Tecsup
TECNOLOGÍA
CON SENTIDO

Desarrollo de Aplicaciones Web

LABORATORIO Nº 9

Spring Security

Alumno(s):	Salas Arenas Marcelo Aldahir	***Nota***	
Grupo:	B	***Ciclo: IV***	

Criterio de Evaluación	*Excelente (4pts)*	*Bueno (3pts)*	*Requiere mejora (2pts)*	*No acept. (0pts)*	*Puntaje Logrado*
Identifica la importancia de las anotaciones.					
Implementa el login y logout usando Spring Security.					
Logra crear la conexión e insertar información a la BD.					
Implementa el MVC propuesto.					
Es puntual y redacta el informe adecuadamente					

TEMA: Spring Security

OBJETIVOS

- Crear un proyecto usando anotaciones, Spring Security, Hibernate JPA y el patrón MVC.

REQUERIMIENTOS

- **Java SDK 17**
- **MySQL**

PROCEDIMIENTO

- ***El laboratorio se ha diseñado para ser desarrollado en grupos de 2.***
- ***Solo un integrante sube el documento al Canvas.***

MARCO TEÓRICO

Características principales:

- **Autenticación**: Verifica la identidad de los usuarios (por ejemplo, mediante login con usuario/contraseña, OAuth, LDAP, etc.).
- **Autorización**: Controla el acceso a recursos según roles o permisos (por ejemplo, solo los administradores acceden a ciertas páginas).
- **Protección contra ataques**: Incluye defensas contra vulnerabilidades como CSRF (Cross-Site Request Forgery), inyección de código, y ataques de fuerza bruta.
- **Integración con Spring**: Se integra fácilmente con aplicaciones Spring Boot, Spring MVC, y otros módulos de Spring.
- **Personalización**: Permite configurar flujos de seguridad personalizados, como autenticación multifactor o single sign-on (SSO).

Componentes clave:

- **SecurityContext**: Almacena la información del usuario autenticado.
- **AuthenticationManager**: Gestiona el proceso de autenticación.
- **AccessDecisionManager**: Decide si un usuario tiene permiso para acceder a un recurso.
- **Filters**: Cadena de filtros que procesan las solicitudes HTTP para aplicar reglas de seguridad.

java

```java
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.web.SecurityFilterChain;

@Configuration
public class SecurityConfig {

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/admin/**").hasRole("ADMIN")
                .requestMatchers("/user/**").hasRole("USER")
                .anyRequest().authenticated()
            )
            .formLogin()
            .and()
            .logout();
        return http.build();
    }
}
```

Esto asegura que las rutas `/admin/**` solo sean accesibles para usuarios con el rol "ADMIN", y las rutas `/user/**` para usuarios con el rol "USER", requiriendo autenticación para cualquier otra solicitud.

Spring Security es ampliamente utilizado en aplicaciones empresariales por su flexibilidad y robustez. ;

A. Crear un proyecto utilizando Spring Security en Spring Boot.

1. Crear el ambiente usando Maven y las siguientes dependencias, use el inicializador del Spring Boot para generar el archivo .pom y que el proyecto sea un .jar

```xml
<dependencies>
    <!-- Spring Boot Starter Web -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <!-- Spring Boot Starter Security -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
    </dependency>
    <!-- Spring Boot Starter Data JPA -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <!-- MySQL Connector -->
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>8.0.33</version>
    </dependency>
    <!-- Lombok (opcional, para reducir código) -->
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>
</dependencies>
```

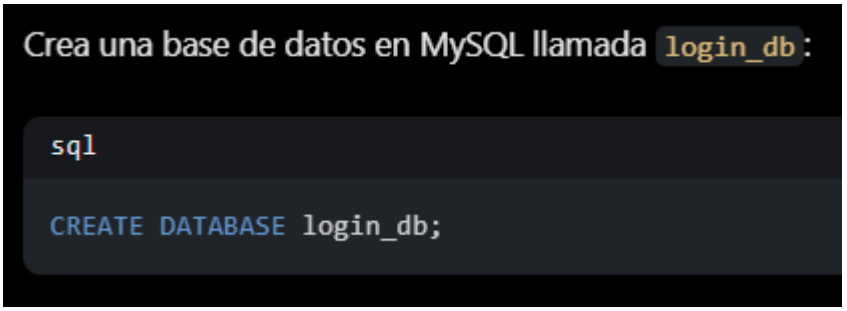
Para las Vistas

```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

2. **Configurar la base de datos.**

- **En src/main/resources/application.properties, configura la conexión a MySQL.**

```
spring.datasource.url=jdbc:mysql://localhost:3306/login_db?useSSL=false&serverTimezone=UTC
 spring.datasource.username=root
spring.datasource.password=tu_contraseña
 spring.jpa.hibernate.ddl-auto=update
 spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
```



3. **Crea una clase User para representar los usuarios en la base de datos (Models).**

```
package com.example.demo.model;

import lombok.Data;
import javax.persistence.*;

@Entity
@Table(name = "users")
@Data
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

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

    @Column(nullable = false)
    private String password;

    @Column(nullable = false)
    private String role; // Ejemplo: "USER" o "ADMIN"
}
```

4. **Crea un repositorio para interactuar con la base de datos.**

```
package com.example.demo.repository;

import com.example.demo.model.User;
import org.springframework.data.jpa.repository.JpaRepository;
import java.util.Optional;

public interface UserRepository extends JpaRepository<User, Long> {
    Optional<User> findByUsername(String username);
}
```

5. **Configuración del Spring Security.**

- **Crea una clase de configuración para Spring Security.**
- **Flujo:**

```
package com.example.demo.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;

import com.example.demo.model.User;
import com.example.demo.repository.UserRepository;

@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/login", "/logout").permitAll()
                .anyRequest().authenticated()
            )
            .formLogin(form -> form
                .loginPage("/login")
                .defaultSuccessUrl("/home", true)
                .permitAll()
            )
            .logout(logout -> logout
                .logoutUrl("/logout")
                .logoutSuccessUrl("/login?logout")
                .permitAll()
            );
        return http.build();
    }


@Bean
    public UserDetailsService userDetailsService(UserRepository userRepository) {
        return username -> {
            User user = userRepository.findByUsername(username)
                    .orElseThrow(() -> new UsernameNotFoundException("User not found"));
            return org.springframework.security.core.userdetails.User
                    .withUsername(user.getUsername())
                    .password(user.getPassword())
                    .roles(user.getRole())
                    .build();
        };
    }

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

```java
package com.example.demo.config;
```

This sets the namespace for the class. It's in the `config` package of the `demo` application.

The imported classes handle:

- `HttpSecurity`, `SecurityFilterChain` – core Spring Security components.
- `UserDetailsService`, `UsernameNotFoundException` – used to load user-specific data.
- `PasswordEncoder`, `BCryptPasswordEncoder` – for securely encoding passwords.
- `@Configuration`, `@EnableWebSecurity`, `@Bean` – for declaring configuration classes and beans.

```java
@Configuration
@EnableWebSecurity
public class SecurityConfig {
```

- `@Configuration` marks this class as a Spring configuration class.
- `@EnableWebSecurity` enables Spring Security web security support.

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

This method defines how **HTTP security** is handled:

1. Authorization

```java
http.authorizeHttpRequests(auth -> auth
    .requestMatchers("/login", "/logout").permitAll()
    .anyRequest().authenticated()
)
```

- Allows anyone to access `/login` and `/logout`.
- All other requests require authentication.

2. Form Login

```java
.formLogin(form -> form
    .loginPage("/login")
    .defaultSuccessUrl("/home", true)
    .permitAll()
)
```

- Uses a custom login page (`/login`).
- On successful login, users are redirected to `/home` .

3. Logout

```java
.logout(logout -> logout
    .logoutUrl("/logout")
    .logoutSuccessUrl("/login?logout")
    .permitAll()
)
```

- Defines `/logout` as the logout URL.
- After logging out, users are redirected to `/login?logout` .

```java
return http.build();
```

- Builds and returns the configured security filter chain.

```java
@Bean
public UserDetailsService userDetailsService(UserRepository userRepository) {
```

Defines a custom user retrieval logic:

java Copiar

```java
return username -> {
    User user = userRepository.findByUsername(username)
        .orElseThrow(() -> new UsernameNotFoundException("User not found"));
```

- Looks up a user in the database using `UserRepository`.
- If not found, throws an exception.

java Copiar

```java
return org.springframework.security.core.userdetails.User
    .withUsername(user.getUsername())
    .password(user.getPassword())
    .roles(user.getRole())
    .build();
```

- Converts your custom `User` object into a Spring Security-compatible `UserDetails`.

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

This bean uses **BCrypt** to encode and verify passwords. BCrypt is strong and adaptive, recommended for storing passwords securely.

Component	Purpose
`@Configuration`, `@EnableWebSecurity`	Marks class as security config
`SecurityFilterChain`	Configures login/logout, URL access control
`UserDetailsService`	Fetches user details from DB
`PasswordEncoder`	Encrypts passwords with BCrypt

6. **Crear un controlador para manejar las rutas de login, logout y home.**

```
package com.example.demo.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class AuthController {

@GetMapping("/login") public
     String login() {
return "login"; // Nombre de la plantilla Thymeleaf o JSP
     }

@GetMapping("/home") public
     String home() {
return "home"; // Página principal después del login
     }
 }
```

7. **Crear las Vistas.**

- **login.html**

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>Login</title>
</head>
<body>
<h2>Login</h2>
<form th:action="@{/login}" method="post">
<div>
<label>Username:</label>
<input type="text" name="username" required/>
</div>
<div>
<label>Password:</label>
<input type="password" name="password" required/>
</div>
<div>
```

```
<button type="submit">Login</button>
</div>
</form>
<div th:if="${param.error}">
<p style="color:red;">Invalid username or password.</p>
</div>
<div th:if="${param.logout}">
<p style="color:green;">You have been logged out.</p>
</div>
</body>
</html>
```

- **home.html**

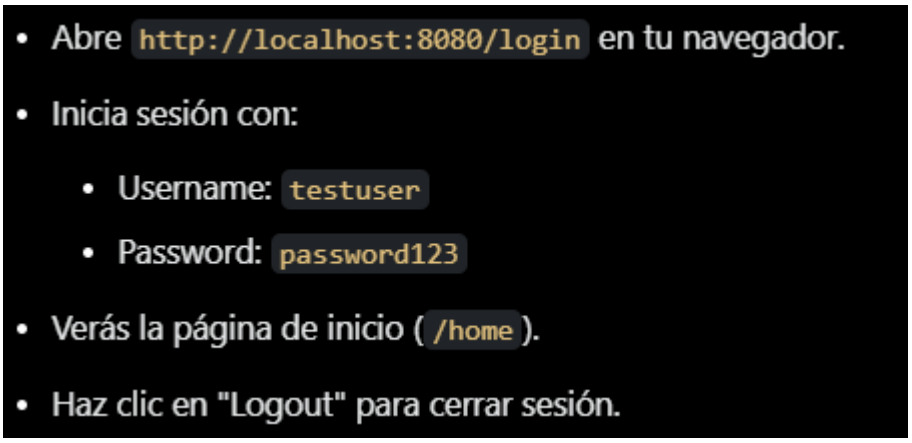
```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>Home</title>
</head>
<body>
<h2>Welcome to the Home Page!</h2>
<form th:action="@{/logout}" method="post">
<button type="submit">Logout</button>
</form>
</body>
</html>
```

8. Crear un usuario de prueba.

```
package com.example.demo;

import com.example.demo.model.User;
import  com.example.demo.repository.UserRepository;
 import org.springframework.boot.CommandLineRunner;
 import org.springframework.boot.SpringApplication;
import  org.springframework.boot.autoconfigure.SpringBootApplication;
 import org.springframework.context.annotation.Bean;
import org.springframework.security.crypto.password.PasswordEncoder;

@SpringBootApplication
public class DemoApplication {

    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }

@Bean
    CommandLineRunner init(UserRepository userRepository, PasswordEncoder passwordEncoder) {
        return args -> {
User user = new User(); user.setUsername("testuser");
user.setPassword(passwordEncoder.encode("password123"));
            user.setRole("USER");
userRepository.save(user);
```

```
        );
    }
}
```

9. **Compilar el proyecto.**



B. En el proyecto anterior crear un MVC del Objeto Dashboard, los datos del objeto deben ser almacenados en una tabla en la BD login_db. Poner estilos, etc.

```
// src/main/java/com/example/demo/model/Dashboard.kt
package com.example.demo.model

import jakarta.persistence.*

@Entity
@Table(name = "dashboard")
data class Dashboard(
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    val id: Long = 0,

    @Column(nullable = false)
    val titulo: String = "",

    @Column(nullable = false)
    val descripcion: String = "",

    @Column(nullable = false)
```

```
    val estado: String = ""
)

// src/main/java/com/example/demo/repository/DashboardRepository.kt
package com.example.demo.repository

import com.example.demo.model.Dashboard
import org.springframework.data.jpa.repository.JpaRepository

interface DashboardRepository : JpaRepository<Dashboard, Long>

// src/main/java/com/example/demo/service/DashboardService.kt
package com.example.demo.service

import com.example.demo.model.Dashboard
import com.example.demo.repository.DashboardRepository
import org.springframework.stereotype.Service

@Service
class DashboardService(val repo: DashboardRepository) {

    fun listar(): List<Dashboard> = repo.findAll()

    fun guardar(dashboard: Dashboard): Dashboard = repo.save(dashboard)

    fun buscarPorId(id: Long): Dashboard = repo.findById(id).orElseThrow()

    fun eliminar(id: Long) = repo.deleteById(id)
}

// src/main/java/com/example/demo/controller/DashboardController.kt
package com.example.demo.controller

import com.example.demo.model.Dashboard
import com.example.demo.service.DashboardService
import org.springframework.stereotype.Controller
import org.springframework.ui.Model
import org.springframework.web.bind.annotation.*
```

```
@Controller
@RequestMapping("/dashboard")
class DashboardController(val servicio: DashboardService) {

    @GetMapping
    fun index(model: Model): String {
        model.addAttribute("dashboards", servicio.listar())
        return "dashboard/index"
    }

    @GetMapping("/nuevo")
    fun nuevo(model: Model): String {
        model.addAttribute("dashboard", Dashboard())
        return "dashboard/form"
    }

    @PostMapping("/guardar")
    fun guardar(@ModelAttribute dashboard: Dashboard): String {
        servicio.guardar(dashboard)
        return "redirect:/dashboard"
    }

    @GetMapping("/editar/{id}")
    fun editar(@PathVariable id: Long, model: Model): String {
        val dashboard = servicio.buscarPorId(id)
        model.addAttribute("dashboard", dashboard)
        return "dashboard/form"
    }

    @GetMapping("/eliminar/{id}")
    fun eliminar(@PathVariable id: Long): String {
        servicio.eliminar(id)
        return "redirect:/dashboard"
    }
}
```

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <title>Dashboard</title>
  <link rel="stylesheet" th:href="@{/css/style.css}">
</head>
<body>
<h1>Dashboard</h1>
<a th:href="@{/dashboard/nuevo}">Nuevo Registro</a>
<table>
  <thead>
    <tr>
      <th>ID</th>
      <th>Título</th>
      <th>Descripción</th>
      <th>Estado</th>
      <th>Acciones</th>
    </tr>
  </thead>
  <tbody>
    <tr th:each="dash : ${dashboards}">
      <td th:text="${dash.id}"></td>
      <td th:text="${dash.titulo}"></td>
      <td th:text="${dash.descripcion}"></td>
      <td th:text="${dash.estado}"></td>
      <td>
        <a th:href="@{/dashboard/editar/{id}(id=${dash.id})}">Editar</a>
        <a th:href="@{/dashboard/eliminar/{id}(id=${dash.id})}">Eliminar</a>
      </td>
    </tr>
  </tbody>
</table>
</body>
</html>
```

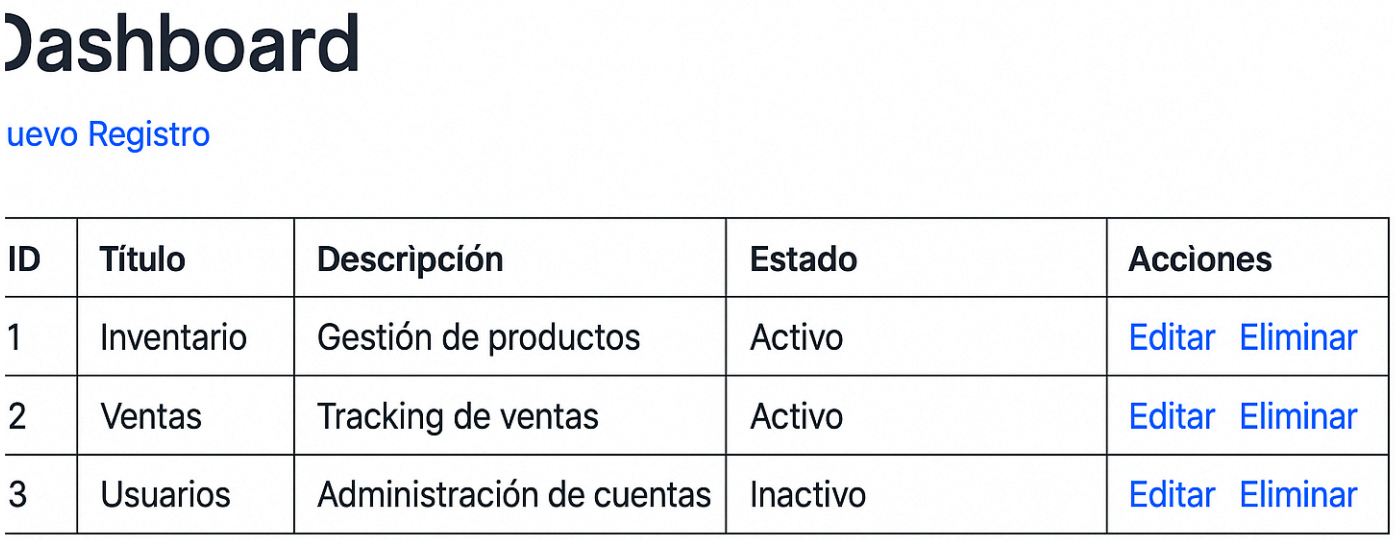
```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
```

```
<head>
  <title>Formulario Dashboard</title>
  <link rel="stylesheet" th:href="@{/css/style.css}">
</head>
<body>
<h1 th:text="${dashboard.id == 0} ? 'Nuevo Dashboard' : 'Editar Dashboard'"></h1>
<form th:action="@{/dashboard/guardar}" th:object="${dashboard}" method="post">
  <input type="hidden" th:field="*{id}">
  <label>Título:</label>
  <input type="text" th:field="*{titulo}" required><br>
  <label>Descripción:</label>
  <input type="text" th:field="*{descripcion}" required><br>
  <label>Estado:</label>
  <input type="text" th:field="*{estado}" required><br>
  <button type="submit">Guardar</button>
</form>
</body>
</html>
```

```
body {
  font-family: Arial, sans-serif;
  margin: 20px;
}
table {
  width: 80%;
  border-collapse: collapse;
  margin-top: 20px;
}
table, th, td {
  border: 1px solid #888;
}
th, td {
  padding: 10px;
  text-align: left;
}
a {
  margin-right: 10px;
```

```
    text-decoration: none;
    color: blue;
}
a:hover {
    text-decoration: underline;
}
```

```
CREATE TABLE dashboard (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    titulo VARCHAR(100) NOT NULL,
    descripcion TEXT NOT NULL,
    estado VARCHAR(50) NOT NULL
);
```



Dashboard

uevo Registro

ID	Título	Descripción	Estado	Acciones
1	Inventario	Gestión de productos	Activo	Editar Eliminar
2	Ventas	Tracking de ventas	Activo	Editar Eliminar
3	Usuarios	Administración de cuentas	Inactivo	Editar Eliminar

Conclusiones:

Indicar las conclusiones que llegó después de los temas tratados de manera práctica en este laboratorio.

Aprendimos a implementar seguridad en aplicaciones web usando Spring Security y anotaciones.

Logramos conectar la base de datos y manejar usuarios de forma segura con encriptación.

Comprendimos mejor el patrón MVC y su integración con Spring Boot.