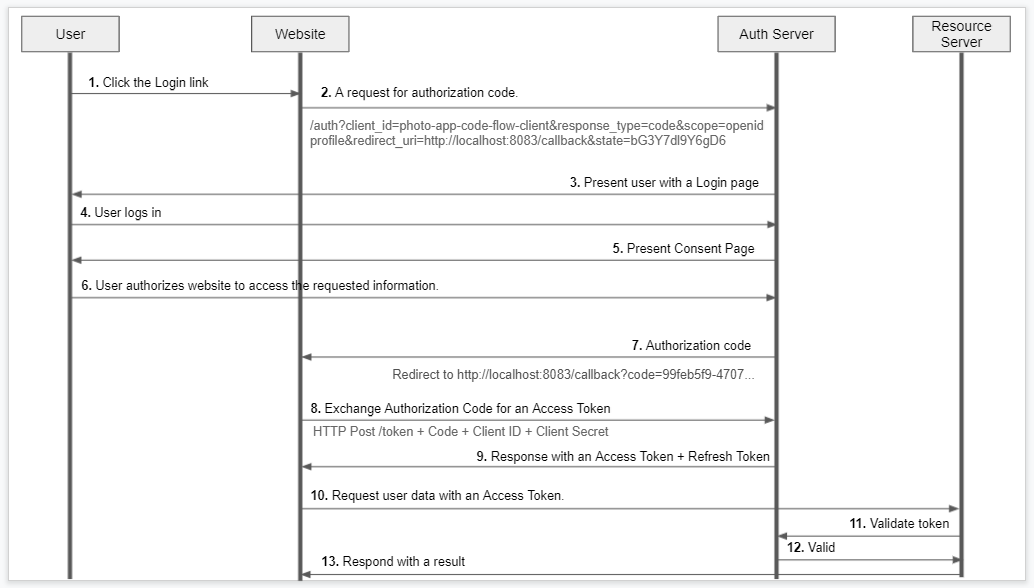
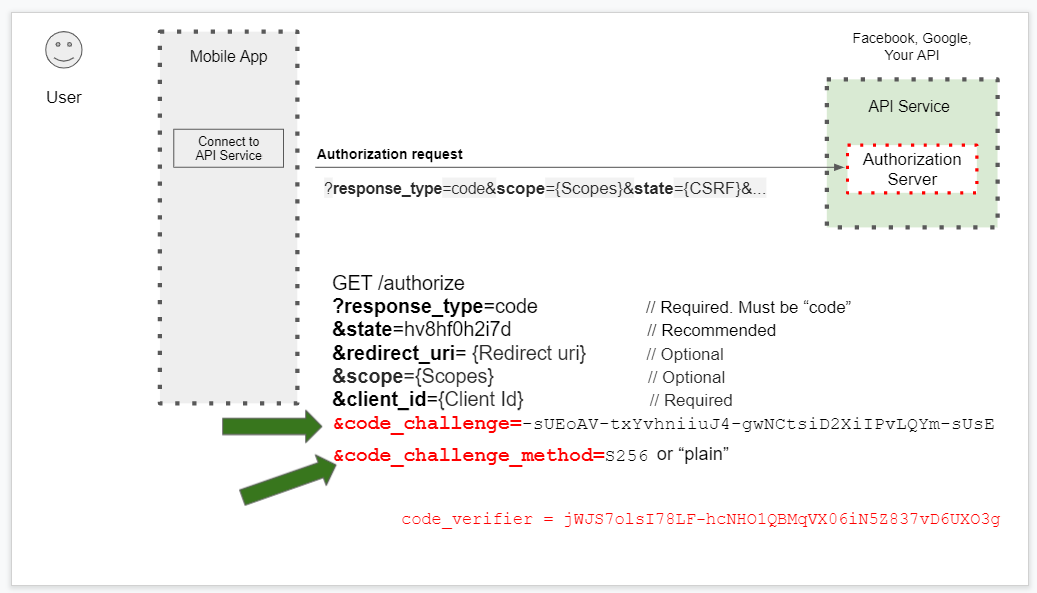
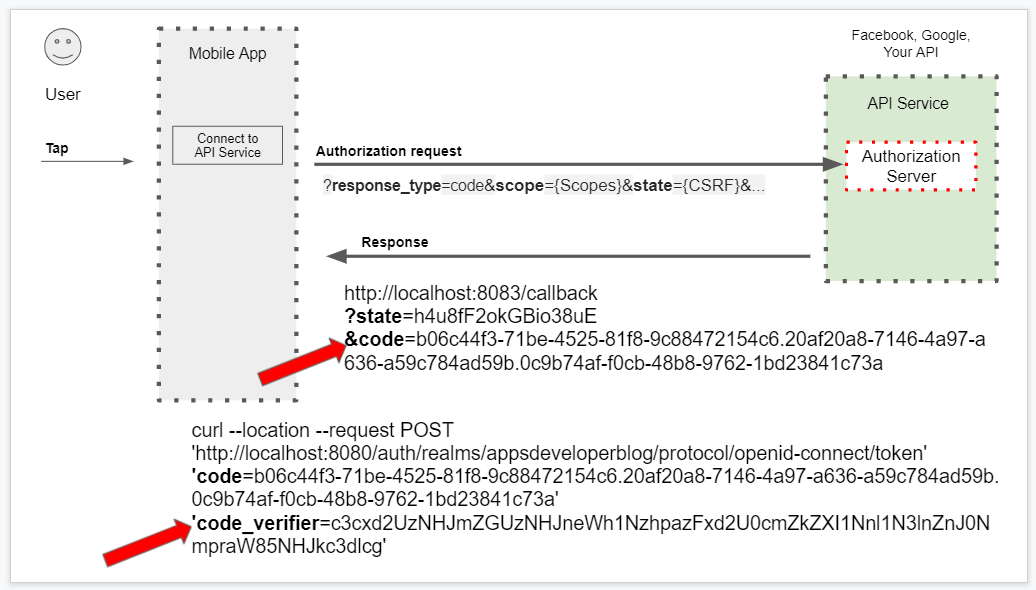
1. Introduction
   1. OAuth 2.0
      1. OAuth = Open + Authorization
      2. OAuth to is now an industry standard protocol for authorization, and it provides specific authorization flows for different types of applications that developers create.
      3. OAuth 2.0 is the **authorization framework** that enables applications to access user data or perform some operation on user behalf.
      4. OAuth also called a **delegated authorization framework** would be helpful with users can authorize a third-party application to access their data or to perform certain operations without giving away their username and password.
   2. OAuth 2.0 Roles
      1. Resource Owner
         1. Resource is the information or the data that is being accessed, and the owner is whoever owns that information.
         2. Resource owner is a user who owns the information and is capable of granting access to this information.
      2. Client
         1. A client is an application that is access information on user's behalf.
      3. Resource Server (Microservices)
         1. Resource Server is a server that is hosting user information, it could be a Google server, or it could be a Facebook server, or it could be your own server that stores user data and user’s data on a resource server is usually available via an API.
      4. Authorization Server
         1. The authorization server is the server that issues access tokens to client application after successfully authenticating the resource owner and obtaining authorization.
         2. All major players like Google, Facebook, Twitter or Microsoft, they have their own authorization servers.
   3. Spring Security
      1. It’s a community driven open-source project.
      2. It provides client support. It provides resource server support and the authorization server support all in one single project.
      3. In New Spring OAuth Project, it provides Client support and resource server, but authorization server is not fully available.
   4. OAuth Client Types
      1. There are two different client types based on their ability to authenticate securely with the authorization server.
         1. Confidential Client
            1. Confidential clients are applications that are able to keep the client secret, key securely and safe on their site.
         2. Public Client
            1. These are applications that cannot guarantee the safety and confidentiality of their client and client secret key
      2. For a client application to be able to communicate with authorization server. It needs to register with authorization server and get its client I.D. and client secret key.
   5. OAuth Access Token
      1. the authorization server will issue, and access token and it will pass this access token to a client application.
      2. The client's application will then use this access token whenever it needs to communicate with the resource server to access user’s data.
      3. To check of the access token is valid, and if it can be used to access the requested information resource server, we will send the request to an authorization server to check if the access token is not fake and it is valid if the authorization server confirms that the access token is valid or not.
      4. The application has the access token. It can access user’s data on the resource server on user's behalf without asking user for username and password anymore.
      5. Access Token Types
         1. Identifier type
            1. An identified type access token does not contain any authorization information and it is being used to look at this information that is associated with this token in the database table.
         2. Self-contain the authorization information
            1. The self-contained access token is a json object that is base 64 encoded, it is separated into three parts, the header section, the payload section, and the signature.
   6. OAuth 2.0 and OpenID Connect (OIDC)
      1. Open Connect was designed as an additional layer to provide our client application with information about the currently authenticated user.
      2. Open ID connect comes as an additional layer on top of or two, and it is an **identity layer** that can provide the client application with an identity information about the user.
      3. The authorization server that supports OpenID. They do connect and provide the client application with the identity Information is also called an identity provider.
      4. The authorization server additionally to an **access token**, will also provide client application with and they determine the **ID token** contains user identity information and the client application can use this I.D. token to extract from it some basic information about the current authenticated user.
2. OAuth 2.0 Grant Types and Authorization Flows
   1. Grant Type
      1. Grant Type is a way an application gets an access token
      2. Different types of grant
         1. Authorization Code
         2. Client Credentials
         3. PKCE Enhanced Authorization Code
         4. Device Code
         5. Implicit Flow (Legacy)
         6. Password Grant (Legacy)
   2. Refresh Token Grant Type
      1. It is used to exchange a refresh token for an access token.
   3. Authorization Code Grant



* 1. PKCE Enhanced Authorization Code
     1. PKCE – Proof Key for Code Exchange
     2. it provides an additional level of security for applications that cannot keep its client secret key safe
     3. Generating Code Verifier
     4. Generating Code Challenge

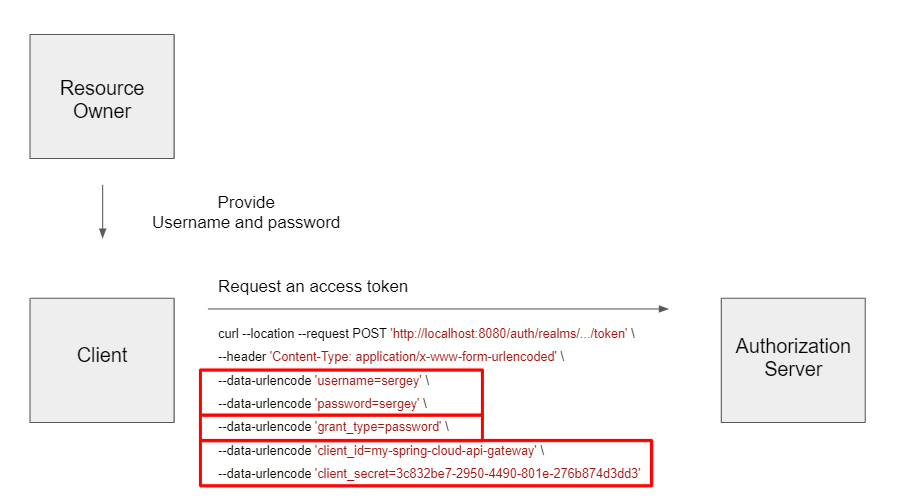




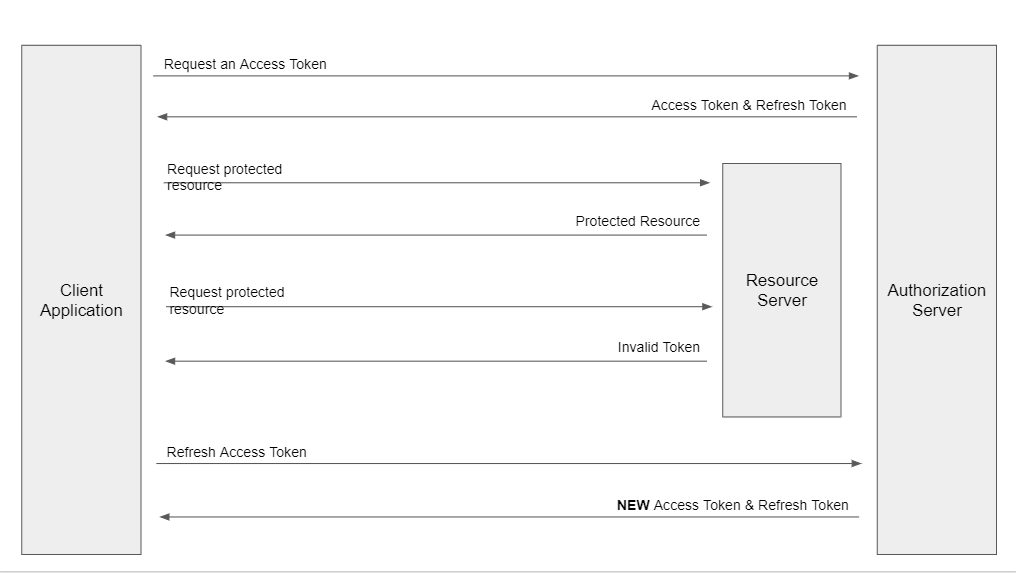
* 1. Client Credentials
     1. Machine to Machine requests
     2. There is no user involved in this file and there is no login page for user to type in their username and password but will use client credentials grant type to authorize machine to machine and request.



* 1. Password Flow

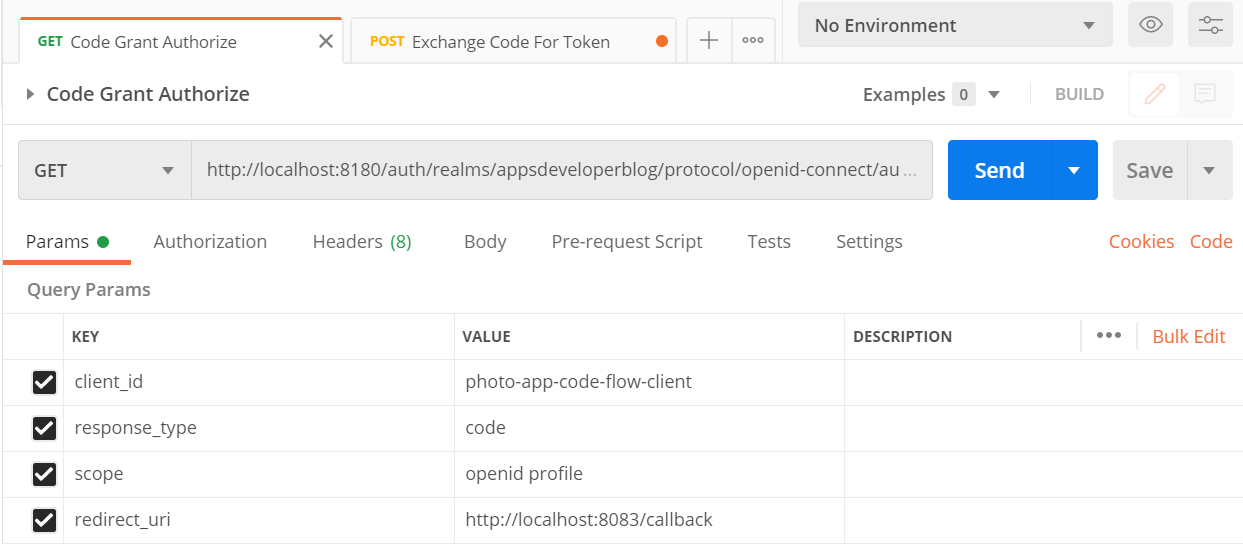


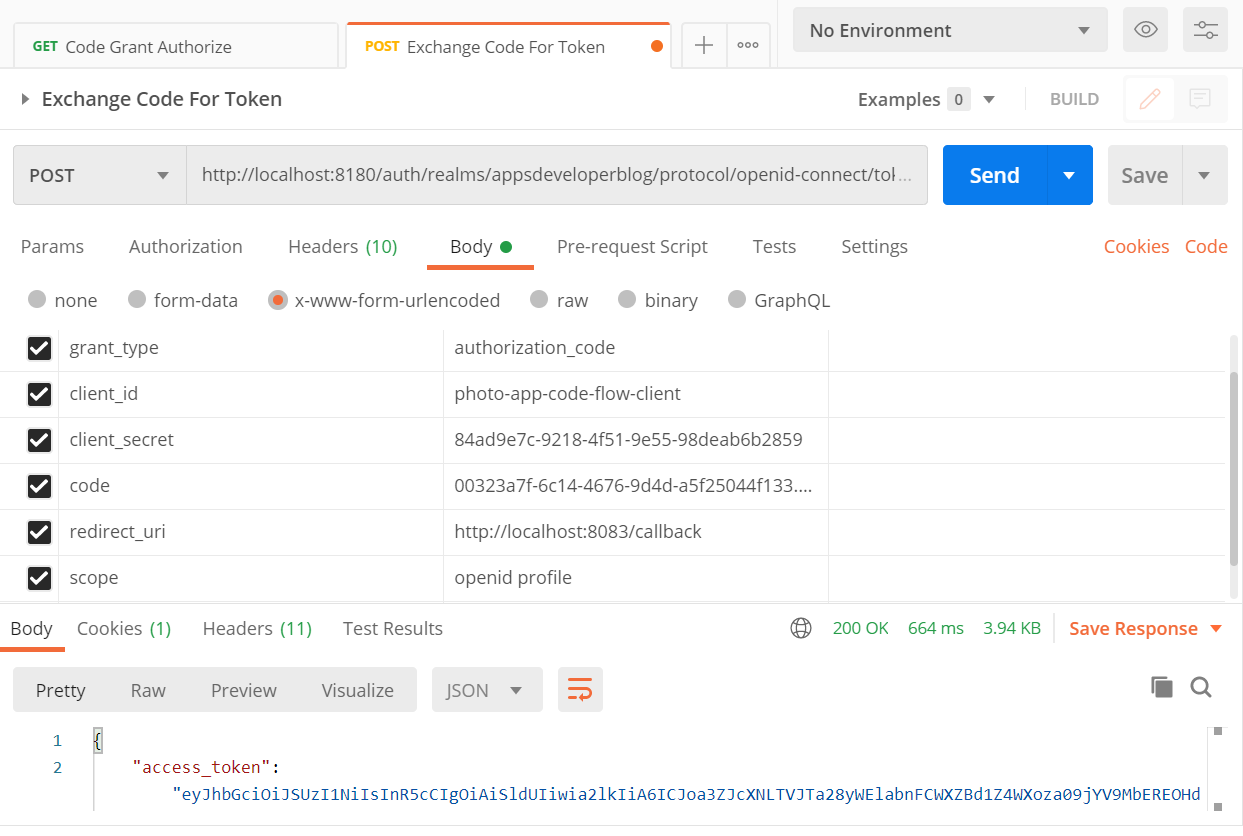
1. Refreshing Access Token
   1. Pass “offline\_access” in scope to request refresh token that never expires.



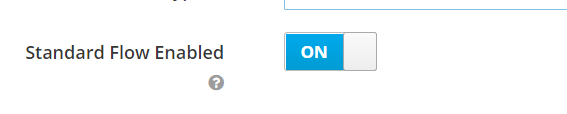


1. Keycloak the standalone authorization server
   1. It is an open-source identity and access management server that supports OpenID connect and with that authorization flows.
   2. It supports Single-Sign On (SSO)
   3. Also supports social login
   4. It also supports active directory.
   5. It can be used with our spring boot application as authorization server. We can use it to let users authenticate with their application and then generate an access token and refresh token.
   6. Setup Keycloak server
      1. Download zip file from <https://www.keycloak.org/> and unzip in local
      2. Go to bin folder
      3. Run “standalone.bat” file will start the server and access using <http://localhost:8080>
      4. To run in different port: **standalone.bat -Djboss.socket.binding.port-offset=100** and then access using <http://locahost:8180>
      5. Creating Initial Admin User
         1. Username: unknowns, Password: unknowns
      6. Creating a new Realm
         1. Realm is like a tenant in server. It is a space that allows us to manage a set of users their rules and their credentials.
         2. By default it has Master Realm. And it’s not recommended to use master Realm.
         3. Create New Realm “appsdeveloperblog”
      7. Creating a new user
         1. Use for user login http://localhost:8180/auth/realms/appsdeveloperblog/account/#/
      8. Creating a new OAuth Client application
         1. Go to Clients and click on create button
         2. Enter client id
         3. Select Client protocol as openid-connect
         4. Enter valid redirect url http://localhost:8083/callback
      9. Configuring Client Application Secrets
         1. Change Access Type to “confidential” and save it.
         2. Go Confidential Tab to get secret.
      10. Requesting Access Token and Refresh Token





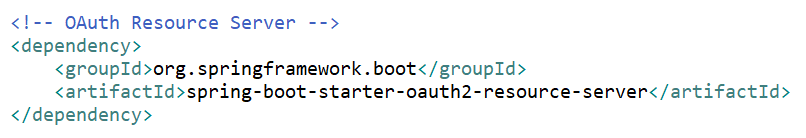
* + 1. Enable / Disable OAuth 2.0 Authorization Flow





* + 1. OAuth 2.0 Client Scopes
       1. Go to Admin -> Client Scopes to create new scopes
       2. In Clients -> Client Scopes Tab for assigning scope to client

1. OAuth Resource Server
   1. Add dependency

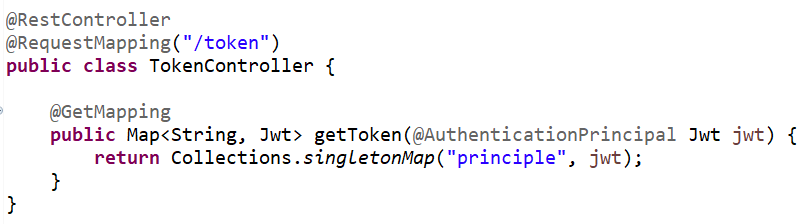


* 1. Add below properties in application.properties file either one of them

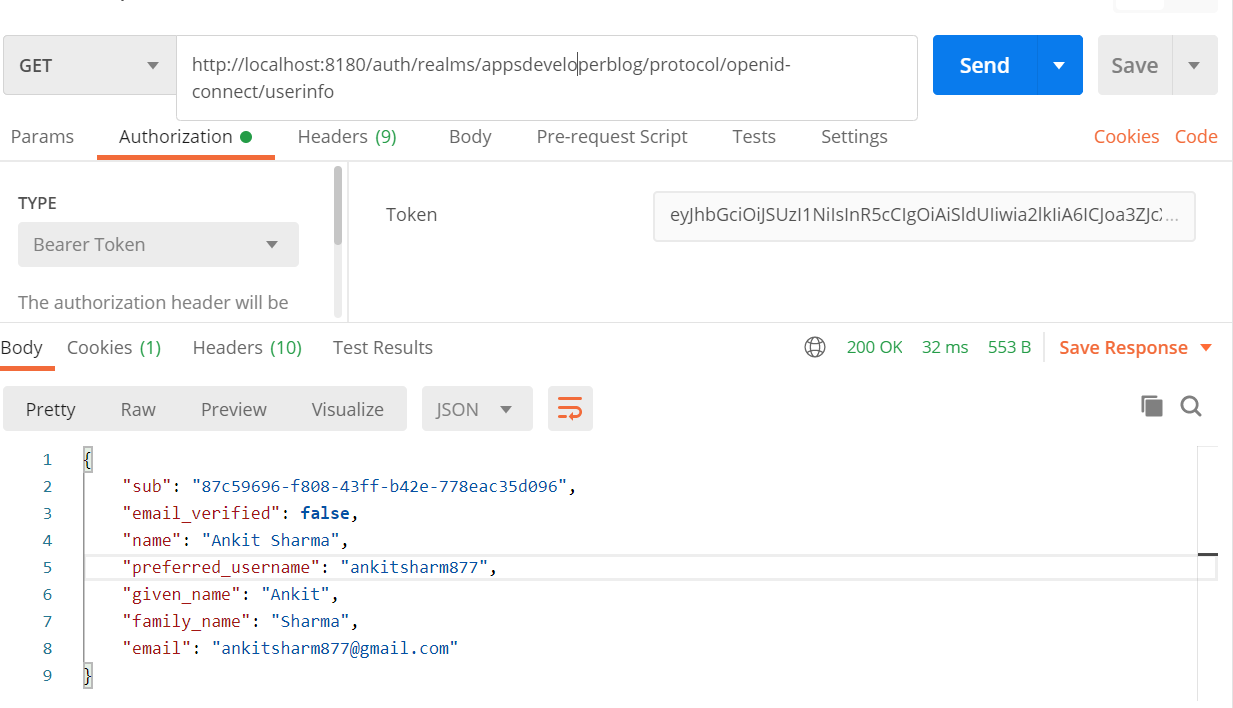
**spring.security.oauth2.resourceserver.jwt.issuer-uri**=http://localhost:8180/auth/realms/appsdeveloperblog

**spring.security.oauth2.resourceserver.jwt.jwk-set-uri**=http://localhost:8180/auth/realms/appsdeveloperblog/protocol/openid-connect/certs

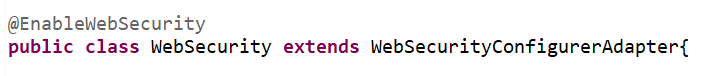
* 1. Call <http://localhost:8081/users/status/check> and also pass token to access resource
  2. Accessing Principle and JWT Claims



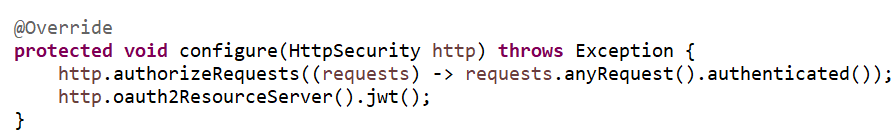
1. Resource Server – Scope Based Access Control
   1. Scope is a mechanism in OAuth 2.0 to limit an application’s access to a user’s account. An application can request one or more scopes, this information is then presented to the user in the consent screen, and the access token issued to the application will be limited to the scopes granted.
   2. OpenID Connect Scopes
      1. The default scopes are **openId** and **profile**.
      2. The openId scope: It enables client applications to verify the identity of a user as well as obtain some basic information about the end user.
      3. The profile scope: it tells the authorization server that application, would like to have access to a user profile details and in openId connect user profile information contains the following. It contains name, family name, given the middle name, nickname and so on, but it does not contain email address.
      4. Other Scopes: email, address, phone and offline\_access
   3. Access user info from authentication server



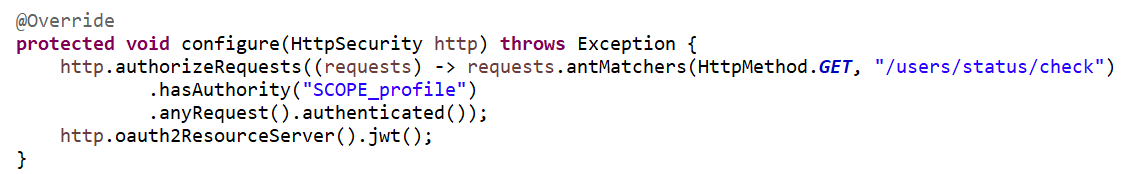
* 1. Implementing Scope Based Access
     1. Create Class extends WebSecurityConfigurerAdapter and add @EnableWebSecurity annotation at class level



* + 1. Override configure function

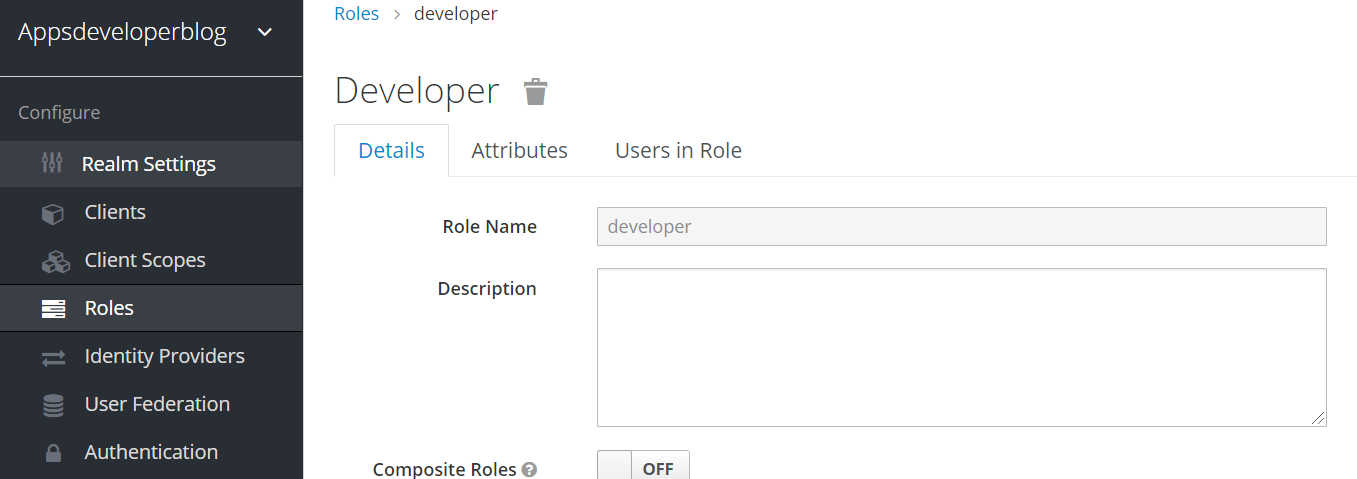


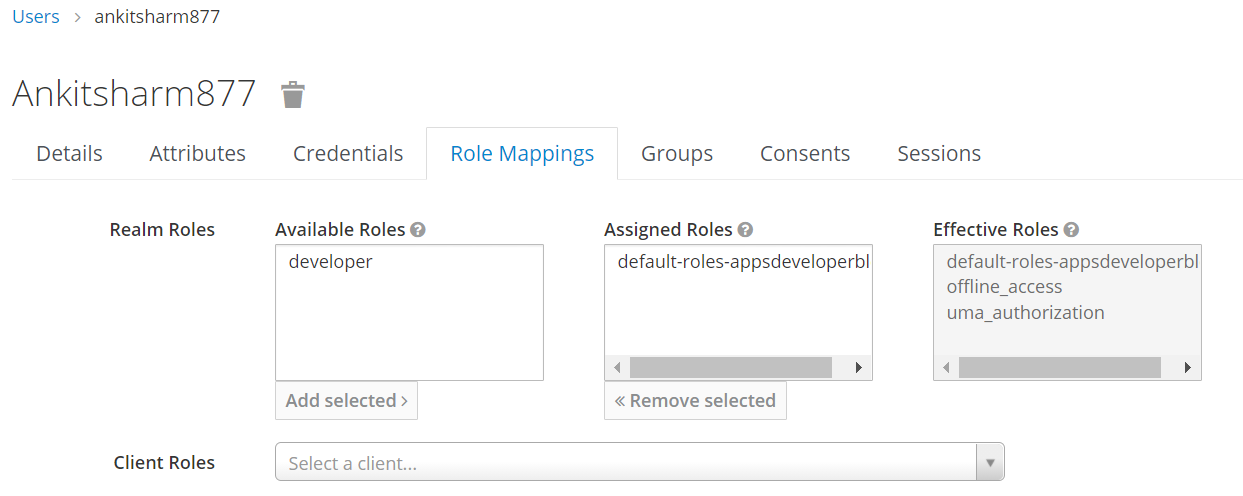
* + 1. Add scope-based access in request and throw 403 (Forbidden)



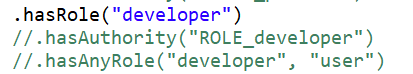
* + 1. Removed profile scope from default and add in optional client scopes.
    2. Get token with profile scope and without profile scope to call request if scope include profile then only it will return the response else it will throw 403 (Forbidden) http error code.

1. Role bases access control with keycloak
   1. Any application can have multiple roles with different privileges or authorities.
   2. Creating a role and assigned to user

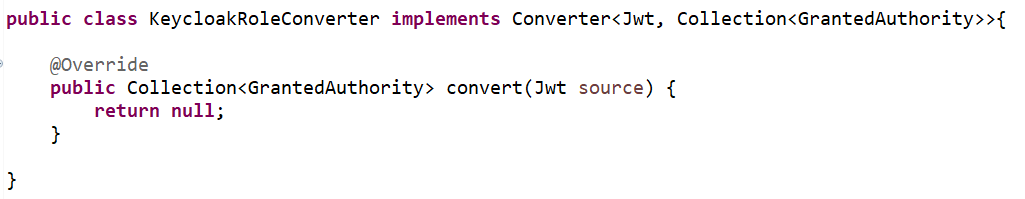




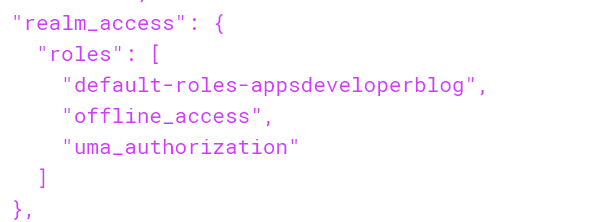
* 1. Securing endpoints to a specific role



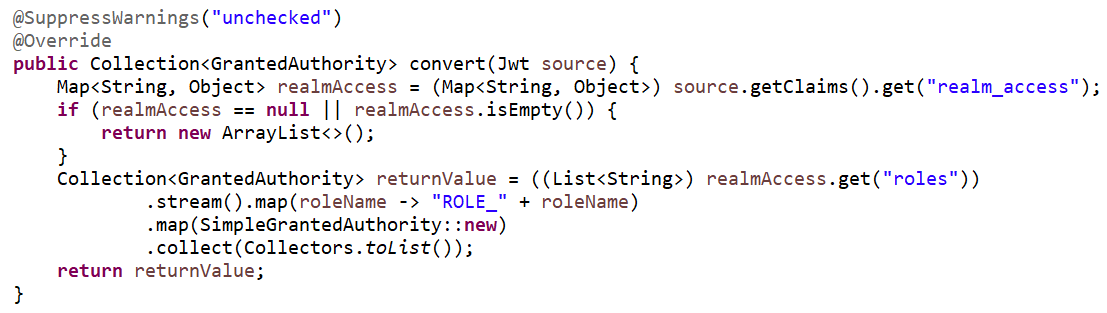
* 1. Creating role converter class



* 1. Decoding JWT to find user roles (<https://jwt.io/>)



* 1. Converting roles into GrantedAuthority objects

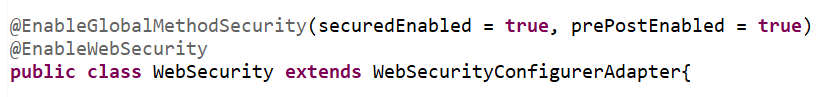


* 1. Register Custom Converter to Http Security

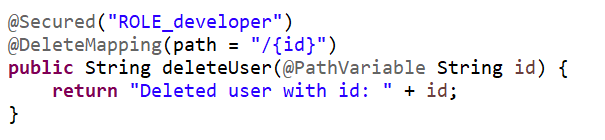


* 1. If we try to access resource with user who has role developer assigned, then only it returns response else it will throw Http Error Code 403 (Forbidden)

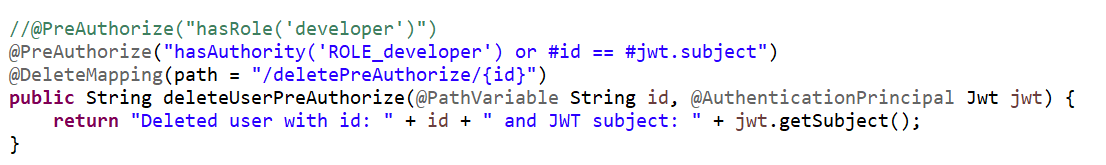
1. Resource Server: Method Level Security
   1. We can add method level security at controller and services layer as well.
   2. We can add method level security at class level (common for all methods) or method level.
   3. We use @Secured() annotation for method level security.
   4. Other annotations are @PreAuthorize() and @PostAuthorize()
   5. Enable method level security
      1. Add @EnableGlobalMethodSecurity annotation at class level in WebSecurity class. For enabling **@Secured** annotation add **securedEnabled** as true and for enabling **@PreAuthrize** and @PostAuthorize annotation add **prePostEnabled** as true.



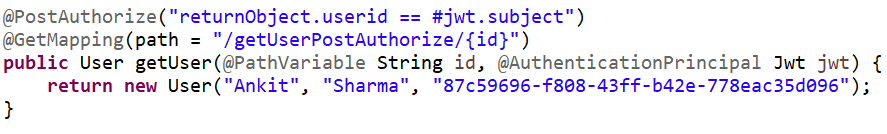
* 1. @Secured annotation



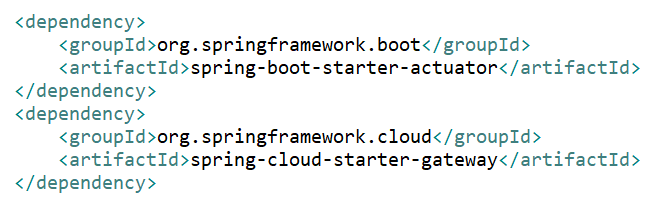
* 1. @PreAuthorize() annotation



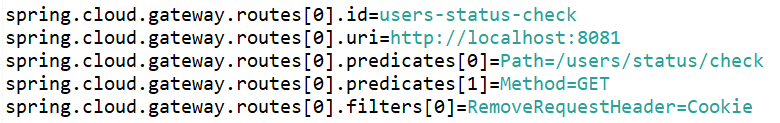
* 1. @AuthenticationPrincipal is used to get Jwt object in request argument
  2. @PostAuthorize() annotation



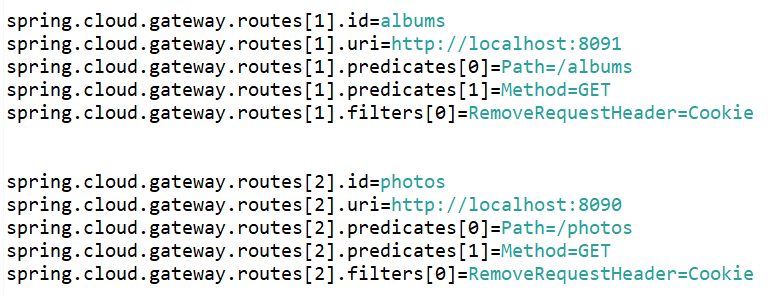
1. Resource Server Behind API Gateway
   1. Create New API Gateway service and add dependency



* 1. Configuring API gateway routes using application properties file



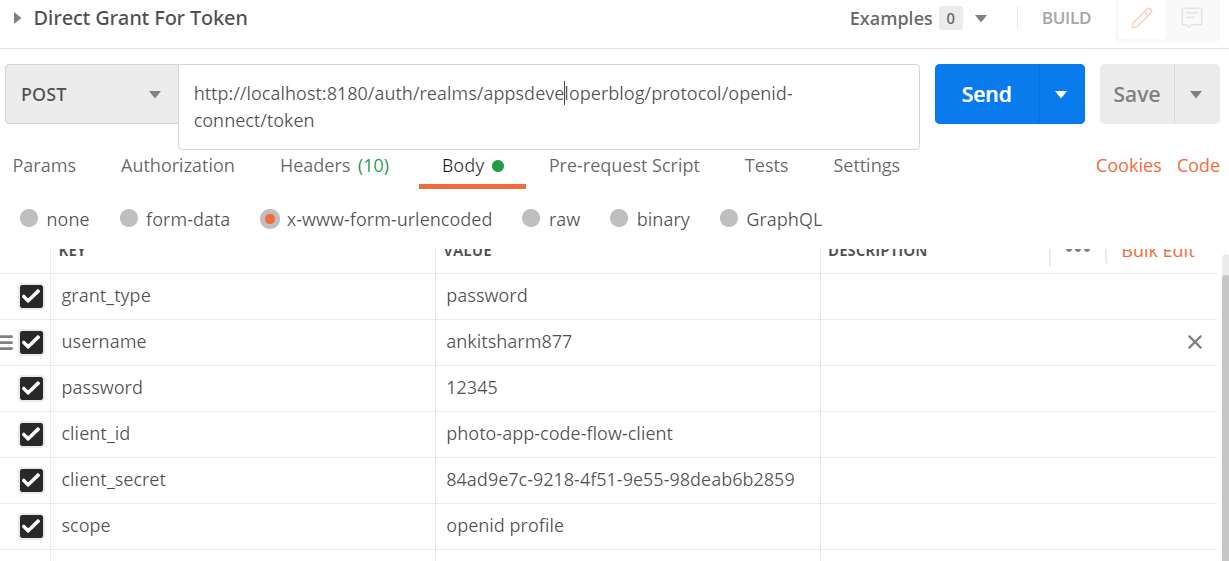
* 1. Add multiple routes



1. Access Token using Grant Type password
   1. This way using username and password we can generate token in single request
   2. Enable Direct Grant for client



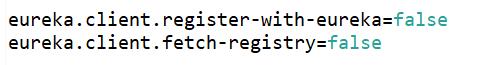
* 1. Direct access token using grant type password



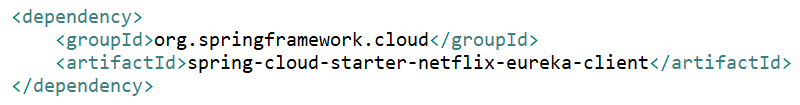
1. Eureka Discovery
   1. It is used to register multiple instances if each microservice, so API gateway take instance from eureka server and send request to microservices
   2. Create project and add dependency



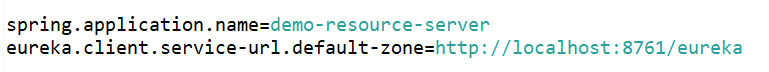
* 1. Add @EnableEurekaServer annotation at class level in Application class
  2. Add below properties in application properties file (not to fetch or register this eureka server with other eureka server)



1. Configure Eureka Client
   1. Add dependency in microservice



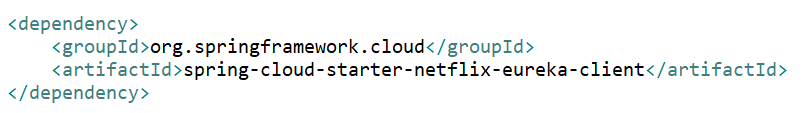
* 1. Add @EnableDiscoveryClient annotation at class level in Application class
  2. Add properties in application properties



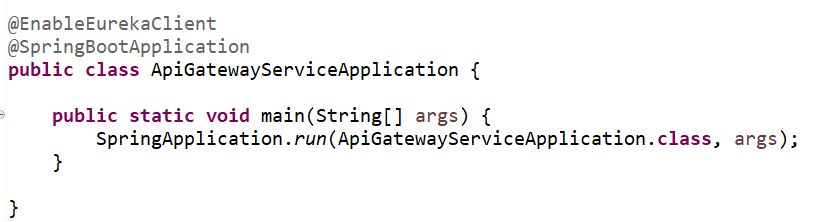
1. Load Balancing
   1. Load Balancing between multiple instances of microservices in API gateway. We register all instance in eureka server. API Gateway uses load balancer to send request between multiple instances by taking instance URI ( **lb://<application-name>** ) from eureka server.
   2. Create Multiple Instance of microservices
      1. In application properties file, if we mention server.port = 0 then it automatically pick unique random port number when we start application. Also set instance-id in application properties to assign unique instance id to each instance.



* + 1. Other we can set port number in run configuration -> arguments -> vm arguments -Dserver.port=8081 and duplicate instance
  1. Configure API Gateway in Eureka Server
     1. Add dependency in pom xml file.



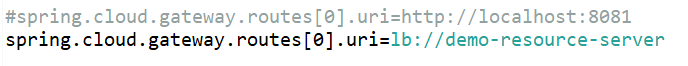
* + 1. Add @EnableEurekaClient annotation in Application class file.



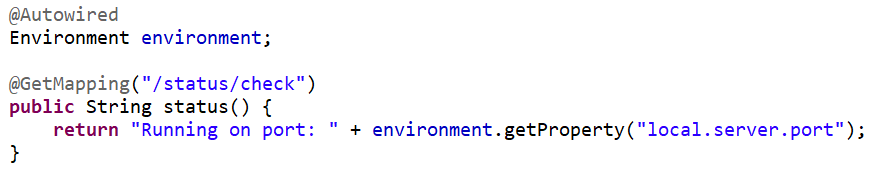
* + 1. Add property in application properties file



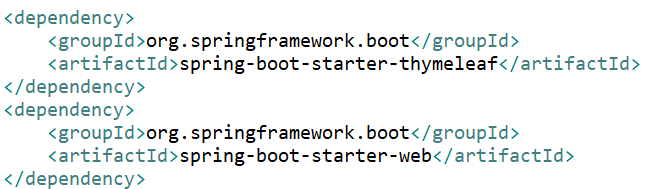
* 1. Configure API Gateway Routes for Load Balancer
     1. Change routes uri to lb://<application-name>



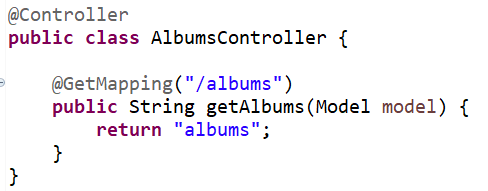
* + 1. Check port number in microservices



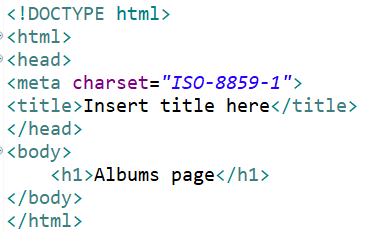
1. OAuth 2.0 in MVC Web App
   1. Create Project and add below dependency



* 1. Spring Boot Thymeleaf
     1. A modern server-side Java template engine for both web and standalone environments. Allows HTML to be correctly displayed in browsers and as static prototypes.
     2. We can add html files in resources/templates folder and access through browser with the help of controller.
     3. Add controller with @Controller annotation as it returns html page in response.



* + 1. Add albums.html file in resources/templates folder and html file name should match with the request path in @GetMapping annotation.



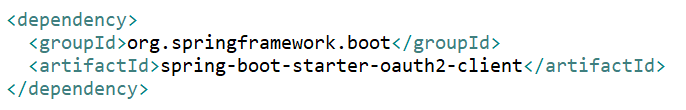
* + 1. Hit <http://localhost:8087/albums> and it render albums.html in browser.
    2. Set data in attribute from controller which can be used in html file



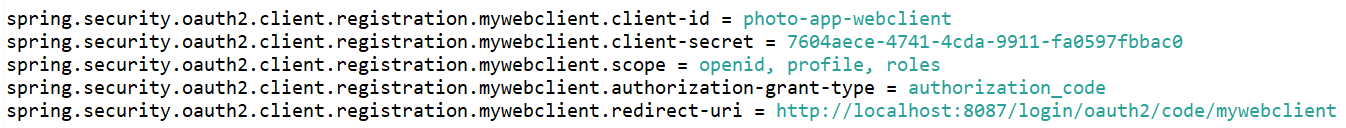
* + 1. Add metadata and use tag attributes in html file



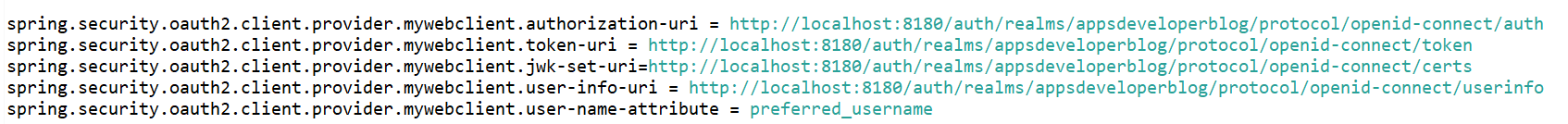
* 1. Adding OAuth 2.0 Client Dependency



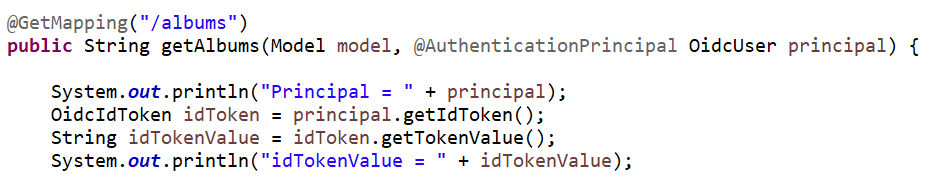
* 1. Configuring OAuth 2.0 Client Properties
     1. Create new client in keycloak and use that for web client



* 1. Configuring OAuth 2.0 Provider Properties



* 1. Reading ID Token
     1. Using @AuthenticationPrincipal annotation for getting principal object in argument.



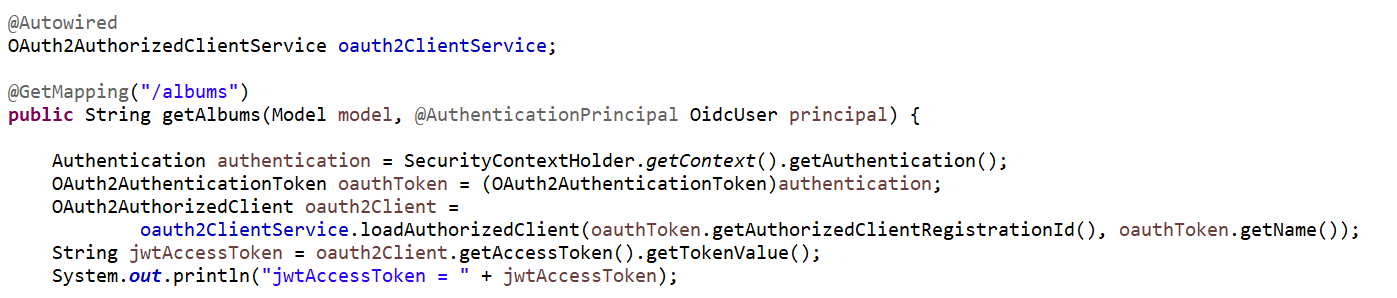
* 1. Reading JWT Access Token
     1. Add Authentication Object in argument (1st way)



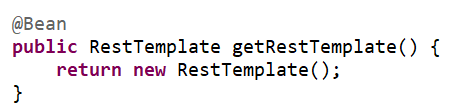
* + 1. Create Authentication Object using SecurityContextHolder (2nd way)



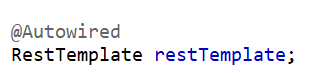
* + 1. Get JWT Access token
       1. @Autowired OAuth2AuthorizedClientService and use that to get OAuth2AuthorizedClient.
       2. Then JWT token value using OAuth2AuthorizedClient.



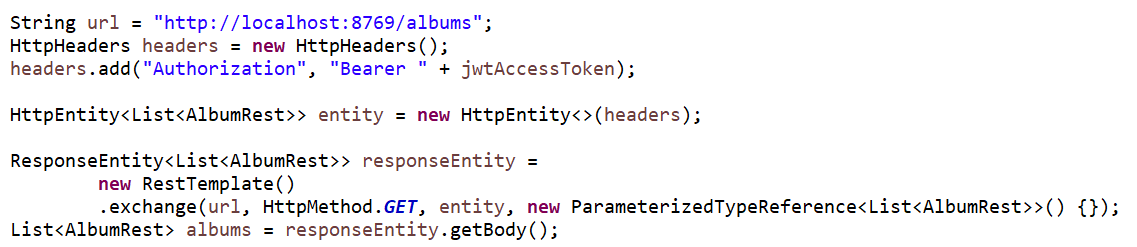
* + 1. HTTP Request to an Internal Resource Server
       1. For using RestTemplate as bean need to add function in Application class with @Bean



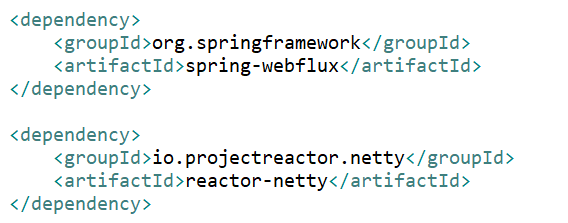
* + - 1. Then @Autowired RestTemplate



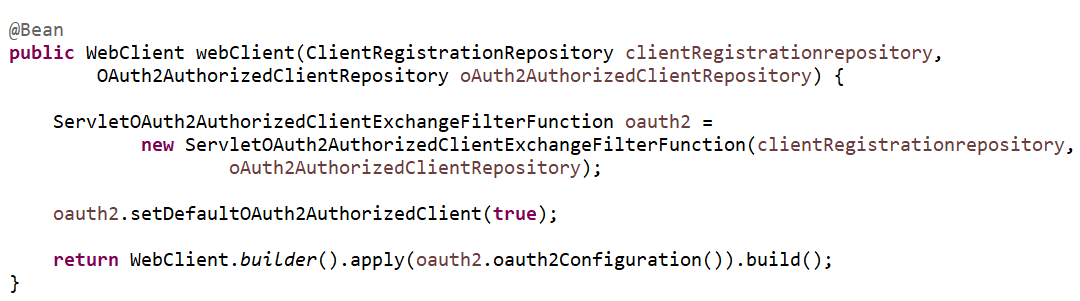
* + - 1. Or directly create object wherever need to rest template.

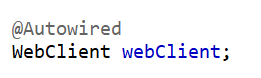


* + 1. A different approach to adding access token
       1. We can use WebClient to get Access token
       2. Add dependency in pom xml file

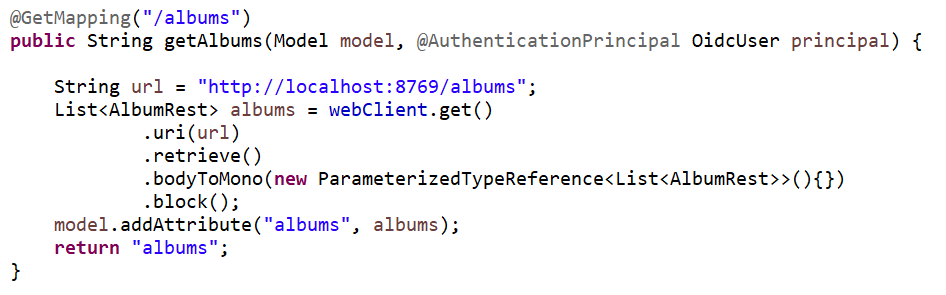


* + - 1. To @Autowired WebClient

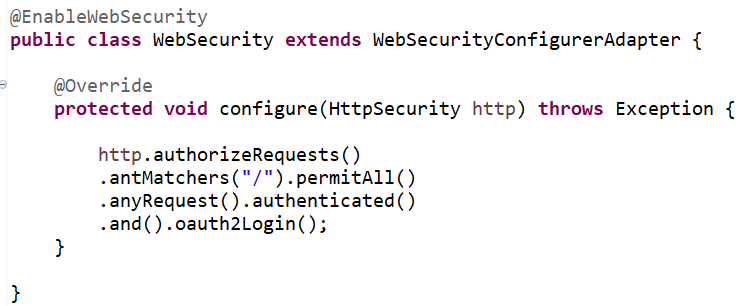




* + - 1. Use web client to send request



1. OAuth 2.0 – Social Login
   1. Configure Http Security



* 1. OAuth 2 Client and Provider Configurations