**OAuth2**

****

**OAuth2** es un protocolo de autorización ampliamente utilizado en el mundo de la seguridad informática y la gestión de identidad en aplicaciones web y móviles. Su objetivo principal es permitir que un usuario autorice a una aplicación o servicio a acceder a sus recursos protegidos en un servidor sin necesidad de compartir sus credenciales.

En esencia, **OAuth2** proporciona un mecanismo seguro y estandarizado para que los usuarios autoricen a terceros a acceder a sus datos almacenados en un servidor, como sus perfiles en redes sociales, cuentas de correo electrónico, fotos, contactos, entre otros. Esto se logra mediante la emisión de tokens de acceso que los terceros pueden usar para realizar solicitudes en nombre del usuario autorizado.

**Roles y Flujos en OAuth2**

El protocolo OAuth2 define diferentes roles y flujos para llevar a cabo el proceso de autorización de manera segura:

1. **Propietario de los recursos (Resource Owner)**: Es el usuario final que posee los recursos protegidos, como datos personales o archivos.
2. **Cliente (Client)**: Es la aplicación o servicio que solicita acceso a los recursos en nombre del propietario de los recursos. Por ejemplo, una aplicación móvil o un sitio web.
3. **Servidor de autorización (Authorization Server)**: Es el servidor responsable de autenticar al propietario de los recursos y de emitir tokens de acceso al cliente después de que el propietario haya dado su consentimiento.
4. **Servidor de recursos (Resource Server)**: Es el servidor que almacena y gestiona los recursos protegidos. Verifica la validez de los tokens de acceso y decide si permitir o denegar el acceso a los recursos solicitados por el cliente.

El protocolo OAuth2 define varios flujos de autorización para diferentes escenarios de uso, como el flujo de autorización estándar (authorization code), el flujo implícito (implicit flow), el flujo de contraseña (password flow), el flujo de credenciales de cliente (client credentials flow), entre otros. Cada flujo tiene sus propias características y se elige según los requisitos de seguridad y las capacidades del cliente.

**Ventajas de OAuth2**

Utilizar OAuth2 en el desarrollo de aplicaciones web y móviles ofrece una serie de ventajas significativas en términos de seguridad, flexibilidad y experiencia del usuario, veamos algunas de ellas:

1. **Seguridad Mejorada:** Uno de los aspectos más importantes de OAuth2 es su capacidad para mejorar la seguridad al evitar la exposición de las credenciales de usuario. Al utilizar tokens de acceso en lugar de contraseñas, OAuth2 reduce significativamente el riesgo de robo de credenciales y compromiso de la cuenta del usuario.
2. **Control Granular de Acceso:** OAuth2 permite un control granular sobre los recursos a los que pueden acceder los terceros en nombre del usuario. Los usuarios pueden autorizar selectivamente a las aplicaciones a acceder a recursos específicos, lo que garantiza la privacidad y la seguridad de sus datos.
3. **Escalabilidad:** El protocolo OAuth2 es altamente escalable y puede manejar grandes volúmenes de solicitudes de autenticación y autorización sin comprometer el rendimiento. Esto lo hace ideal para aplicaciones web y móviles de alto tráfico que requieren un sistema robusto de gestión de identidad.
4. **Interoperabilidad:** OAuth2 es un estándar abierto y ampliamente adoptado en la industria, lo que garantiza la interoperabilidad entre diferentes servicios y plataformas. Esto facilita la integración con proveedores de servicios externos y la construcción de ecosistemas de aplicaciones interconectadas.
5. **Experiencia de Usuario Mejorada:** Al permitir que los usuarios autoricen el acceso a sus datos sin compartir sus credenciales, OAuth2 proporciona una experiencia de usuario más fluida y sin fricciones. Esto aumenta la confianza del usuario y mejora la adopción de la aplicación.

**¿Por qué utilizar OAuth2 con Spring Security?**

En combinación con **Spring Security**, **OAuth2** ofrece una solución completa y robusta para la autenticación y autorización en aplicaciones Spring y existen diferentes causas para usar ambas tecnologías combinadas, entre ellas:

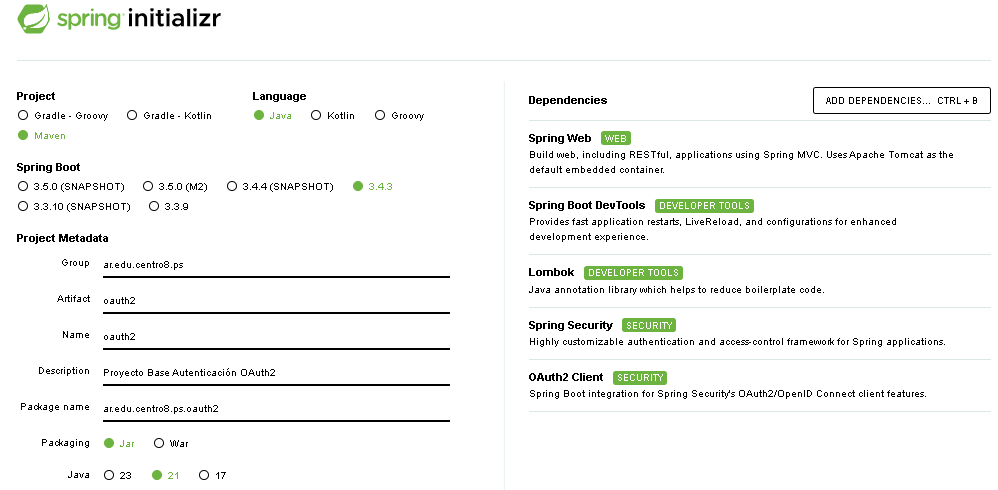
1. **Integración Nativa:** Spring Security proporciona soporte nativo para OAuth2, lo que facilita la implementación de la autenticación y autorización basada en OAuth2 en aplicaciones Spring.
2. **Configuración Simplificada:** Spring Security simplifica la configuración y la gestión de la seguridad en aplicaciones Spring, lo que permite una implementación más rápida y eficiente de OAuth2.
3. **Personalización Flexible:** Spring Security ofrece una amplia gama de opciones de configuración y personalización que permiten adaptar OAuth2 a los requisitos específicos de seguridad de la aplicación.
4. **Integración con Otros Módulos:** Spring Security se integra perfectamente con otros módulos de Spring, como Spring Boot y Spring MVC, lo que facilita la creación de aplicaciones web seguras y de alto rendimiento.
5. **Comunidad Activa:** Spring Security cuenta con una comunidad activa de desarrolladores y una amplia documentación que proporciona soporte y recursos adicionales para implementar OAuth2 de manera efectiva en aplicaciones Spring.

## ****OAuth2 con Spring Security****

Antes de comenzar con el paso a paso tenés que saber que, dada la complejidad que implica implementar OAuth2 con Spring Security vamos dividir el paso a paso de la implementación en 3 partes: Introducción (esta lección dónde estás ahora), Autenticación via GitHub y Autenticación vía Google.

##### **Paso 1: Dependencias en Initializr**

 Vamos a usar las siguientes dependencias:



[https://start.spring.io/#!type=maven-project&language=java&platformVersion=3.4.3&packaging=jar&jvmVersion=21&groupId=ar.edu.centro8.ps&artifactId=oauth2&name=oauth2&description=Proyecto%20Base%20Autenticaci%C3%B3n%20OAuth2&packageName=ar.edu.centro8.ps.oauth2&dependencies=web,devtools,lombok,security,oauth2-client](https://start.spring.io/#!type=maven-project&language=java&platformVersion=3.4.3&packaging=jar&jvmVersion=21&groupId=ar.edu.centro8.ps&artifactId=oauth2&name=oauth2&description=Proyecto%20Base%20Autenticaci%C3%B3n%20OAuth2&packageName=ar.edu.centro8.ps.oauth2&dependencies=web,dev)

##### **Paso 2: Crear un Controller**

Haremos nuevamente dos clásicos endpoints de HelloWorld, uno securizado y otro no securizado para hacer pruebas:

package ar.edu.centro8.ps.oauth2.controller;

import org.springframework.security.access.prepost.PreAuthorize;

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

import org.springframework.web.bind.annotation.RestController;

@RestController

@PreAuthorize("denyAll()")

public class SaludoController {

    @GetMapping("/decirhola")

    @PreAuthorize("permitAll()")

    public String decirHola(){

        return "Hola! Este endpoint no esta securizado";

    }

    @GetMapping("/decirholasec")

    @PreAuthorize("isAuthenticated()")

    public String decirHolaSec(){

        return "Hola! Este endpoint esta securizado";

    }

}

##### **Paso 3: Configuración de application.properties + variables de entorno**

spring.application.name=oauth2

#Configuraciones de seguridad

spring.security.user.name=${SS\_USER}

spring.security.user.password=${SS\_PASSWORD}

spring.security.user.roles=${ROLE}

**Recordá**configurar tus variables de entorno en tu IDE. En este caso vamos a usar los siguientes datos de prueba:

* **SS\_USER:** centro8
* **SS\_PASSWORD:** 1234
* **ROLE:** ADMIN

##### **Paso 4: Configuraciones varias**

Vamos a crear:

* Paquete config y dentro el paquete security
* Creamos nuestra clase Security Config
* Creamos nuestro método securityFilterChain

Todo mediante las diferentes porciones de código a continuación (tal como venimos haciendo en proyectos anteriores).

#### ****Clase Security Config****

package ar.edu.centro8.ps.oauth2.config.security;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.config.Customizer;

import org.springframework.security.config.annotation.method.configuration.EnableMethodSecurity;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

import org.springframework.security.web.SecurityFilterChain;

@Configuration

@EnableWebSecurity

@EnableMethodSecurity

public class SecurityConfig {

    @Bean

    public SecurityFilterChain securityFilterChain (HttpSecurity httpSecurity) throws Exception {

        return httpSecurity

                .csrf(csrf -> csrf.disable())

                .formLogin(Customizer.withDefaults())

                .oauth2Login(Customizer.withDefaults())

                .build();

    }

}

**formLogin(Customizer.withDefaults()):**  
Esta línea configura la autenticación basada en formularios en tu aplicación. Con formLogin(), Spring Security proporciona un formulario de inicio de sesión predeterminado en el que los usuarios pueden ingresar sus credenciales (nombre de usuario y contraseña) para autenticarse. El método Customizer.withDefaults() se utiliza para aplicar la configuración predeterminada proporcionada por Spring Security para la autenticación basada en formularios. Esto incluye la configuración de rutas predeterminadas para el formulario de inicio de sesión (/login) y la página de error de inicio de sesión (/login?error), entre otros.

* **oauth2Login(Customizer.withDefaults()):**Esta línea configura la autenticación mediante OAuth 2.0 en tu aplicación. Con oauth2Login(), Spring Security proporciona un flujo de inicio de sesión predeterminado para permitir que los usuarios inicien sesión utilizando proveedores de identidad externos que admiten OAuth 2.0, como Google, Facebook o GitHub. El método Customizer.withDefaults() se utiliza aquí de manera similar para aplicar la configuración predeterminada proporcionada por Spring Security para la autenticación mediante OAuth 2.0

#### ****Saludo Controller****

package ar.edu.centro8.ps.oauth2.controller;

import org.springframework.security.access.prepost.PreAuthorize;

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

import org.springframework.web.bind.annotation.RestController;

@RestController

@PreAuthorize("denyAll()")

public class SaludoController {

    @GetMapping("/decirhola")

    @PreAuthorize("permitAll()")

    public String decirHola(){

        return "Hola! Este endpoint no esta securizado";

    }

    @GetMapping("/decirholasec")

    @PreAuthorize("isAuthenticated()")

    public String decirHolaSec(){

        return "Hola! Este endpoint esta securizado";

    }

}