Seguridad, registro y login

Hasta este punto, hemos implementado la capa de datos y controladores que envían la información que se encuentra en nuestra base de datos. Ahora llega el momento de implementar la seguridad para poder realizar el inicio de sesión con diferentes roles.

- Implementar Spring Security.
- Modificar la configuración de autenticación.
- Modificar la configuración de SecurityFilterChain.
- Crear los roles "USER", "MANAGER", "ADMIN".
- Encriptar las contraseñas de la base de datos.
- Crear un controlador para el registro.
- Testear la API.

Como mencionamos antes, vamos a tener tres roles principales en nuestra API: User, Manager y Admin. Cada uno tendrá sus permisos especiales de acuerdo a su rol, y solo se podrán registrar Usuarios y Organizadores. Los Administradores solo podrán ser creados desde la cuenta de otro Administrador o desde la base de datos.

Para este sprint, es necesario aplicar las funcionalidades requeridas. Pero además, también se puede ampliar a la recuperación de contraseñas vía email y/o validación de cuentas con email al momento de registrarse. Para eso, puedes utilizar la dependencia Java Mail Sender, que permite el envío de emails desde nuestra API de forma sencilla y utilizando una configuración básica. Esta dependencia es escalable y permite el envío de mails más detallados o con archivos como plantilla de email.

Una vez completada la seguridad y el inicio de sesión de los usuarios, podemos pasar al siguiente sprint donde desarrollaremos la lógica de negocio de nuestra API.

Authentication Manager

Para modificar la forma en que autenticamos a nuestros usuarios, es necesario sobrescribir el método `init` de la clase `GlobalAuthenticationConfigurerAdapter`.

Para lograr esto, debemos copiar y pegar el siguiente código:

```
@Configuration
public class AuthSecurity extends GlobalAuthenticationConfigurerAdapter {
@Autowired
private CustomerRepository customerRepository;
@Override
public void init(AuthenticationManagerBuilder auth) throws Exception{
```

```
auth.userDetailsService(username -> {
Customer user = customerRepository.findByEmail(username);
if(user != null){
return new User(user.getEmail(),user.getPassword(),
AuthorityUtils.createAuthorityList(user.getRol().toString()));
}else {
throw new UsernameNotFoundException("Email invalid");
}
});
}}
```

Cuando nuestra API recibe una solicitud de inicio de sesión, verificará que el usuario exista en nuestra base de datos y que la contraseña coincida con la guardada. El problema aquí es que nuestra contraseña en la base de datos debe estar encriptada para que coincida con la contraseña ingresada por el usuario al iniciar sesión. Por lo tanto, debemos utilizar PasswordEncoder para poder realizar esta tarea. De lo contrario, se nos indicará que el inicio de sesión ha fallado.

SecurityFilterChain

Para proteger los recursos de nuestra API y modificar la forma de realizar el inicio de sesión, debemos realizar una serie de configuraciones en nuestros filtros de seguridad.

A continuación, dejaremos el código básico necesario para realizar estas configuraciones y permitir que la API permita el inicio de sesión y proteja algunos recursos. Esto puede ser modificado e incluso mejorado de muchas formas, pero para comenzar, este es el código mínimo necesario que debemos tener:

```
@EnableWebSecurity
@Configuration
public class FilterSecurity {

@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception{
http.authorizeHttpRequests( path -> {
path.requestMatchers("/index.html", "/v3/api-docs/**", "/swagger-ui/**").permitAll()
.requestMatchers(HttpMethod.POST, "/api/crearevento").hasAnyAuthority("ADMIN",
"MANAGER")
.requestMatchers("/api/events").hasAuthority("USER")
.anyRequest().denyAll();
} )
.csrf(httpSecurityCsrfConfigurer -> httpSecurityCsrfConfigurer.disable())
```

```
.headers(httpSecurityHeadersConfigurer ->
httpSecurityHeadersConfigurer.frameOptions(
frameOptionsConfig -> frameOptionsConfig.disable()))
.formLogin( formLogin -> {
formLogin.loginPage("/index.html")
.loginProcessingUrl("/api/login")
.usernameParameter("email")
.passwordParameter("password")
.permitAll()
.successHandler((request, response, authentication) ->
clearAuthenticationAttributes(request))
.failureHandler((request, response, exception) -> response.sendError(401, "Invalid
user or password"));
})
.logout(httpSecurityLogoutConfigurer ->
httpSecurityLogoutConfigurer
.logoutUrl("/api/logout")
.logoutSuccessHandler(new HttpStatusReturningLogoutSuccessHandler())
.deleteCookies("JSESSIONID"))
.rememberMe(Customizer.withDefaults());
return http.build();
}
private void clearAuthenticationAttributes(HttpServletRequest request) {
HttpSession session = request.getSession(false);
if (session != null) {
session.removeAttribute(WebAttributes.AUTHENTICATION_EXCEPTION);
}
}
}
```

Aquí podemos observar que Spring Security utiliza funciones Lambda para cada uno de los métodos dentro de HttpSecurity. Además, es importante destacar que para que OpenAPI 3 funcione correctamente, debe permitirse el acceso a "**/v3/api-docs/" y "/swagger-ui/**". De lo contrario, no podremos acceder a la interfaz para realizar pruebas de los endpoints.

También podemos ver que tenemos deshabilitado CSRF y FrameOptions. La desactivación de CSRF se debe a que no estaremos enviando formularios desde nuestra API, sino que estaremos enviando y recibiendo información en formato JSON. La desactivación de FrameOptions es para poder hacer uso de la consola de H2.

FormLogin nos permite modificar el inicio de sesión por defecto que utiliza Spring Security con un formulario. Generamos un endpoint con la ruta "/api/login" a donde enviaremos los queryParams del correo electrónico y la contraseña para iniciar sesión. En caso de que el usuario no esté logueado, será redirigido a index.html.

Spring Security

Spring Security es un marco de seguridad altamente personalizable y fácil de usar para aplicaciones Java. Proporciona una amplia gama de características de seguridad para proteger tu aplicación contra amenazas, desde la autenticación básica hasta la autorización avanzada y la protección contra ataques comunes.

Una de las características principales de Spring Security es su capacidad para manejar la autenticación de usuarios de manera flexible y segura. Puedes configurar la autenticación utilizando diversos mecanismos, como la autenticación basada en formularios, la autenticación basada en tokens, la autenticación basada en OAuth, entre otros. Además, Spring Security proporciona soporte para la autenticación de usuarios almacenados en una base de datos, en servicios LDAP, en proveedores de autenticación externos y más.

Además de la autenticación, Spring Security también ofrece un sólido soporte para la autorización, lo que te permite definir quién tiene acceso a qué recursos dentro de tu aplicación. Puedes configurar reglas de autorización utilizando expresiones de seguridad, anotaciones en métodos y clases, o mediante configuración basada en roles.

Otra característica importante de Spring Security es su capacidad para proteger tu aplicación contra ataques comunes, como la falsificación de solicitudes entre sitios (CSRF), los ataques de inyección de código, los ataques de denegación de servicio (DoS) y más. Spring Security incluye mecanismos integrados para mitigar estos tipos de ataques y proteger tu aplicación de forma proactiva.

Basic Authentication

La autenticación básica (Basic Authentication) es un método simple de autenticación utilizado en aplicaciones web para verificar la identidad de los usuarios que intentan acceder a recursos protegidos. Este método se basa en el envío de credenciales de usuario (nombre de usuario y contraseña) en texto plano a través de las cabeceras HTTP de la solicitud.

Cuando un usuario intenta acceder a un recurso protegido, el servidor solicita las credenciales de autenticación. El usuario proporciona su nombre de usuario y

contraseña, que se codifican en texto plano y se envían al servidor como parte de la cabecera `Authorization` de la solicitud HTTP. El servidor verifica las credenciales comparándolas con las credenciales almacenadas en su base de datos de usuarios autorizados. Si las credenciales son válidas, el servidor permite al usuario acceder al recurso solicitado; de lo contrario, se le niega el acceso y se le solicitan credenciales válidas nuevamente.

La autenticación básica es simple de implementar y ampliamente compatible con los navegadores web y otras herramientas de cliente HTTP. Sin embargo, tiene algunas limitaciones importantes, como la falta de cifrado de las credenciales durante la transmisión, lo que las hace vulnerables a ataques de intercepción. Por esta razón, la autenticación básica generalmente se usa en conjunto con HTTPS (HTTP seguro) para garantizar la seguridad de las credenciales durante la transmisión.

A pesar de sus limitaciones de seguridad, la autenticación básica sigue siendo útil en situaciones donde la simplicidad y la facilidad de implementación son más importantes que la seguridad extrema, como en aplicaciones internas o en desarrollo rápido de prototipos.

ecurityFilterChain

SecurityFilterChain es una característica clave de Spring Security que permite configurar múltiples filtros de seguridad en una cadena para procesar las solicitudes HTTP entrantes en una aplicación web. Cada filtro en la cadena se encarga de una tarea específica relacionada con la seguridad, como autenticación, autorización, protección contra ataques, entre otros.

La SecurityFilterChain está diseñada para proporcionar una forma flexible y modular de configurar la seguridad en una aplicación Spring. Permite encadenar varios filtros de seguridad en un orden específico, donde cada filtro puede procesar una solicitud de manera independiente o pasar la solicitud al siguiente filtro en la cadena. Esto proporciona un alto grado de personalización y control sobre cómo se aplica la seguridad en diferentes partes de la aplicación.

Uno de los usos más comunes de SecurityFilterChain es en la configuración de Spring Security a través de la clase `WebSecurityConfigurerAdapter`. Esta clase permite definir múltiples SecurityFilterChain, cada una con su propia configuración de seguridad específica. Por ejemplo, puedes tener una SecurityFilterChain para autenticar solicitudes a recursos públicos y otra para autenticar solicitudes a recursos protegidos.

Cada SecurityFilterChain puede contener una variedad de filtros de seguridad, como `UsernamePasswordAuthenticationFilter` para la autenticación básica,

`AccessControlFilter` para la autorización basada en roles, `CsrfFilter` para la protección contra ataques CSRF, entre otros. Esto permite adaptar la seguridad de la aplicación según los requisitos específicos de cada parte de la aplicación.