



Módulo 4.- Seguridad en Aplicaciones Back- End Java

- Principios de seguridad en aplicaciones web.
- Introducción a Spring Security.
- Autenticación y autorización.
- Seguridad basada en tokens (JWT).
- Prácticas de hardening y validación de entradas.

Módulo 4.- Seguridad en Aplicaciones Back-End Java

En este módulo abordaremos cómo proteger nuestras aplicaciones Java Back-End, con técnicas modernas de seguridad, utilizando Spring Security y JWT.



Módulo 4.- Seguridad en Aplicaciones Back-End Java

Principios de seguridad en aplicaciones web.

1. Principios Fundamentales

- **Confidencialidad:** Proteger los datos de accesos no autorizados (ej: cifrado).
- **Integridad:** Garantizar que los datos no sean modificados sin control (ej: firmas digitales, checksums).
- **Disponibilidad:** Asegurar que los servicios estén siempre operativos (ej: protección contra DoS).
- **Autenticación:** Verificar la identidad del usuario.
- **Autorización:** Determinar qué puede hacer un usuario autenticado.
- **Auditoría:** Registrar actividades para trazabilidad.



Módulo 4.- Seguridad en Aplicaciones Back-End Java

Principios de seguridad en aplicaciones web.

🚨 Riesgos comunes ([según OWASP Top 10](#)):

- **Inyección (SQL, NoSQL, etc.)**
- **Exposición de datos sensibles**
- **Fallos en la autenticación**
- **Mala gestión de sesiones**
- **Cross-Site Scripting (XSS)**




Módulo 4.- Seguridad en Aplicaciones Back-End Java

Principios de seguridad en aplicaciones web.

Inyección (SQL, NoSQL, etc.)

Permite al atacante insertar comandos maliciosos en consultas a bases de datos para leer, modificar o eliminar datos.

 *Ejemplo:* `1' OR '1'='1` puede burlar un login si no se usa consulta parametrizada.

Exposición de datos sensibles

Ocurre cuando datos privados (como contraseñas, tarjetas o RUT) se almacenan o transmiten sin cifrado adecuado.

 *Ejemplo:* enviar contraseñas en texto plano por HTTP en vez de HTTPS.

Fallos en la autenticación

Errores en el proceso de verificación de identidad permiten que atacantes accedan como otros usuarios.

 *Ejemplo:* permitir contraseñas débiles o no bloquear múltiples intentos fallidos.

Módulo 4.- Seguridad en Aplicaciones Back-End Java

Principios de seguridad en aplicaciones web.

Mala gestión de sesiones

La sesión del usuario no se protege bien, permitiendo que un atacante la robe o la reutilice.

 *Ejemplo:* IDs de sesión predecibles o no caducar tokens después de logout.

Cross-Site Scripting (XSS)

El atacante inyecta código malicioso (como JavaScript) en una página que ven otros usuarios.

 *Ejemplo:* un mensaje de comentario con `<script>` que roba cookies.

Módulo 4.- Seguridad en Aplicaciones Back-End Java

Introducción a Spring Security.

2. ¿Qué es Spring Security?

Spring Security es un **framework robusto y extensible para proteger aplicaciones Spring**, que maneja autenticación, autorización y protección contra ataques comunes (CSRF, XSS, etc.).

3. Componentes clave

Componente	Descripción
SecurityFilterChain	Define reglas de seguridad para URLs y métodos HTTP
AuthenticationManager	Valida credenciales del usuario
UserDetailsService	Carga los datos del usuario desde la base de datos
PasswordEncoder	Codifica/decodifica contraseñas (Ej: BCrypt)

CSRF: Es un ataque que engaña al navegador de un usuario autenticado para que realice **acciones no autorizadas** en una aplicación en la que está logueado.

Ejemplo: Estando logueada en tu banca online y alguien te envía un correo con un enlace fraudulento.



Módulo 4.- Seguridad en Aplicaciones Back-End Java

Autenticación y autorización.

4. Autenticación

Proceso de **verificar la identidad** (login).

Spring Security por defecto ofrece:

- Formulario de login
- Login básico con HTTP Basic
- Customización con JWT, OAuth, LDAP, etc.

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http
        .authorizeHttpRequests(auth -> auth.anyRequest().authenticated())
        .httpBasic();
    return http.build();
}
```



Módulo 4.- Seguridad en Aplicaciones Back-End Java

Autenticación y autorización.

5. Autorización

Define **qué recursos puede acceder un usuario autenticado**.


Ejemplo:

```
.authorizeHttpRequests(auth -> auth
    .requestMatchers("/admin/**").hasRole("ADMIN")
    .requestMatchers("/api/**").authenticated()
)
```



Módulo 4.- Seguridad en Aplicaciones Back-End Java

Autenticación y autorización.

 Roles y usuarios en memoria (para pruebas):

```
@Bean
public UserDetailsService users() {
    UserDetails user = User.withUsername("ingrid")
        .password(passwordEncoder().encode("1234"))
        .roles("USER")
        .build();
    return new InMemoryUserDetailsManager(user);
}
```



Módulo 4.- Seguridad en Aplicaciones Back-End Java

Seguridad basada en tokens (JWT).

¿Qué es un JWT?

- **JSON Web Token:** estándar para intercambio seguro de datos.
- Contiene **identidad del usuario, roles, y fecha de expiración.**

```
{  
  "sub": "ingrid",  
  "roles": ["USER"],  
  "exp": 1717029600  
}
```



Módulo 4.- Seguridad en Aplicaciones Back-End Java

Seguridad basada en tokens (JWT).

Flujo típico:

- Usuario se loguea → servidor genera un JWT
- El cliente guarda el token (en memoria o localStorage)
- El token se envía en cada request:
`Authorization: Bearer <token>`
- El servidor valida el JWT y extrae los datos

Ventajas:

- Stateless (no guarda sesión en el servidor)
- Escalable
- Compatible con microservicios



Módulo 4.- Seguridad en Aplicaciones Back-End Java

Prácticas de hardening y validación de entradas.

Hardening:

- **Desactivar endpoints innecesarios**
- **Configurar correctamente CORS y CSRF**
- **Evitar mensajes de error explícitos**
- **Actualizar dependencias de seguridad**

Validación de entradas:

- Usar `@Valid`, `@Size`, `@NotNull`, etc.
- Validar del lado del cliente y del servidor
- Evitar entrada directa de parámetros en SQL o rutas

```
@PostMapping("/registro")
public ResponseEntity<?> registrar(@Valid @RequestBody UsuarioDTO dto) {
    ...
}
```



Módulo 4.- Seguridad en Aplicaciones Back-End Java

Flujo típico de autenticación con JWT

- 1 El usuario se autentica (envía credenciales: usuario y contraseña)
- 2 El servidor verifica las credenciales y genera un token JWT
- 3 El cliente guarda el JWT (en localStorage o memoria)
- 4 En cada request, el cliente envía:
Authorization: Bearer <token>
- 5 El servidor valida el token (firma, expiración, roles)
- 6 Si es válido → acceso permitido. Si no → acceso denegado.



Módulo 4.- Seguridad en Aplicaciones Back-End Java

Herramientas complementarias y escenarios reales

Contenido para bullets:

- **Spring Authorization Server:** Solución oficial de Spring para OAuth2 y JWT.
- **Keycloak:** Servidor de identidad Open Source que gestiona usuarios, roles y tokens.
- **Auth0:** Plataforma SaaS para autenticación segura con soporte para JWT y SSO.
- **JWT.io:** Herramienta para decodificar, verificar y entender tokens JWT.



Módulo 4.- Seguridad en Aplicaciones Back-End Java

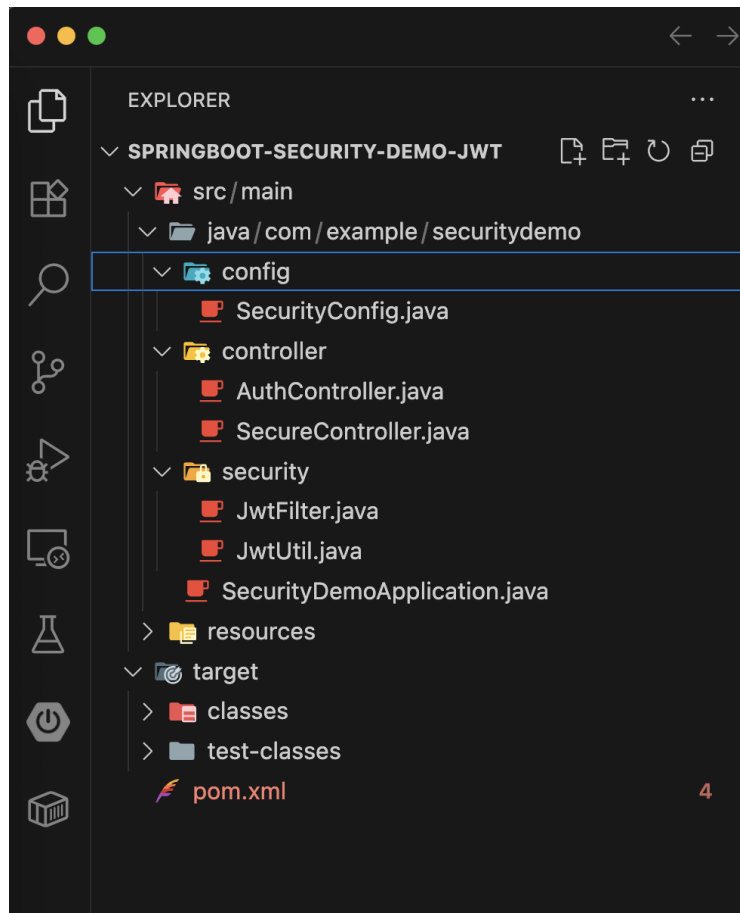


<https://github.com/Joselota/Relatorias/tree/main/CursoJavaAvanzado/springboot-security-demo-jwt>



Módulo 4.- Seguridad en Aplicaciones Back-End Java

Ejemplo




Módulo 4.- Seguridad en Aplicaciones Back-End Java

Ejemplo


1. SecurityConfig.java

 Ubicación: config/


 **Propósito:** Configura la seguridad de la aplicación.


2. AuthController.java

 Ubicación: controller/

 **Propósito:** Proporciona endpoints públicos de autenticación.

3. SecureController.java

 Ubicación: controller/

 **Propósito:** Proteger recursos solo accesibles con JWT.


4. JwtUtil.java

 Ubicación: security/

 **Propósito:** Crear, firmar y validar tokens JWT.

5. JwtFilter.java

 Ubicación: security/

 **Propósito:** Intercepta las peticiones entrantes y valida el JWT.



Módulo 4.- Seguridad en Aplicaciones Back-End Java

1. POST /api/auth/login (con usuario y contraseña)

Cliente (Postman / Frontend)

- Recibe credenciales
- Llama a → JwtUtil.java para generar token

AuthController.java

2. Responde con: Bearer <JWT>

3. Cliente guarda el token y hace nueva petición
GET /api/private/data con header Authorization: Bearer <JWT>

JwtFilter.java

- Intercepta la request
- Valida el JWT usando JwtUtil.java
- Si es válido:
 - crea objeto de autenticación
 - inyecta en SecurityContext

SecurityConfig.java

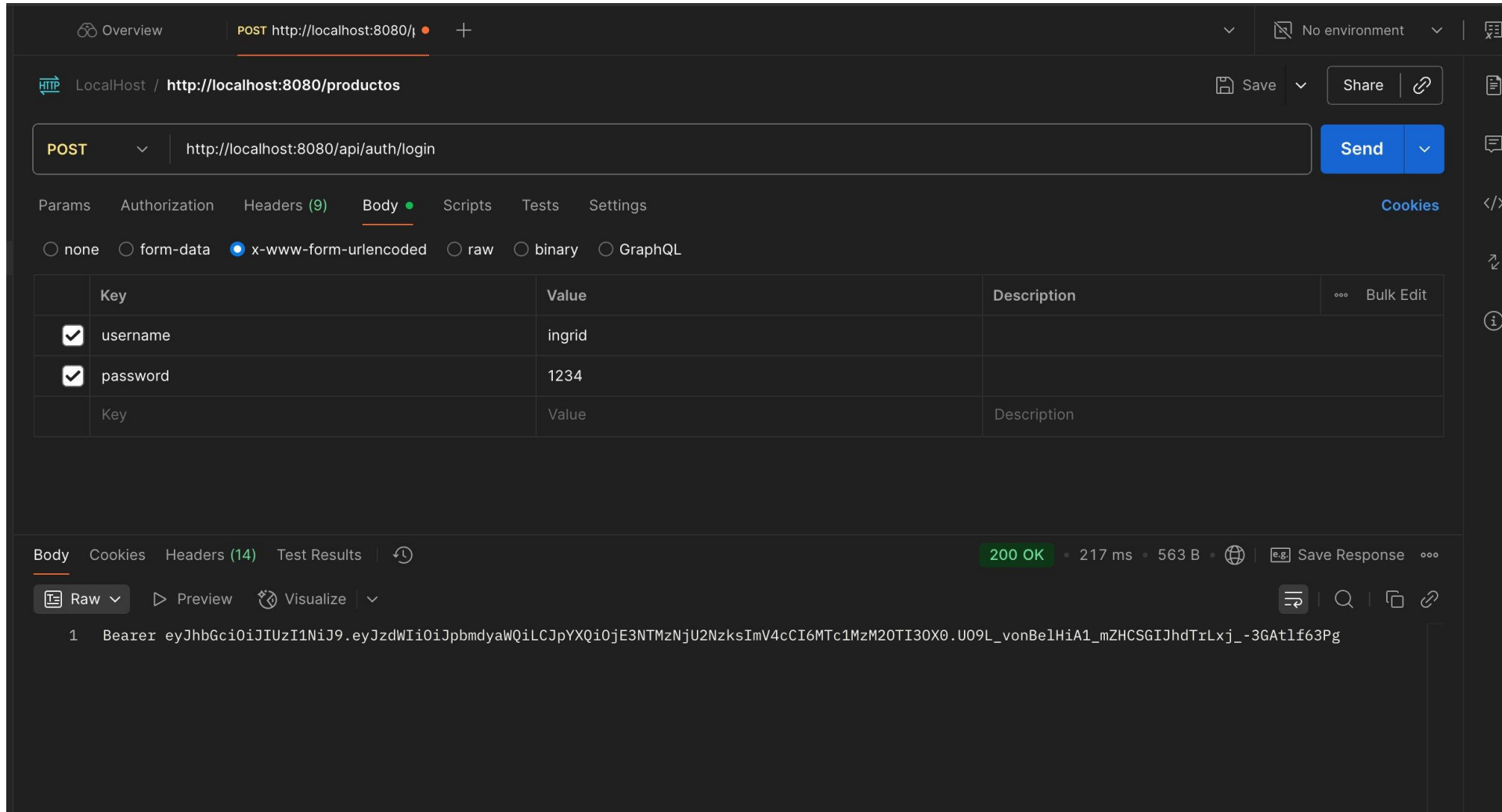
- Configura que /api/private/** requiera autenticación
- Permite acceso si SecurityContext tiene un usuario válido

SecureController.java

Devuelve los datos protegidos

Autorización

Módulo 4.- Seguridad en Aplicaciones Back-End Java



The screenshot displays a REST client interface with the following details:

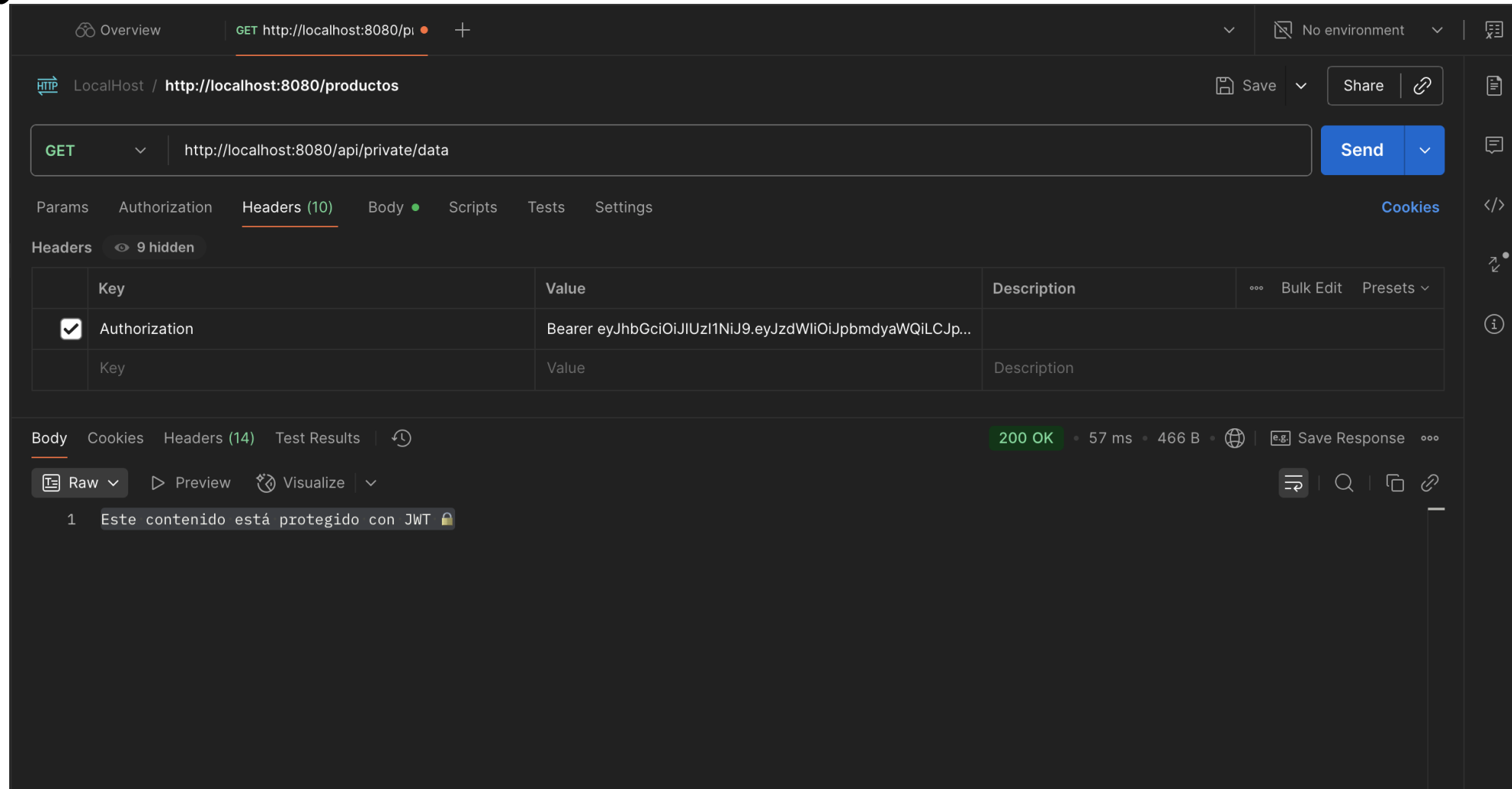
- Request:** A POST request to `http://localhost:8080/api/auth/login` using the `x-www-form-urlencoded` body type. The body contains the following data:

Key	Value	Description
<input checked="" type="checkbox"/> username	ingrid	
<input checked="" type="checkbox"/> password	1234	

- Response:** The response is a 200 OK status with a response time of 217 ms and a size of 563 B. The response body is shown in raw format as a Bearer token: `Bearer eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJpbmdyYWQiLCJpYXQiOiJlE3NTMzNjU2NzksImV4cCI6MTc1MzM2OTI3OX0.U09L_vonBelHiA1_mZHCSGIJhdTrLxj_-3GAtlf63Pg`.



Módulo 4.- Seguridad en Aplicaciones Back-End Java



Overview | GET http://localhost:8080/pi

LocalHost / http://localhost:8080/productos

GET http://localhost:8080/api/private/data

Params Authorization Headers (10) Body Scripts Tests Settings

Headers 9 hidden

	Key	Value	Description
<input checked="" type="checkbox"/>	Authorization	Bearer eyJhbGciOiJIUzI1NiJ9.eyJzdWliOiJpbmdyaWQiLCJp...	
	Key	Value	Description

Body Cookies Headers (14) Test Results

200 OK • 57 ms • 466 B • Save Response

Raw Preview Visualize

1 Este contenido está protegido con JWT









Módulo 4.- Seguridad en Aplicaciones Back-End Java



Cierre

¿Qué lograste?

-  Enviar credenciales → recibir JWT
-  Usar ese JWT en un header Authorization
-  Validar automáticamente el token con `JwtFilter.java`
-  Acceder a una ruta protegida si el token es válido
-  Obtener una respuesta segura:
"Este contenido está protegido con JWT  "

