Microservices with Spring Boot

Software Required

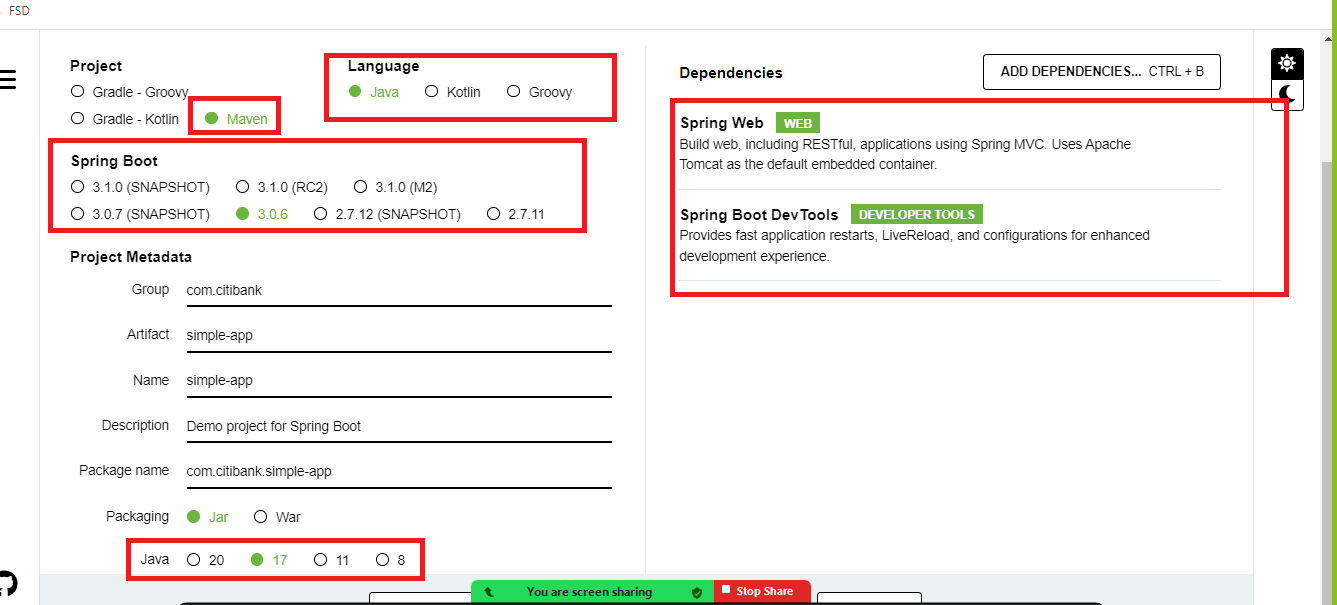
* Java 8 or later
* Eclipse / STS
* Postman
* MySQL

Spring Boot:

It automates all the generic setup for the applications so that you can quickly create spring related applications by concentrating only the application level instead of concentrating on the infrastructure level

Note: Spring Boot can be used only with build tools like Maven or Gradle, they will take care of downloading all the libraries for your project

Project Template



In Spring Boot, Java objects are automatically converted to JSON, we can use any custom objects or predefined APIs in Java like Map, List and so on.

Pre-requisites for microservices in Spring Boot

* Dependency Injection
* Layered architecture like @Service, @RestController, @Repository
* Annotations like @Autowired, @Bean, @Configuration, @Component

Postman app:

It is used to test the webservices

Most of the webservices will have CRUD operations with HTTP methods like post, put, get, delete, we have corresponding annotations from spring boot to create webservices of different methods

* GetMapping
* PostMapping
* PutMapping
* DeleteMapping



Today’s Agenda

* Configuring the project to use base machine JRE
* Creating an executable JAR & running through command prompt
* Microservices

Configuring the project to use base machine JRE

1. Setup the version in pom.xml
2. Change the project build path to use base machine JDK
3. Change the compiler version that matches to base machine JDK
4. Update the project

Building the project into a single artifact

1. You need to use a maven package command from the eclipse or through command prompt, here jar will be created inside target folder
2. You need to run the jar file using “java -jar *filename.jar* <<options>>”

Microservice:

These are loosely coupled services which are independent from other services

Benefits

* Services are loosely coupled
* Testing will be easier as we need to only test a particular service we want
* We can choose any language we want
* We can scale the service which we want
* If a particular service goes down it affects only the services that are down, and other services will be available.

Challenges:

1. Expensive - because we need to many resources
2. Important to identify the right approach
3. Adapting to the changes
4. Global competition

Design principles in microservices

1. Service Discovery: Registers the microservices Instance Id & physical address
2. Discovery Client: These are microservices which locates other microservices using the instance ID, it must send acknowledgement to the service discovery frequently about its health status
3. Client Side Load Balancer: Takes care of distributing the load across multiple instances of microservices
4. Fault Tolerance - Circuit breaker pattern: It takes care of breaking the circuit to stop sending the requests to the already down service
5. Distributed configuration
6. API gateway
7. Distributed log tracing

Spring provides a project called Spring Cloud which has already implemented these design patterns for us

Spring cloud: It uses Netflix OSS library to implement all these design patterns

Microservices with spring uses two projects of spring

* Spring Cloud - gives all the tools & design patterns for microservices
* Spring Boot - to easily create applications using automated features available

Note: We can use simple annotations to add the design patterns we want in our code

ex: @EnableEurekaServer is used to create service discovery

@EnableEurekaClient is used to create discovery client

@LoadBalanced is used to create client side load balancer

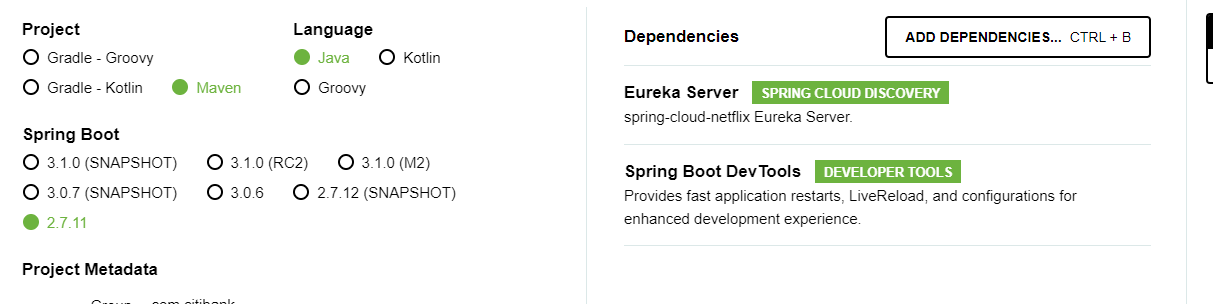
We need to create minimum 2 projects

1. Service Discovery
2. Discovery Client

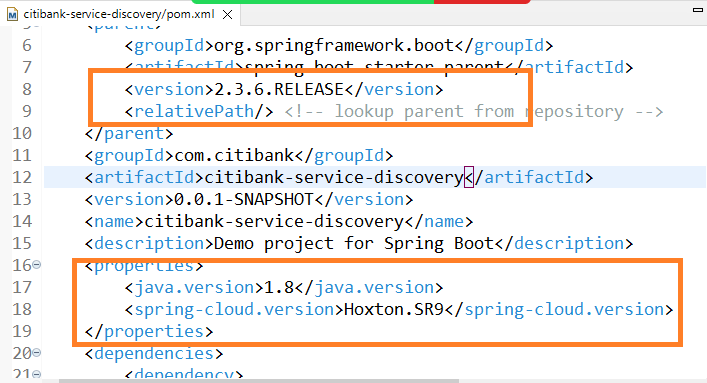
Note: Since we are using spring cloud & spring boot projects, we must refer to the spring website to see the version compatibility

Service discovery project

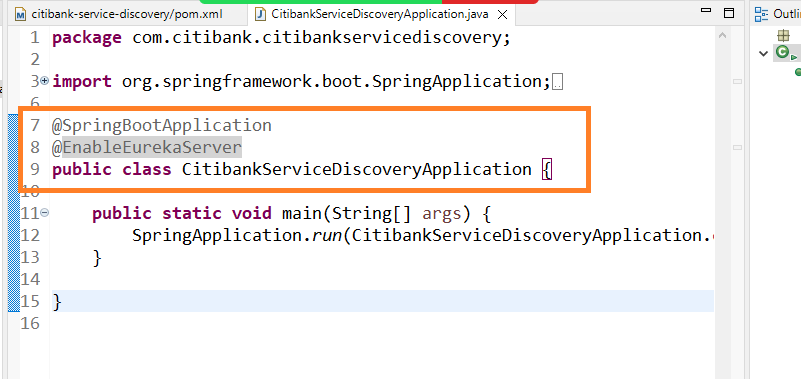
1. Eureka Server
2. Devtools



pom.xml



Service Discovery Enabling

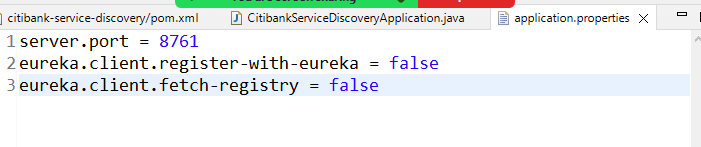


Since Eureka Server downloads Eureka Client library we need to disable client feature in the application

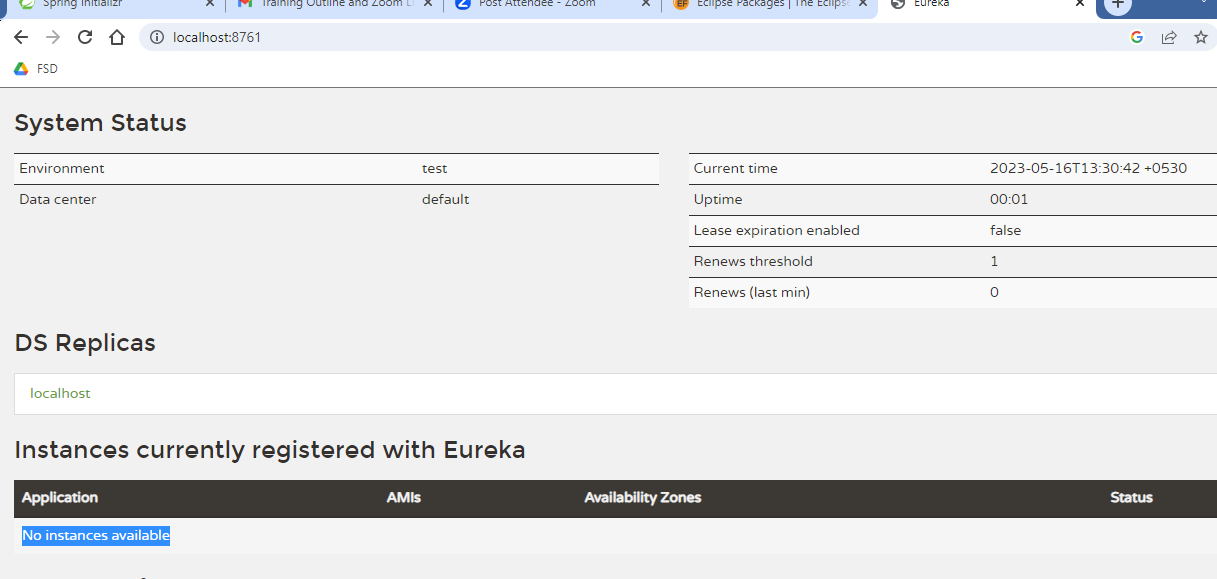
like fetching the registry, registering as a service

Note: By default all the microservice searches eureka server in 8761 port, hence you need to run eureka server in 8761

application.properties



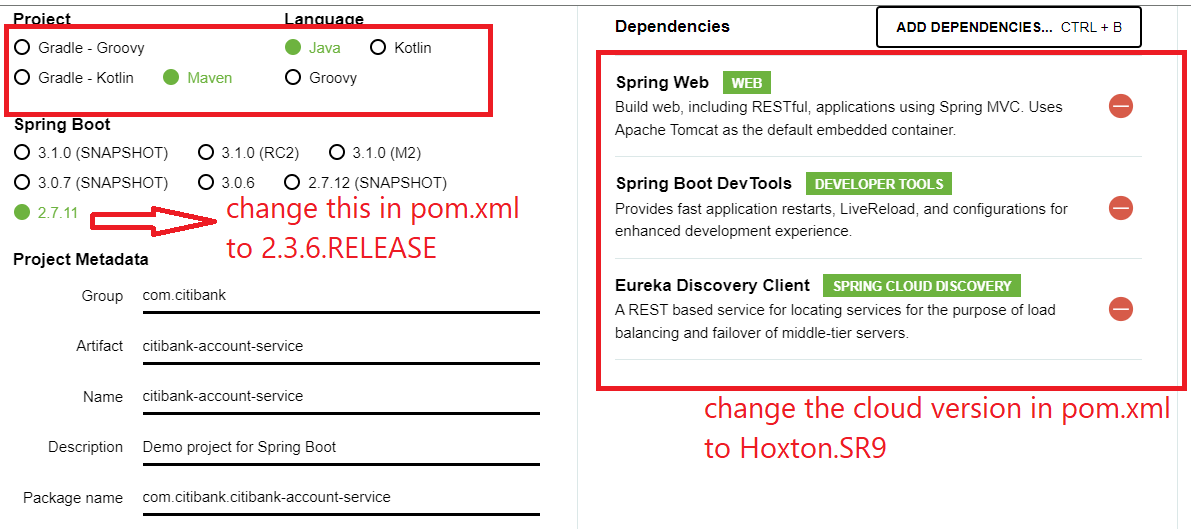
Note: Eureka Server gives you a client dashboard so that you can see all the services



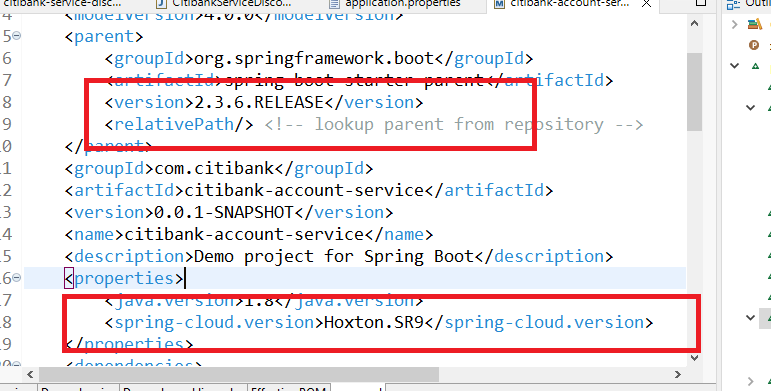
Note: While using microservices avoid running the programs in eclipse, instead use executable jars

Discovery client project

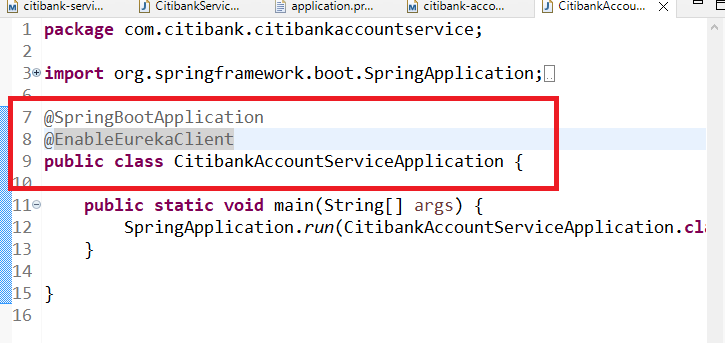
1. Eureka Client
2. Web
3. Devtools



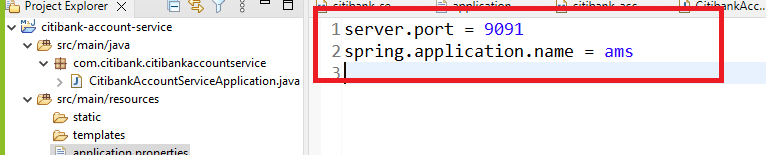
Change the pom.xml as per the below configuration



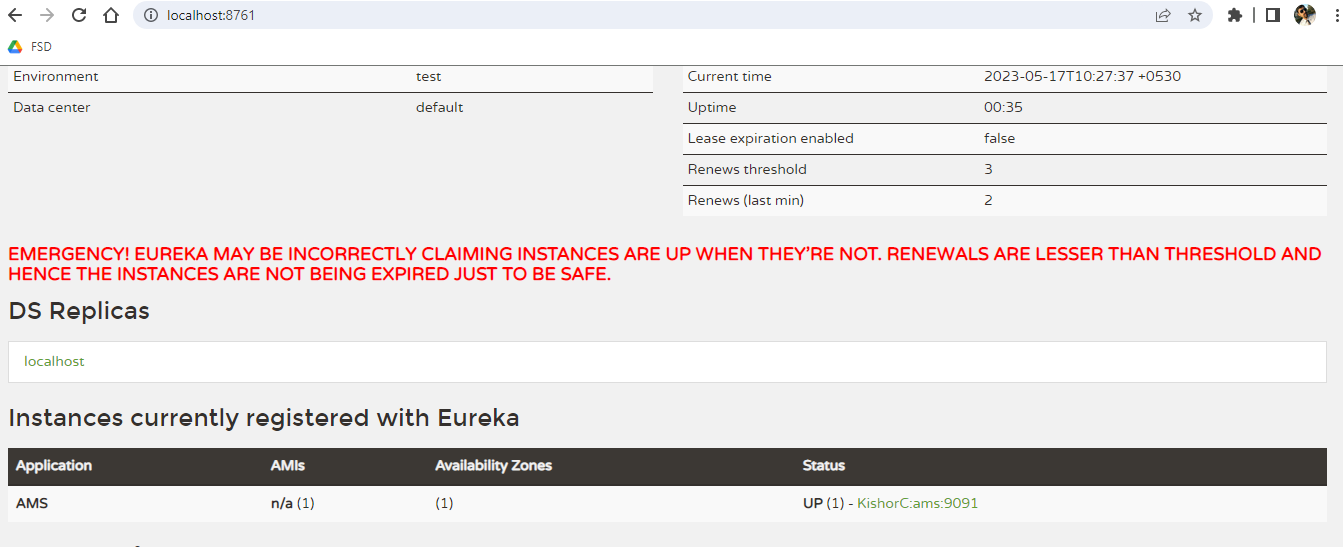
To register this as a microservice we must use @EnableEurekaClient in our application



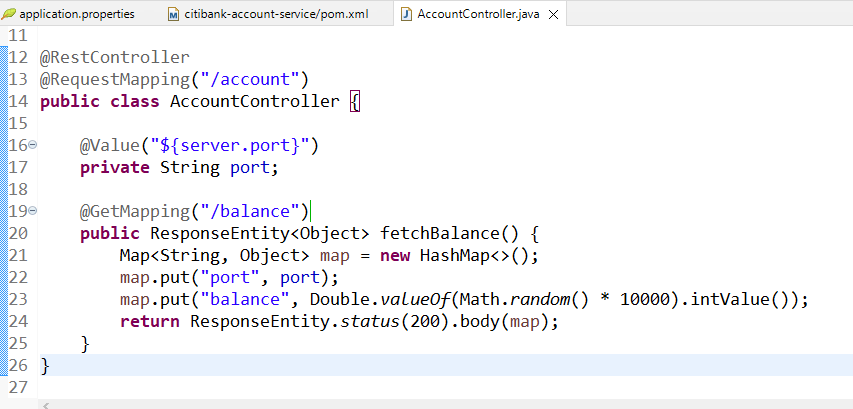
application.properties



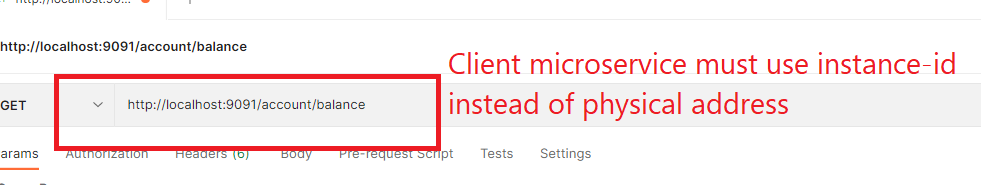
Create an executable jar & run it in command prompt



Create one controller that can return some data to the client



We need to access this microservice from another microservice but to test we can use postman with the physical address.



Another microservice that acts as a client to call a remote microservice (AMS)

RestTemplate: It is an object used to call the remote service

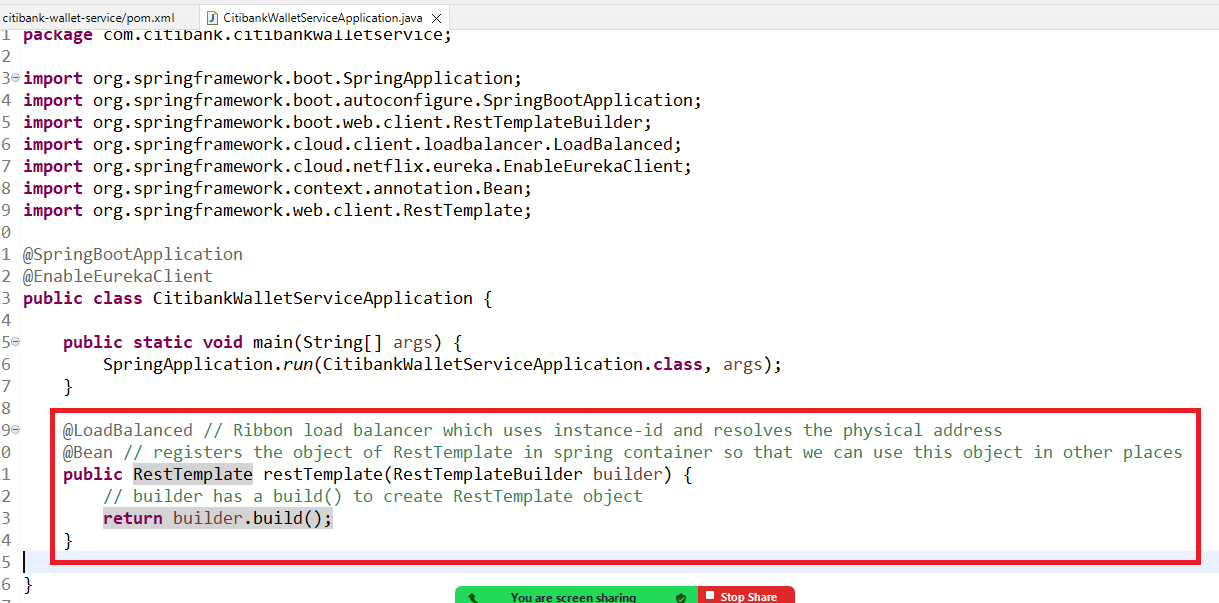
LoadBalancer: Spring cloud gives a library called Ribbon which provides the client side load balancer, we need to use @LoadBalanced annotation to add the load balancer to the RestTemplate

@LoadBalanced  
@Bean  
public RestTemplate restTemplate() {   
 ….  
}

Dependencies

1. Eureka Client
2. Web
3. Dev tools

Enable the microservice and Ribbon load balancer in the application



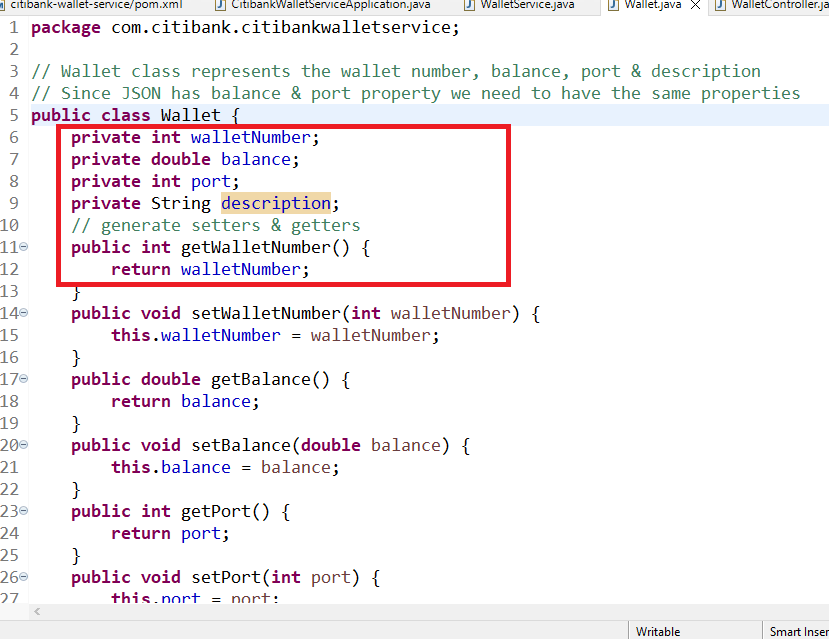
RestTemplate: It is an object used to access remote service using URL and HTTP methods like get, post, put, delete

We need to create a Service layer that would access the remote service & a controller that would access the Service layer

Create 3 classes

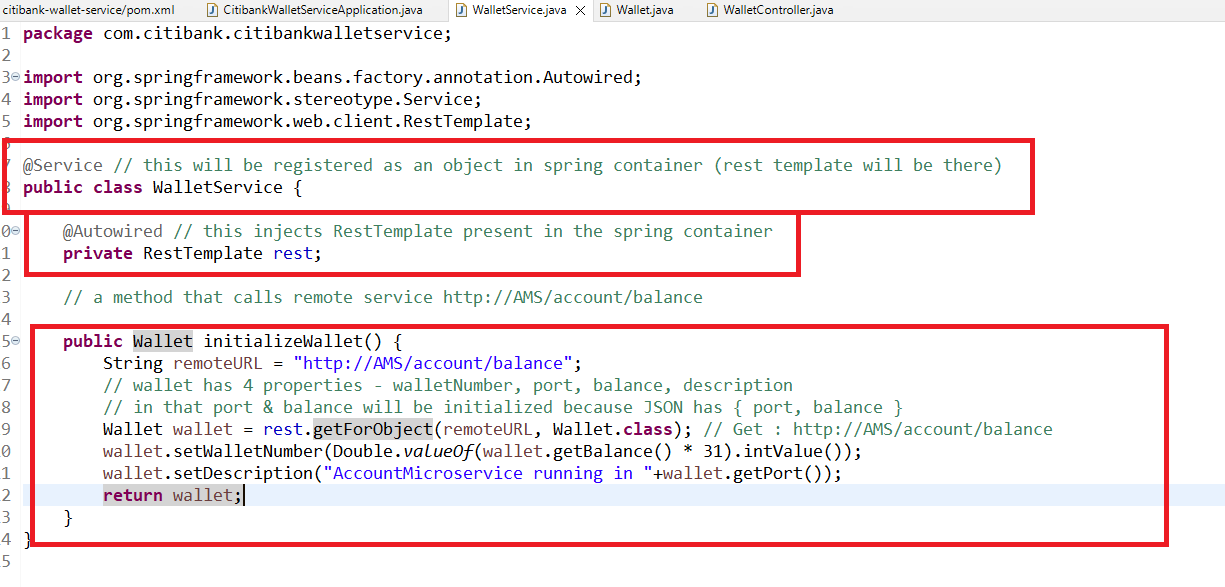
1. WalletService : This will call remote service & gets its data
2. Wallet: a model that represents the remote service & current service data
3. WalletController: this calls the wallet service methods

Wallet.java

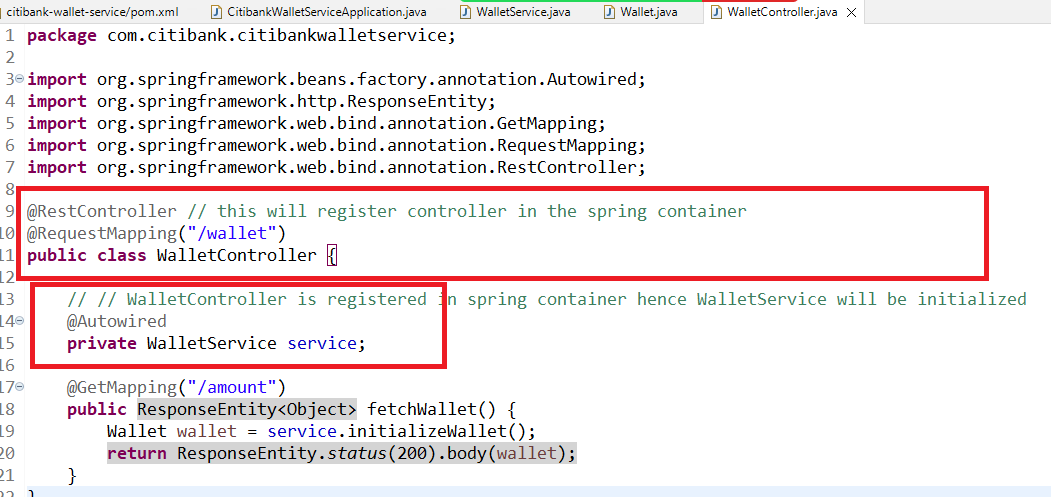


WalletService.java

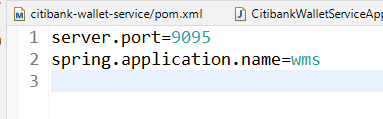
Bind the RestTemplate object here & call Account Microservice using RestTemplate & initialize the Wallet in any one method



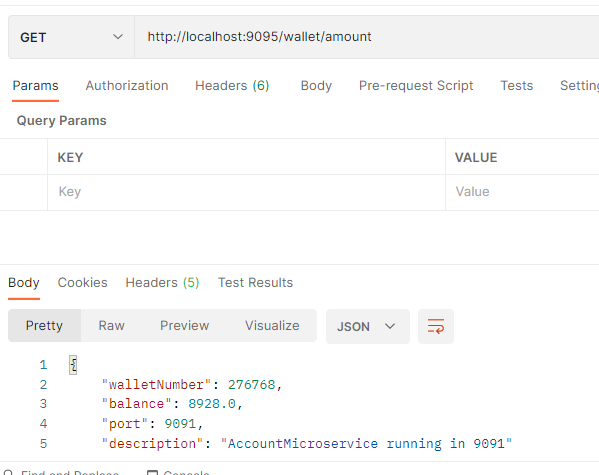
Now you can call this intializeWallet from the controller



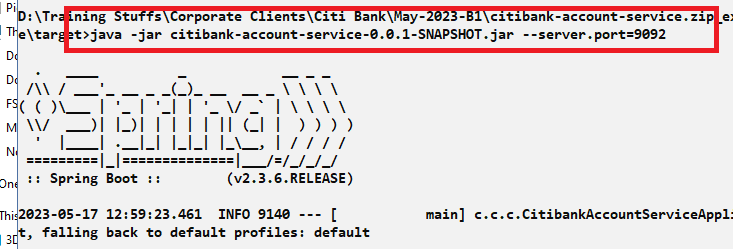
application.properties



Now you can run this project & send request to wallet/balance



Creating another instance of account microservice to verify the load-balancer job



Try to communicate with the microservice without service discovery and check whether load balancer is able to delegate request.