











HERE ORDER OF ANTMATCHER IS IMPORTANT .IF /\*\* IS SPECIFIED ON TOP ALL USERS WILL BE ABLE TO ACCCESS THE URLS AND OTHER ROLE CHECKS WILL BE SKIPPED

**SNIPPET OF HOW WE HAVE ACHIEVED SECURITY IN PROJECT**

package com.practice.graph;  
  
@Slf4j  
@EnableWebFluxSecurity  
@EnableReactiveMethodSecurity  
public class SecurityConfig {  
  
 @Resource  
 private ApplicationProperties applicationProperties;  
  
 //Filter for Auth0 token validation coming from UI requests  
 @Bean  
 SecurityWebFilterChain springSecurityFilterChain(ServerHttpSecurity http) {  
  
 http.securityMatcher(ServerWebExchangeMatchers.pathMatchers(AUTH0\_SUPPORTED\_APIS))  
 .authorizeExchange()  
 .pathMatchers(HttpMethod.GET, "/metadata/\*\*").permitAll()  
 .pathMatchers(HttpMethod.GET, "/version/\*\*").permitAll()  
 .pathMatchers(HttpMethod.POST, "/issuerinfo/\*\*").permitAll()  
 .anyExchange()  
 .authenticated();  
  
 http.oauth2ResourceServer()  
 .jwt()  
 .jwtDecoder(jwtDecoder());  
  
 http.csrf().disable();  
  
 return http.build();  
 }  
  
 private ReactiveJwtDecoder jwtDecoder() {  
 return NimbusReactiveJwtDecoder.withJwkSetUri(applicationProperties.getAuth0JwksUrl()).build();  
 }  
  
 //Filter for M2M token validation coming from Databricks Client requests  
 @Bean  
 SecurityWebFilterChain springSecurityFilterChainM2M(ServerHttpSecurity http) {  
  
 http.securityMatcher(ServerWebExchangeMatchers.pathMatchers(M2M\_AUTH\_SUPPORTED\_APIS))  
 .authorizeExchange()  
 .anyExchange()  
 .authenticated();  
  
 http.oauth2ResourceServer()  
 .jwt()  
 .jwtDecoder(jwtDecoderM2M());  
  
 http.csrf().disable();  
  
 return http.build();  
 }  
  
 private ReactiveJwtDecoder jwtDecoderM2M() {  
  
 NimbusReactiveJwtDecoder jwtDecoder = (NimbusReactiveJwtDecoder)  
 ReactiveJwtDecoders.fromIssuerLocation(applicationProperties.getAzureAdAppIdUri());  
  
 OAuth2TokenValidator<Jwt> tokenValidator = new DelegatingOAuth2TokenValidator<>(  
 new JwtTimestampValidator(),  
 new JwtClaimValidator<List<String>>("aud", (aud) -> {  
 log.info("Validating JWT token, {} with {}", aud, applicationProperties.getAzureAdClientId());  
 return aud != null && aud.contains(applicationProperties.getAzureAdClientId());  
 }));  
  
 jwtDecoder.setJwtValidator(tokenValidator);  
  
 return jwtDecoder;  
 }  
}