

Integrantes: Leonardo Espinoza, Santiago Jiménez, Isabel Morocho

Informe analítico sobre vulnerabilidades comunes (OWASP Top 10) y medidas de mitigación aplicadas en el proyecto “Sistema FitSIL”

1. Resumen técnico del proyecto

El proyecto que se está llevando a cabo por nuestro equipo de desarrollo, trata de una aplicación que busca ser una ayuda para las personas que están empezando en el mundo del gimnasio. Con el fin de poder realizar este proyecto tenemos que utilizar 3 herramientas fundamentales, el entorno de desarrollo, el lenguaje de programación y el framework, después de discutir con el grupo hemos decidido utilizar los siguientes:

El entorno que hemos decidido utilizar para el desarrollo del proyecto es “Visual Studio Code”, debido a que estamos familiarizados con este IDE, lo que hará que el trabajo avance de una forma más rápida y fluida.

El lenguaje de programación que decidimos utilizar es el de java, pues nuestros últimos proyectos los hemos trabajado con este lenguaje, por lo que conocemos bien su sintaxis, lo que lo vuelve bastante cómodo.

Finalmente, el framework que hemos decidido utilizar es “Spring Boot”, debido a que es bastante intuitivo de utilizar y ahorra mucho tiempo de desarrollo, debido a que elimina la necesidad de configurar manualmente varias cosas como archivos XML o dependencias.

2. Identificación de al menos 5 vulnerabilidades del OWASP Top 10 (2021) relevantes para su backend

Para la implementación en nuestro backend hemos identificado las 5 primeras vulnerabilidades las cuales hemos considerado más importantes

- **A01:2021**-Pérdida del control de acceso
- **A02:2021**-Fallas criptográficas
- **A03:2021**-Inyección
- **A04:2021**-Configuración de Seguridad Incorrecta (Security Misconfiguration):
- **A05:2021**-Fallas de Identificación y Autenticación (Identification and Authentication Failures)

3. Descripción del riesgo y del posible impacto de cada vulnerabilidad en su sistema.

A01:2021 - Pérdida de Control de Acceso

Riesgos: generación de accesos no autorizados que podrían resultar en acceso a funcionalidades no disponibles, además de la visualización de información personal y recursos que no se deberían modificar. Incluye significativamente a los accesos de endpoints ya que deberían exigirse de manera restringida, omisión en la verificación de los roles y el uso de tokens invalidados para sacar privilegio de ello.

Riesgo en nuestro sistema: En el microservicio de gestión de usuarios se podría permitir que un usuario normal, tenga acceso a endpoints de administrador, esto generaría modificaciones a otros usuarios del sistema como en sus estadísticas/reportes

Posibles impactos en el sistema:

- Un usuario con intenciones maliciosas podría visualizar, modificar o incluso eliminar rutinas y datos de otros usuarios, llevando a la pérdida de privacidad y el compromiso de los datos.
- Escalar hasta las funciones de un administrador y modificar las configuraciones globales del sistema, dar acceso o roles y eliminar a otros usuarios.
- Fallo en nuestro sistema de microservicios que pueden comprometer el funcionamiento de otros microservicios en cadena, por ejemplo en estadísticas, este confía en los otros microservicios sin antes tener un control. Cuando se genera un token para un usuario normal y se reutiliza para otro servicio, se puede establecer un abuso de este criterio.

A02:2021-Fallas criptográficas:

Riesgo: Hace énfasis en el fallo de la protección de datos sensibles a través de mecanismos criptográficos. No únicamente se trata de usar cualquier cifrado, sino de implementar adecuadamente algoritmos seguros, gestión y claves protegidas y almacenamiento seguro.

Riesgo en nuestro sistema: En el servicio de gestión de usuarios si se usa un almacenamiento de contraseñas con texto plano o algoritmos inseguros, un atacante podría entrar fácilmente a la base de datos y sustraer todas las contraseñas de los usuarios, además en la comunicación de microservicios usuarios-estadísticas si esto no se utiliza mediante métodos HTTP o TLS los atacantes interceptarían los tokens o datos personales de los usuarios.

Posibles impactos en el sistema:

- Robos de identidad por parte de los atacantes para tener acceso a contraseñas y tokens, y entrar por otros usuarios o administradores
- Exposición de las rutinas, historial de ejercicios y métricas de las estadísticas, además de información personal de cada usuario, lo que repercute en la pérdida de confianza de los usuarios

A03:2021-Inyección:

Riesgo: las vulnerabilidades mediante inyección se generan cuando los datos no confiables se envían a intérprete como SQL, No SQL, comandos del sistema, etc, esto ofrece al atacante ejecutar código malicioso manipulando principalmente a la base de datos y tenga acceso a información no autorizada

Riesgo en nuestro sistema: en el módulo de gestión de ejercicios, existe la posibilidad de que un endpoint construya consultas dinámicas sin antes parametrizar o emplear filtros concatenados que admiten la inyección de SQL/JPQL

Posibles impactos en el sistema:

- Los atacantes ejecutan consultas arbitrarias hacia la base de datos que pueden traer consecuencias como el robo de información personal de los usuarios, rutinas de entrenamiento, estadísticas y reportes.

- Modificación o borrado de los datos, provocando fallas de corrupción de datos o interrupción indefinida en cuanto a la disponibilidad del servicio
- Existe una posibilidad de que se podría escalar a la ejecución de código del mismo sistema si la inyección de los atacantes permite acceder al sistema operativo, pero en Spring Boot esto es muy poco probable.

4. A04:2021-Configuración de Seguridad Incorrecta (Security Misconfiguration):

Riesgo: La posibilidad de tener configuraciones incorrectas tales como; permisos, cabeceras HTTP ausentes, documentación interna expuesta, y mostrar errores en producción crean grandes oportunidades de ataque.

Riesgo en nuestro sistema: En un entorno con Spring Boot y microservicios, podemos tener que uno o varios de los microservicios se encuentren de manera expuesta como “dev” (por ejemplo, actuators habilitados), también que los endpoints estén públicos o que el servidor de base de datos acepte conexiones remotas sin tener un control de este, además que no se configuren adecuadamente los CORS.

Posibles impactos en el sistema

- Los atacantes tienen acceso a endpoints de administración, modificación de configuraciones y recabar información del sistema y explotar servicios que no se encuentren protegidos debidamente
- En caso de detectar una mal configuración se pueden realizar ataques del tipo inyección, SSRF y dar acceso mediante puertas traseras

5. A05:2021-Fallas de Identificación y Autenticación (Identification and Authentication Failures):

Riesgo: en esta categoría se encuentran fallas en cuanto a mecanismos de identificación como, por ejemplo; enumeración de usuarios, autenticación por contraseñas débiles, sesiones ineficientes, generación de tokens incorrectamente emitidos, etc.

Riesgo en nuestro sistema: en el servicio de autenticación y gestión de usuarios se puede tener contraseñas que están almacenadas sin un hash seguro, posibilidad de tener un logueo sin un máximo de intentos, tokens JWT sin un tiempo de caducidad, reutilización de tokens antiguamente generados o las existencias de mecanismos que permitan resetear contraseñas de otros usuarios.

Posibles impactos en el sistema

- Un usuario malintencionado podría robar credenciales de otros usuarios y recabar información como rutinas estadísticas y otros datos que no le pertenecen
- Las sesiones no autenticadas podrían generarse indefinidamente, lo que conlleva a intentos de suplantación de identidad
- Para sistemas con diferentes roles, es importante la autenticación como administrador ya que esto implica que tendrá un dominio total del entorno, por lo tanto, se deberá tener un control de qué usuarios pueden desarrollar esta función.

4. Medidas de mitigación implementadas o planificadas, explicando cómo se aplicaron en el código o la configuración.

1. A01:2021: PÉRDIDA DE CONTROL DE ACCESO

Esta vulnerabilidad se presenta cuando usuarios pueden tener acceso a funciones que no puede realizar tal como es el caso de un administrador

En nuestro proyecto de spring implementamos como medida de seguridad un filtro denominado SecurityConfig que establece los permisos de acceso a cada endpoint.

- **Spring Security para definir reglas de acceso**

```
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    http
        .csrf().disable()
        .cors().and()
        .authorizeHttpRequests(auth -> auth
            // Rutas públicas para usuarios normales
            .requestMatchers("/usuarios/registro", "/usuarios/login").permitAll()
            // Rutas públicas para autenticación general (si tienes)
            .requestMatchers("/auth/**").permitAll()
            // Registro de administradores solo para admins
            .requestMatchers("/administradores/registro").hasRole("ADMINISTRADOR")
            // Cualquier otra ruta requiere autenticación
            .anyRequest().authenticated()
        )
}
```

En este archivo denominado SecurityConfig se determina un filtro de seguridad unificado que establece qué rutas pueden ser accedidas sin restricción y aquellas que necesitan de autenticación o rol específico.

Gracias a esto podemos establecer que solo los administradores pueden registrar otros administradores, los usuarios podrán registrarse e iniciar sesión libremente.

- **Autenticación basada en JWT (JSON Web Token)**

Creamos un archivo JwtService que nos ayudará a generar un token que consta del correo y su rol en específico el cual se utilizara para dar una validación de acceso a cada petición.

```
// Genera token con correo y rol
public String generarToken(String correo, String rol) {
    return Jwts.builder()
        .setSubject(correo)
        .claim("rol", rol)
        .setIssuedAt(new Date())
        .setExpiration(new Date(System.currentTimeMillis() + EXPIRATION_TIME))
        .signWith(SignatureAlgorithm.HS256, SECRET_KEY)
        .compact();
}
```

El token incluye una fecha de expiración de 1 hora y se firma en un algoritmo HS256 usando una clave secreta que radica en un archivo .env

- **Validación del token en cada solicitud**

Se añade un filtro de seguridad encargado de interceptar todas las peticiones HTTP y valida el token antes de permitir el acceso

```
if (username != null && SecurityContextHolder.getContext().getAuthentication() == null) {
    UserDetails userDetails = userDetailsService.loadUserByUsername(username);

    if (jwtService.validarToken(jwt, userDetails.getUsername())) {
        String rol = jwtService.extraerRol(jwt);
        List<SimpleGrantedAuthority> authorities = List.of(new SimpleGrantedAuthority("ROLE_" + rol));

        UsernamePasswordAuthenticationToken authToken =
            new UsernamePasswordAuthenticationToken(
                userDetails,
                null,
                authorities
            );

        SecurityContextHolder.getContext().setAuthentication(authToken);
    }
}
```

Con esta implementación se garantiza que el token es de un usuario previamente autenticado en el sistema, el rol que se presenta en el token se aplica dinámicamente por el contexto de seguridad y si el token no es válido o ha expirado su tiempo se recibe una petición 403.

2. A02:2021: FALLAS CRIPTOGRÁFICAS

Se generan cuando las contraseñas, claves o tokens se guardan o transmite sin antes establecer un método seguro de cifrado

- **Cifrado de contraseñas con BCrypt**

Aquí las contraseñas que escribimos se cifran antes de ser almacenadas, usando el encoder que hemos definido en el archivo SecurityConfig.

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

Cuando un usuario se está logeando, el sistema valida las credenciales haciendo una comparación entre la contraseña que el usuario escribe con su versión cifrada.

```
// Validar contraseña con BCrypt
if (!passwordEncoder.matches(contrasenia, admin.getContrasenia())) {
    return ResponseEntity.status(HttpStatus.UNAUTHORIZED)
        .body(Map.of("error", "Contraseña incorrecta"));
}
```

Con ByCrypt las contraseñas ingresadas no se almacenarán en texto plano, además en cada hash se incluye una salida aleatoria por lo que estaremos dificultando los ataques por fuerza bruta.

- **Firma segura de tokens JWT**

Los tokens Jwt se firman con una clave privada y no se almacenan en una base de datos, con ello garantizamos autenticidad e integridad.

```
@Value("${jwt.secret}")
private String secretKey;
```

```
sistema-FitSIL > .env
1  JWT_SECRET=claveSuperSeguraYAleatoria1234567890
2
```

Con esto la integridad y autenticidad del token está garantizada, así se evita la falsificación o cambios maliciosos.

- **Eliminación de datos sensibles en las respuestas**

En los controladores de autenticación y registro, el sistema se encarga de evitar mostrar datos sensibles como las contraseñas en archivos JSON

```
// Validar contraseña con BCrypt
if (!passwordEncoder.matches(contrasenia, admin.getContrasenia())) {
    return ResponseEntity.status(HttpStatus.UNAUTHORIZED)
        .body(Map.of("error", "Contraseña incorrecta"));
}

// Generar token JWT válido incluyendo el rol del administrador
String token = jwtService.generarToken(correo, admin.getRol().name());

Map<String, Object> response = new HashMap<>();
response.put("mensaje", "Inicio de sesión exitoso");
response.put("correo", correo);
response.put("rol", admin.getRol().toString());
response.put("token", token);

return ResponseEntity.ok(response);
}
```

```
@PutMapping("/perfil")
public ResponseEntity<Usuario> actualizar(
    @RequestParam String email,
    @RequestHeader("Authorization") String authHeader,
    @RequestBody Usuario datos) {

    String token = authHeader.replace("Bearer ", "");
    String correoToken = jwtService.obtenerCorreoDesdeToken(token);

    if (!correoToken.equals(email)) {
        return ResponseEntity.status(403).build();
    }

    Usuario actualizado = usuarioService.actualizarPerfil(email, datos);
    return ResponseEntity.ok(actualizado);
}
```

```
// Crear administrador
@PostMapping("/registro")
public ResponseEntity<Administrador> registrar(@RequestBody Administrador admin) {
    admin.setContrasenia(passwordEncoder.encode(admin.getContrasenia()));
    return ResponseEntity.ok(adminService.registrarAdmin(admin));
}
```

Solo se envían el correo, el rol y el token JWT, cumpliendo con el principio de mínima exposición de datos sensibles.

3. A03:2021-INYECCIÓN

Son las entradas de usuario mal validadas que podrían inyectar SQL o JS, afectando la base de datos o la respuesta de la API.

En el código se validó los datos de entrada con Bean Validation

Casos de prueba de inyección:

- **Caso 1: Correo Malicioso:** Para este caso se validó estrictamente el formato del correo y se evita aceptar valores que contengan operadores o código malicioso.

En el código se aplicó en la entidad persona, la cual hace herencia a usuario,

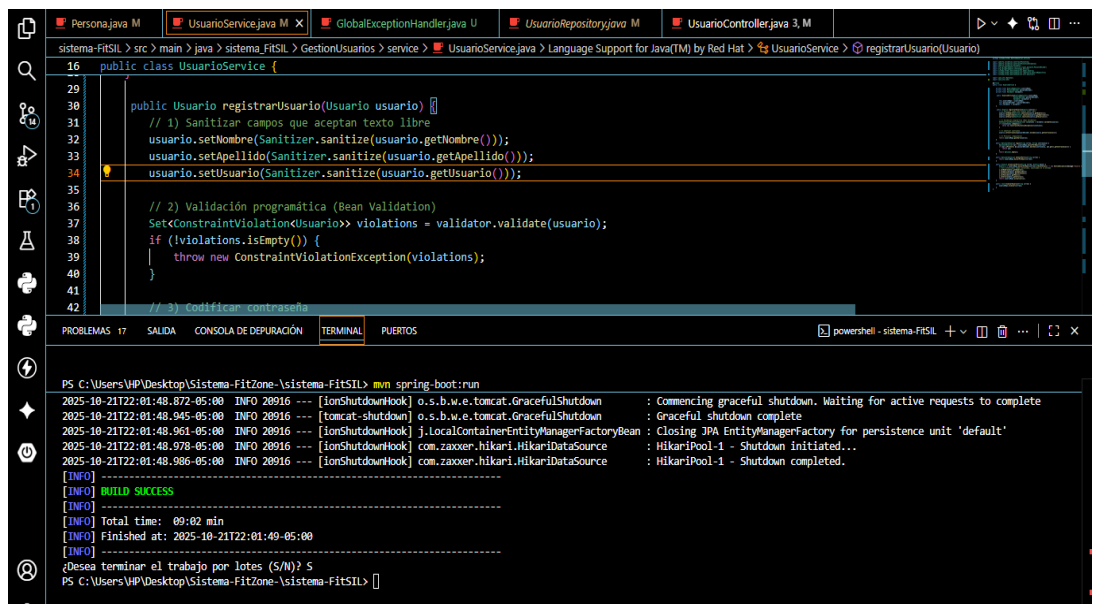

```

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 e
33 @Pattern(regexp = "^[a-zA-Z0-9_]+$", message = "Usuario sol
34 private String usuario;

```

En el UsuarioController se usa @Valid @RequestBody para activar la validación automática.

Y en UsuarioService se valida de forma programática con el Validator antes de persistir:



```

16 public class UsuarioService {
29
30 // 1) Sanitizar campos que aceptan texto libre
31 usuario.setNombre(Sanitizer.sanitize(usuario.getNombre()));
32 usuario.setApellido(Sanitizer.sanitize(usuario.getApellido()));
33 usuario.setUsuario(Sanitizer.sanitize(usuario.getUsuario()));
34
35 // 2) Validación programática (Bean Validation)
36 Set<ConstraintViolation<Usuario>> violations = validator.validate(usuario);
37 if (!violations.isEmpty()) {
38     throw new ConstraintViolationException(violations);
39 }
40
41 // 3) Codificar contraseña
42

```

```

PS C:\Users\VP\Desktop\Sistema-FitZone\Sistema-FitSIL> mvn spring-boot:run
2025-10-21T22:01:48.872-05:00 INFO 20916 --- [ionShutdownHook] o.s.b.w.e.tomcat.GracefulShutdown : Commencing graceful shutdown. Waiting for active requests to complete
2025-10-21T22:01:48.945-05:00 INFO 20916 --- [tomcat-shutdown] o.s.b.w.e.tomcat.GracefulShutdown : Graceful shutdown complete
2025-10-21T22:01:48.961-05:00 INFO 20916 --- [ionShutdownHook] j.localContainerEntityManagerFactoryBean : Closing JPA EntityManagerFactory for persistence unit 'default'
2025-10-21T22:01:48.978-05:00 INFO 20916 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown initiated...
2025-10-21T22:01:48.986-05:00 INFO 20916 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown completed.
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 00:02 min
[INFO] Finished at: 2025-10-21T22:01:49-05:00
[INFO] -----
¿Desea terminar el trabajo por lotes (S/N)? S
PS C:\Users\VP\Desktop\Sistema-FitZone\Sistema-FitSIL>

```

- **Caso 2: Usuario Malicioso:** se restringió la forma del nombre de usuario a caracteres seguros y longitudes razonables; además se sanitiza y valida antes de guardar.

```

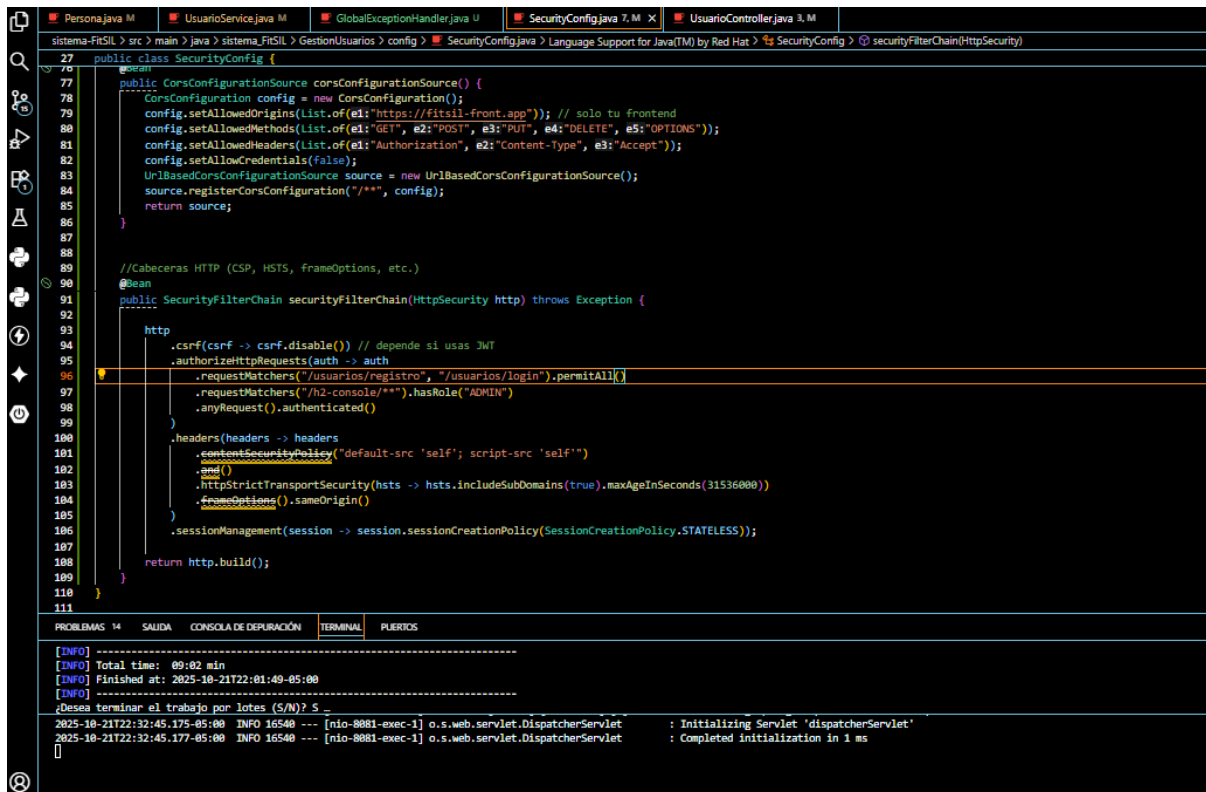
@NotBlank(message = "El nombre de usuario es obligatorio")
@Size(min = 3, max = 20, message = "El usuario debe tener entre 3 y 20 caracteres")
@Pattern(regexp = "^[a-zA-Z0-9_]+$", message = "Usuario solo puede contener letras, números y guiones bajos")
private String usuario;

```


Estas medidas impiden que valores maliciosos lleguen al repositorio y que se almacenen o se usen para manipular lógica (reduce riesgo de SQL/logic injection y XSS).

4. A04:2021-CONFIGURACIÓN DE SEGURIDAD INCORRECTA

En el proyecto se fortaleció la seguridad de la aplicación configurando correctamente el archivo de seguridad (SecurityConfig.java).



```

27 public class SecurityConfig {
28     @Bean
29     public CorsConfigurationSource corsConfigurationSource() {
30         CorsConfiguration config = new CorsConfiguration();
31         config.setAllowedOrigins(List.of("https://fitsil-front.app")); // solo tu frontend
32         config.setAllowedMethods(List.of("GET", "POST", "PUT", "DELETE", "OPTIONS"));
33         config.setAllowedHeaders(List.of("Authorization", "Content-Type", "Accept"));
34         config.setAllowCredentials(false);
35         UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
36         source.registerCorsConfiguration("/**", config);
37         return source;
38     }
39
40     //Cabeceras HTTP (CSP, HSTS, frameOptions, etc.)
41     @Bean
42     public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
43         http
44             .csrf(csrf -> csrf.disable()) // depende si usas JWT
45             .authorizeHttpRequests(auth -> auth
46                 .requestMatchers("/usuarios/registro", "/usuarios/login").permitAll()
47                 .requestMatchers("/h2-console/**").hasRole("ADMIN")
48                 .anyRequest().authenticated()
49             )
50             .headers(headers -> headers
51                 .contentSecurityPolicy("default-src 'self'; script-src 'self'")
52                 .and()
53                 .httpStrictTransportSecurity(hsts -> hsts.includeSubDomains(true).maxAgeInSeconds(31536000))
54                 .frameOptions().sameOrigin()
55             )
56             .sessionManagement(session -> session.sessionCreationPolicy(SessionCreationPolicy.STATELESS));
57         return http.build();
58     }
59 }

```

PROBLEMAS 14 SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS

```

[INFO] -----
[INFO] Total time: 02:02 min
[INFO] Finished at: 2025-10-21T22:01:49-05:00
[INFO] -----
¿Desea terminar el trabajo por lotes (S/N)? S
2025-10-21T22:32:45.175-05:00 INFO 16540 --- [nio-8081-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2025-10-21T22:32:45.177-05:00 INFO 16540 --- [nio-8081-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms

```

Gracias a esta configuración, se evitan accesos no autorizados a endpoints administrativos y se bloquea la carga del sistema en iframes o sitios externos.

Las rutas restringidas no se puedan acceder sin token o con rol incorrecto.

Cuando se inicia sesión correctamente el servidor te devuelve un **token JWT**

5. A05:2021-FALLAS DE IDENTIFICACIÓN Y AUTENTICACIÓN

- **Limitar intentos de login**

Si llegas al límite (MAX_INTENTOS), ya no dejará intentar más hasta que reinicie el sistema

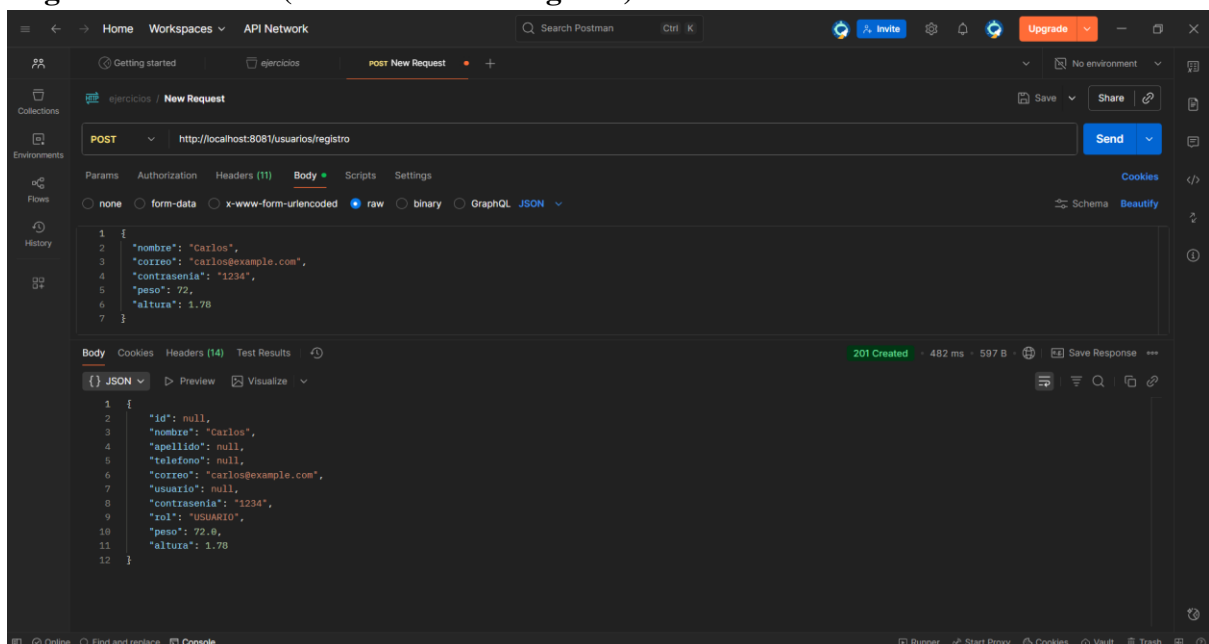
```

55
56
57 public Optional<Usuario> login(String correo, String contrasenia) {
58     int intentos = intentosFallidos.getDefault(correo, defaultValue:0);
59
60     // Verificar si está bloqueado
61     if (intentos >= MAX_INTENTOS) {
62         throw new RuntimeException("Usuario bloqueado por " + MAX_INTENTOS + " intentos fallidos");
63     }
64
65     Optional<Usuario> opt = usuarioRepo.buscarPorEmail(correo);
66
67     if (opt.isPresent() && passwordEncoder.matches(contrasenia, opt.get().getContrasenia())) {
68         // Login exitoso -> resetear contador
69         intentosFallidos.put(correo, value:0);
70         return opt;
71     } else {
72         // Incrementar intentos fallidos
73         intentosFallidos.put(correo, intentos + 1);
74         int restantes = MAX_INTENTOS - (intentos + 1);
75         if (restantes > 0) {
76             throw new RuntimeException("Contraseña incorrecta. Te quedan " + restantes + " intentos");
77         } else {
78             throw new RuntimeException("Usuario bloqueado por " + MAX_INTENTOS + " intentos fallidos");
79         }
80     }

```

5. Evidencias de verificación: capturas de pruebas, fragmentos de código o resultados de herramientas de análisis.

Registro de usuario (POST /usuarios/registro)



The screenshot shows a Postman interface for a new request. The URL is `http://localhost:8081/usuarios/registro` and the method is `POST`. The request body is raw JSON:

```

1 {
2   "nombre": "Carlos",
3   "correo": "carlos@example.com",
4   "contrasenia": "1234",
5   "peso": 72,
6   "altura": 1.78
7 }

```

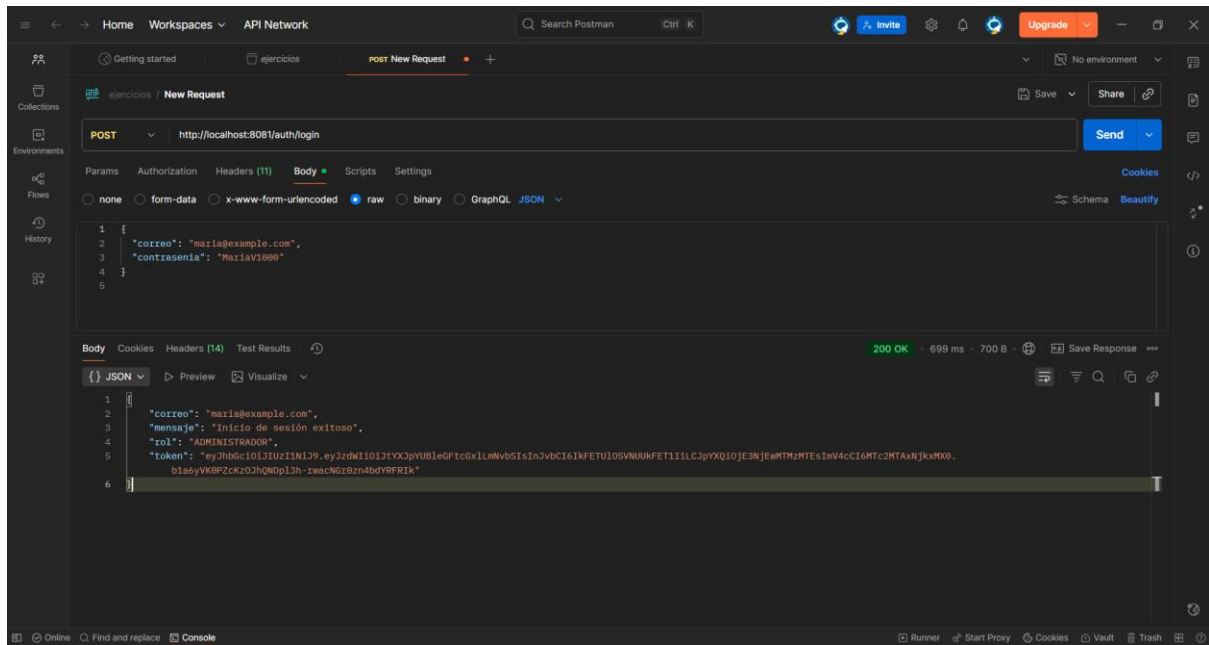
The response is a 201 Created status with a response body:

```

1 {
2   "id": null,
3   "nombre": "Carlos",
4   "apellido": null,
5   "telefono": null,
6   "correo": "carlos@example.com",
7   "usuario": null,
8   "contrasenia": "1234",
9   "rol": "USUARIO",
10  "peso": 72.0,
11  "altura": 1.78
12 }

```

Inicio de sesión (POST /usuarios/login)



POST http://localhost:8081/auth/login

Body (raw):

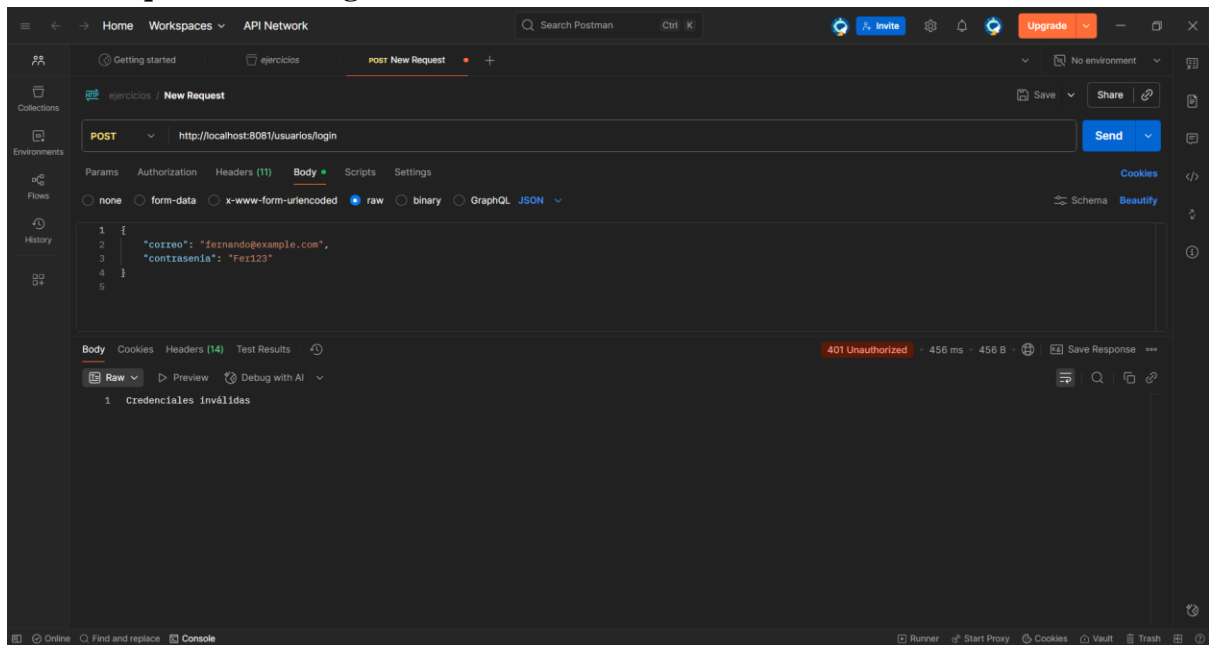
```
1 {
2   "correo": "maria@example.com",
3   "contrasenia": "MariaV1999"
4 }
5
```

Response: 200 OK - 699 ms - 700 B

Body (JSON):

```
1 {
2   "correo": "maria@example.com",
3   "mensaje": "Inicio de sesión exitoso",
4   "rol": "ADMINISTRADOR",
5   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IHYXZjYU81eGFTcGx1bWV5SisIn3VbCjE6MTU0OSVNUUkFET111L3pYXQ10jE3NjEwMTZMTesIMW4ccjE6MTc2MTAxNjksMK0.
6   b1a6yVKBpZcK20JhQNDp13h-twacNgz8zn4bdVRFRIk"
7 }
```

Usuarios que no están registrados en el sistema



POST http://localhost:8081/usuarios/login

Body (raw):

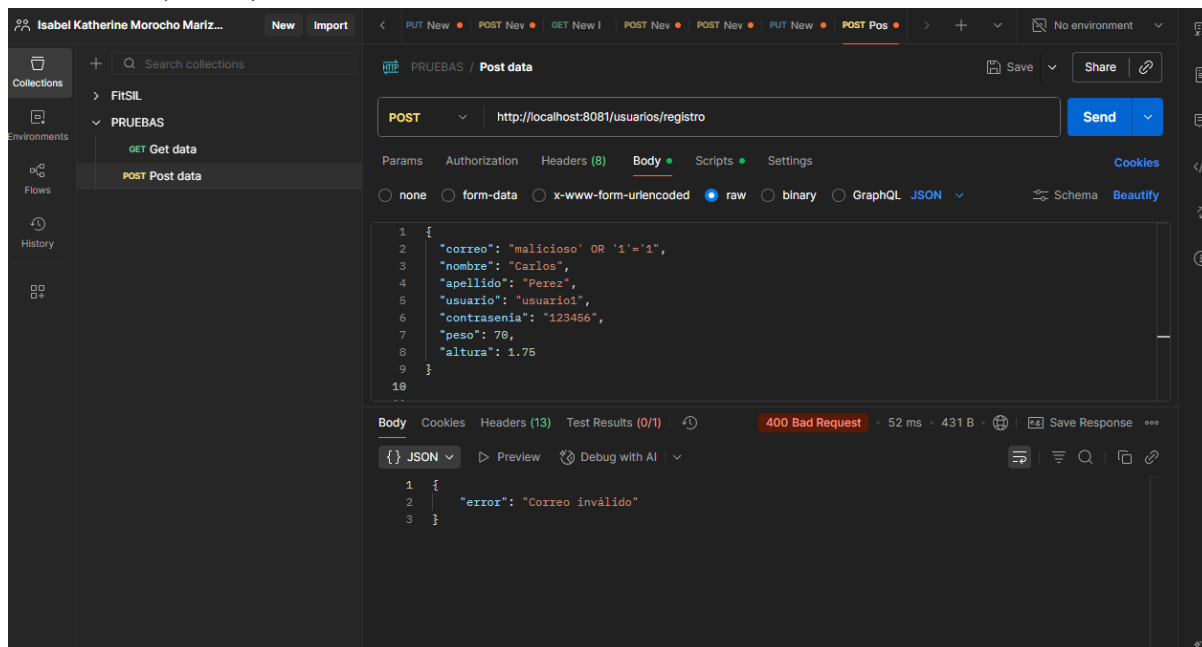
```
1 {
2   "correo": "fernando@example.com",
3   "contrasenia": "Fer123"
4 }
5
```

Response: 401 Unauthorized - 456 ms - 456 B

Body (Raw):

```
1 Credenciales inválidas
```

Correo Malicioso (POST)



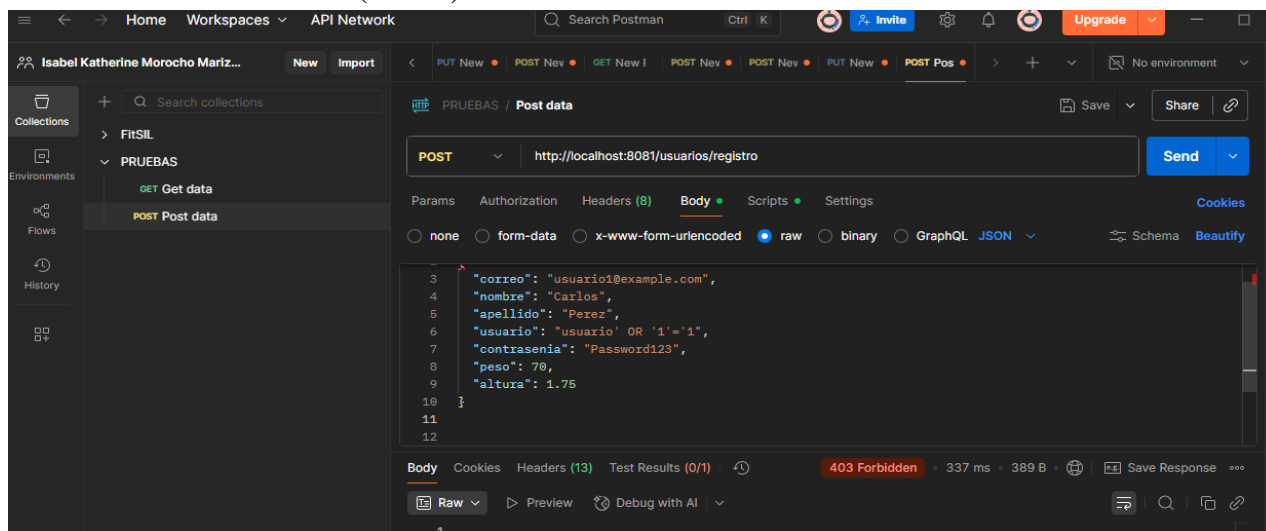
The screenshot shows a Postman interface with a POST request to `http://localhost:8081/usuarios/registro`. The request body is a JSON object:

```
1 {
2   "correo": "malicioso" OR '1'='1',
3   "nombre": "Carlos",
4   "apellido": "Perez",
5   "usuario": "usuario1",
6   "contrasenia": "123456",
7   "peso": 70,
8   "altura": 1.75
9 }
```

The response is a **400 Bad Request** with a status of 52 ms and 431 B. The response body is:

```
1 {
2   "error": "Correo inválido"
3 }
```

Usuario malicioso (POST)



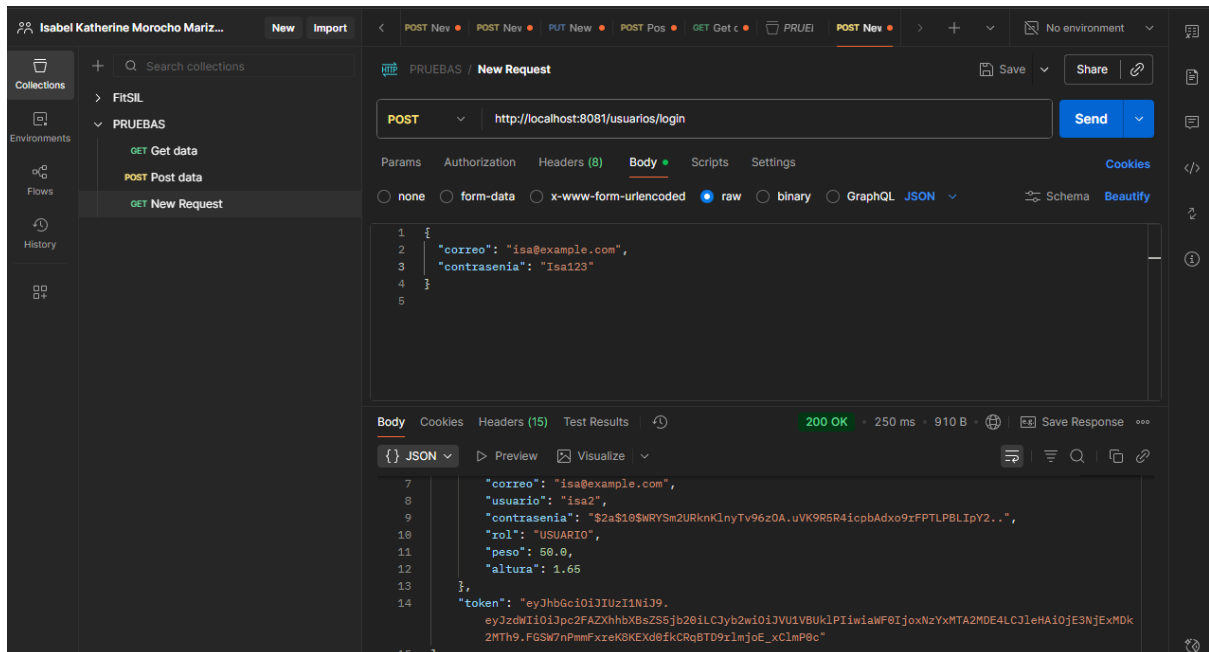
The screenshot shows a Postman interface with a POST request to `http://localhost:8081/usuarios/registro`. The request body is a JSON object:

```
3 "correo": "usuario1@example.com",
4 "nombre": "Carlos",
5 "apellido": "Perez",
6 "usuario": "usuario" OR '1'='1',
7 "contrasenia": "Password123",
8 "peso": 70,
9 "altura": 1.75
10 }
11
12
```

The response is a **403 Forbidden** with a status of 337 ms and 389 B. The response body is:

```
1
```

Inicio de sesión correcto (POST)



Isabel Katherine Morocho Mariz... New Import

PRUEBAS / New Request

POST http://localhost:8081/usuarios/login

Params Authorization Headers (8) Body Scripts Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Schema Beautify

```

1 {
2   "correo": "isa@example.com",
3   "contrasenia": "Isa123"
4 }
5

```

Body Cookies Headers (15) Test Results 200 OK 250 ms 910 B Save Response

JSON Preview Visualize

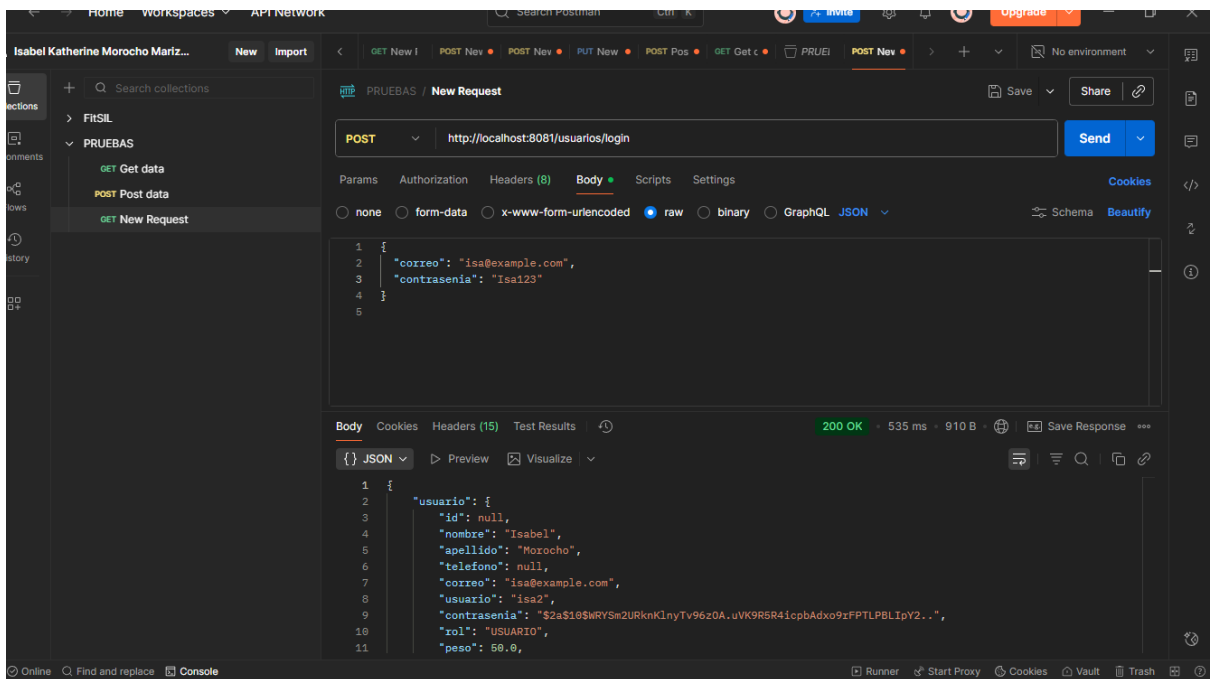
```

7 {
8   "correo": "isa@example.com",
9   "usuario": "isa2",
10  "contrasenia": "$2a$10$NRYSm2URknKlNyTv96z0A.uVK9R5R4icpbAdxo9zFPTLPBLIpY2..",
11  "rol": "USUARIO",
12  "peso": 50.0,
13  "altura": 1.65
14 }
15 "token": "eyJ3bGciOiJIUzI1NiJ9.eyJ3dWUiOiJpc2FAZXhhbXBsZS5jb28iLCJyb2wiOiJVVU1V8Uk1PIiwiaWF6IjoxNzYxMTA2MDE4LjEhAiojE3NjExMDk2NTh9.FGSW7nPmmFxeK0KEXd0fkCRqBDT9r1mjoE_xClmP8c"

```

Límite de intentos al iniciar sesión (POST)

- Inicio correcto de sesión:



Home Workspaces API Network

Isabel Katherine Morocho Mariz... New Import

PRUEBAS / New Request

POST http://localhost:8081/usuarios/login

Params Authorization Headers (8) Body Scripts Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Schema Beautify

```

1 {
2   "correo": "isa@example.com",
3   "contrasenia": "Isa123"
4 }
5

```

Body Cookies Headers (15) Test Results 200 OK 535 ms 910 B Save Response

JSON Preview Visualize

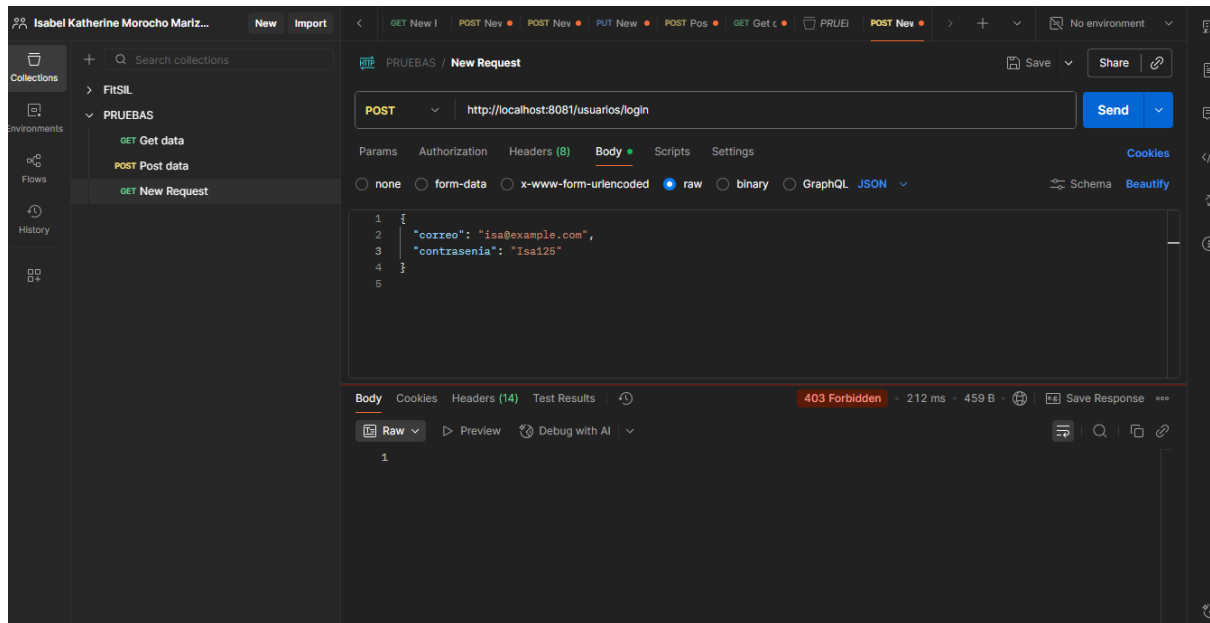
```

1 {
2   "usuario": {
3     "id": null,
4     "nombre": "Isabel",
5     "apellido": "Morocho",
6     "telefono": null,
7     "correo": "isa@example.com",
8     "usuario": "isa2",
9     "contrasenia": "$2a$10$NRYSm2URknKlNyTv96z0A.uVK9R5R4icpbAdxo9zFPTLPBLIpY2..",
10    "rol": "USUARIO",
11    "peso": 50.0,

```

Online Find and replace Console Runner Start Proxy Cookies Vault Trash

- **Máximo de intento de inicio de sesión:**



6. Reflexión personal: qué aprendió sobre seguridad durante esta unidad y cómo mejoraría el diseño futuro del sistema.

La seguridad en el sistema FitSIL, es muy importante ya que nos hace tomar conciencia de como errores comunes pueden comprometer todo el sistema, hasta incluso funcionalidades básicas si no están implementadas correctamente pueden verse vulnerables ante un atacante.

La validación de datos, el control de acceso, la autenticación segura y la gestión de errores son esenciales que, si se descuidan, pueden comprometer no solo la información de los usuarios, sino también la integridad y disponibilidad del sistema completo.

Esta actividad nos llevo a conocer la importancia de adoptar una mentalidad preventiva en lugar de reactiva: anticipar los posibles riesgos antes de que ocurran, por otra parte para el futuro, mejoraríamos el diseño del sistema aplicando principios de seguridad por diseño, implementando pruebas de penetración tempranas y asegurando que cada capa de la aplicación mantenga buenas prácticas de cifrado, validación y control de privilegios.

Bibliografía:

[1]“Inicio - OWASP Top 10:2021,” *owasp.org*. <https://owasp.org/Top10/es/>