

Texto color Verde: Completado.

Texto color Naranja: comentario Mio.

Texto color Rojo: No completado.

## Implementación de Seguridad en una Aplicación Bancaria

**Objetivo:** Desarrollar una aplicación bancaria simple utilizando Spring Boot y Spring Security, implementando medidas de seguridad robustas.

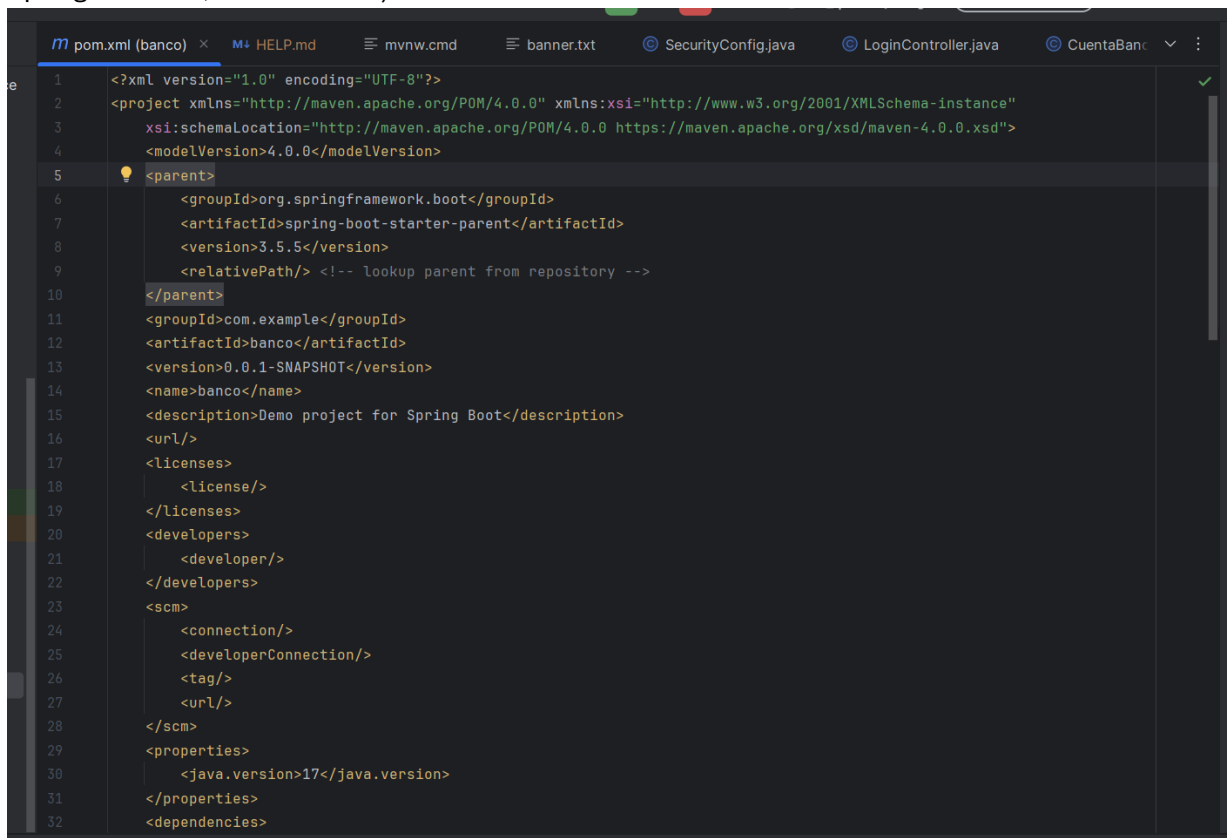
### Requisitos:

1. La aplicación debe tener tres tipos de usuarios: CLIENTE, EMPLEADO y ADMIN. //aplicado
2. Implementar autenticación basada en formulario. //aplicado
3. Configurar autorización basada en roles. //aplicado
4. Proteger contra ataques CSRF y XSS. //aplicado
5. Implementar gestión de sesiones segura. //aplicado
6. Integrar con Spring Data JPA para el almacenamiento de usuarios y cuentas bancarias. //solicitado en el punto 8 no supe aplicarlo.

### Tareas:

#### 1. Configuración inicial: //aplicado

- Crea un Proyecto Spring Boot con las dependencias necesarias (Spring Web, Spring Security, Spring Data JPA, H2 Database).



```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
4   <modelVersion>4.0.0</modelVersion>
5   <parent>
6     <groupId>org.springframework.boot</groupId>
7     <artifactId>spring-boot-starter-parent</artifactId>
8     <version>3.5.5</version>
9     <relativePath/> <!-- lookup parent from repository -->
10  </parent>
11  <groupId>com.example</groupId>
12  <artifactId>banco</artifactId>
13  <version>0.0.1-SNAPSHOT</version>
14  <name>banco</name>
15  <description>Demo project for Spring Boot</description>
16  <url/>
17  <licenses>
18    <license/>
19  </licenses>
20  <developers>
21    <developer/>
22  </developers>
23  <scm>
24    <connection/>
25    <developerConnection/>
26    <tag/>
27    <url/>
28  </scm>
29  <properties>
30    <java.version>17</java.version>
31  </properties>
32  <dependencies>
```

#### 2. Configuración de Spring Security: //aplicado

- Implementa una clase de configuración que extienda `WebSecurityConfigurerAdapter` Ya no se usa en las nuevas versiones - en Spring Boot 3 ya está deprecado `SecurityFilterChain`

```

14 public class SecurityConfig {
27     public DaoAuthenticationProvider authenticationProvider(CustomUserDetailsService userDetailsService,
33     }
34
35     // Configuración principal de seguridad web
36     @Bean
37     public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
38         http
39             // Definimos quienes pueden acceder a que rutas
40             .authorizeHttpRequests( AuthorizationManagerRequestMat... auth -> auth
41                 .requestMatchers( ...patterns: "/", "/home", "/publico", "/css/**", "/js/**").permitAll() // publico
42                 .requestMatchers( ...patterns: "/cliente/**").hasRole("CLIENTE") // solo cliente
43                 .requestMatchers( ...patterns: "/empleado/**").hasRole("EMPLEADO") // solo empleado
44                 .requestMatchers( ...patterns: "/admin/**").hasRole("ADMIN") // solo admin
45                 .anyRequest().authenticated() // todo lo demas requiere login
46             )
47
48             // CSRF habilitado (proteccion contra ataques de tipo Cross-Site Request Forgery)
49             .csrf(Customizer.withDefaults())
50
51             // Configuraciones de cabeceras de seguridad
52             .headers( HeadersConfigurer<HttpSecurity> headers -> headers
53                 // Política de seguridad de contenido (CSP) para scripts, estilos, fuentes, imagenes
54                 .contentSecurityPolicy( ContentSecurityPolicyConfig csp -> csp.policyDirectives(
55                     "default-src 'self'; " +
56                     "script-src 'self' 'unsafe-inline' https://cdn.jsdelivr.net https://cdnjs.cloudflare.c
57                     "style-src 'self' 'unsafe-inline' https://cdn.jsdelivr.net https://fonts.googleapis.co
58                     "font-src 'self' https://fonts.gstatic.com; " +
59                     "img-src 'self' data:; " +
60                     "object-src 'none'; " +
61                     "frame-ancestors 'none'; " +

```

Configura la autenticación basada en formulario.

```

// Definimos quienes pueden acceder a que rutas
.authorizeHttpRequests( AuthorizationManagerRequestMat... auth -> auth
    .requestMatchers( ...patterns: "/", "/home", "/publico", "/css/**", "/js/**").permitAll() // publico
    .requestMatchers( ...patterns: "/cliente/**").hasRole("CLIENTE") // solo cliente
    .requestMatchers( ...patterns: "/empleado/**").hasRole("EMPLEADO") // solo empleado
    .requestMatchers( ...patterns: "/admin/**").hasRole("ADMIN") // solo admin
    .anyRequest().authenticated() // todo lo demas requiere login
)

```

- Define las reglas de autorización para diferentes endpoints.  
No puedes acceder directamente a un usuario sin antes haberte registrado.
- http://localhost:8080/publico → acceso libre.
- http://localhost:8080/cliente/dashboard → pide login, entra con cliente / 1234.
- http://localhost:8080/empleado/dashboard → pide login, entra con empleado / 1234.
- http://localhost:8080/admin/dashboard → pide login, entra con admin / 1234.

### 3. Modelo de datos y repositorios: //aplicado

- Crea entidades para Usuario y CuentaBancaria.

```

1 package com.banco.banco.model;
2
3 import jakarta.persistence.*;
4 import java.util.HashSet;
5 import java.util.Set;
6
7 @Entity
8 @Table(name = "usuarios")
9 public class Usuario {
10
11     // Id unico de cada usuario, autogenerado por la base de datos
12     @Id
13     @GeneratedValue(strategy = GenerationType.IDENTITY)
14     private Long id;
15
16     // Nombre de usuario unico y obligatorio (para login)
17     @Column(nullable = false, unique = true)
18     private String username;
19
20     // Contraseña del usuario, obligatoria
21     @Column(nullable = false)
22     private String password;
23
24     // Rol del usuario: CLIENTE, EMPLEADO, ADMIN
25     @Column(nullable = false)
26     private String rol;
27
28     // Relacion con las cuentas bancarias de este usuario
29     // mappedBy indica que la relacion se maneja desde CuentaBancaria.usuario
30     // Cascade ALL: si se borra el usuario, se borran sus cuentas
31     // orphanRemoval true: si se quita una cuenta del Set, tambien se borra de la BD
32     @OneToOne(mappedBy = "usuario", cascade = CascadeType.ALL, orphanRemoval = true)

```

- Implementa repositorios JPA para estas entidades.

```

1 package com.banco.banco.repository;
2
3 import com.banco.banco.model.Usuario;
4 import org.springframework.data.jpa.repository.JpaRepository;
5 import java.util.Optional;
6
7 // Repositorio para manejar los Usuarios en la base de datos
8 // Extiende JpaRepository que ya trae todos los metodos CRUD (guardar, borrar, buscar, actualizar)
9 public interface UsuarioRepository extends JpaRepository<Usuario, Long> {
10
11     // Metodo para obtener todos los usuario en especifico
12     // Spring Data JPA lo interpreta automaticamente y hace la consulta por ti
13     Optional<Usuario> findByUsername(String username);
14 }
15

```

#### 4. Autenticación y autorización: //aplicado

- Implementa un UserDetailsService personalizado que cargue usuarios desde la base de datos.

```

1 package com.banco.banco.service;
2
3 import com.banco.banco.model.Usuario;
4 import com.banco.banco.repository.UsuarioRepository;
5 import org.springframework.security.core.GrantedAuthority;
6 import org.springframework.security.core.authority.SimpleGrantedAuthority;
7 import org.springframework.security.core.userdetails.User;
8 import org.springframework.security.core.userdetails.UserDetails;
9 import org.springframework.security.core.userdetails.UserDetailsService;
10 import org.springframework.security.core.userdetails.UsernameNotFoundException;
11 import org.springframework.stereotype.Service;
12
13 import java.util.Collections;
14
15 @Service
16 public class CustomUserDetailsService implements UserDetailsService {
17
18     // Repositorio para acceder a los usuarios en la base de datos
19     private final UsuarioRepository usuarioRepository;
20
21     public CustomUserDetailsService(UsuarioRepository usuarioRepository) {
22         this.usuarioRepository = usuarioRepository;
23     }
24
25     // Metodo que Spring Security llama para autenticar un usuario por username
26     @Override
27     public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
28         // Buscamos el usuario en la base de datos
29         Usuario usuario = usuarioRepository.findByUsername(username)
30             .orElseThrow(() -> new UsernameNotFoundException("Usuario no encontrado: " + username));
31
32         // Nos aseguramos que el rol tenga el prefijo ROLE_
33     }
34

```

- Configura un PasswordEncoder para el almacenamiento seguro de contraseñas.

```

1 @Configuration
2 public class SecurityConfig {
3
4     // Creamos un Bean para encriptar las contraseñas usando BCrypt
5     // Esto sirve para guardar contraseñas seguras en la base de datos
6     @Bean
7     public PasswordEncoder passwordEncoder() {
8         return new BCryptPasswordEncoder();
9     }
10

```

#### 5. Controladores y vistas: //aplicado

- Crea controladores para login, dashboard de usuario, y operaciones bancarias.

```

1 package com.banco.banco.controller;
2
3 import org.springframework.stereotype.Controller;
4 import org.springframework.web.bind.annotation.GetMapping;
5
6 @Controller no usages 2 sebasramirez830
7 public class DashboardController {
8
9     // Cuando un cliente entra a su dashboard
10    @GetMapping("/cliente/dashboard") no usages 2 sebasramirez830
11    public String clienteDashboard() {
12        // devuelve la vista cliente-dashboard.html ubicada en templates
13        return "cliente-dashboard";
14    }
15
16    // Cuando un empleado entra a su dashboard
17    @GetMapping("/empleado/dashboard") no usages 2 sebasramirez830
18    public String empleadoDashboard() {
19        // devuelve la vista empleado-dashboard.html
20        return "empleado-dashboard";
21    }
22
23    // Cuando un admin entra a su dashboard
24    @GetMapping("/admin/dashboard") no usages 2 sebasramirez830
25    public String adminDashboard() {
26        // devuelve la vista admin-dashboard.html
27        return "admin-dashboard";
28    }
29
30    // Cuando alguien entra a la raíz del sitio "/"
31    @GetMapping("/") no usages 2 sebasramirez830
32    public String slashRoot () {

```

- Implementa vistas correspondientes (puedes usar Thymeleaf).

```

resources
├── static
└── templates
    ├── admin-dashboard.html
    ├── cliente-dashboard.html
    ├── empleado-dashboard.html
    ├── home.html
    ├── login.html
    └── prueba.html

```

## 6. Protección contra amenazas: //aplicado

- Habilita la protección CSRF. Y Implementa encabezados de seguridad para prevenir XSS.

```

37 public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
38     http
39         .authorizeHttpRequests((AuthorizationManagerRequestMatcher) auth -> auth
40             .requestMatchers(...patterns: "/empleado/**").hasRole("EMPLEADO") // solo empleado
41             .requestMatchers(...patterns: "/admin/**").hasRole("ADMIN") // solo admin
42             .anyRequest().authenticated() // todo lo demas requiere login
43         )
44         .csrf(csrf -> csrf
45             .csrfTokenRepository(CookieCsrfTokenRepository.withHttpOnlyFalse())
46             .csrfTokenRepository(CookieCsrfTokenRepository.withHttpOnlyFalse())
47         )
48         .headers(headersConfigurer -> headersConfigurer
49             .contentSecurityPolicy(ContentSecurityPolicyConfigurer.contentSecurityPolicy()
50                 .policyDirectives(
51                     "default-src 'self'; " +
52                     "script-src 'self' 'unsafe-inline' https://cdn.jsdelivr.net https://cdnjs.cloudflare.com; " +
53                     "style-src 'self' 'unsafe-inline' https://cdn.jsdelivr.net https://fonts.googleapis.com; " +
54                     "font-src 'self' https://fonts.gstatic.com; " +
55                     "img-src 'self' data:; " +
56                     "object-src 'none'; " +
57                     "frame-ancestors 'none'; " +
58                     "form-action 'self';"
59                 )
60             )
61         )
62         .httpStrictTransportSecurity(HttpStrictTransportSecurityConfigurer.httpStrictTransportSecurity()
63             .includeSubDomains(true)
64             .preload(true)
65         )
66     );
67 }
68
69 // Evita que otros sitios puedan mostrar nuestra pagina en iframe
70 frameOptions(FrameOptionsConfigurer frameOptions -> frameOptions
71     .deny()
72 );

```

## 7. Gestión de sesiones: //aplicado

- Configura el manejo de sesiones en Spring Security.

```

27 public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
74     .formLogin( FormLoginConfigurer<HttpSecurity> form -> form
86     })
87     .permitAll()
88     }
89
90 // Configuración del logout
91 .logout( LogoutConfigurer<HttpSecurity> logout -> logout
92     .logoutUrl("/logout") // URL para cerrar sesión
93     .logoutSuccessUrl("/login?logout") // a donde redirige luego de cerrar sesión
94     .invalidateHttpSession(true) // invalida la sesión actual
95     .deleteCookies( CookieNamesToClear "JSESSIONID") // elimina cookies de sesión
96     .permitAll()
97 )
98
99 // Gestión de sesiones
100 .sessionManagement( SessionManagementConfigurer<HttpSecurity> session -> session
101     .invalidSessionUrl("/login?invalid-session") // si la sesión es inválida
102     .maximumSessions(1) // solo 1 sesión por usuario
103     .maxSessionsPreventsLogin(false) // si se loguea otra vez, expulsa la sesión anterior
104     .expiredUrl("/login?expired") // si la sesión expira, redirige a login
105 )
106
107 // devolvemos la configuración completa
108 return http.build();
109 }
110
111 }

```

- Implementa un límite de sesiones concurrentes por usuario.

```

1 spring.application.name=banco
2
3 # Si el usuario no interactúa en la página durante 15 minutos la sesión caducará.
4 server.servlet.session.timeout=15m
5
6 # Configuración de la base de datos H2 en memoria
7 spring.datasource.url=jdbc:h2:mem:bancoDB
8 spring.datasource.driverClassName=org.h2.Driver
9 spring.datasource.username=sa
10 spring.datasource.password=
11
12 # Hibernate no crea tablas, usaremos schema.sql y data.sql
13 spring.jpa.hibernate.ddl-auto=none
14 spring.jpa.show-sql=true
15
16 # Permitir que Spring ejecute schema.sql y data.sql
17 spring.sql.init.mode=always
18 spring.jpa.defer-datasource-initialization=true
19
20 # Habilitar consola H2
21 spring.h2.console.enabled=true
22 spring.h2.console.path=/h2-console
23
24
25

```

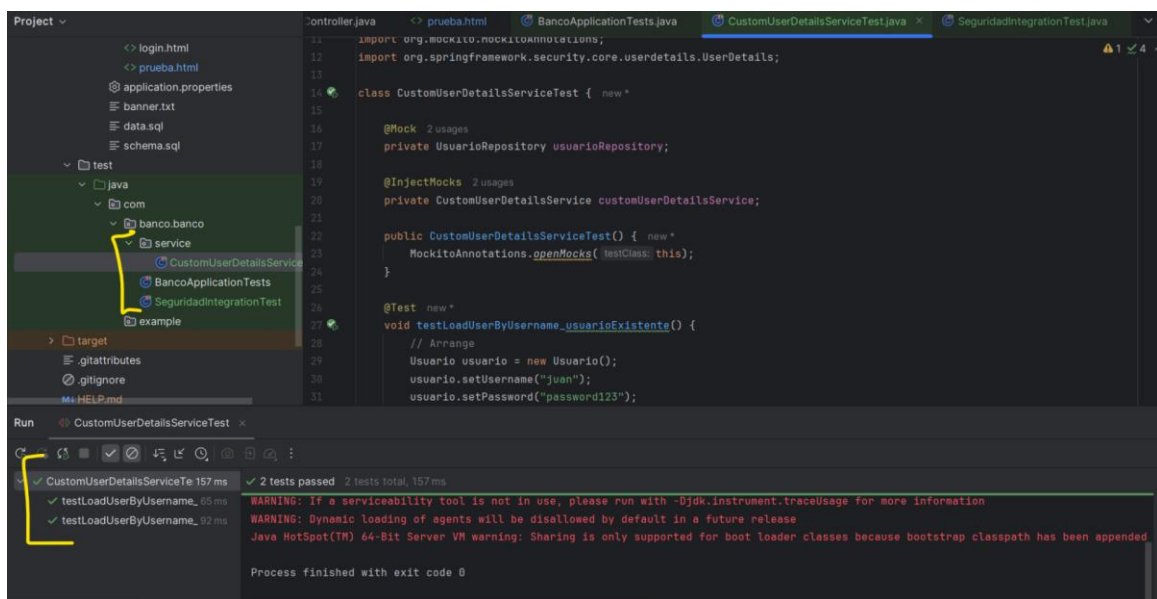
## 8. Integración con Spring Data: //no supe aplicarlo

- Utiliza los repositorios JPA en los servicios de la aplicación.

## 9. Pruebas: //aplicado

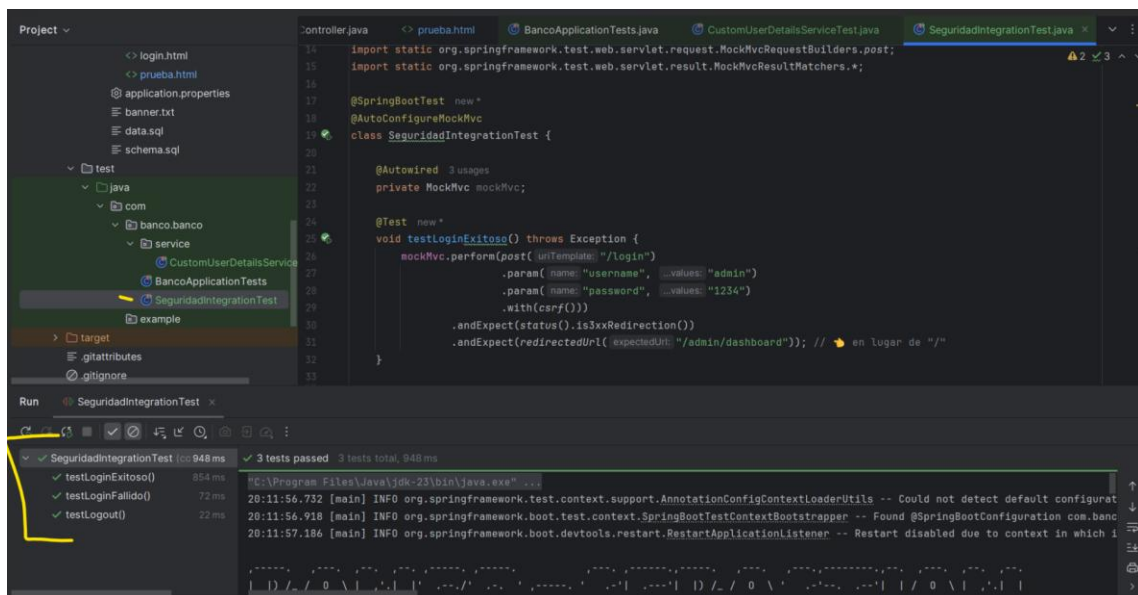
- Escribe pruebas unitarias para los servicios de seguridad.





La prueba se ejecutó correctamente y fue el resultado esperado de cada uno de los test

- Implementa pruebas de integración para los flujos de autenticación y autorización.



La prueba se ejecutó correctamente y fue el resultado esperado de cada uno de los test

### Endpoints a implementar:

- GET /login: Página de inicio de sesión. //aplicado
- GET /dashboard: Dashboard del usuario (accesible para todos los usuarios autenticados). //aplicado
- GET /account/{id}: Detalles de la cuenta (CLIENTE puede ver solo sus cuentas, EMPLEADO y ADMIN pueden ver todas). //aplicado
- POST /transfer: Realizar una transferencia (solo para CLIENTE).
- GET /admin: Panel de administración (solo para ADMIN). //aplicado

### Criterios de evaluación:

1. Correcta implementación de la configuración de Spring Security. //aplicado
2. Funcionamiento adecuado de la autenticación y autorización. //aplicado
3. Implementación efectiva de medidas contra CSRF y XSS. //aplicado
4. Gestión segura de sesiones. //aplicado

5. Integración correcta con Spring Data JPA. //solicitado en el punto 8 no supe aplicarlo.

6. Calidad y cobertura de las pruebas implementadas. //aplicado