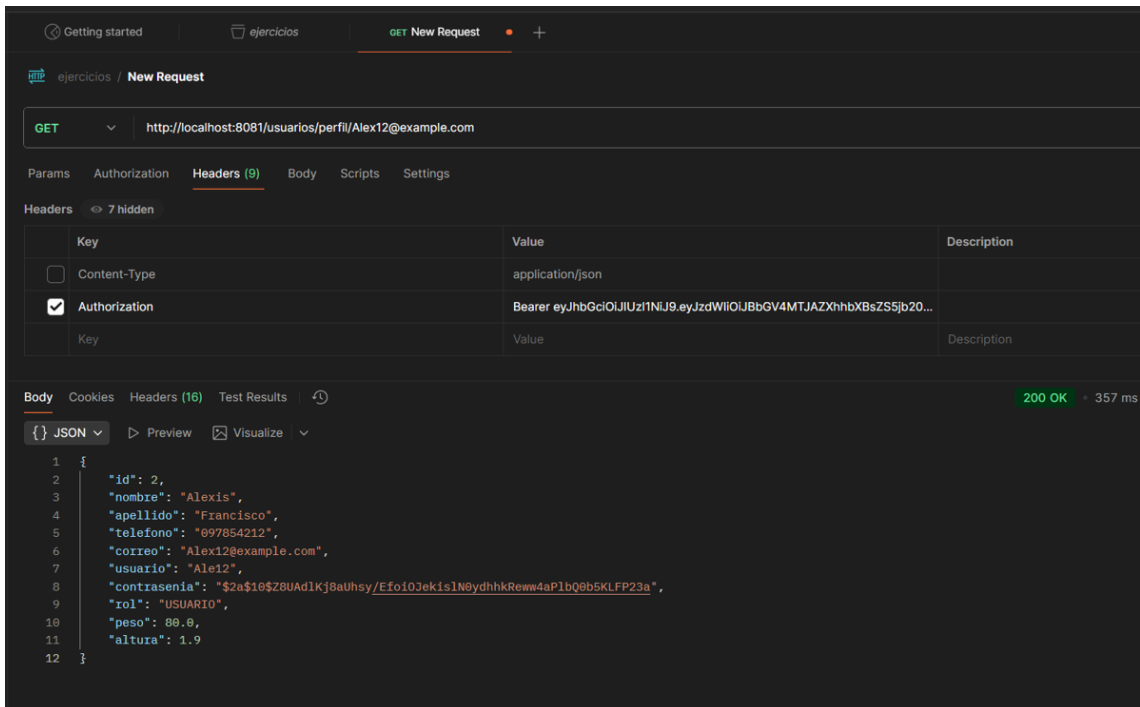
Registro de un nuevo usuario



Acceso al sistema al nuevo usuario



Visualización del perfil del usuario



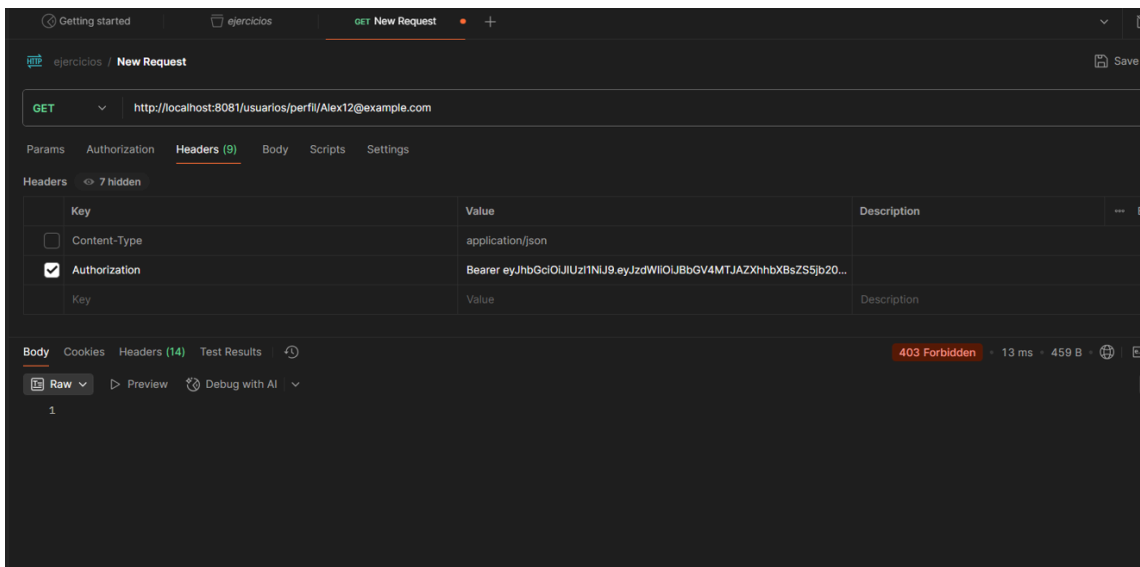
The screenshot shows a Postman interface with a GET request to `http://localhost:8081/usuarios/perfil/Alex12@example.com`. The request is configured with the following headers:

Key	Value	Description
<input type="checkbox"/> Content-Type	application/json	
<input checked="" type="checkbox"/> Authorization	Bearer eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJBbGV4MTJAZXhhbXBsZS5jb20...	
Key	Value	Description

The response is a 200 OK status with a response time of 357 ms. The body is displayed in JSON format:

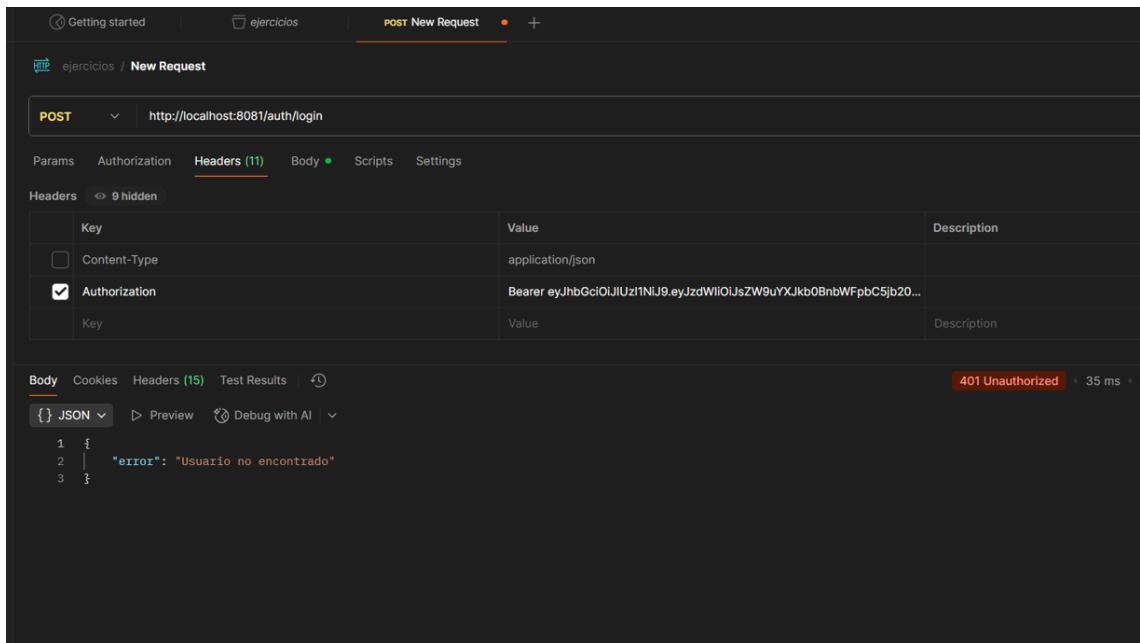
```
1 {
2   "id": 2,
3   "nombre": "Alexis",
4   "apellido": "Francisco",
5   "telefono": "097854212",
6   "correo": "Alex12@example.com",
7   "usuario": "Ale12",
8   "contrasenia": "$2a$10$ZBUAdlKj8aUhsy/Efoi0Jekis1N0ydhhkRww4aP1bQ0b5KLP23a",
9   "rol": "USUARIO",
10  "peso": 80.0,
11  "altura": 1.9
12 }
```

Acceso de un usuario con un token invalido o expirado



The screenshot shows the same GET request in Postman, but the response is a 403 Forbidden status with a response time of 13 ms and a body size of 459 B. The body is displayed in Raw format.

Intento de acceso de un usuario a funciones de administrador



3. Configuración CORS y validaciones

- Definir orígenes permitidos y métodos HTTP.

Dentro de la clase `SecurityConfig` se creó un bean de configuración CORS que establece los orígenes, métodos y encabezados permitidos

```
@Bean
public CorsConfigurationSource corsConfigurationSource() {
    CorsConfiguration config = new CorsConfiguration();
    config.setAllowedOrigins(List.of("https://fitsil-front.app", "http://localhost:3000"));
    config.setAllowedMethods(List.of("GET", "POST", "PUT", "DELETE", "OPTIONS"));
    config.setAllowedHeaders(List.of("Authorization", "Content-Type", "Accept"));
    config.setAllowCredentials(false);

    UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
    source.registerCorsConfiguration("/**", config);
    return source;
}
```

esta configuración solo podrá aceptar solicitudes desde <https://fitsil-front.app> y rechazara cualquier otro dominio distinto. Se establecen también métodos seguros y controlados como GET, POST, PUT y DELETE, al igual que omite encabezados y credenciales poco relevantes.

- **Agregar validación de entrada (request body y params).**

Se usó Bean Validation con anotaciones en los modelos, y al poner las validaciones allí (@NotBlank, @Email, @Size, etc.), todas las partes del sistema que usen ese modelo aplican automáticamente las mismas reglas, sin tener que repetirlas en cada controlador.

```
11
12 @Entity
13 @Inheritance(strategy = InheritanceType.JOINED)
14 public class Persona {
15
16     @Id
17     @GeneratedValue(strategy = GenerationType.IDENTITY)
18     private Integer id;
19
20     @NotBlank
21     @Size(min = 3, max = 50)
22     private String nombre;
23
24     private String apellido;
25     private String telefono;
26
27     @NotBlank(message = "Correo obligatorio")
28     @Email(message = "Correo inválido")
29     private String correo;
30
31     @NotBlank(message = "El nombre de usuario es obligatorio")
32     @Size(min = 3, max = 20, message = "El usuario debe tener entre 3 y 20 caracteres")
33     @Pattern(regexp = "^[a-zA-Z0-9_]+$", message = "Usuario solo puede contener letras, números y guiones bajos")
34     private String usuario;
35
36     private String contrasenia;
```

- **Implementar manejo de errores uniforme (códigos HTTP y mensajes JSON).**

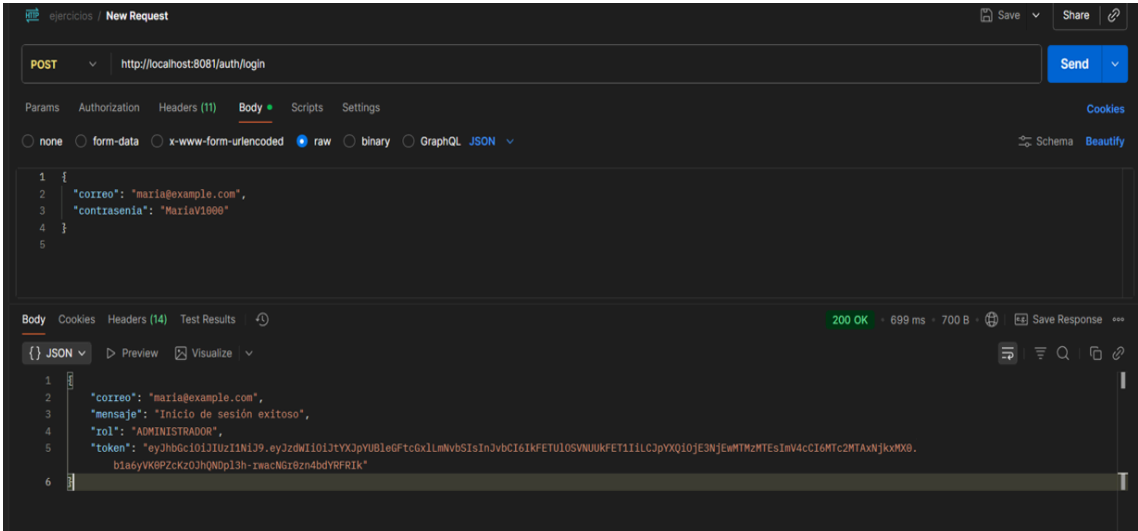
Se definió una clase global para el manejo de excepciones denominada GlobalExceptionHandler, lo que hace es interceptar los errores que hay en toda la aplicación y devolverlos en un formato JSON.

```
@ControllerAdvice
public class GlobalExceptionHandler {

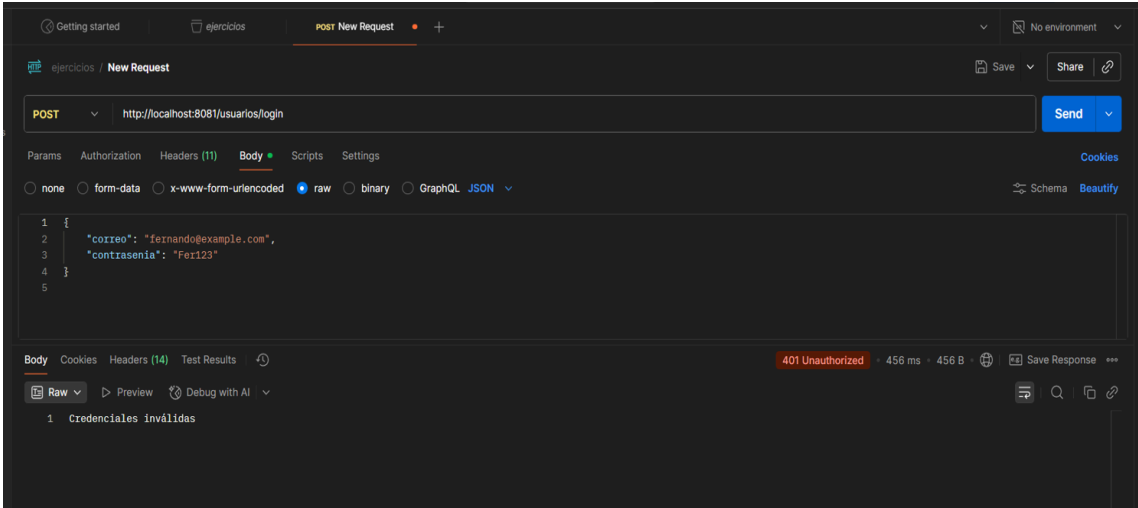
    @ExceptionHandler({ConstraintViolationException.class})
    public ResponseEntity<?> handleConstraintViolation(ConstraintViolationException ex) {
        String errores = ex.getConstraintViolations()
            .stream()
            .map(v -> v.getPropertyPath() + ": " + v.getMessage())
            .collect(Collectors.joining(", "));
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(Map.of("errores", errores));
    }

    @ExceptionHandler({IllegalArgumentException.class})
    public ResponseEntity<?> handleIllegalArg(IllegalArgumentException ex) {
        return ResponseEntity.badRequest().body(Map.of("error", ex.getMessage()));
    }
}
```

Inicio de sesión de un administrador

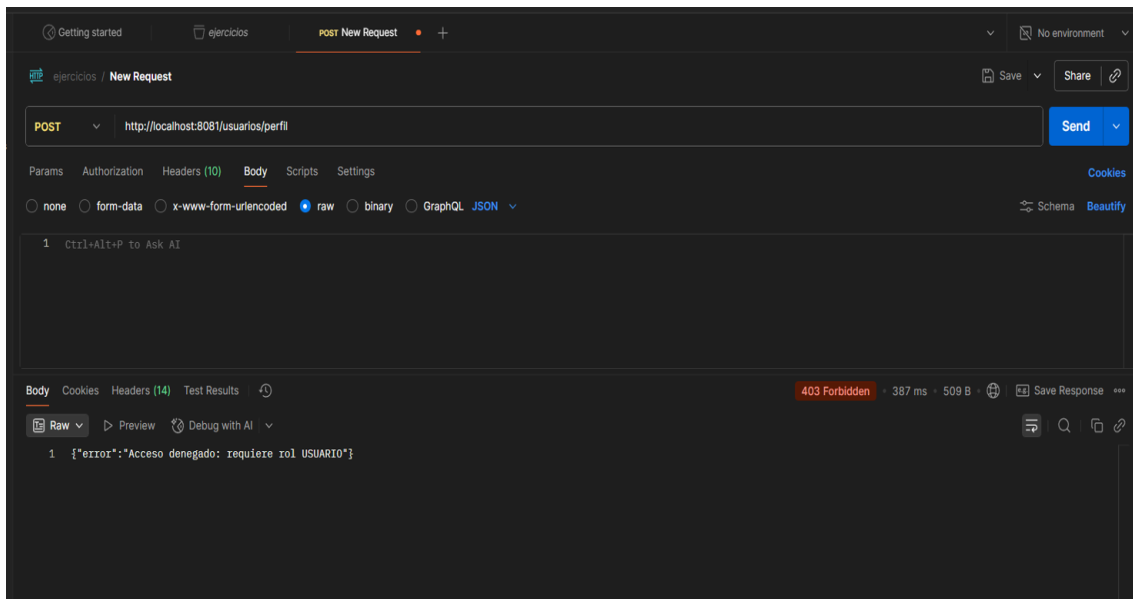


Credenciales que no constan en el sistema





Usuarios no registrados en el sistema



4. Pruebas y verificación

- Probar login y rutas protegidas con Postman/Swagger.
- Documentar resultados (capturas de respuesta 200, 401, 403).

5. Modelo C4 con seguridad

- **Actualizar diagramas Container y Component para incluir los servicios de autenticación y módulos de seguridad.**

Diagrama de Contenedores Actualizado

Sistema FitSIL

Este sistema esta enfocado en el entrenamiento físico personalizado, que permite a los usuarios registrar rutinas, controlar su progreso y acceder a planes de ejercicios según sus objetivos

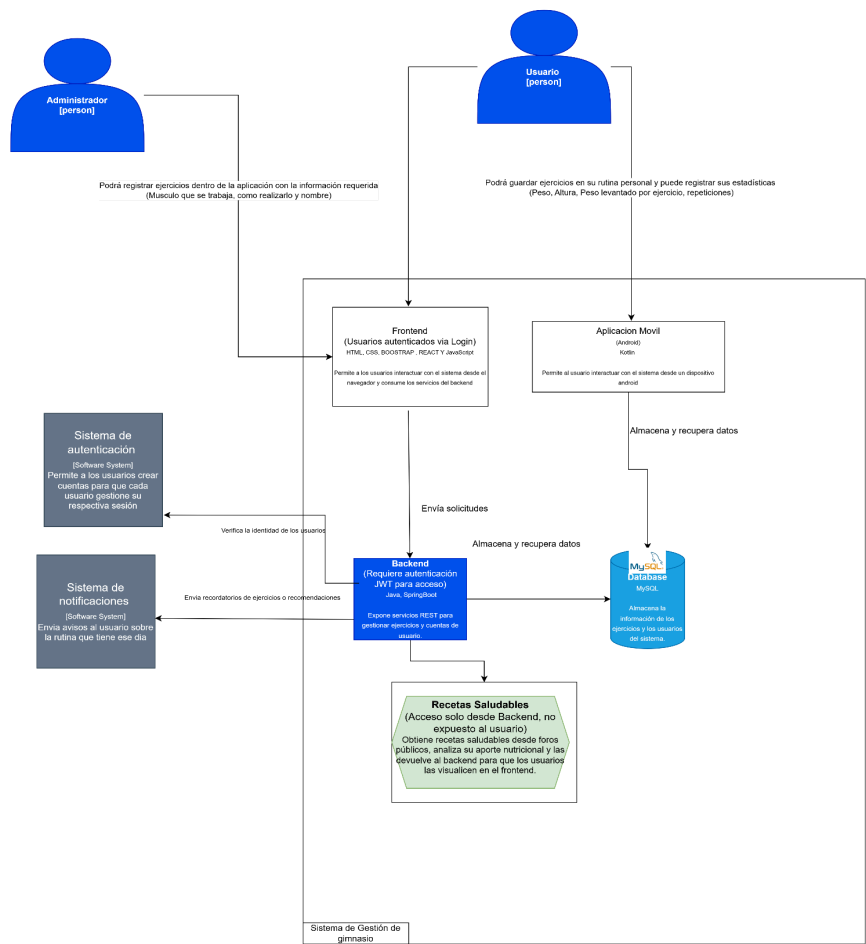
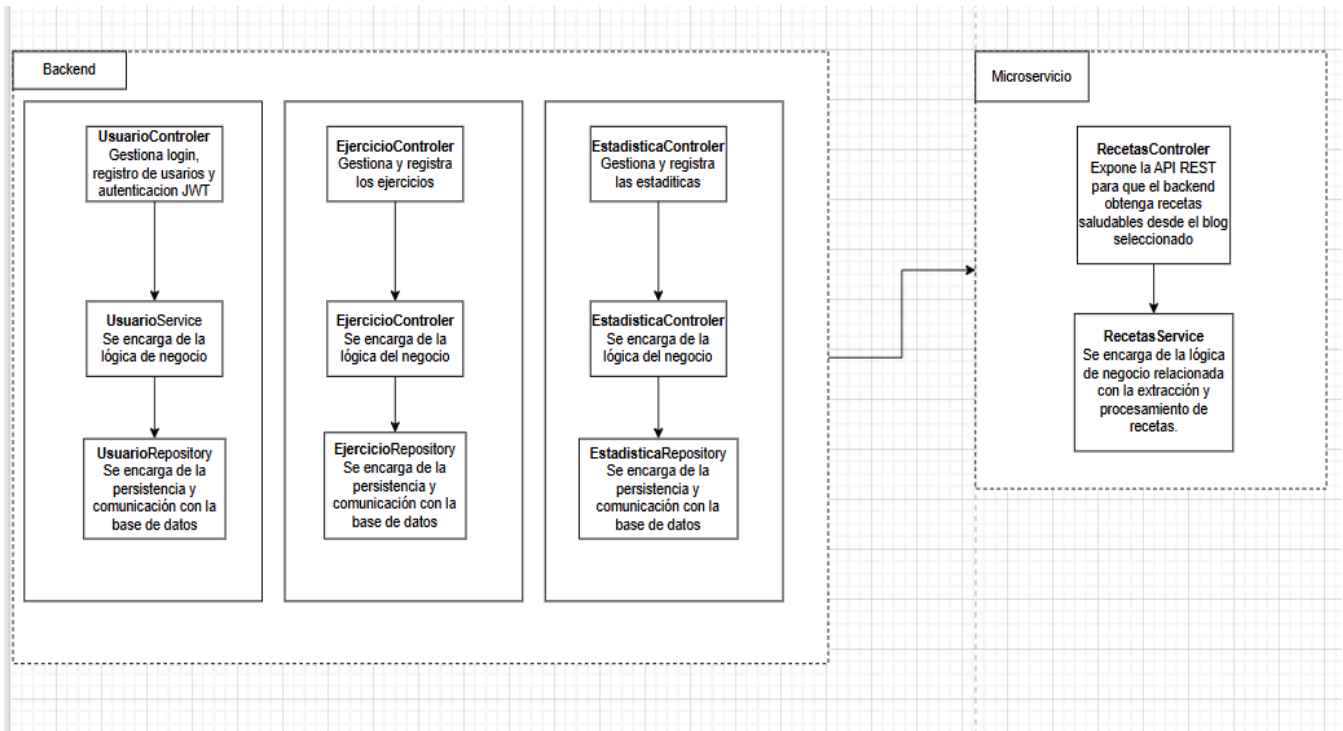


Diagrama de Componentes Actualizado



- Guardar los diagramas actualizados en </docs/architecture/>.

7. Resultados esperados:

- Backend con flujo JWT u OAuth2 funcional.
- Políticas CORS y validaciones implementadas.
- Colección Postman/Swagger con pruebas de autenticación.
- Diagramas C4 actualizados mostrando componentes de seguridad.
- Evidencias (capturas de pantalla + README actualizado).

8. Preguntas de Control:

1. ¿Cuál es la diferencia fundamental entre autenticación y autorización dentro de un sistema backend?

La autenticación se refiere a verificar quién eres, cómo cuándo usuario hace login con su correo y contraseña, el sistema valida sus credenciales, en cambio la autorización es decidir qué se puede hacer dentro del sistema es decir los permisos que tiene.

2. ¿Qué ventajas ofrece JWT frente a sesiones tradicionales de servidor y qué vulnerabilidades puede tener?

Las ventajas que ofrece son:

- El token lleva toda la información (usuario y roles), no hace falta guardar sesiones en memoria.
- JWT fácil de usar en sistemas con múltiples servidores o microservicios

Puede tener vulnerabilidades como:

- Robo de token, si un atacante obtiene el JWT, puede usarlo hasta que expire.
- Falsificación del token, si la clave secreta no es segura, alguien podría generar un token válido.

3. **Explique cómo CORS protege (o restringe) la comunicación entre cliente y servidor.**

CORS (Cross-Origin Resource Sharing), es quien decide qué dominios pueden hacer peticiones en el backend. Por ejemplo en nuestro frontend en `http://localhost:4200` puede hacer solicitudes a `http://localhost:8081`, pero otro dominio no permitido será bloqueado por el navegador. Esto protege el backend de accesos no deseados desde sitios esternos.

4. **Mencione tres vulnerabilidades del OWASP Top 10 que podrían afectar su API y cómo las mitigaría.**

1. A01:2021 – Fallas de autenticación:

- Problema: contraseñas débiles o tokens inseguros.
- Mitigación: bcrypt para contraseñas, JWT con expiración y roles.

2. A05:2021 – Fallas de control de acceso (RBAC):

- Problema: un usuario accede a recursos que no le corresponden.
- Mitigación: middleware JWT + validación de roles antes de cada endpoint.

3. A03:2021 – Inyección de datos (SQL/NoSQL/XSS):

- Problema: enviar datos maliciosos que rompen el sistema o robar información.
- Mitigación: validación de entrada en los modelos (@NotBlank, @Email), consultas parametrizadas y filtrado de contenido.

5. **¿En qué parte del modelo C4 se deben representar las capas o componentes de seguridad y por qué?**

Se puede presentar en el diagrama de componentes o en el diagrama de contenedores, ya que permite visualizar cómo la seguridad está integrada en cada componente y como se encarga de proteger cada capa.

6. **¿Qué buenas prácticas debe seguir al almacenar contraseñas y manejar tokens en su proyecto?**

Para las contraseñas; nunca guardarlas en texto plano, para los token JWT guardar solo en memoria o almacenamiento seguro del cliente, y usar una clave secreta fuerte y de expiración corta.

9. Conclusiones:

- Comprendimos la diferencia entre autenticación y autorización, la importancia de RBAC, la gestión segura de contraseñas y la protección contra vulnerabilidades comunes (OWASP Top 10).
- Se configuró los endpoints de registro y login, usando JWT, garantizando que solo usuarios autenticados puedan acceder a recursos protegidos.

- Los componentes relacionados con la seguridad (controladores de autenticación, filtros JWT, validaciones) se evidencian en el diagrama de Componentes y Containers, mostrando claramente cómo se protege cada parte del sistema.

10. Recomendaciones:

- No exponer claves ni tokens en el código ni repositorio público.
- Probar rutas protegidas antes de fusionar a develop.
- Documentar las decisiones de seguridad en README o en un anexo /docs/security_notes.md.

11. Evaluación

Criterio	2 – Logro Alto	1 – Logro Medio	0 – Bajo / Sin Evidencia
1. Implementación de autenticación y autorización (JWT/OAuth2)	Flujo completo, tokens válidos y rutas protegidas operativas.	Flujo parcial o errores de validación de token.	No implementa autenticación funcional.
2. Configuración de CORS y validaciones	Configuración correcta, verificada en pruebas.	Parcial o con advertencias en consola.	Sin configuración verificable.
3. Aplicación de principios OWASP Top 10	Checklist completo y medidas de mitigación documentadas.	Checklist parcial o sin evidencia de mitigación.	No evidencia revisión OWASP.
4. Actualización del modelo C4 (Container y Component)	Diagramas actualizados y coherentes con las modificaciones de seguridad.	Diagramas incompletos o sin claridad en los componentes de seguridad.	No presenta actualización del C4.
5. Documentación y entrega de evidencias	PDF y README completos con capturas y referencias a la implementación.	Entrega parcial o poco clara.	No entrega evidencias o sin documentación.

12. Bibliografía

- OWASP Foundation. (2023). OWASP Top 10 – Web Application Security Risks.
- Auth0. JWT Handbook. <https://auth0.com/learn/json-web-tokens>
- Spring Security / Django Auth / Express JWT docs.
- PlantUML / Mermaid Model C4 Reference.

13. Elaboración y Aprobación

Elaborado por	Edison L Coronel Romero Docente	
Aprobado por	Edison L Coronel Romero Director de Carrera	