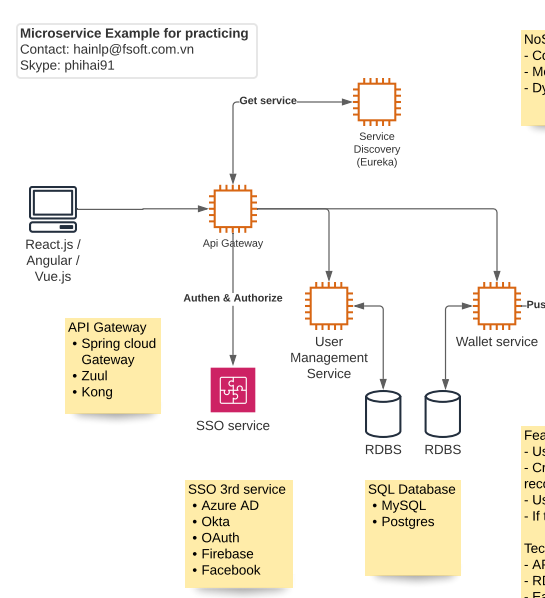
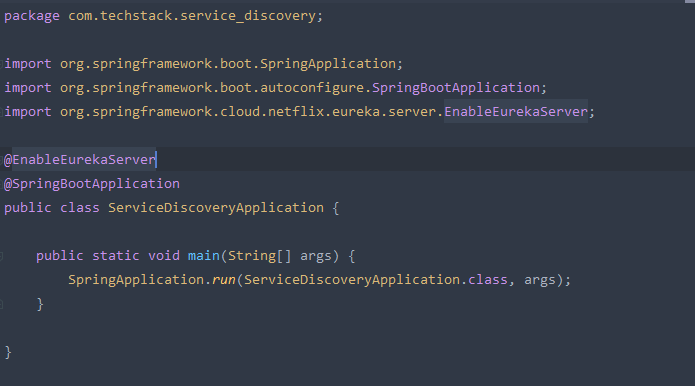
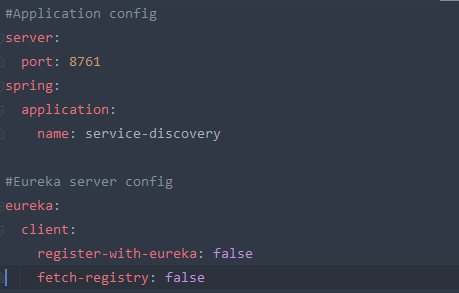
Create a microservice system with Spring

(Spring Boot, Spring Cloud, Eureka, Okta)



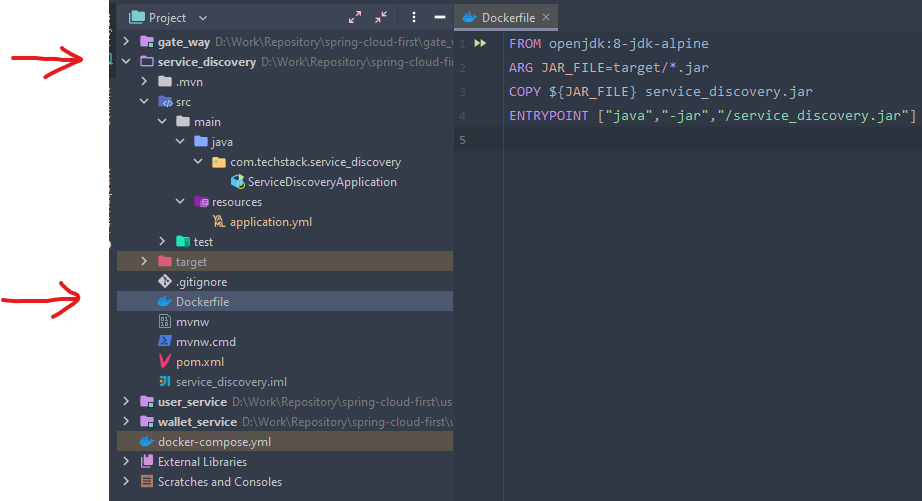
# Create Eureka server for service discovery

1. Go to <https://start.spring.io/> for creating new Spring project, add Eureka Server as Maven Dependency
2. Download and Unzip the package
3. Import project to your favourite IDE
4. In the …Application class, add the annotation **@EnableEurekaServer** to tell Spring that this application will be used as the Eureka Server
5. Add some configurations, in this guidelines, I will use application.yml file:

Explanation

* **server.port** for define which port the application will take
* **spring.application.name** (optional) this could be use later on when you want to get the application name
* **eureka.client.register-with-eureka, fetch-registry** we need to set these to false because the default behavior of Eureka, it will register its own Eureka server.

1. Dockerizing the application:

* Add a simple Dockerfile to the root folder of service\_discovery project to define how to build the Eureka server image

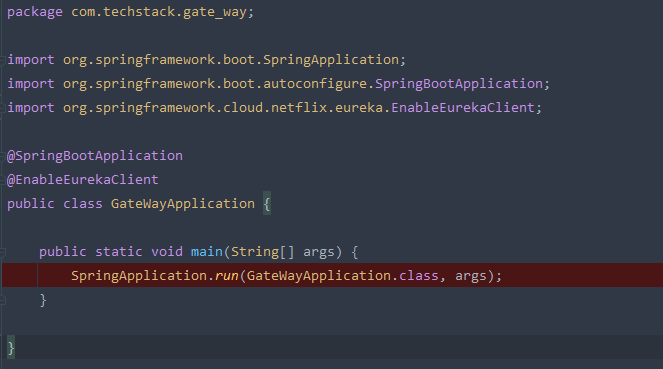
Explanation

(Based on Docker reference: https://docs.docker.com/engine/reference/builder/)

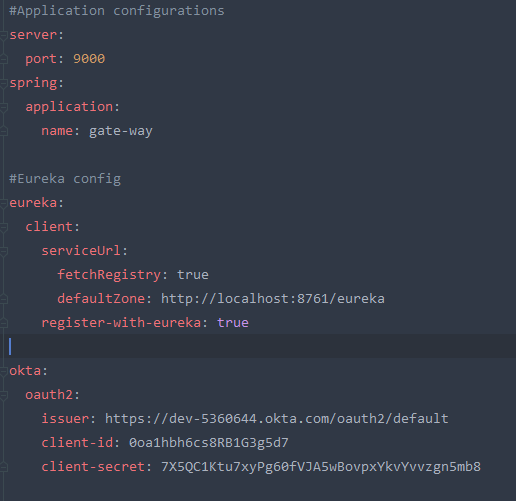
* **FROM -** define which **base image** we would custom to create our image
* **ARG -** The ARG instruction defines a variable that users can pass at build-time to the builder with the docker build command using the --build-arg <varname>=<value> flag. If a user specifies a build argument that was not defined in the Dockerfile, the build outputs a warning.
* **COPY** - instruction copies new files or directories from <src> and adds them to the filesystem of the container at the path <dest>.
* **ENTRYPOINT** - allows you to configure a container that will run as an executable.

# Create API gateway using Spring Cloud Gateway:

1. Go to <https://start.spring.io/> for creating new Spring project, add Eureka discovery client, Gateway, Spring Reactive Web, Okta as Maven Dependency. We need Spring Reactive Web and Okta for securing the system by SSO with Oauth2.
2. Download and Unzip the package
3. Import project to your favourite IDE
4. For the Gateway module, it is an Eureka Client which it will register itself to the Eureka Server. Therefore, we need to use **@EnableEurekaClient** instead.



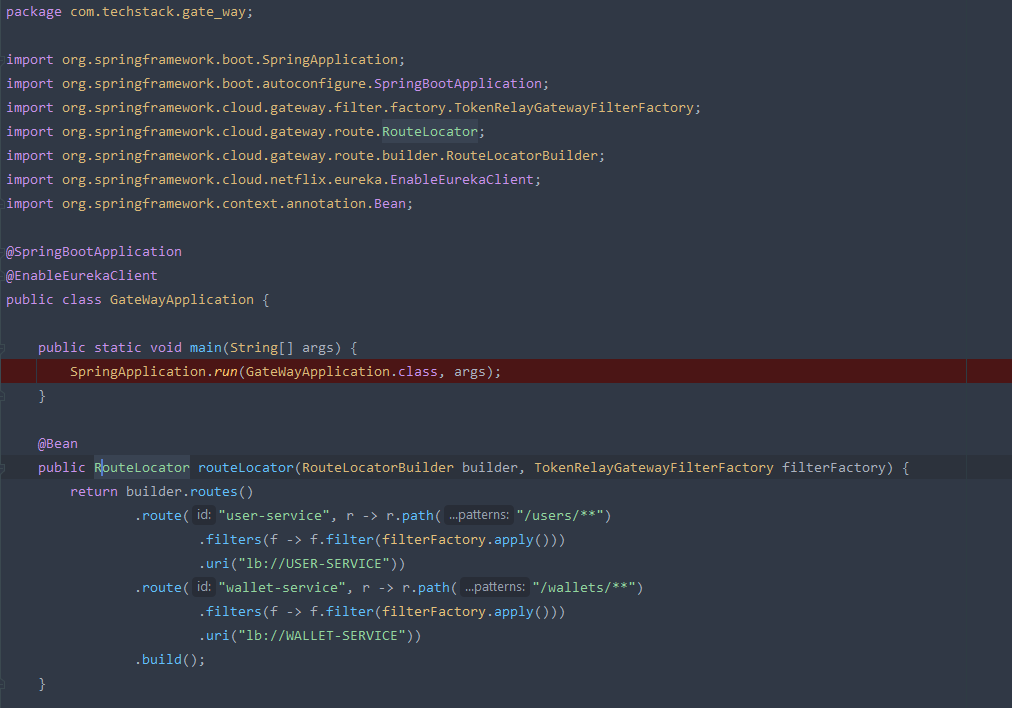
1. Now, we need to add some configurations

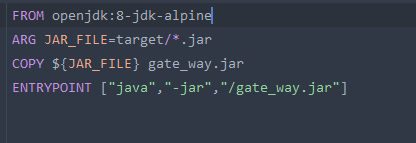


Explanation:

* **eureka.client.serviceUrl.fetchRegistry:** simply means that the Gateway will get the Eureka service registry from Eureka Server.
* **eureka.client.serviceUrl.defaultZone:** to tell that where is the Eureka Server currently hosting.
* **eureka.client.register-with-eureka:** to register the Gateway itself to Eureka Server
* **okta** config will be explained later,

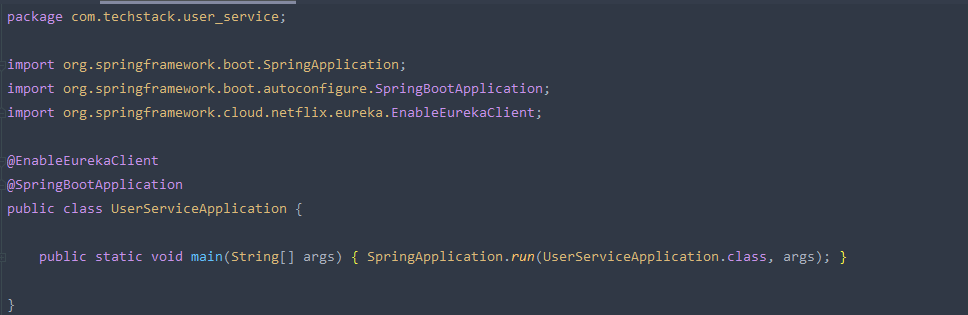
1. A Gateway will redirect the requests to the correct resource server. A good Gateway would route the request without knowing where the resources server are being located (Reason for using Eureka). To make our Gateway works, we need to add @Bean that provides a Route Locator as the Routing configuration (I will explain all of these later)

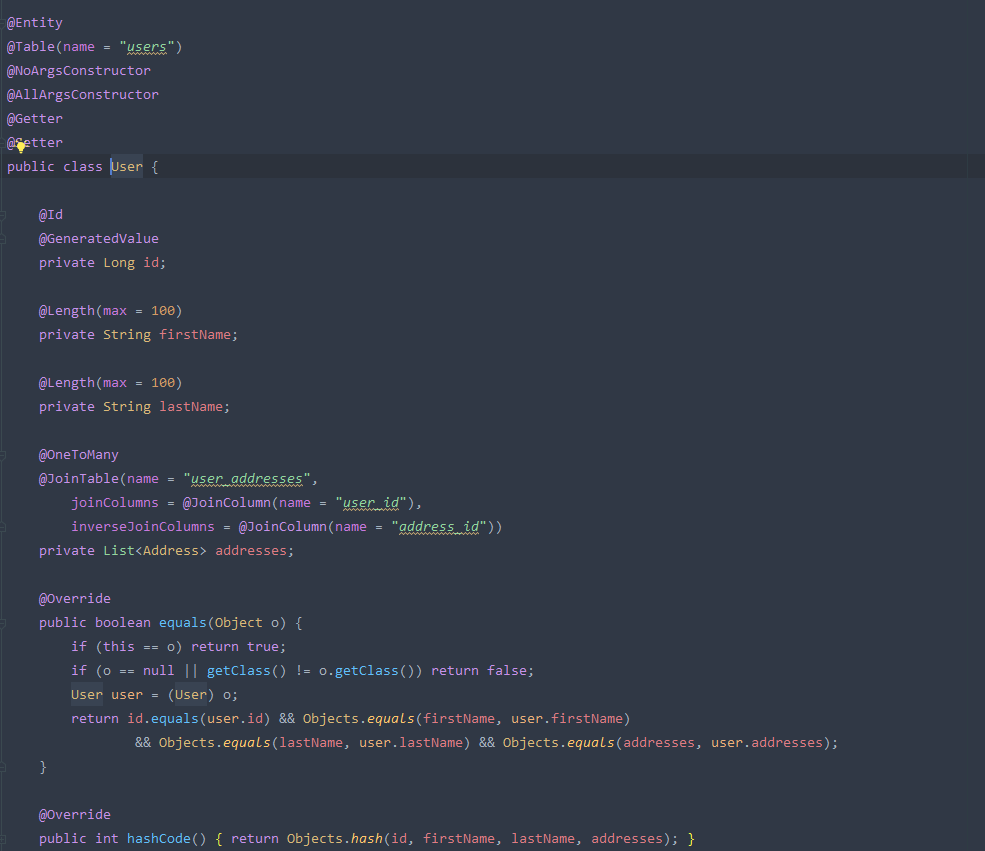


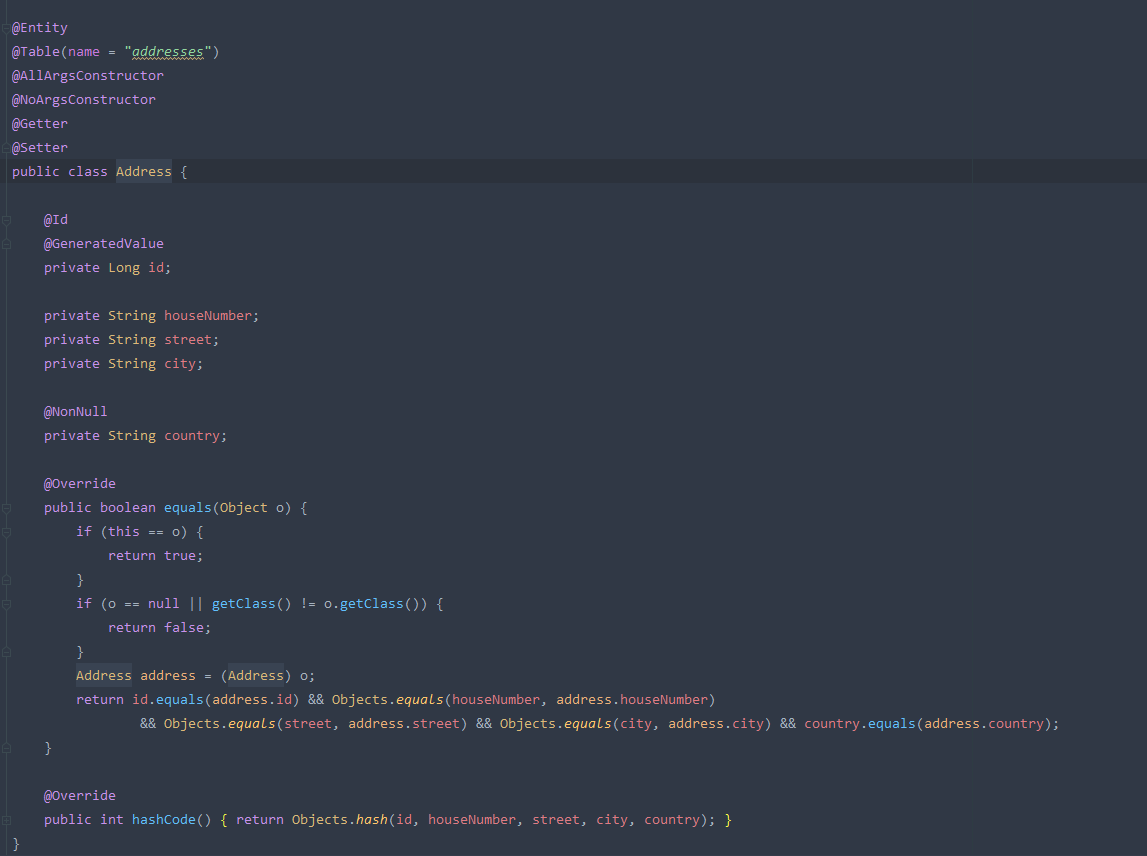
1. :

# Create Service providers

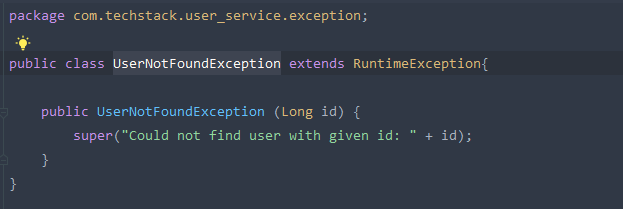
## Create user-service:

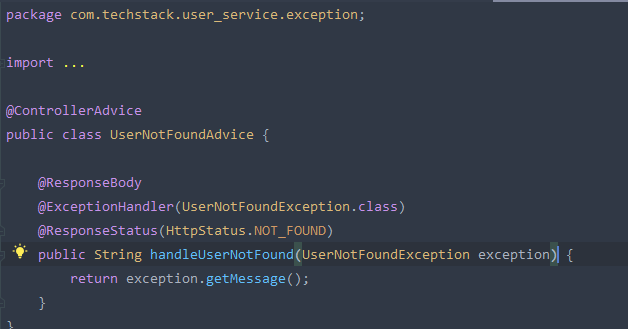
1. Go to <https://start.spring.io/> for creating new Spring project, add Eureka discovery client, Web, HATEOAS, Okta, lombok, JPA (In this example, we will use docker image of MySQL database)
2. Download and Unzip the package
3. Import project to your favourite IDE
4.  Add @EnableEurekaClient to Application class
5. Create the Entities that contain the data of user resource

* User class

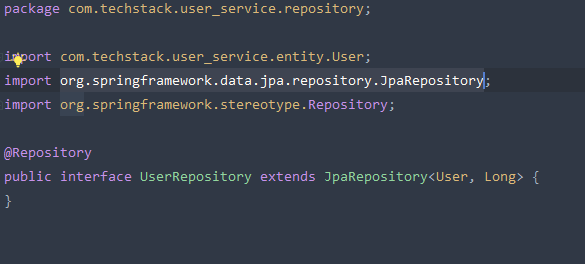
* Address class

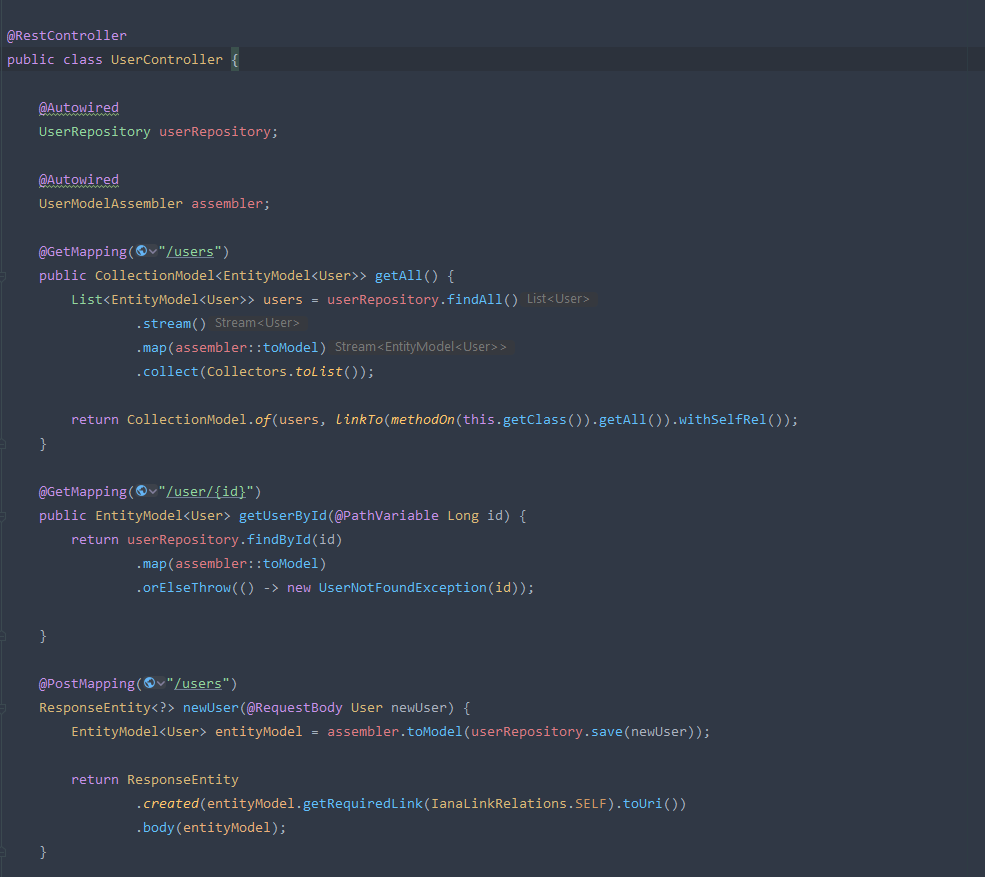
1. To handle the exception, we create UserNotFoundException class which will log out the id of missing user and UserNotFoundAdvice to handle the behavior when the application meet UserNotFoundException.

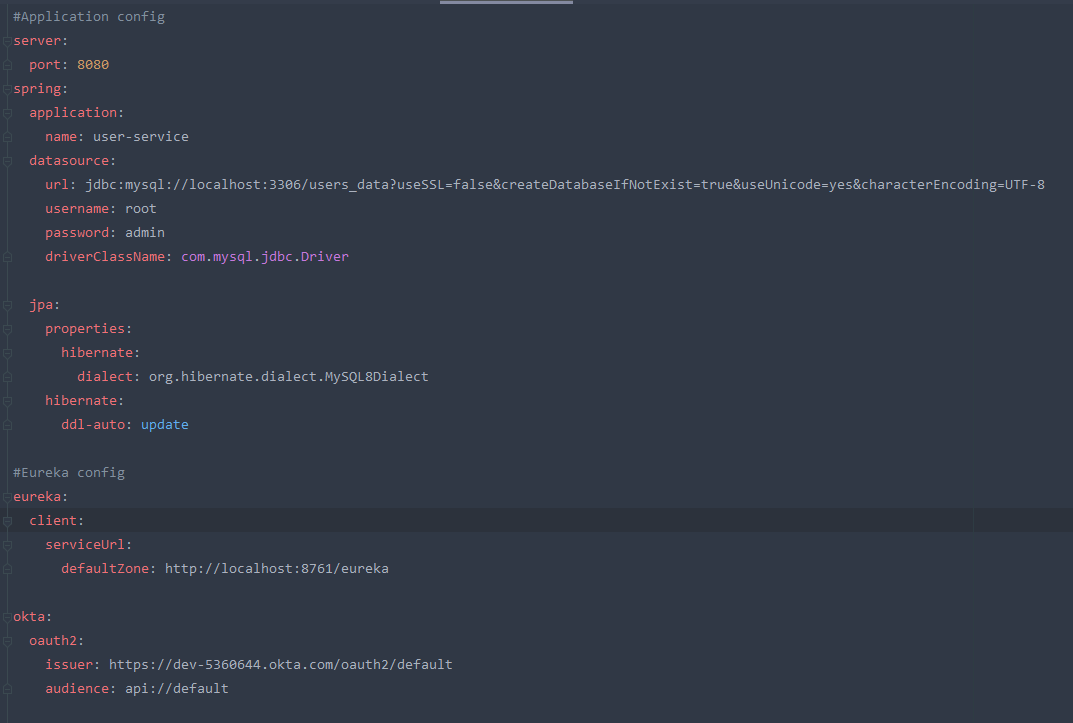


And

In this case, we will return the Error message we defined in UserNotFoundException and return HTTP 404 status in the response

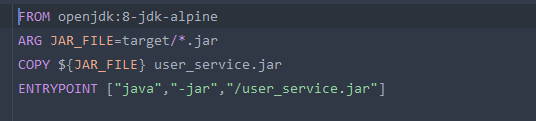
1. Create the service interface that handle the query which usually called repository class. There are multiple implementations of Repository interface of Spring data. In this example, we will extends **org.springframework.data.jpa.repository.JpaRepository** interface
2. To make our API is actually the Rest API and is not RPC according to Roy T. Fielding. We need to implement the helper service that provide the conversion of our entity and **org.springframework.hateoas.EntityModel**
3. Create the controller class, which are the definition of our user-service API. As a said above, we need to convert our entity to some type of response like **EntityModel, ResponseEntity, CollectionModel.** The reason for this is to create the type of response that contains the links of resources.



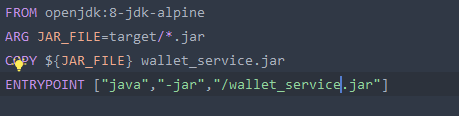
1. Definition of configurations in application.yml file

Explanation

* In this config file, there are some normal declaring such as application port, application name, the datasource configs.
* The config **eureka.client.serviceUrl.defaultZone**: which will define the default location of eureka server;
* The config **okta.oauth2.issuer** to tell Okta which application issuer this user-service project belong to. The other one, **okta.oauth2.audience** means this project is an audience of Okta Authorize server. Both of these config means that, user-service provider is going to used the forwarding token which send from our API gate-way

1. Dockerizing the user-service

## Create wallet-service:

* 1. We will create a similar service which handle the data in user wallet, just make it up with the similar steps in user-service.
  2. Dockerizing the wallet-service

# Starting the project

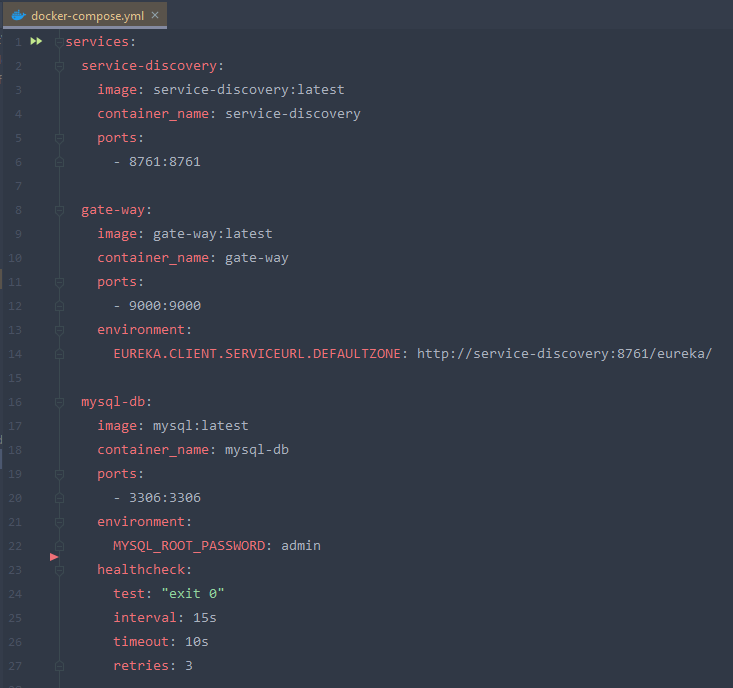
1. To start our project, we could use this command to build our project (Do it case by case for each project)
2. And start each module with the command following the order: Eureka Server -> Gate-way -> User-service, Wallet-service
   * + Ctrl + C to turn off the module

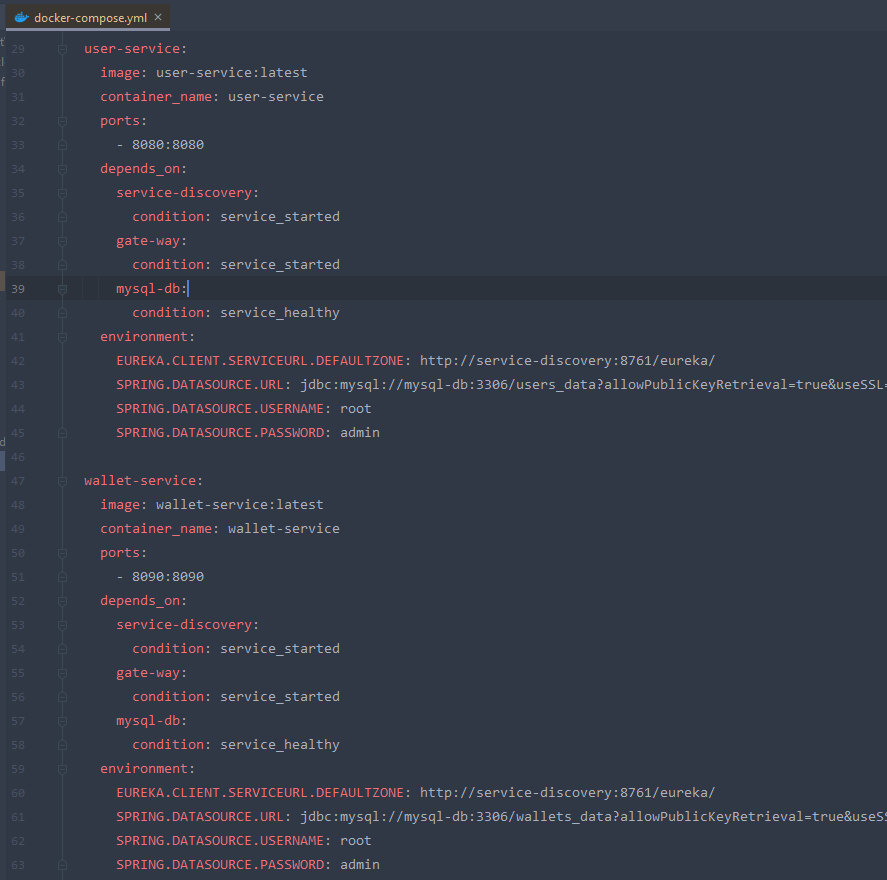
Or, this command to run in the background mode:

And turn off the module in background module

1. Because we use Docker in this example, I prefer a better way to start all the project via Docker

* In the folder location which contain all of our modules (Eureka Server, Gate-way, Service providers), we create a **docker-compose.yml** file which contains the config below:





Explanation

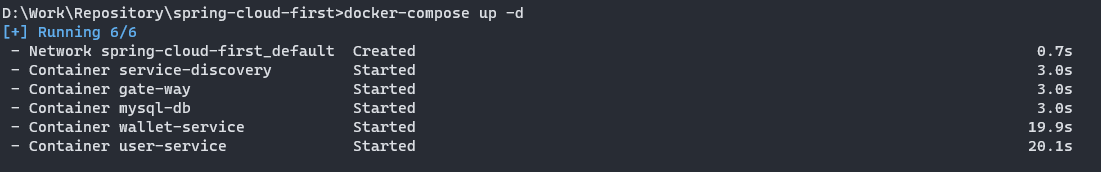
* services: we define our services’ definition inside this
  + service-discovery: this is the service name for our eureka server, this name could be used as the domain name when we define some URL without the need of knowing the exact ip address. (Don’t use underscore \_ in this name, this is the invalid hostname)
    - image: the image that we will use to containerize this service
    - container\_name: the name of this service’s container
    - port: the port mapping, in this example 8761:8761 is the short syntax one which stands for HOST:CONTAINER port
  + gate-way
    - environment: will declare the environment variables of our container. In this case, we define a config EUREKA.CLIENT.SERVICEURL.DEFAULTZONE with an overriding value <http://service-discovery:8761/eureka>. This config will override the value inside application.yml file in gate-way module
  + mysql-db
    - environment: to containerize mysql image, we need to define root password in MYSQL\_ROOT\_PASSWORD config
    - healthcheck: this one will do a healthcheck task base on the config below
      * test: the test condition, in our example, we check if the result is “exit 0” which means everything executed successfully without errors.
      * interval: after the given time the healthcheck task will run again.
      * timeout: the limit time of healthcheck task
      * retries: number of time that the healthcheck will retry.
  + user-service
    - depends\_on: Express dependency between services, when we run the command **docker-compose up** it will start services in dependency order.
      * condition: service\_started (when this service is successfully started)/ service\_healthy (when the healthcheck task of this service shows the healthy result)
  + wallet-service: some definition is similar with user-service

1. Build images for each module by the command

**docker build –t [IMAGE-NAME] .**

After this command, our module will have the docker image

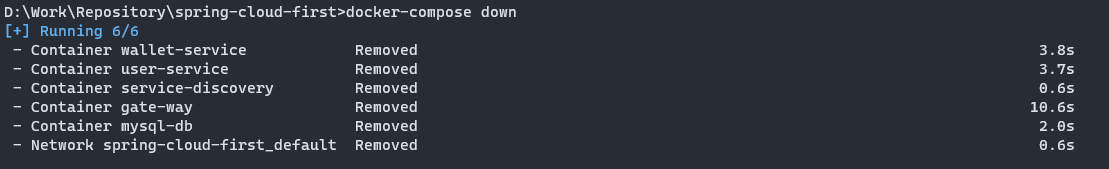
1. Inside the project folder, we run this commands:

Our services will be started, besides that, a default network will be created

It takes wallet-service and user-service about 20s to start because in the first time Docker start them, the mysql-db container healthcheck still return the unhealthy result, after the interval time (15s), the service could be started successfully.

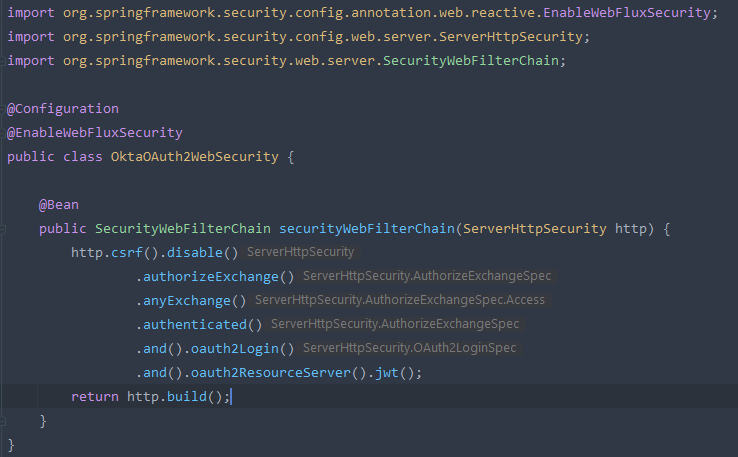
Your docker-compose start-up will be failed. What is the reason?

* + - Your Okta configuration is incorrect.

For the sake of this guidelines, we will run this command to shut down what we have created by **docker-compose up** command

# Okta Configurations:

1. Before you begin, you’ll need a free Okta developer account, Install the Okta CLI.
2. Run Okta CLI redirect to our gate-way project folder.
3. Run **okta login**
4. Run **okta apps create.** Select the default app name, or change it as you see fit
5. Choose Web and press Enter
6. Select Okta Spring Boot Starter. Accept the default Redirect URI values provided for you. That is, a Login Redirect of http://localhost:9000/login/oauth2/code/okta and a Logout Redirect of <http://localhost:9000>. The reason of port 9000 is our gate-way service has the port mapping at 9000.
7. After that, a config file application.properties will be created inside gate-way project which will contains the correct data of 3 configurations which i migrated to the application.yml format
   * **okta.oauth2.issuer**
   * **okta.oauth2.client-id**
   * **okta.oauth2.client-secret**
8. Now, we need to Enable Security in our Gate-way by providing this class



1. Modify the config okta.oauth2.issuer in user-service and wallet-service with the same value as gate-way module

# Testing

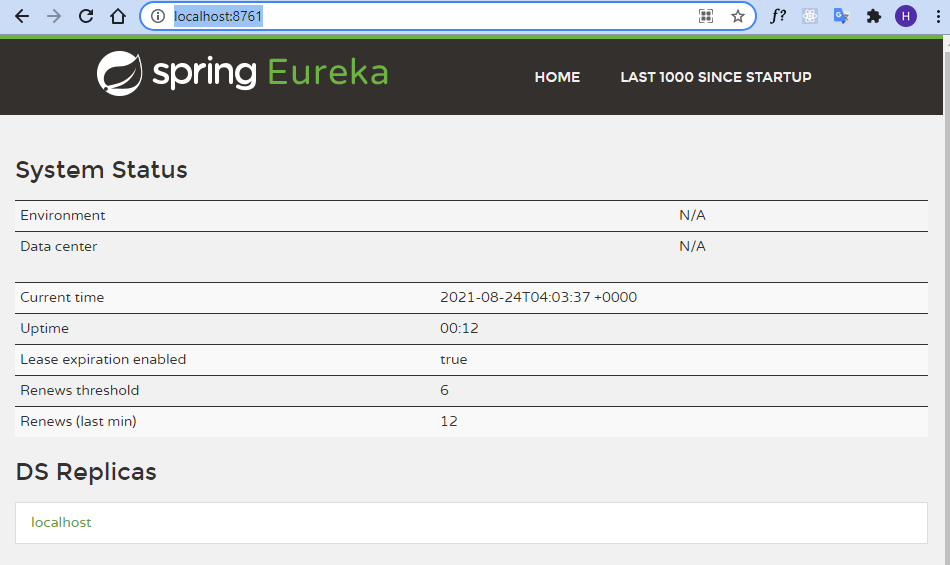
Now we back to our docker-compose step

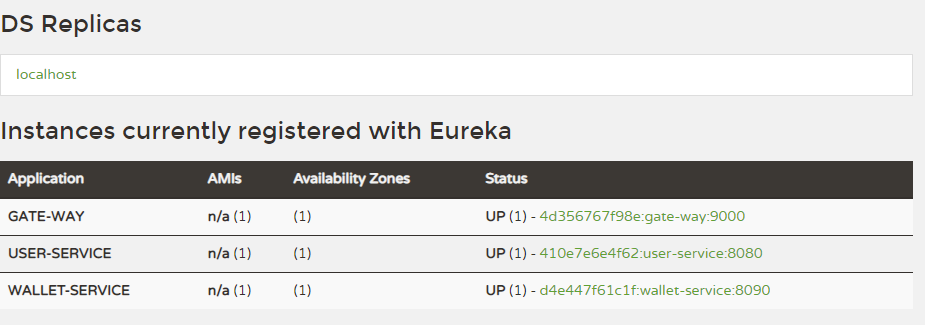
1. First thing first, we need to build the jar files again and update the docker images for each module by the commands

**mvn clean install –DskipTests=True**

and

**docker build –t [IMAGE-NAME] .**

1. Now we run **docker-compose up -d** and the services should be start successfully.
2. Access the url: <http://localhost:8761/> As default, this is the admin page of Eureka, we should see all the service instance register here.



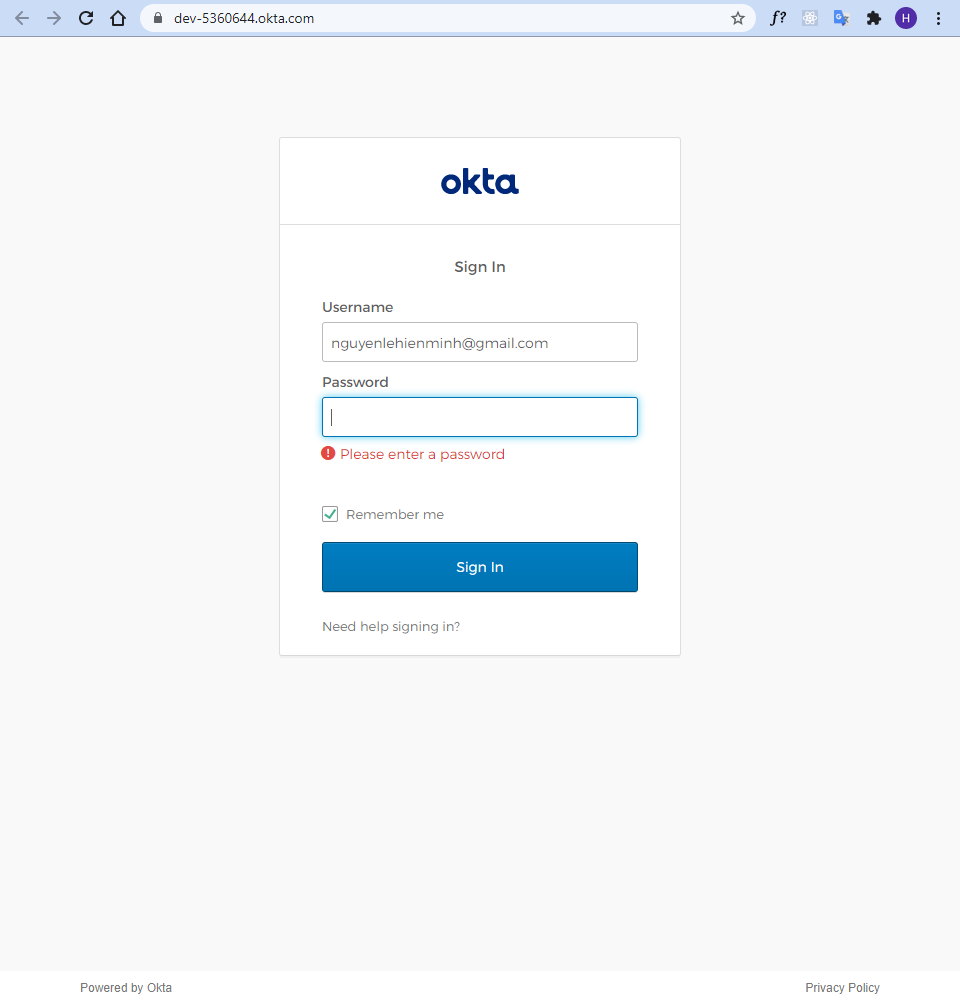
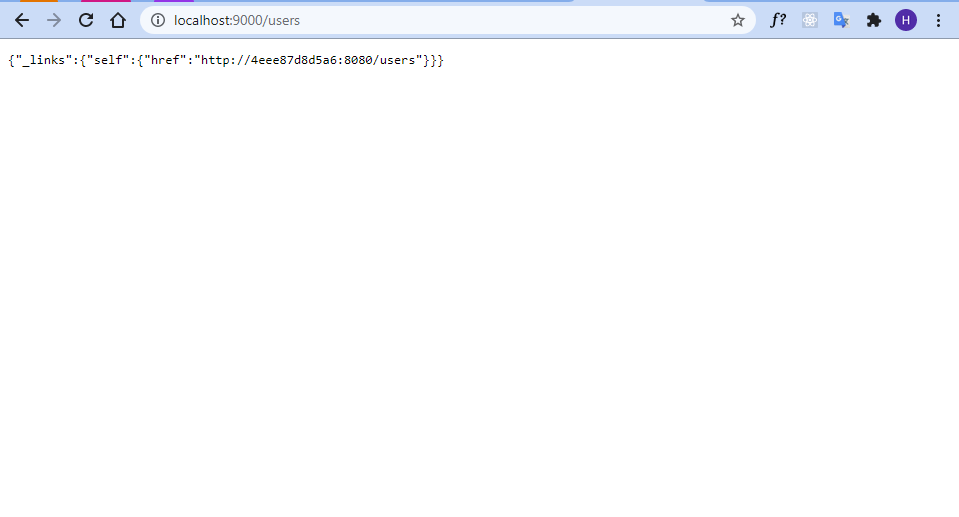
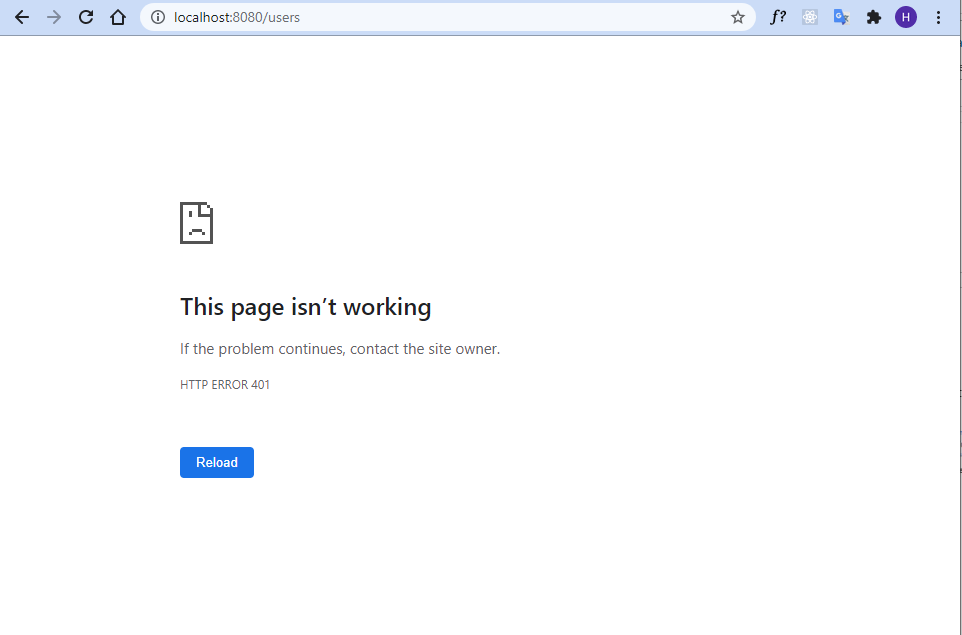
Gate-way Route Locator explanation

As you can see in Eureka registered instance section, there are 3 instance which was our services such as gate-way, service providers (user-service, wallet-service). Notice in the Application column, there are the names with are also the ID of these services

****Take a look back in the RouteLocator Bean:

* This routing shows that every path that has the pattern **/user/\*\*** will be filtered and redirect to the uri **lb://USER-SERVICE.** Notice that the lb://USER-SERVICE is the id of user-service module in Eureka server registering. The scheme lb tells the Spring Cloud LoadBalancerClient to resolve the name USER-SERVICE.
* The similar one is wallet-service

Now we try to user this forwarding through our gate-way module:

1. Access <http://localhost:9000/users>, we will see default Okta login page appears
2. Enter password and the system will redirect us to correct resource with the credential we use in gate-way.
3. If we try to access this resource directly without using gate-way, there is a 401 Unauthorize response appears

# Conclusion:

So far this is all we need for a basic Spring boot system with SSO server is Okta and Service Register Service is Eureka.

* Our service modules could be registered successfully to the Eureka server
* Our gate-way could get the instance from Eureka server and forwarding to the right resource
* SSO is applied in the gate-way level, other service providers is hidden and can not be accessed directly without the credential from gate-way.