Integración Keycloak con microservicios

Keycloak en servidor de recurso

- El servidor de recurso necesita comunicar con Keycloak para la verificación del token JWT.
- Además de Spring Security, requiere el starter oauth2-resource-server:

```
<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-oauth2-resource-server</artifactId>
</dependency>
```

Propiedades configuración Keycloak

- En el archivo .properties o .yml se deben definir una serie de propiedades para la comunicación con Keycloak:

```
spring:
  security:
    oauth2:
      resourceserver:
        jwt:
          issuer-uri: http://localhost:8080/realms/ContactosRealm
          jwk-set-uri: ${spring.security.oauth2.resourceserver.jwt.issuer-uri}/protocol/openid-connect/certs

jwt:
  auth:
    converter:
      resource-id: login
      principal-attribute: preferred_username
```

Propiedades conversor

- Se define una clase que extraiga las propiedades establecidas para el conversor:

```
@Data
@Validated
@Configuration
@ConfigurationProperties(prefix = "jwt.auth.converter")
public class JwtAuthConverterProperties {
        private String resourceId;
        private String principalAttribute;
}
```

Configuración conversor

- Se define la clase de configuración del conversor:

```
@Component
public class JwtAuthConverter implements Converter<Jwt, AbstractAuthenticationToken> {

    private final JwtGrantedAuthoritiesConverter jwtGrantedAuthoritiesConverter = new JwtGrantedAuthoritiesConverter();
    private final JwtAuthConverterProperties properties;
    public JwtAuthConverter(JwtAuthConverterProperties properties) {
                        this.properties = properties;
    }
    @Override
    public AbstractAuthenticationToken convert(Jwt jwt) {
        Collection<GrantedAuthority> authorities = Stream.concat(
                jwtGrantedAuthoritiesConverter.convert(jwt).stream(),
                extractResourceRoles(jwt).stream()).collect(Collectors.toSet());
       return new JwtAuthenticationToken(jwt, authorities, getPrincipalClaimName(jwt));
    }

    private String getPrincipalClaimName(Jwt jwt) {
       String claimName = JwtClaimNames.SUB;
       if (properties.getPrincipalAttribute() != null) {
          claimName = properties.getPrincipalAttribute();
       }
       return jwt.getClaim(claimName);
    }

    private Collection<? extends GrantedAuthority> extractResourceRoles(Jwt jwt) {
       Map<String, Object> resourceAccess = jwt.getClaim("resource_access");
       System.out.println("resource access!!!!!"+resourceAccess);
       Map<String, Object> resource;
       Collection<String> resourceRoles;
       if (resourceAccess == null
             || (resource = (Map<String, Object>) resourceAccess.get(properties.getResourceId())) == null
```

Clase configuración de seguridad

- Políticas de acceso a los recursos:

```
@RequiredArgsConstructor
@Configuration
@EnableWebSecurity
public class SecurityConfig {
    public static final String ADMIN = "ADMINS";
    public static final String USER = "USERS";
    private final JwtAuthConverter jwtAuthConverter;
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http.authorizeHttpRequests(authorizeHttpRequest->{
                authorizeHttpRequest.requestMatchers(HttpMethod.GET,  "/contactos/**").permitAll()
                    .requestMatchers(HttpMethod.POST, "/contactos").hasRole(USER);
                }
            )                                   //HttpSecurity
        .oauth2ResourceServer(oauth2ResourceServer->
                oauth2ResourceServer.jwt(jwt->
                    jwt.jwtAuthenticationConverter(jwtAuthConverter))
            )                                   //HttpSecurity
        .sessionManagement(sessionManagement-
>sessionManagement.sessionCreationPolicy(SessionCreationPolicy.STATELESS));
        return http.build();
    }
}
```

Keycloak en el cliente

- El cliente no requiere ninguna dependencia ni configuración especial.
- Debe encargar de lanzar la petición post a Keycloak con los datos para realizar el proceso de autenticación y obtención del token
- Una vez recibido el token, lo incluirá en las cabeceras de las peticiones a microservicio