
DATUM

El día de ayer vimos:

- Lectura de properties
- PathParam
- QueryParam
- Método Get

Agenda del día

- Processor
- Autenticación Básica
- Autenticación utilizando Jwt
- Swagger

Processor

Los procesadores personalizados en Apache Camel son útiles cuando necesitas realizar operaciones específicas o personalizadas en tus rutas de integración. Puedes utilizar procesadores personalizados para manipular, transformar o enriquecer los mensajes que fluyen a través de tus rutas.

```
from("direct:start")
    .process(new MyProcessor())
    .to("direct:finish");
```

Spring Boot Starter Security

Proporciona una amplia gama de funciones de seguridad, como autenticación, autorización, protección contra ataques CSRF (Cross-Site Request Forgery), manejo de sesiones, entre otras.

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

PasswordEncoder

Es una interfaz de servicio en Spring Security que se utiliza para codificar contraseñas. Su función principal es codificar las contraseñas de los usuarios antes de almacenarlas en la base de datos. La codificación de contraseñas es esencial para la seguridad, ya que evita el almacenamiento de contraseñas en texto plano.

```
PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}
```

BCryptPasswordEncoder

Es una implementación concreta de la interfaz PasswordEncoder que utiliza la función de hash fuerte BCrypt. BCrypt es un algoritmo de hash de contraseñas diseñado para ser lento y resistente a ataques de fuerza bruta.

```
PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}
```

SecurityFilterChain

Se encarga de proteger las URLs de la aplicación, validar los nombres de usuario y contraseñas enviados, redirigir a formularios de inicio de sesión, entre otras tareas relacionadas con la seguridad.

```
SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {

    http.csrf(csrf -> csrf.disable()).authorizeHttpRequests((authorize) -> {
        authorize.anyRequest().authenticated();
    }).httpBasic(Customizer.withDefaults());
    return http.build();
}
```

HttpSecurity

Se utiliza para definir reglas de seguridad que controlan qué recursos están protegidos y cómo se aplican las restricciones de acceso. Puede configurar puntos como autenticación, autorización, manejo de sesiones, entre otras opciones.

```
SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {

    http.csrf(csrf -> csrf.disable()).authorizeHttpRequests((authorize) -> {
        authorize.anyRequest().authenticated();
    }).httpBasic(Customizer.withDefaults());
    return http.build();
}
```

UserDetailsService

Utilizando esta interfaz se puede personalizar la forma en que Spring Security obtiene detalles específicos del usuario, como su nombre de usuario, contraseña y roles. Esto es esencial para la autenticación y la autorización.

```
UserDetailsService userDetailsService() {

    UserDetails admin = User.builder().username("admin").password(passwordEncoder().encode("admin")).roles("ADMIN")
            .build();

    return new InMemoryUserDetailsManager(admin);
}
```

Swagger

Swagger es una herramienta ampliamente utilizada en el desarrollo de aplicaciones para facilitar la documentación, prueba y consumo de APIs (Interfaces de Programación de Aplicaciones).

```
restConfiguration()
    .component("servlet")
    .bindingMode(RestBindingMode.auto)
    .apiContextPath("/api-doc")
    .apiProperty("api.title", "Ejercicio de Apache Camel")
    .apiProperty("api.description", "Uso del Framework Apache Camel operaciones matematicas")
    .apiProperty("api.version", "1.0.0");
```

DATUM

Gracias por la Atención