UNIDADES TECNOLÓGICAS DE SANTANDER
Programa: *Tecnología en Desarrollo de Sistemas Informaticos*
Asignatura: *Programación Web en Java*
Grupo: *B192*

Proyecto: *Aplicación Web con Inicio de Sesión en Spring Boot*

DOCUMENTACION SPRING SEGURITY

Estudiantes:

- *Jersson Sebastián Fuentes*
- *Juan Pablo Espinel*
- *Guadalupe García*
- *María José Guarde*

Docente:
Magíster Carlos Beltrán

Fecha: *noviembre de 2025*
Lugar: *Bucaramanga, Santander*

Sistema de Autenticación con Spring Boot, Spring Security y Thymeleaf

1. Introducción

El presente documento explica el funcionamiento del sistema de autenticación implementado en el proyecto del semestre de una aplicativo web de psicología para estudiantes , desarrollado con ***Spring Boot*** *como backend,* ***Spring Security*** *para la gestión de la seguridad, y* ***Thymeleaf*** *como motor de plantillas del frontend.*

El objetivo principal de este documento es relatar la implementacion del modulo de ***login*** *que permita el registro, autenticación y gestión del perfil de usuario de manera segura, moderna y extensible.*

---

2. Arquitectura General

El sistema está compuesto por cuatro capas principales:

- ***Modelo:*** *Define la entidad Usuario, mapeada la tabla usuarios de la base datos mediante JPA.*
- ***Repositorio:*** *Gestiona las operaciones CRUD con la base de datos.*
- ***Servicio:*** *Contiene la lógica de negocio y la integración con Spring Security.*
- ***Controlador:*** *Gestiona las peticiones HTTP y la interacción con las vistas Thymeleaf.*
- ***Configuración:*** *Define las reglas de seguridad y las rutas protegidas.*
- ***Vistas (Thymeleaf):*** *Interfaz de usuario para el login, registro y perfil.*

---

3. Modelo (Usuario)

La clase Usuario representa un usuario dentro del sistema. Implementa la interfaz UserDetails de Spring Security, lo que permite que la clase se integre directamente con el mecanismo de autenticación del framework.

Atributos principales:

- ***correo*** *→ se utiliza como nombre de usuario (username).*
- ***contraseñaHash*** *→ almacena la contraseña cifrada con* ***BCrypt.***
- ***rol*** *→ define los permisos del usuario (por ejemplo,* ***ROLE_USER).***
- ***fotoUrl y descripcion*** *→ inf****ormación adicional del perfil.***

```java
package Modelo;

import jakarta.persistence.*;

/**
 * Clase de Entidad que representa a un usuario en la base de datos.
 * Implementa UserDetails de Spring Security para facilitar la autenticación.
 */
@Entity
@Table(name = "usuarios")
public class Usuario implements UserDetails {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String nombre;

    @Column(nullable = false, unique = true)
    private String correo;

    @Column(name = "contraseña_hash", nullable = false)
    private String contraseñaHash;

    @Column(nullable = false)
    private String rol; // Mantenemos el rol para Spring Security


    @Column(name = "foto_url", columnDefinition = "TEXT")
    private String fotoUrl;

    @Column(columnDefinition = "TEXT") // Columna para la descripción (biografía)
    private String descripcion;

    // Constructores, Getters y Setters
    public Usuario() {
    }

    // Getters y Setters (Añadidos para Foto y Descripción)
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
```

Métodos clave:

La implementación de UserDetails obliga a definir métodos como:

```
@Override
public Collection<? extends GrantedAuthority> getAuthorities() {
    return List.of(new SimpleGrantedAuthority(rol));
}
```

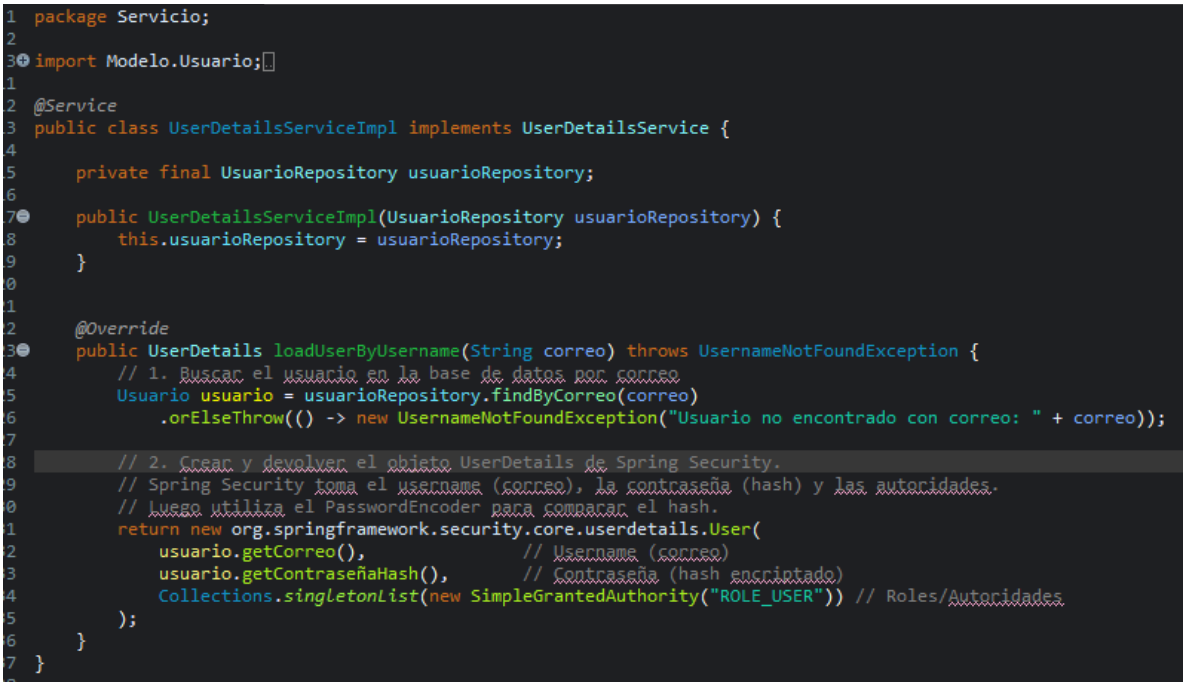
Esto permite a Spring Security conocer los roles y permisos del usuario autenticado.

---

4. Capa de Servicio (UserDetailsServiceImpl y UsuarioService)

a) UserDetailsServiceImpl

Esta clase implementa UserDetailsService, interfaz obligatoria para que Spring Security pueda cargar los datos del usuario durante el proceso de autenticación.



```java
package Servicio;

import Modelo.Usuario;

@Service
public class UserDetailsServiceImpl implements UserDetailsService {

    private final UsuarioRepository usuarioRepository;

    public UserDetailsServiceImpl(UsuarioRepository usuarioRepository) {
        this.usuarioRepository = usuarioRepository;
    }


    @Override
    public UserDetails loadUserByUsername(String correo) throws UsernameNotFoundException {
        // 1. Buscar el usuario en la base de datos por correo
        Usuario usuario = usuarioRepository.findByCorreo(correo)
            .orElseThrow(() -> new UsernameNotFoundException("Usuario no encontrado con correo: " + correo));

        // 2. Crear y devolver el objeto UserDetails de Spring Security.
        // Spring Security toma el username (correo), la contraseña (hash) y las autoridades.
        // Luego utiliza el PasswordEncoder para comparar el hash.
        return new org.springframework.security.core.userdetails.User(
            usuario.getCorreo(),              // Username (correo)
            usuario.getContraseñaHash(),      // Contraseña (hash encriptado)
            Collections.singletonList(new SimpleGrantedAuthority("ROLE_USER")) // Roles/Autoridades
        );
    }
}
```

Su método principal es:

```
@Override
public UserDetails loadUserByUsername(String correo) throws
UsernameNotFoundException {
    Usuario usuario = usuarioRepository.findByCorreo(correo)
```

```
        .orElseThrow(() -> new UsernameNotFoundException("Usuario no encontrado"));
    return new org.springframework.security.core.userdetails.User(
        usuario.getCorreo(),
        usuario.getContraseñaHash(),
        Collections.singletonList(new SimpleGrantedAuthority("ROLE_USER"))
    );
}
```

Aquí Spring obtiene el correo y la contraseña encriptada del usuario y valida su acceso con el PasswordEncoder.

b) UsuarioService

```java
@Service
public class UsuarioService {

    private final UsuarioRepository usuarioRepository;
    private final PasswordEncoder passwordEncoder;

    // La inyección se resuelve correctamente aquí
    public UsuarioService(UsuarioRepository usuarioRepository, PasswordEncoder passwordEncoder) {
        this.usuarioRepository = usuarioRepository;
        this.passwordEncoder = passwordEncoder;
    }

    // --- MÉTODOS DE REGISTRO ---
    public Usuario registrarNuevoUsuario(Usuario usuario) {
        // 1. Validar unicidad
        if (usuarioRepository.findByCorreo(usuario.getCorreo()).isPresent()) {
            throw new IllegalStateException("El correo ya está registrado en la base de datos local.");
        }

        // 2. Hashear la contraseña
        String rawPassword = usuario.getContraseñaHash();
        String hashedPassword = passwordEncoder.encode(rawPassword);

        usuario.setContraseñaHash(hashedPassword);

        // 3. Asignar rol por defecto (IMPORTANTE para Spring Security)
        if (usuario.getRol() == null || usuario.getRol().isEmpty()) {
            usuario.setRol("ROLE_USER");
        }

        // 4. Guardar
        return usuarioRepository.save(usuario);
    }

    // --- MÉTODOS DEL PERFIL ---

    /**
     * Busca un usuario por su ID. Necesario para cargar el perfil completo en la vista.
     * @param id El ID del usuario.
     * @return Un Optional con el Usuario, o vacío si no se encuentra.
     */
    public Optional<Usuario> findById(Long id) {
        return usuarioRepository.findById(id);
    }

    /**
     * Actualiza el nombre, la descripción y la URL de la foto del perfil.
     * @param id ID del usuario a modificar.
     * @param nombre Nuevo nombre.
     * @param descripcion Nueva biografía/descripción.
     * @param fotoUrl Nueva URL de la foto.
     */
    @Transactional
    public void actualizarPerfil(Long id, String nombre, String descripcion, String fotoUrl) {
        // Buscar el usuario por ID. Si no existe, lanza una excepción.
        Usuario usuario = usuarioRepository.findById(id)
                .orElseThrow(() -> new UsernameNotFoundException("Usuario no encontrado con ID: " + id));
```

Contiene la lógica de negocio principal del sistema, incluyendo:

- ***Registro de nuevos usuarios (registrarNuevoUsuario), donde se:***
  - ***Verifica si el correo ya existe.***
  - ***Se cifra la contraseña con BCryptPasswordEncoder.***
  - ***Se asigna un rol por defecto.***
- ***Actualización del perfil (actualizarPerfil).***
- ***Eliminación de cuenta (borrarUsuario).***

Ejemplo del uso de PasswordEncoder:

String hashedPassword = passwordEncoder.encode(usuario.getContraseñaHash());
usuario.setContraseñaHash(hashedPassword);

---

5. Controlador de Perfil (PerfilControlador)

```java
package Controlador;

import Modelo.Usuario;

/**
 * Controlador dedicado a la gestión de la vista, edición y borrado del perfil del usuario.
 */
@Controller
public class PerfilControlador {

    private final UsuarioService usuarioService;

    @Autowired
    public PerfilControlador(UsuarioService usuarioService) {
        this.usuarioService = usuarioService;
    }

    /**
     * Obtiene el objeto Usuario completo actualmente autenticado de la DB.
     * @return El objeto Usuario (modelo) completo o null si no está autenticado o no existe en la DB.
     */
    private Optional<Usuario> getAuthenticatedUserFromDb() {
        Authentication authentication = SecurityContextHolder.getContext().getAuthentication();

        if (authentication == null || !authentication.isAuthenticated() ||
            authentication.getPrincipal().equals("anonymousUser")) {
            return Optional.empty();
        }

        Object principal = authentication.getPrincipal();
        String correo = null;

        // Si el principal es el objeto Usuario (ideal)
        if (principal instanceof Usuario) {
            correo = ((Usuario) principal).getCorreo();
        // Si el principal es el UserDetails de Spring Security (común)
        } else if (principal instanceof UserDetails) {
            correo = ((UserDetails) principal).getUsername();
        } else {
            // Último recurso: intentar como String (el username/correo)
            correo = principal.toString();
        }

        // Usamos el correo obtenido para buscar el objeto Usuario COMPLETO en la DB.
        if (correo != null) {
            return usuarioService.findByCorreo(correo);
        }

        return Optional.empty();
    }


    // 1. Mostrar el formulario de perfil
    @GetMapping("/perfil")
    public String mostrarPerfil(Model model) {

        Optional<Usuario> usuarioDb = getAuthenticatedUserFromDb();

        if (usuarioDb.isEmpty()) {
```

Gestiona las operaciones relacionadas con el perfil del usuario autenticado:

- ***@GetMapping("/perfil")*** *→ carga los datos del usuario a**ctual desde la sesión.***
- ***@PostMapping("/perfil/editar")*** *→ permite actualizar nombre, descripción y foto.*
- ***@PostMapping("/perfil/borrar")*** *→ elimina la cuenta previa confirmación.*

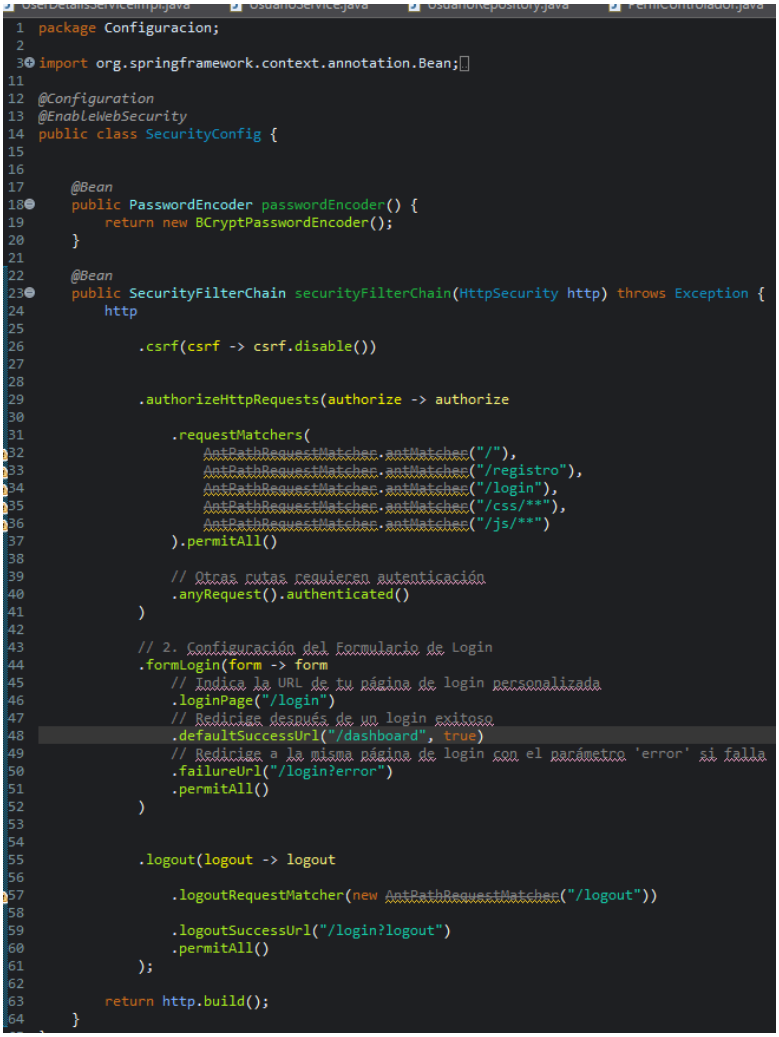
El controlador obtiene el usuario autenticado directamente desde el contexto de seguridad:

Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
String correo = ((UserDetails) authentication.getPrincipal()).getUsername();

---

6. Configuración de Seguridad (SecurityConfig)

Esta clase define las reglas que controlan qué rutas requieren autenticación y cuáles son públicas.



```java
package Configuracion;

import org.springframework.context.annotation.Bean;

@Configuration
@EnableWebSecurity
public class SecurityConfig {


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

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

            .csrf(csrf -> csrf.disable())


            .authorizeHttpRequests(authorize -> authorize

                .requestMatchers(
                    AntPathRequestMatcher.antMatcher("/"),
                    AntPathRequestMatcher.antMatcher("/registro"),
                    AntPathRequestMatcher.antMatcher("/login"),
                    AntPathRequestMatcher.antMatcher("/css/**"),
                    AntPathRequestMatcher.antMatcher("/js/**")
                ).permitAll()

                // Otras rutas requieren autenticación
                .anyRequest().authenticated()
            )

            // 2. Configuración del formulario de Login
            .formLogin(form -> form
                // Indica la URL de tu página de login personalizada
                .loginPage("/login")
                // Redirige después de un login exitoso
                .defaultSuccessUrl("/dashboard", true)
                // Redirige a la misma página de login con el parámetro 'error' si falla
                .failureUrl("/login?error")
                .permitAll()
            )


            .logout(logout -> logout

                .logoutRequestMatcher(new AntPathRequestMatcher("/logout"))

                .logoutSuccessUrl("/login?logout")
                .permitAll()
            );

        return http.build();
    }
```

Configuración del PasswordEncoder

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

Configuración de Seguridad HTTP

```
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    http
        .csrf(csrf -> csrf.disable())
        .authorizeHttpRequests(auth -> auth
            .requestMatchers("/", "/login", "/registro", "/css/**", "/js/**").permitAll()
            .anyRequest().authenticated()
        )
        .formLogin(form -> form
            .loginPage("/login")
            .defaultSuccessUrl("/dashboard", true)
            .failureUrl("/login?error")
            .permitAll()
        )
        .logout(logout -> logout
            .logoutRequestMatcher(new AntPathRequestMatcher("/logout"))
            .logoutSuccessUrl("/login?logout")
            .permitAll()
        );
    return http.build();
}
```

Esto significa:

- *Las rutas /login, /registro y recursos estáticos son públicas.*
- *Todas las demás requieren que el usuario esté autenticado.*
- *El formulario de login usa /login como endpoint y redirige a /dashboard al ingresar correctamente.*

---

7. Vista de Login (login.html)

*Diseñada con **Bootstrap 5** y **Thymeleaf**, permite a los usuarios ingresar sus credenciales o registrarse.*

Características:

- *Modo oscuro y claro con almacenamiento local.*
- *Mensajes de error y éxito usando expresiones Thymeleaf:*

```
<div th:if="${param.error}" class="message-box error-message">
    Credenciales inválidas. Intenta de nuevo.
</div>
```

- *Enlace directo al registro:*

```
<a th:href="@{/registro}">Comienza tu viaje</a>
```

---

8. Flujo de Autenticación

1. *El usuario ingresa su correo y contraseña en /login.*
2. *Spring Security intercepta la petición POST y ejecuta UserDetailsServiceImpl.*

3. ***Se compara la contraseña ingresada con el hash almacenado mediante BCryptPasswordEncoder.***
4. ***Si es correcta:***
    - ***Se crea una sesión segura.***
    - ***El usuario es redirigido al dashboard de la app***

Agregamos link del repositorio en GitHub donde se muestra la implementación al completo https://github.com/tomioka20/App-Psicologia-