# COMMUNICATION OF MICROSERVICES-

Synchronous->Direct Reques/Response

Asynchronous->we need messaging queue like Kafka/Rabbit MQ

Steps to Add RestTemplate:-

1)Add department code field in Employee JpA Enity.

Yes since we are getting details from different with Department Details ,so we need to add that object of a class into a entity class microservice.

2)Create DepartmentDto class

You need to create department class to store the department details.

3)Configure RestTemplate as Spring Bean in the main class to create bean of the main class.

@Bean

**public** RestTemplate restTemplate() {

**return** **new** RestTemplate();

}

4)Inject and use RestTemplate to make call in EmployeeService Impl.class

You can inject the Rest Template to the Required class by using-

@Autowired

**private** RestTemplate restTemplate;

Then create the class of another Microservice for which you want to post the data or fetch the data to store it in a Object with constructor and getter and setter.

**public** **class** BankLoginCreds {

**private** String email;

**private** String password;

}

Then Directly you can create a method to create your own Api to connect to other Microservice-

@PostMapping("/validateBankCreds")

**public** Boolean validateBank(@RequestBody BankLoginCreds loginCreds) {

//restTemplate.getForEntity("http://localhost:9999/obs/consumerlogin")

ResponseEntity<Boolean> responseEntity=restTemplate.postForEntity("http://localhost:9999/obs/data/userlogin", loginCreds,Boolean.**class**);

Boolean b=responseEntity.getBody();

**return** b;

}

//we can use .getForEntity(URL,object,Return Type) to get the required data and we can store the same into the variables of object.

## Microservice communication using WebClient-

Steps:-

1)Add Spring Webflux dependency

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-webflux</artifactId>

</dependency>

2)Configure WebClient as Spring Bean.

@Bean

**public** WebClient webClient() {

**return** WebClient.*builder*().build();

}

3)Inject and use webClient to call Rest API.

Boolean b=webClient.post().uri("http://localhost:9999/obs/data/userlogin").body(Mono.*just*(loginCreds), BankLoginCreds.**class**).retrieve().bodyToMono(Boolean.**class**).block();

4)Test using Postman Client.

And we will get the result code uploaded on github. Railway\_Reservation\_updated/feature\_web\_client

@GetMapping("/getBankID/{mail}")

**public** String getBankID(@PathVariable String mail){

String BankId=webClient.get().uri("http://localhost:9999/obs/data/getId/" + mail).retrieve().bodyToMono(String.**class**).block();

**return** BankId;

}

//Get Mapping using getBankId

## Use Of Feign Client for Communication(My code not working)-

Steps-

1. Add Spring Cloud open feign Maven –dependency to the Service.
2. Enable Feign Client using @EnableFeignClient
3. Create Feign Client
4. Change the empId to use feign Client

Go to start.spring.io🡪search openfeign🡪explore and Copy dependency

Openfeign is a spring cloud deplendency so we have to add spring cloud-starter dependencies to add any spring cloud dependency of type Pom.

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-openfeign</artifactId>

</dependency>

--------------------------------------------------------------------------------------------------------------------------------------

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>${spring-cloud.version}</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

----------------------------------------------------------------------------------

In Properties add-

<spring-cloud.version>2023.0.3</spring-cloud.version>

--------------------------------------------------------------------------------------------------------------------------------------

Add @EnableFeignClients is used to enable component scanning for interfaces.

Now you need to create an interface with @FeignClient annotation.

Create Interface APIClient.-->

@FeignClient(url = "http://localhost:9999" ,value="Bank-service")

**public** **interface** APIClient {

@GetMapping("/obs/data/getId/{email}")

**public** String gettheId(@PathVariable String email);

}

Update the Controller as per requirement-

@GetMapping("/getBankID/{mail}")

**public** String getBankID(@PathVariable String mail){

//String BankId=webClient.get().uri("http://localhost:9999/obs/data/getId/" + mail).retrieve().bodyToMono(String.class).block();

String BankId=apiClient.gettheId(mail);

**return** BankId;

}