Trabajo Final Integrador – Programación Segura

Trabajo Final Integrador - Programación Segura, en Java con Spring Boot

Objetivo

Este proyecto busca validar los conocimientos de desarrollo seguro en Java mediante el uso de Spring Boot, Spring Security, JWT y OAuth2. Además, forma parte del portfolio profesional del estudiante, demostrando su capacidad para construir APIs seguras, escalables y bien documentadas.

Estructura del Proyecto

El sistema implementa:

- CRUD de entidades: `Producto`, `Pedido`
- CRUD de seguridad: `UserSec` (usuario), `Role`, `Permission`
- Autenticación con JWT
- Autorización basada en roles y permisos
- Swagger UI para documentación
- Testing con Postman
- Integración con GitHub OAuth2
- Encriptado de contraseñas con BCrypt

Seguridad y Accesos

Roles:

- **ADMIN**: Acceso total a todos los endpoints.

- **USER**: Acceso de solo lectura a entidades de dominio (`GET /api/productos`, `/api/pedidos`).

- **CLIENTE**: Permisos definidos para operar con una entidad de usuario.

Permisos:

- **CREATE**

- **READ**

- **UPDATE**

- **DELETE**

Asignación por rol:

| Rol | CREATE | READ | UPDATE | DELETE |

|---|---|---|---|---|

| ADMIN | ✅ | ✅ | ✅ | ✅ |

| USER | | ✅ | | |

| CLIENTE | ✅(1) | ✅ | ✅(1) | |

(1): Solo sobre una entidad específica

Entidades principales

- **UserSec**: Usuario autenticado. Contiene `username`, `password` (encriptado), estados, y `Set<Role>`.

- **Role**: Nombre de rol y lista de permisos (`Set<Permission>`).

- **Permission**: Permiso específico (por ejemplo `READ`, `UPDATE`, etc.).

- **Producto / Pedido**: Entidades de prueba con relaciones (Pedido tiene muchos Productos).

JWT: Creación y Validación

Generación del token (clase `JwtUtils`)

- Firmado con `HS256` y una `PRIVATE_KEY` cargada por variable de entorno.

- Contiene:

- `sub` = username

- `authorities` = roles + permisos

- `exp` = expiración (5 horas)

- `iss`, `jti`, `nbf`, `iat`

Validación (filtro `JwtTokenValidator`)

- Valida firma, emisor y expiración

- Si es válido, lo registra en `SecurityContextHolder`

- Si no, bloquea el acceso

Autenticación y Autorización

- `POST /auth/login`: Devuelve JWT si usuario y contraseña son correctos.

- Filtro `JwtTokenValidator` valida todos los JWT entrantes.

- Control de acceso mediante:

- `@PreAuthorize("hasRole(...)")`

- `@PreAuthorize("hasAuthority(...)")`

- `HttpSecurity` en `SecurityConfig`

Testing

- Incluye colección Postman: `SecurityJWTITFPS.postman_collection.json`
- Archivo Word adjunto: **Secuencia de Testing TIFPS.docx**

🔧 Requisitos Técnicos

- Java 17
- Spring Boot 3.4.x
- MySQL 8.x
- Maven 3.8+
- JWT (Auth0)
- Swagger (SpringDoc OpenAPI)
- OAuth2 Client (GitHub)

Variables de Entorno

Estas deben definirse en tu entorno o archivo `.env`:

```
PRIVATE_KEY=clave_para_firmar_jwt
USER_GENERATOR=identificador_emisor_jwt
GH_CLIENT_ID=tu_client_id_github
GH_CLIENT_SECRET=tu_client_secret_github
SS_USER=centro8
```

SS_PASSWORD=1234

```

Instrucciones de Ejecución

1. Clonar el proyecto:

```bash
git clone https://github.com/tuusuario/tpfinalps.git
```

2. Configurar las variables de entorno

3. Ejecutar la aplicación:

```bash
mvn spring-boot:run
```

Endpoints de prueba

Método	URL	Requiere Token	Rol Necesario
POST	/auth/login	❌	-
GET	/api/users	✅	ADMIN
GET	/api/productos	✅	ADMIN / USER
GET	/api/pedidos	✅	ADMIN / USER
POST	/api/roles	✅	ADMIN
```

Usuarios de prueba

| Rol | Usuario | Contraseña | JWT necesario |

|---|---|---|---|

| ADMIN | admin | 1234 | ✅ |

| USER | user | 1234 | ✅ |

| CLIENTE | cliente | 1234 | ✅ |

Swagger & Docs

- Swagger UI: http://localhost:8080/swagger-ui.html

- Documentación API: http://localhost:8080/v3/api-docs

Colección de Postman

Archivo: `SecurityJWTITFPS.postman_collection.json` (incluye login, JWT, pruebas de roles/permisos, etc.)

Autor

Trabajo desarrollado por Maxi — TIF Seguridad y Programación Java — Centro 8.

Secuencia de Testing con Postman y Base de Datos

Secuencia de Testing mediante POSTMAN y BD del TIFPS

Generación del usuario ADMIN en la base de datos

1. Generamos el hash de la contraseña "1234" con BCrypt

Podemos usar este código Java para obtener el hash (podemos pegarlo en un método main):

```
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;


@SpringBootApplication
public class TpfinalpsApplication {


   public static void main(String[] args) {
      SpringApplication.run(TpfinalpsApplication.class, args);
      System.out.println(new BCryptPasswordEncoder().encode("1234"));
   }


}
```

Ejecutamos y copiamos el resultado, será algo como:

$2a$10$bp3JjuYtnQG/gwg6gdeE3O4vbhclC1m4plpUf4DKSzIdj2KmNu0dO

2. Insertamos el usuario en la base de datos

Supongamos que nuestra tabla de usuarios se llama users y tiene los siguientes campos:

id (auto-incremental)

username

password

enabled

account_not_expired

account_not_locked

credential_not_expired

Ejecutamos este SQL (reemplazamos el hash por el que generamos):

INSERT INTO users (username, password, enabled, account_not_expired,
account_not_locked, credential_not_expired)

VALUES ('centro8',
'$2a$10$bp3JjuYtnQG/gwg6gdeE3O4vbhclC1m4plpUf4DKSzIdj2KmNu0dO', 1, 1, 1, 1);

3. Asignamos el rol ADMIN

Supongamos que tenemos una tabla roles y una tabla intermedia user_roles:

-- Si el rol ADMIN no existe:

INSERT INTO roles (role) VALUES ('ADMIN');

-- Asociamos el usuario al rol (ajustamos los nombres de las columnas/tablas según nuestro modelo)

INSERT INTO user_roles (user_id, role_id)

VALUES (

  (SELECT id FROM users WHERE username = 'centro8'),

  (SELECT id FROM roles WHERE role = 'ADMIN')

);

4. Ahora sí podemos hacer login con centro8 y 1234 y obtener el JWT.

5. Creamos los permisos de READ, UPDATE, CREATE y DELETE (mediante bearer JWT) :

POST http://localhost:8080/api/permissions Authorization Bearer Token

en el Body Json

{

"permissionName": "READ" } ,

crearPermisoUpdate: lo que cambia es el Body Json

{

"permissionName": "UPDATE"

},

crearPermisoCreate: lo que cambia es el Body Json

{

"permissionName": "CREATE"

},

crearPermisoDelete: lo que cambia es el Body Json

{

"permissionName": "DELETE"

}

6. Asociamos permisos READ, UPDATE, CREATE y DELETE al rol ADMIN desde la BD (ver security.sql)

-- Asociaciones de permisos READ, UPDATE, CREATE Y DELETE al rol ADMIN

SELECT id FROM roles WHERE role = 'ADMIN';

SELECT id FROM permissions WHERE permission_name = 'READ';

SELECT id FROM permissions WHERE permission_name = 'UPDATE';

```
SELECT id FROM permissions WHERE permission_name = 'CREATE';
 SELECT id FROM permissions WHERE permission_name = 'DELETE';

-- Insertamos las asociaciones en la tabla de enlace

INSERT INTO roles_permissions (role_id, permission_id) VALUES (1, 1); -- ADMIN - READ


INSERT INTO roles_permissions (role_id, permission_id) VALUES (1, 2); -- ADMIN - UPDATE

INSERT INTO roles_permissions (role_id, permission_id) VALUES (1, 3);
-- ADMIN - CREATE

INSERT INTO roles_permissions (role_id, permission_id) VALUES (1, 4); -- ADMIN – DELETE


-- Verifica la asociación

SELECT * FROM roles_permissions WHERE role_id = 1;
```

7. Logueados como ADMIN podemos crear otro rol (USER) y usuario (vendedor)

crearRoleUser: POST Body JSON: {

```
  "role": "USER",
  "permissionsList": [
    { "id": 1 }, // READ
```

```
      { "id": 2 }, // UPDATE
      { "id": 3 }, // CREATE
      { "id": 4 }  // DELETE
    ]
}
```

crearUsuarioVendedor: POST   Body JSON: {

```
  "username": "vendedor",
  "password": "1234",
  "enabled": true,
  "accountNotExpired": true,
  "accountNotLocked": true,
  "credentialNotExpired": true,
  "rolesList": [
    {
      "id": 2 // ID del rol USER
    }
  ]
}
```

8. Logueados como ADMIN podemos crear otro rol (GUEST) y usuario (cliente)

crearRoleGuest: POST   Body JSON: {

```
  "role": "GUEST",
  "permissionsList": [
```

```
    {
      "id": 1 // ID del permiso READ ,
      }
  ]
}
```

crearUsuarioCliente: POST Body JSON: {

```
  "username": "cliente",
  "password": "1234",
  "enabled": true,
  "accountNotExpired": true,
  "accountNotLocked": true,
  "credentialNotExpired": true,
  "rolesList": [
    {
      "id": 3 // ID del rol GUEST
    }
```

9. Loguearse como vendedor y como cliente (para que me de sus tokens para los endpoints de pedido y producto)

10. Probar endpoints de producto y pedido. (1ro crear producto, antes de pedido)

11. : Probar Login con GitHub

Probamos ingresar a nuestro endpoint securizado desde el navegador y al aparecer el login veremos que ya tenemos la opción de logear con GitHub (localhost:8080/decirholasec)

Una vez hecho click para logear con GitHub y autenticados, vamos a poder dirigirnos a nuestro endpoint securizado y acceder sin problema alguno:

Nota: Esta autenticación que acabamos de hacer se guarda en una cookie. Si queremos eliminar la cookie de la sesión podemos hacerlo mediante F12 en el navegador (estando dentro de la url de la aplicación y ya logueados), luego dirigirnos al apartado cookies, identificar la cookie, hacer click derecho sobre ella y luego delete o eliminar. Luego de hacer esto nos va a pedir nuevamente nuestros datos de inicio de sesión.

12. Logout (sólo si ya esta logueado)

13. Link para ver documentación con Swagger: http://localhost:8080/swagger-ui/index.html