**Encriptado de Contraseñas**

**¿Qué significa Encriptar?**

**Encriptar**, también conocido como **cifrar o codificar**, es el proceso de transformar información legible, como texto o datos, en un formato ininteligible para que solo las personas o sistemas autorizados puedan acceder a ella.

Este proceso se realiza utilizando algoritmos matemáticos complejos que alteran la estructura de la información original, haciéndola imposible de entender sin la clave de descifrado adecuada.

La **encriptación** se utiliza en diversas áreas para proteger información confidencial, como:

* **Seguridad informática:** Encriptar contraseñas, datos bancarios, comunicaciones en línea y archivos confidenciales protege contra el acceso no autorizado, robo de datos y ciberataques.
* **Comunicaciones seguras:** La encriptación se utiliza en protocolos como HTTPS para garantizar la confidencialidad e integridad de las comunicaciones en internet, como al realizar compras en línea o acceder a servicios bancarios.
* **Almacenamiento de datos:** La encriptación de datos en discos duros, memorias USB y otros dispositivos de almacenamiento protege la información en caso de pérdida o robo del dispositivo.
* **Protección de la propiedad intelectual:** La encriptación se utiliza para proteger contenido digital como software, música, películas y libros electrónicos contra la piratería y la distribución ilegal.

**¿Por qué es importante encriptar contraseñas?**

Encriptar las contraseñas es crucial para los desarrolladores y para el futuro uso correcto y seguro de las aplicaciones que lleven a cabo por varias razones:

* **Protección de datos confidenciales:**Los desarrolladores trabajan con información sensible como datos de usuarios, bases de datos, código fuente y propiedad intelectual. Encriptar las contraseñas protege esta información de accesos no autorizados, filtraciones de datos y ciberataques.
* **Prevención de robo de identidad:**Los «ciberdelincuentes» pueden robar contraseñas para suplantar la identidad de usuarios, acceder a sus cuentas y realizar acciones en su nombre, como compras, modificar datos o acceder a información que sea sumamente confidencial. La encriptación dificulta este tipo de ataques.
* **Cumplimiento de normas y regulaciones:**Muchas industrias, como la financiera y la de salud, tienen regulaciones estrictas sobre la protección de datos. Encriptar las contraseñas es una medida esencial para cumplir con estas regulaciones y evitar sanciones en los sistemas que sean desarrollados.
* **Mejora de la seguridad general:**La encriptación de contraseñas es un elemento fundamental de una estrategia de seguridad sólida. Al proteger las contraseñas, se reduce el riesgo de que otros elementos de seguridad, como firewalls y sistemas de detección de intrusiones, sean vulnerados.
* **Buenas prácticas de desarrollo:**La encriptación de contraseñas es considerada una buena práctica de desarrollo. Los desarrolladores de sistemas en general deben adoptar esta práctica para proteger la información confidencial y demostrar su compromiso con la seguridad. Anteriormente

**Algoritmos de Encriptación**

Existen diversos algoritmos de encriptación de contraseñas, cada uno con sus propias características y niveles de seguridad. Los más comunes se pueden clasificar en dos categorías principales:

**1. Algoritmos de encriptación simétrica:**

* **bcrypt:** Un algoritmo robusto y ampliamente utilizado que ofrece un alto nivel de seguridad frente a ataques de fuerza bruta y rainbow table. Es la opción recomendada para almacenar contraseñas en bases de datos.
* **scrypt:** Un algoritmo relativamente nuevo que se destaca por su resistencia a ataques de minería de contraseñas. Es una buena alternativa a bcrypt, especialmente cuando se requiere un mayor nivel de protección.
* **PBKDF2 (Password-Based Key DerivationFunction 2):** Un algoritmo versátil que puede utilizarse para derivar claves de cifrado a partir de contraseñas. Es menos común que bcrypt o scrypt, pero puede ser útil en algunas aplicaciones específicas.

**2. Algoritmos de encriptación asimétrica:**

* **RSA (Rivest-Shamir-Adleman):** Un algoritmo ampliamente utilizado para la firma digital y la autenticación de comunicaciones. No se recomienda para el encriptado directo de contraseñas debido a su elevado costo computacional.
* **Elliptic curve cryptography (ECC):** Una alternativa más moderna a RSA que ofrece un mayor nivel de seguridad con menor costo computacional. Se está volviendo cada vez más popular para el encriptado de contraseñas.

**Factores a considerar al elegir un algoritmo de encriptación de contraseñas:**

* **Nivel de seguridad:** El algoritmo debe ofrecer un nivel de seguridad adecuado para el tipo de información que se protege y el riesgo de ataques.
* **Costo computacional:** El algoritmo debe ser lo suficientemente eficiente como para no afectar el rendimiento del sistema.
* **Compatibilidad:** El algoritmo debe ser compatible con los sistemas y lenguajes de programación que se utilizan.
* **Facilidad de implementación:** El algoritmo debe ser fácil de implementar y utilizar.

**¿Qué tener en cuenta a la hora de elegir un algoritmo de encriptado?**

*Algunas cuestiones básicas a tener en cuenta a la hora de elegir un algoritmo de encriptado pueden ser:*

* Para el almacenamiento de contraseñas en bases de datos, se recomienda utilizar algoritmos como **bcrypt** o **scrypt**.
* Para la autenticación de usuarios en aplicaciones web, se recomienda utilizar mecanismos de autenticación seguros, como el **hash de contraseñas** y **tokens de autenticación**.

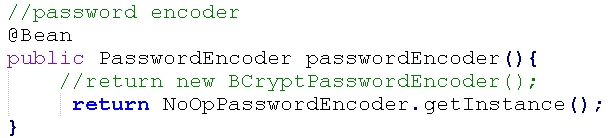
**Implementación Práctica**

**Paso 1: BCrypt en Spring Security**

Anteriormente en nuestro proyecto usábamos las contraseñas en formato de texto plano. Esto, a nivel de seguridad es un GRAVE ERROR y una GRAN VULNERABILIDAD que podríamos llegar a tener en nuestro sistema.

El encargado de codificar y decodificar las contraseñas (en base a la arquitectura que planteamos anteriormente) es nuestro método passwordEncoder() dentro de nuestro archivo de SecurityConfig.

El problema, es que en él establecimos la opción «NoOpPasswordEncoder» tal como vemos en el código:



NoOpPasswordEncoder es una clase de Spring Security que implementa la interfaz PasswordEncoder pero no realiza ninguna encriptación. En otras palabras, deja las contraseñas sin encriptar. se utiliza principalmente para **propósitos de prueba y desarrollo**. En un entorno de producción, **nunca se debe utilizar NoOpPasswordEncoder** para almacenar contraseñas reales, ya que esto las deja completamente expuestas y vulnerables a ataques.

Dado esto, vamos ahora a agregar un algoritmo de encriptación a nuestro PasswordEncoder, en este caso, basándonos en las recomendaciones que vimos anteriormente, vamos a utilizar **BCrypt**:

//password encoder

    @Bean

    public PasswordEncoder passwordEncoder(){

         return new BCryptPasswordEncoder();

    }

**Paso 2: Codificar las contraseñas con BCrypt**

Si bien ya agregamos **BCrypt** a nuestro **PasswordEncoder**, aún no agregamos ninguna forma de encriptado a la hora de guardar nuestras contraseñas. Actualmente, si recibimos la contraseña «1234» mediante una **request**, la misma se almacena tal cual en la base de datos sin codificar, por lo que, vamos a realizar las modificaciones necesarias para encriptar nuestras contraseñas a la hora de guardarlas.

Vamos a ir a nuestro endpoint de creación de usuarios y vamos a agregar un llamado a nuestra instancia de UserService para que la capa de servicio se encargue de encriptar la contraseña que recibamos en el endpoint de forma plana:

package ar.edu.centro8.ps.encriptado.service;

import ar.edu.centro8.ps.encriptado.model.UserSec;

import ar.edu.centro8.ps.encriptado.repository.IUserRepository;

import org.springframework.beans.factory.annotation.Autowired;

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

import org.springframework.stereotype.Service;

import java.util.List;

import java.util.Optional;

@Service

public class UserService implements IUserService {

    @Autowired

    private IUserRepository userRepository;

    @Override

    public List<UserSec> findAll() {

        return userRepository.findAll();

    }

    @Override

    public Optional<UserSec> findById(Long id) {

        return userRepository.findById(id);

    }

    @Override

    public UserSec save(UserSec userSec) {

        return userRepository.save(userSec);

    }

    @Override

    public void deleteById(Long id) {

        userRepository.deleteById(id);

    }

    @Override

    public void update(UserSec userSec) {

        save(userSec);

    }

    @Override

    public String encriptPassword(String password) {

        return new BCryptPasswordEncoder().encode(password);

    }

}

Por supuesto, el método **encriptPassword** no existe en nuestro **userService**, por lo que vamos a crearlo en la interfaz y luego implementarlo en la clase de implementación:

package ar.edu.centro8.ps.encriptado.service;

import ar.edu.centro8.ps.encriptado.model.UserSec;

import java.util.List;

import java.util.Optional;

public interface IUserService {

    public List<UserSec> findAll();

    public Optional<UserSec> findById(Long id);

    public UserSec save(UserSec userSec);

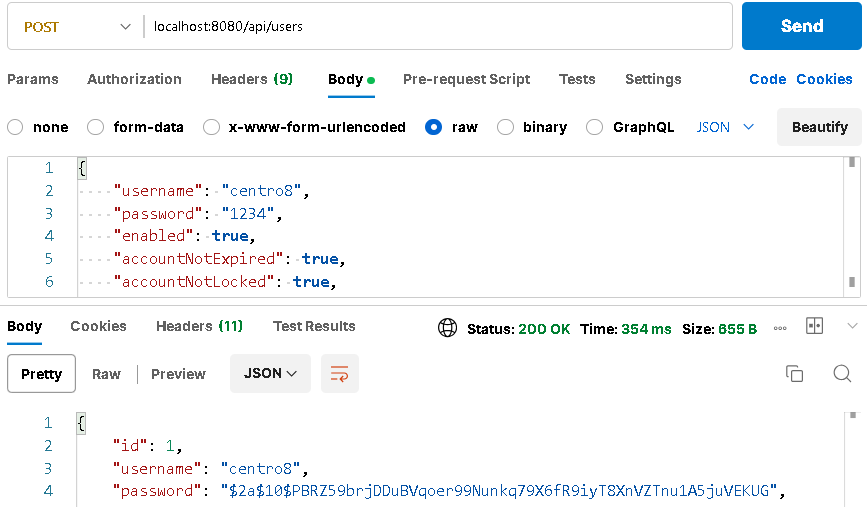
    public void deleteById(Long id);

    public void update(UserSec userSec);

    public String encriptPassword(String password);

}

Con esto, ahora todas nuestras contraseñas se deberían de encriptar al dar de alta a los usuarios. Probemos crear un nuevo usuario con Postman a ver si es así:



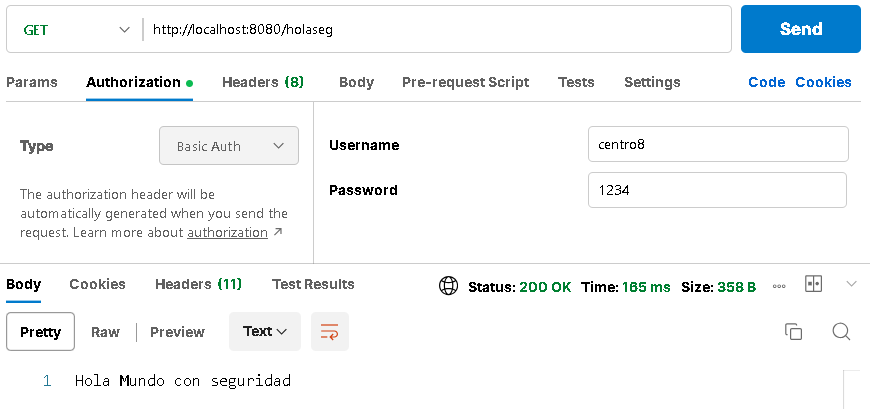
Como podemos ver, la contraseña se encriptó correctamente y se guardó en la base de datos de igual manera, a diferencia de los usuarios del ejercicio anterior que habían sido guardados sin encriptado:

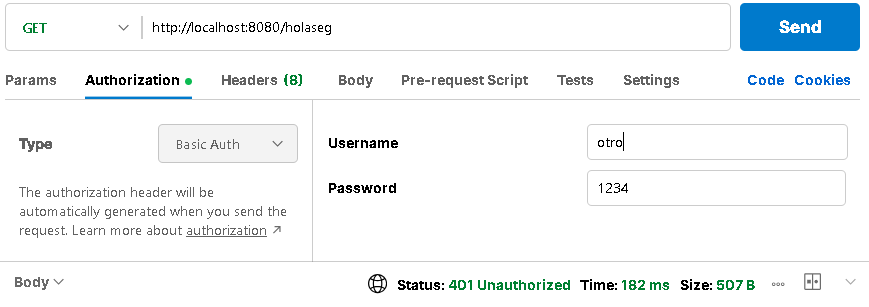
**Paso 3: Decodificar contraseñas**

Ya logramos encriptar nuestras contraseñas a la hora de guardarlas pero recordemos que a la hora de llevar a cabo la autenticación, el PasswordEncoder debe desencriptar la contraseña para validarla con la contraseña en texto plano que envíe el usuario que intente loguearse.

Por suerte, no es necesario implementar ningún tipo de código extra. Automáticamente, el PasswordEncoder al ya tener definido que trabajamos con BCrypt, cuando recibe una cadena encriptada, la compara desencriptada con la contraseña en texto plano que pueda enviar un usuario.

Probemos esto en Postman:





Como podemos observar, el PasswordEncoder hace correctamente la decodificación de la contraseña y, cuando esta se trate de 1234 valida la autenticación y, cuando se trate de 12345 o una incorrecta, al no coincidir, no permitirá el acceso a la información.

