

Diseño y Construcción de Microservicios

LAB Microservicios – Información Adicional del Token

Objetivos

- Mostrar al participante el procedimiento para agregar información adicional en el token.

Procedimiento

1. En el proyecto **springboot-servicio-oauth** agrega la interface **IUsuarioService.java** en el paquete **com.cognos.app.oauth.service**.

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

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

public interface IUsuarioService {

    public Usuario findByUsername(String username);

}
```

2. Implementamos la interface IUsuarioService en la clase UsuarioService del paquete **com.cognos.app.oauth.service**.

```
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;

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

@Service
public class UsuarioService implements IUsuarioService, 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> authorities = 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,
            authorities);
}

@Override
public Usuario findByUsername(String username) {
    Usuario usuario = cliente.findByUsername(username);
    return usuario;
}
}

```

3. En el proyecto **springboot-servicio-oauth** , en el paquete **com.cognos.app.oauth.security** crea una nueva clase llamada **InfoAdicionalToken.java**.

```

package com.cognos.app.oauth.security;

import java.util.HashMap;
import java.util.Map;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.oauth2.common.DefaultOAuth2AccessToken;
import org.springframework.security.oauth2.common.OAuth2AccessToken;
import org.springframework.security.oauth2.provider.OAuth2Authentication;
import org.springframework.security.oauth2.provider.token.TokenEnhancer;
import org.springframework.stereotype.Component;

import com.cognos.app.oauth.service.IUsuarioService;
import com.cognos.app.usuarios.commons.model.entity.Usuario;

@Component
public class InfoAdicionalToken implements TokenEnhancer {

```

```

@Autowired
private IUserarioService usuarioService;

@Override
public OAuth2AccessToken enhance(OAuth2AccessToken accessToken,
                                OAuth2Authentication authentication) {
    Map<String, Object> info = new HashMap<String, Object>();
    Usuario usuario = usuarioService.findByUsername(authentication.getName());
    info.put("nombre", usuario.getNombre());
    info.put("apellido", usuario.getApellido());
    info.put("Email", usuario.getEmail());
    ((DefaultOAuth2AccessToken) accessToken).setAdditionalInformation(info);
    return accessToken;
}
}

```

4. Modificar el servidor de autorizaciones. Edita la clase **AuthorizationServerConfig.java** del paquete **com.cognos.app.oauth.security**.

```

package com.cognos.app.oauth.security;

import java.util.Arrays;

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.ClientDetailsServiceConfigurer;
import org.springframework.security.oauth2.config.annotation.web.configuration.AuthorizationServerConfigurerAdapter;
import org.springframework.security.oauth2.config.annotation.web.configuration.EnableAuthorizationServer;
import org.springframework.security.oauth2.config.annotation.web.configurers.AuthorizationServerEndpointsConfigurer;
import org.springframework.security.oauth2.config.annotation.web.configurers.AuthorizationServerSecurityConfigurer;
import org.springframework.security.oauth2.provider.token.TokenEnhancerChain;
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 InfoAdicionalToken infoAdicionalToken;

    @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 {
        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 {
        TokenEnhancerChain tokenEnhancerChain =new TokenEnhancerChain();

        tokenEnhancerChain.setTokenEnhancers(Arrays.asList(infoAdicionalToken
            ,accessTokenConverter()));
        endpoints.authenticationManager(authenticationManager)
            .tokenStore(tokenStore())
            .accessTokenConverter(accessTokenConverter())
            .tokenEnhancer(tokenEnhancerChain);
    }

    @Bean
    public JwtTokenStore tokenStore() {
        // Para almacenar 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;
}
}

```

5. Edita el archivo application.properties del servicio servicio-zuul-server.

```

spring.application.name=servicio-zuul-server
server.port=8090

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

zuul.routes.usuarios.service-id=servicio-usuarios
zuul.routes.usuarios.path=/api/usuarios/**

zuul.routes.productos.service-id=servicio-productos
zuul.routes.productos.path=/api/productos/**

zuul.routes.items.service-id=servicio-items
zuul.routes.items.path=/api/items/**

zuul.routes.oauth.service-id=servicio-oauth
zuul.routes.oauth.path=/api/security/**

#excluir las cookies de los headers
zuul.routes.oauth.sensitive-headers=Cookie,Set-Cookie

hystrix.command.default.execution.isolation.thread.timeoutInMilliseconds: 20000
ribbon.ConnectTimeout: 3000
ribbon.ReadTimeout: 10000

```

6. Prueba con Postman que el token tiene información adicional.

Realiza una petición de prueba y copia el token en jwt.io y verifica que la información adicional como apellido, email se muestra.

