

# ➤ Control de Acceso mediante Spring Security

---

**Plan formativo:** Desarrollo de Aplicaciones Full Stack Java Trainee V2.0

# HOJA DE RUTA

¿Cuáles **skill** conforman el programa?



# REPASO CLASE ANTERIOR

En la clase anterior trabajamos :

- ✓ *Aprender a implementar el uso de anotaciones para securizar desde el controlador*

# LEARNING PATHWAY

¿Sobre qué temas trabajaremos?

6.4

Start! 🏁

Control de Acceso  
mediante Spring  
Security

Autenticación con JPA

Encriptando

Autenticación contra una base de datos utilizando JPA

# OBJETIVOS DE APRENDIZAJE

¿Qué aprenderemos?



*Autenticación contra una base de datos  
utilizando JPA*



# › Autenticación contra una base de datos utilizando JPA



# Autenticación utilizando JPA

La autenticación es un componente esencial en cualquier aplicación web para garantizar que solo usuarios autorizados tengan acceso a los recursos protegidos. Spring Security proporciona un sólido marco para implementar la autenticación, y cuando se combina con Java Persistence API (JPA), la autenticación contra una base de datos se vuelve más potente y flexible.





# Autenticación utilizando JPA

## Rol de JPA en la Autenticación:

Java Persistence API (JPA) es una especificación de Java para el mapeo objeto-relacional, lo que facilita el almacenamiento y la recuperación de objetos Java en bases de datos relacionales. Al utilizar JPA en combinación con Spring Security, puedes aprovechar su capacidad para interactuar directamente con la capa de persistencia de la base de datos para autenticar usuarios.





# ➤ Autenticación y encriptación de contraseñas mediante JPA

# Autenticación y encriptación de contraseñas mediante JPA

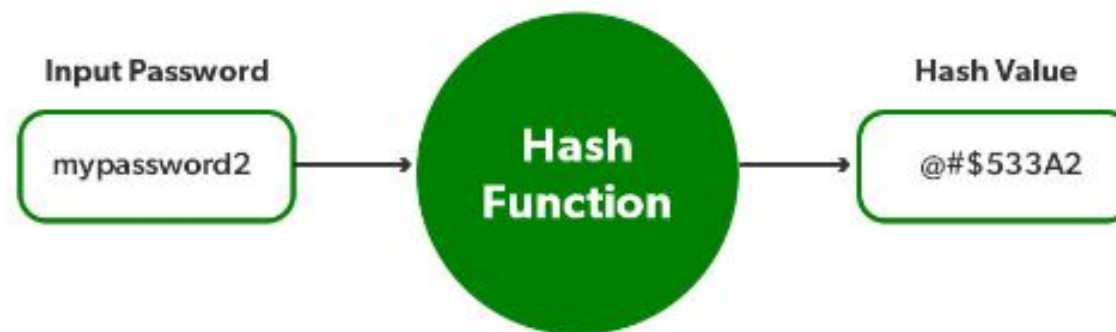
Un mecanismo de seguridad que hay que tener en cuenta a la hora de guardar los datos del usuario es **proteger las contraseñas**.

Usualmente no se almacena la contraseña como tal, sino que lo que guardamos en la base de datos es el **valor hash** de las mismas. De este modo, si alguien logra tener acceso a nuestra base de datos de usuarios, no podrá leer las contraseñas.



# Autenticación y encriptación de contraseñas mediante JPA

Para esta tarea Spring Security utiliza la interfaz `PasswordEncoder`, podemos crear nuestra propia implementación o utilizar una de las proporcionadas por el Framework, para este ejemplo utilizaremos la clase `BCryptPasswordEncoder`.



# Autenticación y encriptación de contraseñas mediante JPA

En el método `configureGlobal` de la clase de seguridad, tendremos el objeto `AuthenticationManagerBuilder` que se encarga de realizar el manejo de la autenticación de usuarios.

A continuación del llamado a `userDetailsService` para autenticar, colocaremos otro punto para llamar al `passwordEncoder`.

```
@Autowired
public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception{
    auth.userDetailsService(usuarioServicio)
        .passwordEncoder(new BCryptPasswordEncoder());
}
```

# Autenticación y encriptación de contraseñas mediante JPA

Una vez definido el método de autenticación y encriptación de contraseñas, solo nos resta **setear** el **encoder** al **momento de guardar la contraseña** generada por el usuario.

**Es necesario que en la base de datos quede persistida la contraseña ya codificada.** Esto lo realiza JPA con referencia de las anotaciones ya mencionadas.



# Autenticación y encriptación de contraseñas mediante JPA

Para poder definir la encriptación al momento de la creación de la contraseña, vamos a agregar el llamado al encoder en el método de creación de usuario en el servicio. Por ejemplo:

```
UserEntity user = new UserEntity();

user.setName(name);
user.setEmail(email);

user.setPassword(new BCryptPasswordEncoder().encode(password));

user.setRole(Role.USER);

userRepository.save(user);
```

# Autenticación y encriptación de contraseñas mediante JPA

De esta manera estamos pidiéndole a JPA que **codifique la contraseña en el mismo momento en el que se setee en la variable password**. Si realizamos una consulta en la base de datos, podremos ver que la contraseña está compuesta de caracteres alfanuméricos en cadena larga.

```
UserEntity user = new UserEntity();

user.setName(name);
user.setEmail(email);

user.setPassword(new BCryptPasswordEncoder().encode(password));

user.setRole(Role.USER);

userRepository.save(user);
```

# Autenticación y encriptación de contraseñas mediante JPA

Por ejemplo, si al momento de crear el usuario colocamos como contraseña la cadena “123456”, la clase `BCryptPasswordEncoder`, generará una contraseña como ésta:

```
$2a$10$fYoIYBQcszFwBxwT3hbj.QY6TDz1SvDDi9SQgY028  
maC6khfg2di
```

Y ese será el **valor hash** con el que queda guardada en la base de datos.









# Autenticación y encriptación de contraseñas mediante JPA



**Características** de la autenticación con JPA:



1. **Integración Sencilla:** La integración de Spring Security con JPA es fluida y simplifica la implementación de la autenticación, ya que permite utilizar entidades JPA para representar a los usuarios y roles.
2. **Seguridad y Flexibilidad:** Utilizar JPA para la autenticación permite una gestión segura y flexible de usuarios y roles en la base de datos, lo que facilita la adaptación a cambios en los requisitos de seguridad.
3. **Contraseña Segura:** Almacenar contraseñas de manera segura es fundamental. Spring Security facilita esto mediante el uso de codificadores de contraseñas como **BCrypt**, que son difíciles de descifrar.



# Autenticación y encriptación de contraseñas mediante JPA



**Características** de la autenticación con JPA:


- 
- 4. **Personalización de Detalles del Usuario:** La interfaz **UserDetails** permite personalizar los detalles del usuario, como roles y autorizaciones, según los requisitos de la aplicación.
  - 5. **Manejo de Sesiones:** Spring Security maneja automáticamente la creación y administración de sesiones de usuario, proporcionando una capa adicional de seguridad.
- 



# Autenticación y encriptación de contraseñas mediante JPA



Algunas **desventajas** de la autenticación con JPA:


- 
1. **Complejidad en Implementaciones Avanzadas:** A medida que las necesidades de autenticación se vuelven más complejas, como la implementación de flujos de autenticación personalizados, la configuración puede volverse más complicada.
  2. **Posible Sobrecarga de Base de Datos:** Dependiendo del tamaño de la aplicación y la cantidad de usuarios, el acceso frecuente a la base de datos para autenticar usuarios podría generar cierta sobrecarga.
  3. **Seguridad en la Configuración:** Asegúrate de configurar adecuadamente Spring Security para proteger las configuraciones sensibles, como las credenciales de la base de datos y las claves de codificación de contraseñas.



# Autenticación y encriptación de contraseñas mediante JPA



## Consideraciones adicionales:

1. **Auditoría y Registros:** Considera la implementación de registros de auditoría para realizar un seguimiento de eventos de autenticación, incluyendo intentos fallidos y exitosos.
2. **Actualizaciones de Contraseña:** Implementa una funcionalidad para que los usuarios puedan actualizar sus contraseñas y garantiza que las contraseñas antiguas sean almacenadas de manera segura.
3. **Mantenimiento de Usuarios:** Incluye funcionalidades para administrar usuarios, como la capacidad de desactivar cuentas o restablecer contraseñas.

# Evaluación Integradora ✨

¿Listos para un nuevo desafío? En esta clase comenzamos a construir nuestro...

## Trabajo Integrador del Módulo 💪

Iremos completándolo progresivamente clase a clase.



# LIVE CODING

Ejemplo en vivo

## Encriptando contraseñas

*Vamos a agregar las configuraciones de encriptación de contraseñas en la clase de seguridad. Además, vamos a encriptar la contraseña en el momento en que seteamos el atributo.*

*1- Personalizar el método **configureGlobal** para agregar el encriptador*

*2- Agregar el encriptador al seteo de la contraseña, para que se codifique al momento de la creación del usuario.*



# LIVE CODING

Ejemplo en vivo

- 3- Crear un formulario de Registro de Usuario (HTML o JSP)*
- 4- Crear un controlador de registro que reciba los datos del formulario y guarde un usuario creado con su contraseña codificada*
- 5- consultar la base de datos para confirmar que la contraseña fue encriptada con éxito.*

  **Tiempo: 30 minutos**

Momento: ✚

# Time-out!

🕒 5 min.







# **Ejercicio** **Encriptando**



# Securizando



## Contexto: 🙌

Vamos a continuar con el ejercicio de la clase anterior. En este caso debemos agregar la configuración de encriptación de contraseñas.

## Consigna: 📝

- 1- Añadir el método `configureGlobal` a la clase de configuración de seguridad.
- 2- Agregar el llamado al `encoder` en el método de creación de usuarios, cuando se setea la contraseña.
- 3- Crear dos nuevos usuarios y verificar el funcionamiento del `encoder` en la base de datos.

**Tiempo** 🕒: 20 minutos (puedes continuar de manera asíncrona)

○

# ¿Alguna consulta?

+



# RESUMEN

¿Qué logramos en esta clase?

- ✓ **Comprender cómo autenticar usuarios utilizando JPA**
- ✓ **Aprender a encriptar las contraseñas con JPA**



# #WorkingTime

Continuemos ejercitando

**¡Antes de cerrar la clase!** Te invitamos a: 🙌 🙌 🙌

1. Repasar nuevamente la grabación de esta clase
2. Revisar el material compartido en la plataforma de Moodle (lo que se vio en clase y algún ejercicio adicional)
  - a. *Lectura Modulo 6, Lección 4: páginas 14 - 16*
3. Traer al próximo encuentro, todas tus dudas y consultas para verlas antes de iniciar nuevo tema.

# ¡Muchas Gracias!

Nos vemos en la próxima clase 🙌

