Diseño y Construcción de Microservicios

LAB Microservicios y OAuth Servidor de Autorizaciones

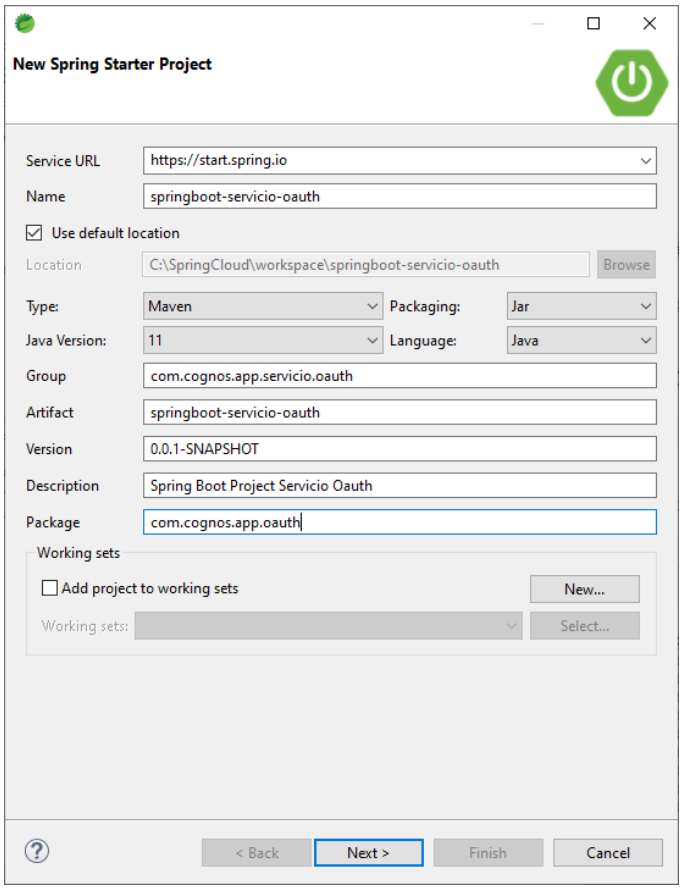
Objetivos

- Mostrar al participante el procedimiento para la configuración de ambientes de Microservicios usando el Servidor de Autorizaciones OAuth2

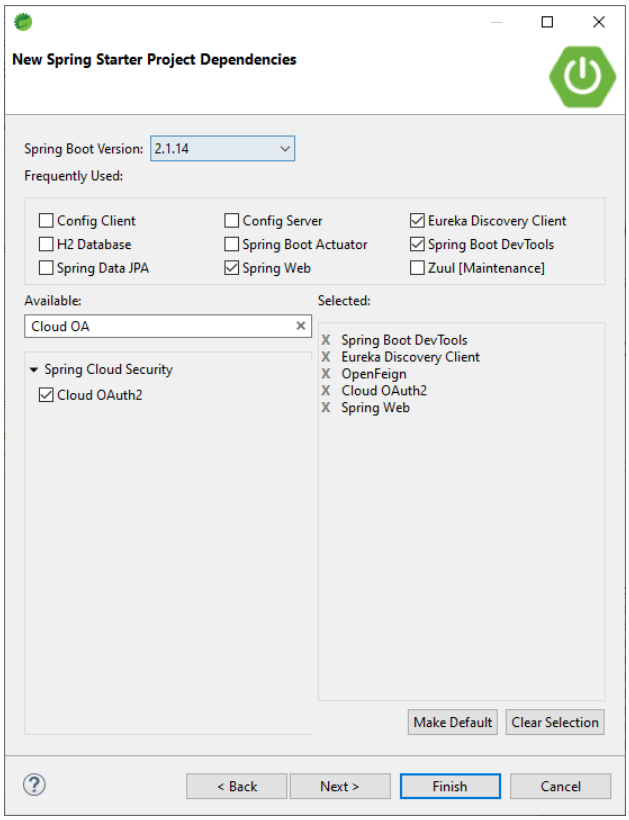
Procedimiento

Creación del Proyecto

1. Crea el proyecto **springboot-servicio-oauth**.



2. Agrega al proyecto **springboot-servicio-oauth**, las dependencias "**Spring Boot DevTools**","**Eureka Discovery Client**","**OpenFeign**","**Cloud OAuth2**" y "Spring Web".



3. Agrega la dependencia springboot-servicio-usuarios-commons al proyecto **springboot-servicio-oauth**. Edita el archivo pom.xml del proyecto y agrega lo siguiente: (sugerencia copia el contenido desde el archivo pom.xml del proyecto springboot-servicio-usuarios)

```
<dependency>
   <groupId>com.cognos.app.usuarios.commons</groupId>
   <artifactId>springboot-servicio-usuarios-commons</artifactId>
   <version>0.0.1-SNAPSHOT</version>
</dependency>
```

4. Edita el archivo **pom.xml** de proyecto **springboot-servicio-oauth** y excluye la dependencia de **spring-boot-starter-data-jpa** que viene la librería **`springboot-servicio-usuarios-commons`**.

```
…
<dependency>
      <groupId>com.cognos.app.usuarios.commons</groupId>
      <artifactId>springboot-servicio-usuarios-commons</artifactId>
      <version>0.0.1-SNAPSHOT</version>
      <exclusions>
            <exclusion>
                  <groupId>org.springframework.boot</groupId>
                  <artifactId>spring-boot-starter-data-jpa</artifactId>
```

```
            </exclusion>
        </exclusions>
</dependency>

…
```

5. En el proyecto **springboot-servicio-oauth** anota la clase SpringbootServicioOauthApplication.java con la anotación @EnableEurekaClient.

```
package com.cognos.app.oauth;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;

@EnableEurekaClient
@SpringBootApplication
public class SpringbootServicioOauthApplication {

 public static void main(String[] args) {
        SpringApplication.run(SpringbootServicioOauthApplication.class, args);
 }

}
```

6. Edita el archivo application.properties del proyecto **springboot-servicio-oauth.**

```
spring.application.name=servicio-oauth
server.port=9100

eureka.client.service-url.defaultZone=http://localhost:8761/eureka
```

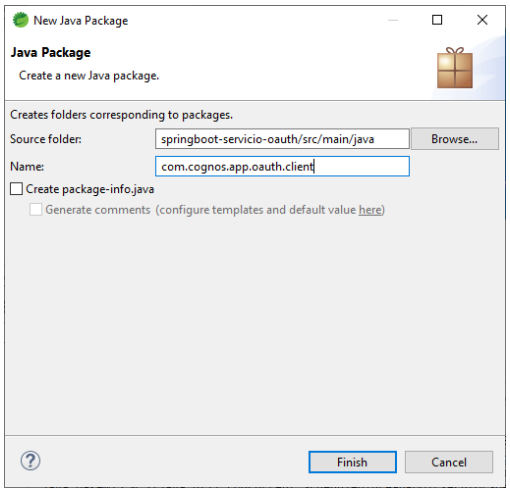
Implementación del Cliente Feign

7. Implementa el cliente feign en el proyecto **springboot-servicio-oauth** para que se comunique con el servicio servicio-usuarios.
    a. Agrega la anotación **@EnableFeignClients** a la clase **SpringbootServicioOauthApplication.java**.

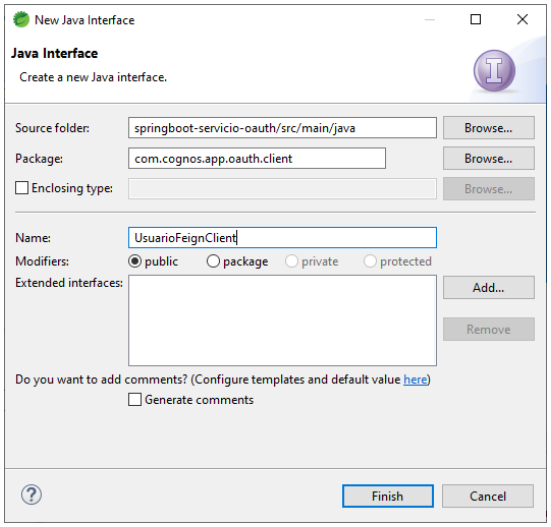
```
package com.cognos.app.oauth;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
import org.springframework.cloud.openfeign.EnableFeignClients;
```

```
@EnableFeignClients
@EnableEurekaClient
@SpringBootApplication
public class SpringbootServicioOauthApplication {

  public static void main(String[] args) {
          SpringApplication.run(SpringbootServicioOauthApplication.class, args);
  }
}
```

b. Crea el paquete com.cognos.app.oauth.client en el proyecto.



c. Agrega la interface **UsuarioFeignClient** al proyecto.



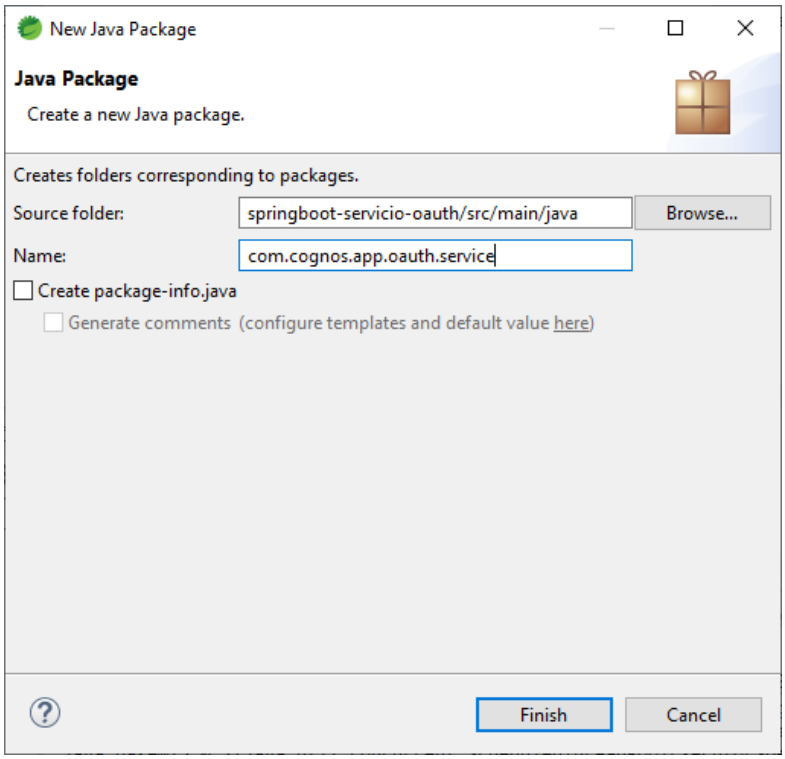
d. Edita la interface UsuarioFeignClient.java para indicar el nombre del servicio y mapear el método a invocar.

```
package com.cognos.app.oauth.client;

import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;

import com.cognos.app.usuarios.commons.model.entity.Usuario;

@FeignClient(name="servicio-usuarios")
public interface UsuarioFeignClient {

    @GetMapping("/usuarios/search/buscar-usuario")
    public Usuario findByUsername(@RequestParam("nombre")String
username);

}
```

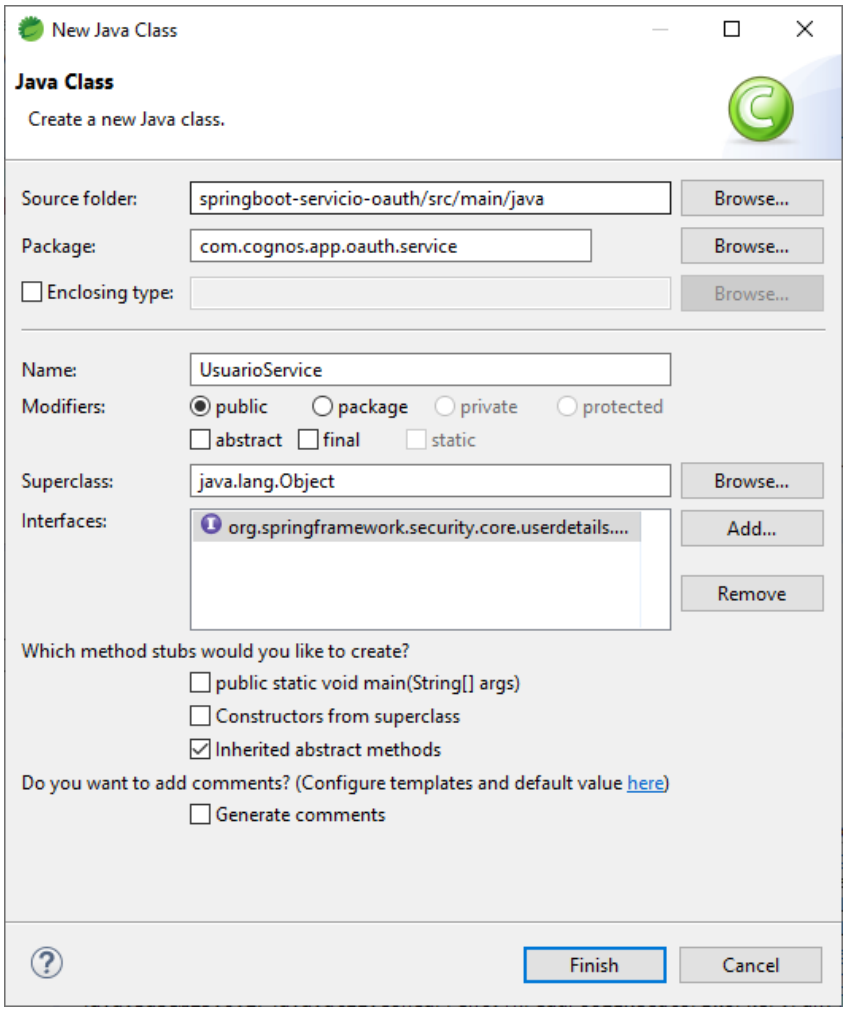
e. Guarda los cambios. [Ctrl] + [Shift] + [S]

Implementando las Clases para la Autenticación

8. Crea el paquete **com.cognos.app.oauth.service** y crea la clase **UserDetails** que implemente la interface de "Spring Security" llamada **UserDetailsService**.



Crea la clase UsuarioService.java



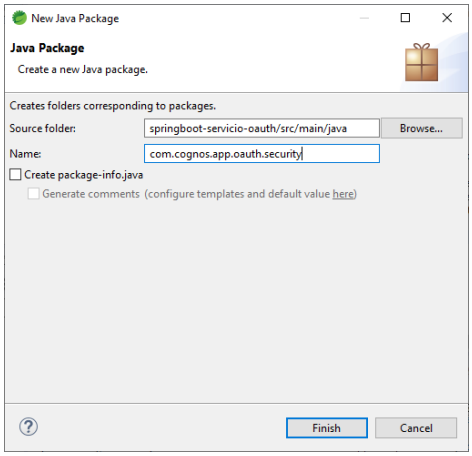
9. Edita la clase UsuarioService.java y conviertelo en un servicio de Spring.

```
package com.cognos.app.oauth.service;

import java.util.List;
import java.util.stream.Collectors;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
```

```java
import com.cognos.app.oauth.client.UsuarioFeignClient;
import com.cognos.app.usuarios.commons.model.entity.Usuario;

@Service
public class UsuarioService implements UserDetailsService {

   private Logger log = LoggerFactory.getLogger(UsuarioService.class);

   @Autowired
   private UsuarioFeignClient cliente;

   @Override
   public UserDetails loadUserByUsername(String username) throws
                                                    UsernameNotFoundException {
   Usuario usuario = cliente.findByUsername(username);
   if(usuario == null) {
       String mensaje = "Error: El usuario " + username+ " no existe. ";
       log.error(mensaje);
       throw new UsernameNotFoundException(mensaje);
   }
   List<GrantedAuthority> autorities = usuario.getRoles()
                          .stream()
                          .map(role -> new
                              SimpleGrantedAuthority(role.getNombre()))
                           .peek(authority -> log.info("Role:" +
                              authority.getAuthority()))
                           .collect(Collectors.toList());
             log.info("Usuario autenticado:"+username);
             return new User(usuario.getUsername(), usuario.getPassword(),
                              usuario.getEnabled(), true, true,true, autorities);
       }
}
```

Registrando las Clases en Spring Security

10. Crea el paquete **com.cognos.app.oauth.security** en el proyecto.



11. En el paquete **com.cognos.app.oauth.security** crea la clase SpringSecurityConfig.java

```
package com.cognos.app.oauth.security;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.config.annotation.authentication.builders.Authentic
ationManagerBuilder;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConf
igurerAdapter;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

@Configuration
public class SpringSecurityConfig extends WebSecurityConfigurerAdapter {

@Autowired
private UserDetailsService usuarioService;

@Bean
public BCryptPasswordEncoder passwordEncoder() {
	return new BCryptPasswordEncoder();
}

@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
auth.userDetailsService(this.usuarioService).passwordEncoder(passwordEncoder());
}

@Override
@Bean
protected AuthenticationManager authenticationManager() throws Exception {
	return super.authenticationManager();
}

}
```

Configurando el Servidor de Autorizaciones

12. En el paquete **com.cognos.app.oauth.security** agrega la clase **AuthorizationServerConfig.java**.

```
package com.cognos.app.oauth.security;

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

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import
org.springframework.security.oauth2.config.annotation.configurers.ClientDetailsS
erviceConfigurer;
import
org.springframework.security.oauth2.config.annotation.web.configuration.Authoriz
ationServerConfigurerAdapter;
import
org.springframework.security.oauth2.config.annotation.web.configuration.EnableAu
thorizationServer;
import
org.springframework.security.oauth2.config.annotation.web.configurers.Authorizat
ionServerEndpointsConfigurer;
import
org.springframework.security.oauth2.config.annotation.web.configurers.Authorizat
ionServerSecurityConfigurer;
import
org.springframework.security.oauth2.provider.token.store.JwtAccessTokenConverter
;
import org.springframework.security.oauth2.provider.token.store.JwtTokenStore;

@Configuration
@EnableAuthorizationServer
public class AuthorizationServerConfig extends
                                        AuthorizationServerConfigurerAdapter {

       @Autowired
       private BCryptPasswordEncoder passwordEncoder;

       @Autowired
       private AuthenticationManager authenticationManager;

       @Override
       public void configure(AuthorizationServerSecurityConfigurer security)
                                                                  throws Exception {
              super.configure(security);
       }

       @Override
       public void configure(ClientDetailsServiceConfigurer clients) throws
                                                                     Exception {
              super.configure(clients);
       }

       @Override
       public void configure(AuthorizationServerEndpointsConfigurer endpoints)
                                                                  throws Exception {
              endpoints.authenticationManager(authenticationManager)
                       .tokenStore(tokenStore())
                      .accessTokenConverter(accessTokenConverter());

       }
```

```
	@Bean
	public JwtTokenStore tokenStore() {
		// Para almacena el token necesitamos convertirlo
		return new JwtTokenStore(accessTokenConverter());
	}

	@Bean
	public JwtAccessTokenConverter accessTokenConverter() {
		JwtAccessTokenConverter tokenConverter = new
								JwtAccessTokenConverter();
		String key = "algo_secreto";
		tokenConverter.setSigningKey(key);
		return tokenConverter;
	}

}
```

Registrando las Aplicaciones Clientes

13. Edita la clase **AuthorizationServerConfig.java** y registra los clientes utilizando los métodos de la clase.

```
package com.cognos.app.oauth.security;
import org.springframework.cloud.context.config.annotation.RefreshScope;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import
org.springframework.security.oauth2.config.annotation.configurers.ClientDetailsS
erviceConfigurer;
import
org.springframework.security.oauth2.config.annotation.web.configuration.Authoriz
ationServerConfigurerAdapter;
import
org.springframework.security.oauth2.config.annotation.web.configuration.EnableAu
thorizationServer;
import
org.springframework.security.oauth2.config.annotation.web.configurers.Authorizat
ionServerEndpointsConfigurer;
import
org.springframework.security.oauth2.config.annotation.web.configurers.Authorizat
ionServerSecurityConfigurer;
import
org.springframework.security.oauth2.provider.token.store.JwtAccessTokenConverter
;
import org.springframework.security.oauth2.provider.token.store.JwtTokenStore;

@RefreshScope
@Configuration
```

```java
@EnableAuthorizationServer
public class AuthorizationServerConfig extends
AuthorizationServerConfigurerAdapter {

	@Autowired
	private BCryptPasswordEncoder passwordEncoder;

	@Autowired
	private AuthenticationManager authenticationManager;

	@Override
	public void configure(AuthorizationServerSecurityConfigurer security)
	                                                        throws Exception {
		security.tokenKeyAccess("permitAll()")
			.checkTokenAccess("isAuthenticated()");
	}

	@Override
	public void configure(ClientDetailsServiceConfigurer clients) throws
	                                                              Exception {
		// frontendapp es el id de la aplicacion
		clients.inMemory().withClient("frontendapp")
				.secret(passwordEncoder.encode("12345"))
				.scopes("read","write")
				.authorizedGrantTypes("password","refresh_token")
				.accessTokenValiditySeconds(3600)
				.refreshTokenValiditySeconds(3600)
				.and()
				.withClient("androidapp")
				.secret(passwordEncoder.encode("54321"))
				.scopes("read","write")
				.authorizedGrantTypes("password","refresh_token")
				.accessTokenValiditySeconds(3600)
				.refreshTokenValiditySeconds(3600);
	}

	@Override
	public void configure(AuthorizationServerEndpointsConfigurer endpoints)
	                                                        throws Exception {
		endpoints.authenticationManager(authenticationManager)
			 .tokenStore(tokenStore())
			.accessTokenConverter(accessTokenConverter());

	}
	@Bean
	private JwtTokenStore tokenStore() {
		// Para almacena el token necesitamos convertirlo
		return new JwtTokenStore(accessTokenConverter());
	}

	@Bean
	public JwtAccessTokenConverter accessTokenConverter() {
		JwtAccessTokenConverter tokenConverter = new
                                                JwtAccessTokenConverter();
		String key = "algo_secreto";
```

```
            tokenConverter.setSigningKey(key);
            return tokenConverter;
        }

}
```