SPRING BOOT MICROSERVICES PROJECT

Name : ABHAY MISHRA Trainer Name : DR. SIDDARTH SHARMA
Batch No. : 3(JAVA)

Index

Objective:

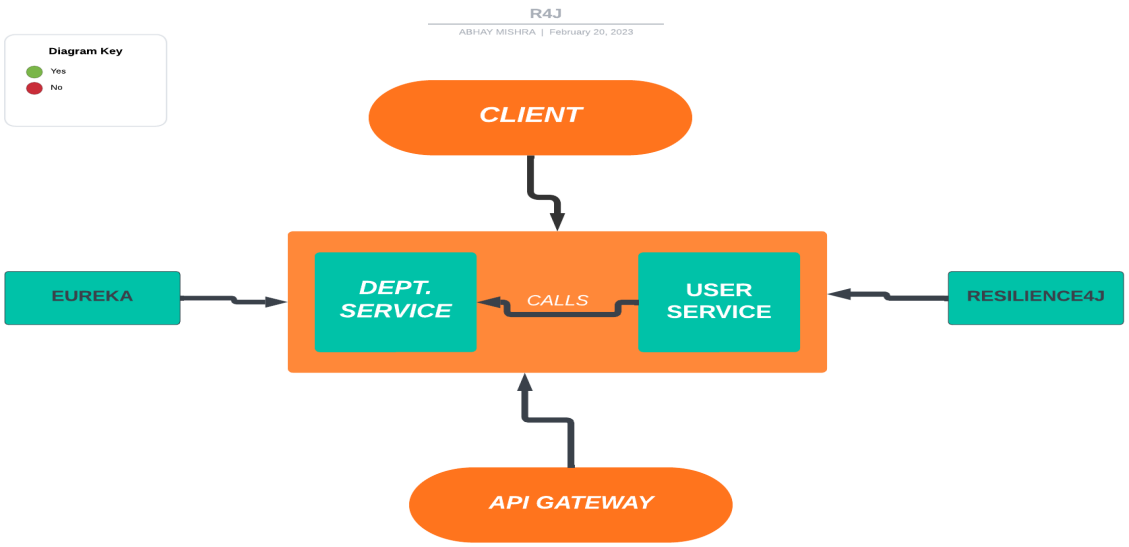
To building circuit breaker with the help of "resilience4j" to trace which of microservices in the project is not responding and to generate proper message to the client by eradicating the white label error page .

Technical Stack Used:

- Java 17
- Maven as Building Tool
- Spring Boot Framework
- IntelliJ IDE
- Postman For API Testing
- H2 in-memory database
- Lombok for BoilerPlate Code
- Resillience4j for CircuitBreaker

- Slf4j for logging information
- Eureka Server as Discovery server

Application Architecture Diagram:



Spring Framework:

Spring is a *lightweight* framework. It can be thought of as a *framework of frameworks* because it provides support to various frameworks such as Struts, Hibernate, Tapestry, EJB, JSF, etc. The framework, in broader sense, can be defined as a structure where we find solution of the various technical problems.

The Spring framework comprises several modules such as IOC, AOP, DAO, Context, ORM, WEB MVC etc. We will learn these modules in next page. Let's understand the IOC and Dependency Injection first.

Microservices:

In the **Microservices** tutorial, we will understand how to implement microservices using **Spring Cloud**. We will learn how to establish communication between microservices, **enable load balancing**, **scaling up and down of microservices**. We will also learn to **centralize the configuration of microservices** with **Spring Cloud Config Server**. We will implement **Eureka Naming Server** and **Distributed tracing** with **Spring Cloud Sleuth** and **Zipkin**. We will create fault tolerance microservices with **Zipkin**.

Our **microservices** tutorial discusses the basic functionalities of **Microservice Architecture** along with relevant examples for easy understanding.

Eureka Server:

Eureka naming server is an application that holds information about all client service applications. Each microservice registers itself with the Eureka naming server. The naming server registers the client services with their **port numbers** and **IP addresses**. It is also known as **Discovery Server.** Eureka naming server comes with the bundle of Spring Cloud. It runs on the default port **8761**. It also comes with a Java-based client component, the eureka client, which makes interactions with the service much easier.

Resilience4J:

Resilience4j is a lightweight fault tolerance library designed for functional programming. Resilience4j provides higher-order functions (decorators) to enhance any functional interface, lambda expression or method reference with a Circuit Breaker, Rate Limiter, Retry or Bulkhead. You can stack more than one decorator on any functional interface, lambda expression or method reference. The advantage is that you have the choice to select the decorators you need and nothing else.

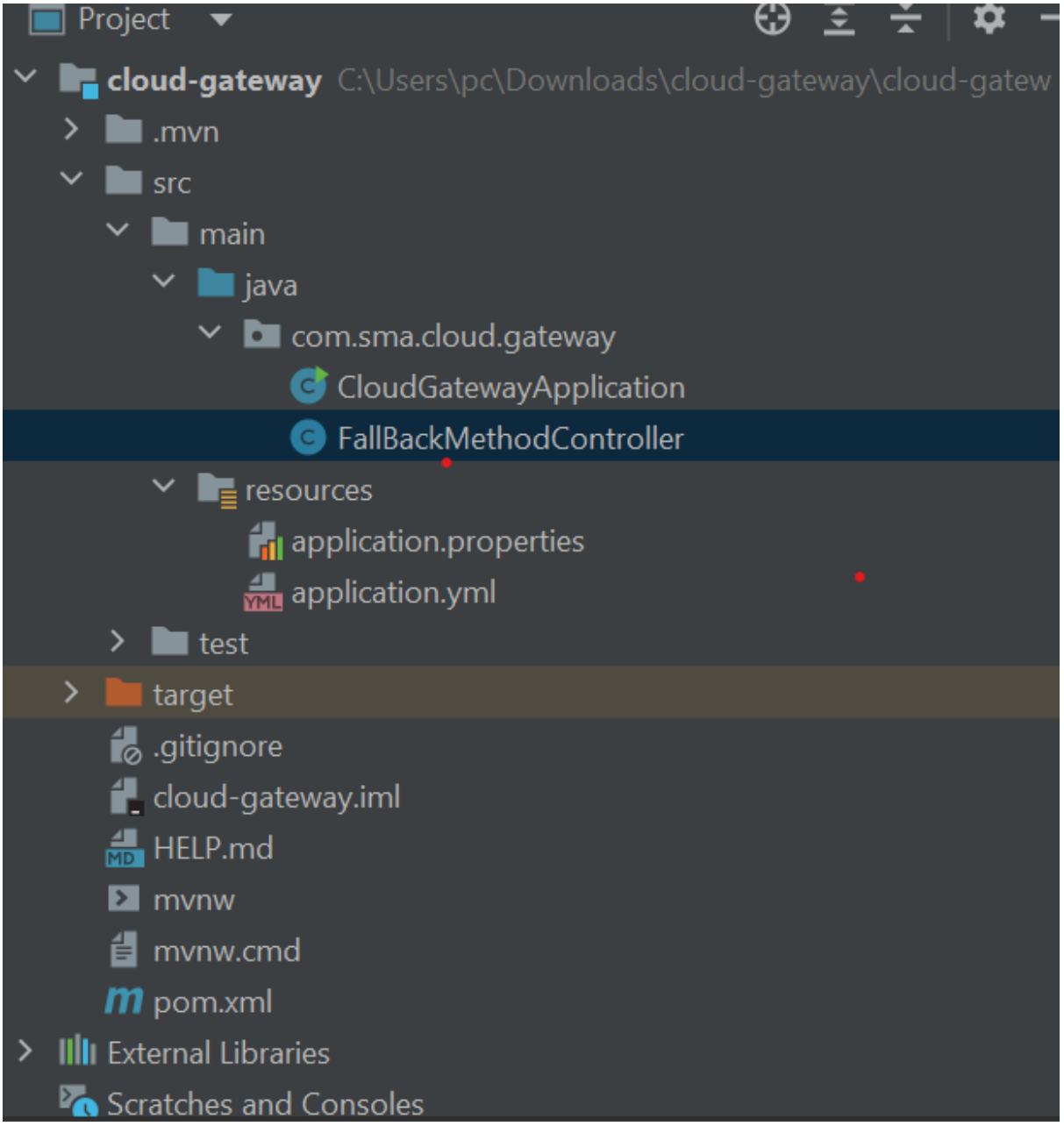
NOTE: Resilience4j 2 requires Java 17.

Resilience4j provides higher-order functions (decorators) to enhance any functional interface, lambda expression or method reference with a Circuit Breaker, Rate Limiter, Retry or Bulkhead. You can stack more than one decorator on any functional interface,

lambda expression or method reference. The advantage is that you have the choice to select the decorators you need and nothing else.

API Gateway for microservices:

File structure:


Project
cloud-gateway C:\Users\pc\Downloads\cloud-gateway\cloud-gatew
.mvn
src
main
java
com.sma.cloud.gateway
CloudGatewayApplication
FallBackMethodController
resources
application.properties
application.yml
test
target
.gitignore
cloud-gateway.iml
HELP.md
mvnw
mvnw.cmd
pom.xml
External Libraries
Scratches and Consoles

An API gateway is one part of an API management system. The API gateway intercepts all incoming requests and sends them through the API management system, which handles a variety of necessary functions.

Exactly what the API gateway does will vary from one implementation to another. Some common functions include authentication, routing, rate limiting, billing, monitoring, analytics, policies, alerts, and security.

PORT NO :9191

To create an API Gateway in Spring Boot, you can follow these steps:

step1. Add dependency to pom.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<project                                xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
     <groupId>org.springframework.boot</groupId>
     <artifactId>spring-boot-starter-parent</artifactId>
     <version>3.0.2</version>
     <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.dailycodebuffer</groupId>
  <artifactId>cloud-gateway</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>cloud-gateway</name>
  <description>cloud-gateway</description>
  <properties>
     <java.version>19</java.version>
     <spring-cloud.version>2022.0.1</spring-cloud.version>
  </properties>
  <dependencies>
     <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
     </dependency>
     <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-aop</artifactId>
     </dependency>
     <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
     </dependency>
     <dependency>
        <groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-circuitbreaker-resilience4j</artifactId>
     </dependency>
```

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

<artifactId>spring-cloud-starter-circuitbreaker-reactor-resilience4j</artifac
tId>
       <version>2.1.0</version>
    </dependency>
    <dependency>
       <groupId>org.springframework.boot</groupId>
       <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>
    <dependency>
       <groupId>org.springframework.cloud</groupId>
       <artifactId>spring-cloud-starter-gateway</artifactId>
    </dependency>
    <dependency>
       <groupId>org.springframework.cloud</groupId>
       <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
    </dependency>

    <dependency>
       <groupId>org.springframework.boot</groupId>
       <artifactId>spring-boot-starter-test</artifactId>
       <scope>test</scope>
    </dependency>
  </dependencies>
  <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>

  <build>
    <plugins>
       <plugin>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-maven-plugin</artifactId>
       </plugin>
    </plugins>
  </build>
  <repositories>
    <repository>
       <id>netflix-candidates</id>
       <name>Netflix Candidates</name>

<url>https://artifactory-oss.prod.netflix.net/artifactory/maven-oss-candidate
s</url>
       <snapshots>
          <enabled>false</enabled>
       </snapshots>
    </repository>
  </repositories>

</project>
```

step2. Configuration in application.yml or application.properties file:

```yaml
server:
 port: 9191

spring:
 application:
   name: API-GATEWAY
 cloud:
   gateway:
     routes:
       - id: USER-SERVICE
         uri: lb://USER-SERVICE
         predicates:
           - Path=/users/**
         filters:
           - name: CircuitBreaker
             args:
               name: USER-SERVICE
               fallbackuri: forward:/userServiceFallBack
       - id: DEPARTMENT-SERVICE
         uri: lb://DEPARTMENT-SERVICE
         predicates:
           - Path=/departments/**
         filters:
           - name: CircuitBreaker
             args:
               name: DEPARTMENT-SERVICE
               fallbackuri: forward:/departmentServiceFallBack

management:
 health:
   circuitbreakers:
     enabled: true
 endpoints:
   web:
     exposure:
       include: health
 endpoint:
   health:
     show-details: always

resilience4j.circuitbreaker:
 configs:
   default:
     slidingWindowSize: 4
     permittedNumberOfCallsInHalfOpenState: 10
     waitDurationInOpenState: 10000
     failureRateThreshold: 60
     eventConsumerBufferSize: 10
     registerHealthIndicator: true
   someShared:
     slidingWindowSize: 3
     permittedNumberOfCallsInHalfOpenState: 10
 instances:
   user-service:
     baseConfig: default
     waitDurationInOpenState: 500000
   departmentService:
     baseConfig: default
     waitDurationInOpenState: 500000
```

```
   backendB:
     baseConfig: someShared



eureka:
 client:
   register-with-eureka: true
   fetch-registry: true
   service-url:
     defaultZone: http://localhost:8761/eureka/
 instance:
   hostname: localhost

           - Path=/service2/**
```

3. Add @EnableDiscoveryClient

```
package com.sma.cloud.gateway;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;

@SpringBootApplication
@EnableDiscoveryClient
public class CloudGatewayApplication {

  public static void main(String[] args) {
     SpringApplication.run(CloudGatewayApplication.class, args);
  }

}

}
```

step4. Java class to handle Fallback methods

```
package com.sma.cloud.gateway;


import io.github.resilience4j.circuitbreaker.annotation.CircuitBreaker;
import lombok.extern.slf4j.Slf4j;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@Slf4j
public class FallBackMethodController {
   @GetMapping("/userServiceFallBack")
               @CircuitBreaker(name="userService",     fallbackMethod     =
"userServiceFallbackMethod")
   public String userServiceFallBackMethod(){
       log.info("Inside userServiceFallBackMethod");
       return "User Service is taking longer than expected."+
               " Please Try again later";

   }

   @GetMapping("/departmentServiceFallBack")
```

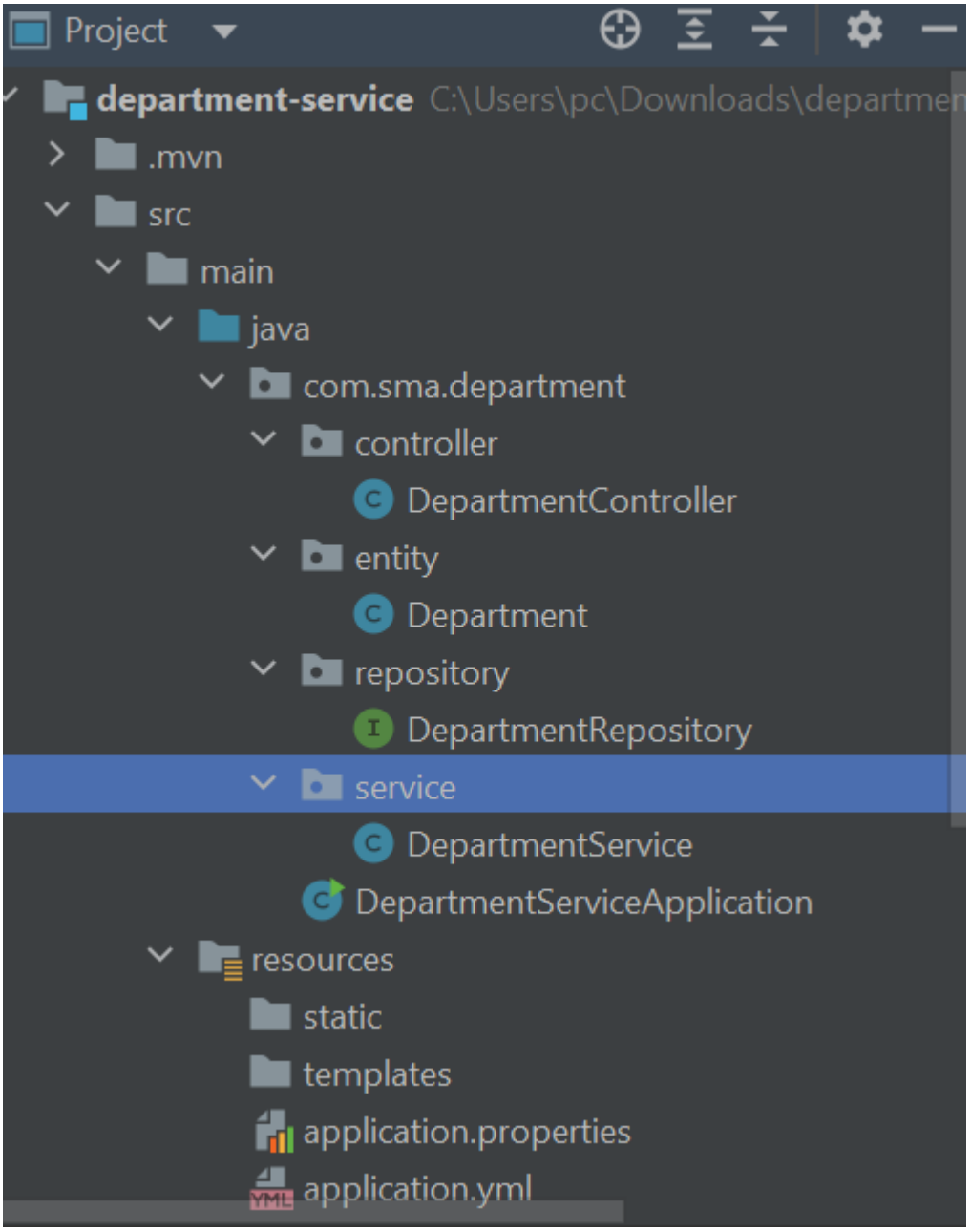
```
            @CircuitBreaker(name="departmentService",    fallbackMethod    =
"departmentServiceFallbackMethod")
   public String departmentServiceFallBackMethod(){
        log.info("Inside departmentServiceFallBackMethod");
        return "Department Service is taking longer than expected."+
                " Please Try again later";

   }
}
```

Screenshot of API Gateway Running Successfully:

```
Discovery Client initialized at timestamp 1676917639047 with initial instances count: 2
Registering application API-GATEWAY with eureka with status UP
Saw local status change event StatusChangeEvent [timestamp=1676917639090, current=UP, previous=STARTING]
DiscoveryClient_API-GATEWAY/192.168.1.33:API-GATEWAY:9191: registering service...
DiscoveryClient_API-GATEWAY/192.168.1.33:API-GATEWAY:9191 - registration status: 204
Netty started on port 9191
Updating port to 9191
Started CloudGatewayApplication in 15.654 seconds (process running for 16.889)
```

Department microservices:

File Structure:



PORT NO :9001

To create an Department microservice in Spring Boot, you can follow these steps:

step1. Add dependency to pom.xml:

```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
```

```
<modelVersion>4.0.0</modelVersion>
<parent>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-parent</artifactId>
   <version>3.0.2</version>
   <relativePath/> <!-- lookup parent from repository -->
</parent>
<groupId>com.sma</groupId>
<artifactId>department-service</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>department-service</name>
<description>department-service </description>
<properties>
   <java.version>17</java.version>
   <spring-cloud.version>2022.0.1</spring-cloud.version>
</properties>
<dependencies>
   <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
   </dependency>
   <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
   </dependency>
   <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
   </dependency>

   <dependency>
      <groupId>com.h2database</groupId>
      <artifactId>h2</artifactId>
      <scope>runtime</scope>
```

```
    </dependency>
    <dependency>
       <groupId>org.projectlombok</groupId>
       <artifactId>lombok</artifactId>
       <optional>true</optional>
    </dependency>
    <dependency>
       <groupId>org.springframework.boot</groupId>
       <artifactId>spring-boot-starter-test</artifactId>
       <scope>test</scope>
    </dependency>



</dependencies>

<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>

<build>
    <plugins>
       <plugin>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-maven-plugin</artifactId>
       </plugin>
    </plugins>
```

```
  </build>

</project>
```

step2.configuration in application.yml

```
server:
 port: 9001

spring:
 application:
   name: DEPARTMENT-SERVICE

eureka:
 client:
   register-with-eureka: true
   fetch-registry: true
   service-url:
     defaultZone: http://localhost:8761/eureka/
 instance:
   hostname: localhost
```

step3.DepartmentController.java

```
package com.sma.department.controller;

import com.sma.department.entity.Department;
import com.sma.department.service.DepartmentService;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/departments")
@Slf4j
public class DepartmentController {
   @Autowired
   private DepartmentService departmentService;

   @PostMapping("/")
   public Department saveDepartment(@RequestBody Department department){
       log.info("Inside saveDepartment method of DepartmentController ");
       return departmentService.saveDepartment(department);
   }

   @GetMapping("/{id}")
   public Department findDepartmentById(@PathVariable("id") Long
departmentId){
       log.info("Inside findDepartmentById method of DepartmentController ");
       return departmentService.findDepartmentById(departmentId);
   }


}
```

step4.Department.java

```
package com.sma.department.entity;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Entity
//The @Entity annotation specifies that the class is an entity
// and is mapped to a database table
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Department {
   @Id
   @GeneratedValue(strategy= GenerationType.AUTO)
   private Long departmentId;
   private String departmentName;
   private String departmentAddress;
   private String departmentCode;
}

}
```

step5.DepartmentRepository.java

```
package com.sma.department.repository;

import com.sma.department.entity.Department;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
//Spring Repository annotation is a specialization of @Component annotation,
//        so Spring Repository classes are autodetected by spring framework
through classpath scanning. Spring Repository is very close to DAO pattern
where DAO classes are responsible for providing CRUD operations on database
tables. However,
//        if you are using Spring Data for managing database operations,
//        then you should use Spring Data Repository interface.
public interface DepartmentRepository extends JpaRepository<Department,Long>
{

   Department findByDepartmentId(Long departmentId);
}
```

step6.DepartmentService.java

```
package com.sma.department.service;


import com.sma.department.entity.Department;
import com.sma.department.repository.DepartmentRepository;
```

```java
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
@Slf4j
//In an application, the business logic resides within the service layer
//so we use the @Service Annotation to indicate that a class belongs to that
layer.
//It is also a specialization of @Component Annotation like the @Repository
Annotation.
public class DepartmentService {
  @Autowired
   private DepartmentRepository departmentRepository;

   public Department saveDepartment(Department department) {
       log.info("Inside saveDepartment method of DepartmentService ");
       return departmentRepository.save(department);
   }

   public Department findDepartmentById(Long departmentId) {
       log.info("Inside findDepartmentById method of DepartmentService ");
       return departmentRepository.findByDepartmentId(departmentId);
   }
}
```

step7.DepartmentServiceApplication.java

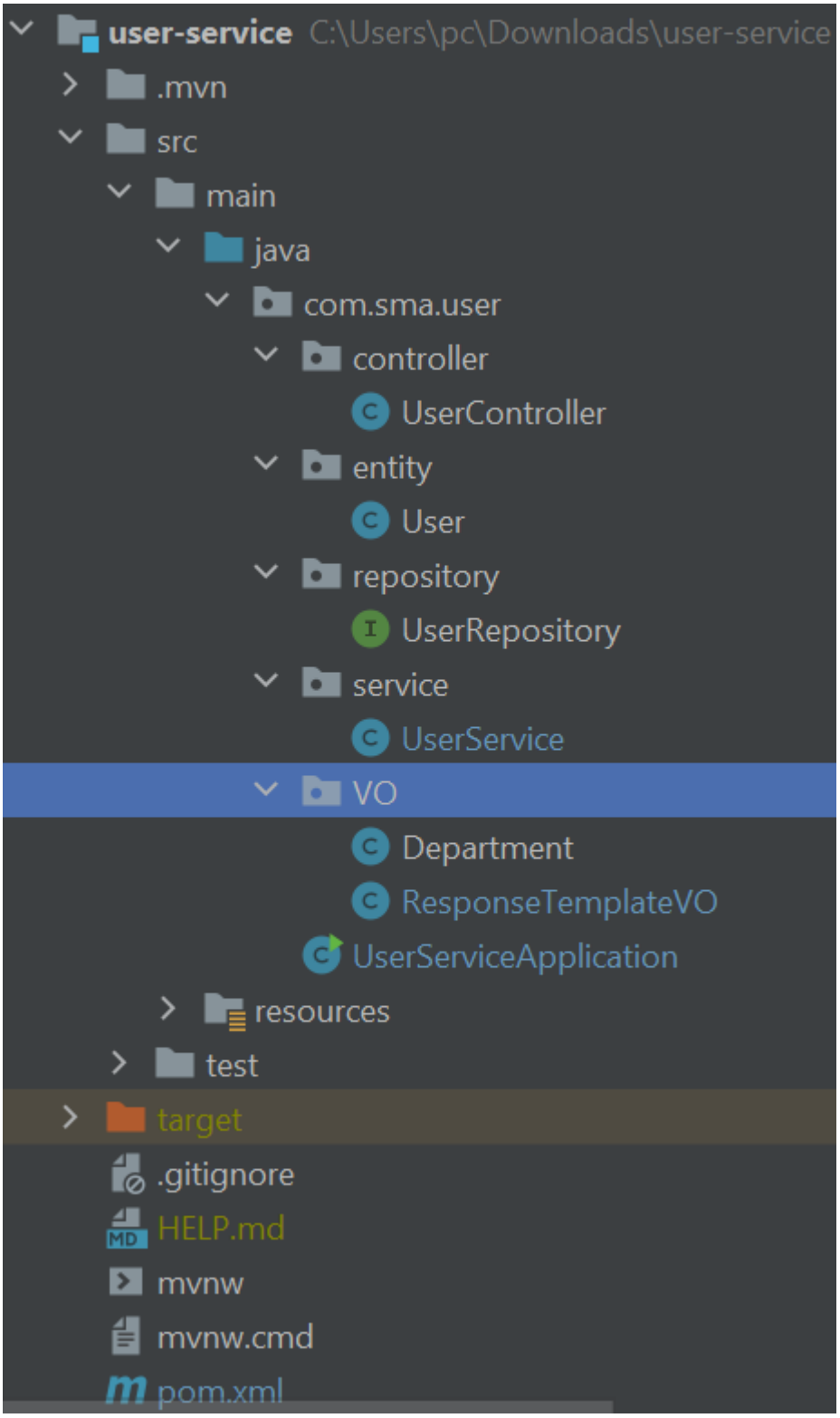
```java
package com.sma.department;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class DepartmentServiceApplication {

  public static void main(String[] args) {
     SpringApplication.run(DepartmentServiceApplication.class, args);
  }

}
```

Screenshot of Department-Service Running Successfully:

```
Registering application DEPARTMENT-SERVICE with eureka with status UP
Saw local status change event StatusChangeEvent [timestamp=1676917461538, current=UP, previous=STARTING]
DiscoveryClient_DEPARTMENT-SERVICE/192.168.1.33:DEPARTMENT-SERVICE:9001: registering service...
Tomcat started on port(s): 9001 (http) with context path ''
Updating port to 9001
DiscoveryClient_DEPARTMENT-SERVICE/192.168.1.33:DEPARTMENT-SERVICE:9001 - registration status: 204
Started DepartmentServiceApplication in 14.541 seconds (process running for 15.877)
```

User microservices:

File Structure:



PORT NO :9002

To create an User microservice in Spring Boot, you can follow these steps:

step1. Add dependency to pom.xml:

```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>

  <parent>

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

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

     <version>3.0.2</version>

     <relativePath/> <!-- lookup parent from repository -->

  </parent>

  <groupId>com.sma</groupId>

  <artifactId>user-service</artifactId>

  <version>0.0.1-SNAPSHOT</version>

  <name>user-service</name>

  <description>user-service</description>

  <properties>

     <java.version>17</java.version>

     <spring-cloud.version>2022.0.1</spring-cloud.version>

  </properties>

  <dependencies>

     <dependency>

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

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

     </dependency>

     <dependency>

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

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

     </dependency>

     <dependency>

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

```
<artifactId>spring-cloud-starter-circuitbreaker-resilience4j</artifactId>
    </dependency>
    <dependency>
       <groupId>org.springframework.cloud</groupId>
       <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
    </dependency>
    <dependency>
       <groupId>org.springframework.boot</groupId>
       <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
       <groupId>org.springframework.boot</groupId>
       <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
       <groupId>com.h2database</groupId>
       <artifactId>h2</artifactId>
       <scope>runtime</scope>
    </dependency>
    <dependency>
       <groupId>org.projectlombok</groupId>
       <artifactId>lombok</artifactId>
       <optional>true</optional>
    </dependency>
    <dependency>
       <groupId>org.springframework.boot</groupId>
       <artifactId>spring-boot-starter-test</artifactId>
       <scope>test</scope>
    </dependency>
  </dependencies>

  <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>

  <build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
  </build>

</project>
```

step2.configuration in application.properties

```
server.port = 9002
spring.datasource.url=jdbc:h2:mem:db;DB_CLOSE_DELAY=-1;MODE=MySQL;NON_KEYWORDS=USER
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.H2Dialect
spring.application.name=USER-SERVICE
eureka.client.service-url.defaultZone=http://localhost:8761/eureka/
eureka.client.fetch-registry=true
eureka.client.register-with-eureka=true
spring.devtools.restart.enabled=true
management.health.circuitBreakers.enabled=true
management.endpoints.web.exposure.include=health
management.endpoint.health.show-details=always
```

```
resilience4j.circuitbreaker.instances.user-service.registerHealthIndicator=tr
ue

resilience4j.circuitbreaker.instances.user-service.eventConsumerBufferSize=10

resilience4j.circuitbreaker.instances.user-service.failureRateThreshold=50

resilience4j.circuitbreaker.instances.user-service.minimumNumberOfCalls=50

resilience4j.circuitbreaker.instances.user-service.automaticTransitionFromOpe
nToHalfOpenEnabled=true

resilience4j.circuitbreaker.instances.user-service.waitDurationInOpenState=5s

resilience4j.circuitbreaker.instances.user-service.permittedNumberOfCallsInHa
lfOpenState=3

resilience4j.circuitbreaker.instances.user-service.slidingWindow=10

resilience4j.circuitbreaker.instances.user-service.slidingWindowType=COUNT_BA
SED
```

step3.UserController.java

```
package com.sma.user.controller;

import com.sma.user.VO.ResponseTemplateVO;
import com.sma.user.entity.User;
import com.sma.user.service.UserService;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/users")
@Slf4j
public class UserController {
    @Autowired
    private UserService userService;

    @PostMapping("/")
    public User saveUser(@RequestBody User user){
        log.info("Inside saveUser method of UserController");
        return userService.saveUser(user);
    }
    @GetMapping("/{id}")
    public ResponseTemplateVO getUserWithDepartment(@PathVariable("id") Long
userId){
        log.info("Inside getUserWithDepartment method of UserController");
        return userService.getUserWithDepartment(userId);
    }

}
```

step4.User.java

```
package com.sma.user.entity;
```

```java
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor

//@AllArgsConstructor generates a constructor requiring argument for every
field in the annotated class
//@NoArgsConstructor generates a constructor with no parameter
public class User {
   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private Long userId;
   private String firstName;
   private String lastName;
   private String email;
   private Long departmentId;
}
```

step5.UserRepository.java

```java
package com.sma.user.repository;

import com.sma.user.entity.User;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface UserRepository extends JpaRepository<User,Long> {

   User findByUserId(Long userId);
}
```

step6.UserService.java

```java
package com.sma.user.service;

import com.sma.user.VO.Department;
import com.sma.user.VO.ResponseTemplateVO;
import com.sma.user.entity.User;
import com.sma.user.repository.UserRepository;
import io.github.resilience4j.circuitbreaker.annotation.CircuitBreaker;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.web.client.RestTemplate;

@Service
@Slf4j
```

```
public class UserService {
    @Autowired
    private UserRepository userRepository;

    @Autowired
    private RestTemplate restTemplate;

    public User saveUser(User user) {
        log.info("Inside saveUser method of UserService");
        return userRepository.save(user);
    }
//     public  ResponseTemplateVO serviceFallback(Exception e){
//         log.info("servicefallbackmethod");
//         ResponseTemplateVO vo=new ResponseTemplateVO();
//         vo.setMessage("THIS IS A FALL BACK PAGE FOR USER-SERVICE");
//         return vo;
//     }
//     @CircuitBreaker(name="user-service",fallbackMethod="serviceFallback")

    public ResponseTemplateVO getUserWithDepartment(Long userId) {
        log.info("Inside getUserWithDepartment method of UserService");
        ResponseTemplateVO vo=new ResponseTemplateVO();
        User user=userRepository.findByUserId(userId);
        Department department=

restTemplate.getForObject("http://DEPARTMENT-SERVICE/departments/"+user.getDe
partmentId()
                        ,Department.class);
        vo.setUser(user);
        vo.setDepartment(department);
        return vo;
    }
}
```

step7.UserServiceApplication.java

```
package com.sma.user;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.loadbalancer.LoadBalanced;
import org.springframework.context.annotation.Bean;
import org.springframework.web.client.RestTemplate;

@SpringBootApplication
public class UserServiceApplication {

  public static void main(String[] args) {
     SpringApplication.run(UserServiceApplication.class, args);
  }
  @Bean
   @LoadBalanced
  public RestTemplate restTemplate(){
     return new RestTemplate();
  }

}
```

step8.Department.java

```
package com.sma.user.VO;


import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;


@Data
@AllArgsConstructor
@NoArgsConstructor
public class Department {
   private Long departmentId;
   private String departmentName;
   private String departmentAddress;
   private String departmentCode;
}
```

step9.ResponseTemplateVO.java

```
package com.sma.user.VO;


import com.sma.user.entity.User;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;


@Data
@AllArgsConstructor
@NoArgsConstructor
public class ResponseTemplateVO {
   private User user;
   private Department department;
```

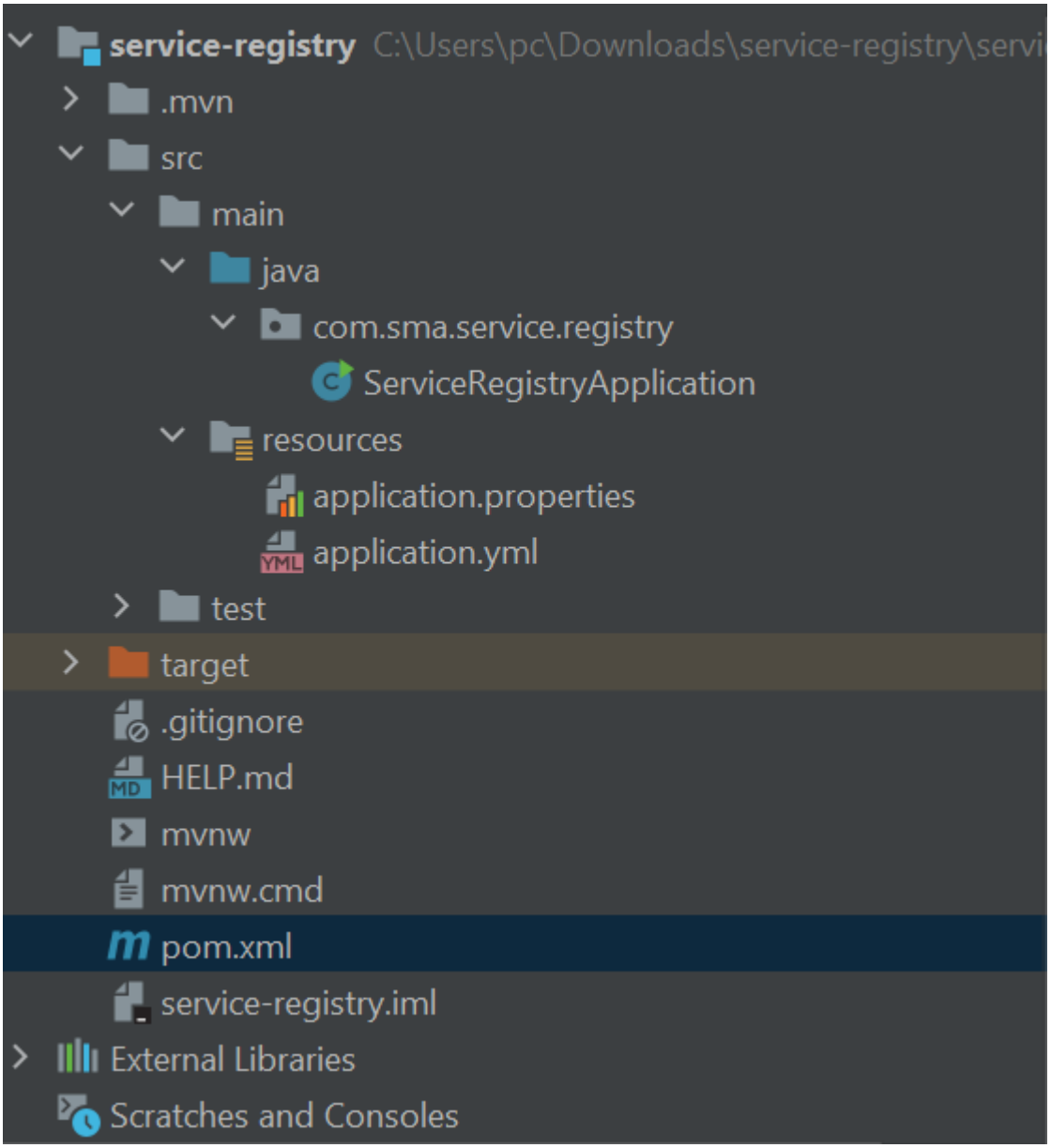
```
}
```

Screenshot of Department-Service Running Successfully:

```
Registering application USER-SERVICE with eureka with status UP
Saw local status change event StatusChangeEvent [timestamp=1676918474161, current=UP, previous=STARTING]
DiscoveryClient_USER-SERVICE/192.168.1.33:USER-SERVICE:9002: registering service...
Tomcat started on port(s): 9002 (http) with context path ''
Updating port to 9002
DiscoveryClient_USER-SERVICE/192.168.1.33:USER-SERVICE:9002 - registration status: 204
Started UserServiceApplication in 19.236 seconds (process running for 20.59)
```

Service Registry:

File Structure:



PORT NO :8761

To create an Service Registry in Spring Boot, you can follow these steps:

step1. Add dependency to pom.xml:

```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
```

```
<modelVersion>4.0.0</modelVersion>
<parent>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-parent</artifactId>
   <version>3.0.2</version>
   <relativePath/> <!-- lookup parent from repository -->
</parent>
<groupId>com.sma</groupId>
<artifactId>service-registry</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>service-registry</name>
<description>service-registry</description>
<properties>
   <java.version>17</java.version>
   <spring-cloud.version>2022.0.1</spring-cloud.version>
</properties>
<dependencies>
   <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
   </dependency>

   <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
   </dependency>
</dependencies>
<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>

  <build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
  </build>
  <repositories>
    <repository>
        <id>netflix-candidates</id>
        <name>Netflix Candidates</name>

<url>https://artifactory-oss.prod.netflix.net/artifactory/maven-oss-candidate
s</url>
        <snapshots>
            <enabled>false</enabled>
        </snapshots>
    </repository>
  </repositories>


</project>
```

step2.configuration in application.yml

```
server:
 port: 8761
```

```yaml
eureka:
 client:
   register-with-eureka: false
   fetch-registry: false
```

step3.ServiceRegistryApplication.java

```java
package com.sma.service.registry;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

@SpringBootApplication
@EnableEurekaServer
public class ServiceRegistryApplication {

  public static void main(String[] args) {
     SpringApplication.run(ServiceRegistryApplication.class, args);
  }

}
```

Screenshot of Service Registry Running Successfully:

```
Changing status to UP
Tomcat started on port(s): 8761 (http) with context path ''
Updating port to 8761
Started Eureka Server
Started ServiceRegistryApplication in 11.627 seconds (process running for 12.735)
Initializing Spring DispatcherServlet 'dispatcherServlet'
Initializing Servlet 'dispatcherServlet'
Completed initialization in 2 ms
```

SCREENSHOT SHOWS HERE THAT ALL CLIENTS ARE REGISTERED IN EUREKA SERVER DASHBOARD:

DS Replicas

localhost

Instances currently registered with Eureka

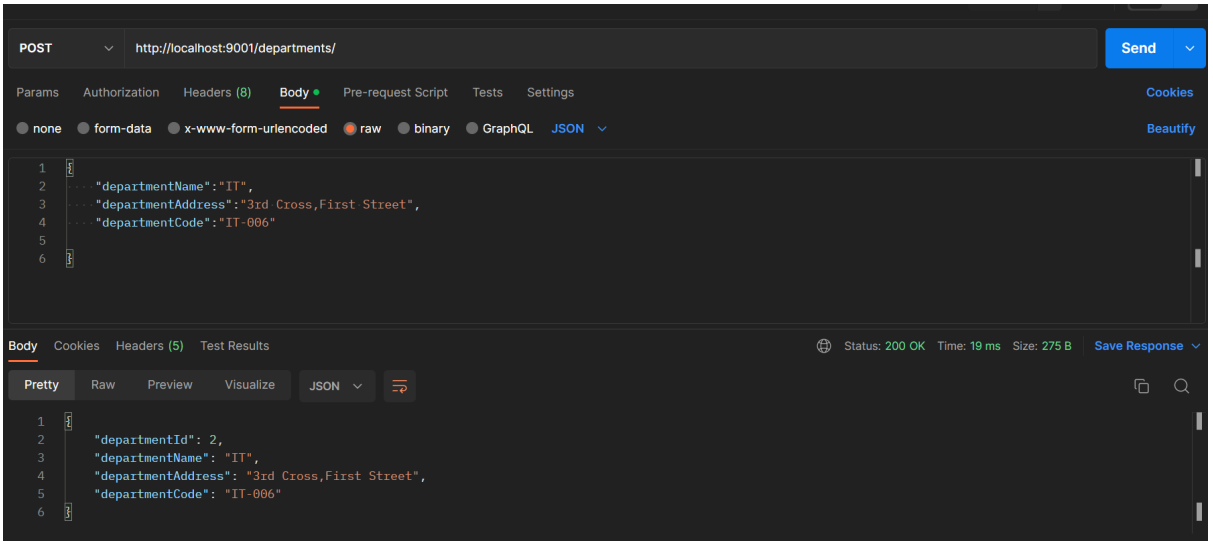
Application	AMIs	Availability Zones	Status
API-GATEWAY	n/a (1)	(1)	UP (1) - 192.168.1.33:API-GATEWAY:9191
DEPARTMENT-SERVICE	n/a (1)	(1)	UP (1) - 192.168.1.33:DEPARTMENT-SERVICE:9001
USER-SERVICE	n/a (1)	(1)	UP (1) - 192.168.1.33:USER-SERVICE:9002

General Info

Name	Value
total-avail-memory	74mb
num-of-cpus	8
current-memory-usage	44mb (59%)

SCREENSHOTS OF API TESTING ON POSTMAN :

1)Save department



Description: API hits @RequestMapping("/departments") and thereafter @PostMapping("/") in DepartmentController

url – http://localhost:9001/departments/

end point - /departments/{id}

id=1,2,3…etc

request – JSON body

```
{
```

```
  "departmentName":"IT",
  "departmentAddress":"3rd Cross,First Street",
  "departmentCode":"IT-006"

}
```

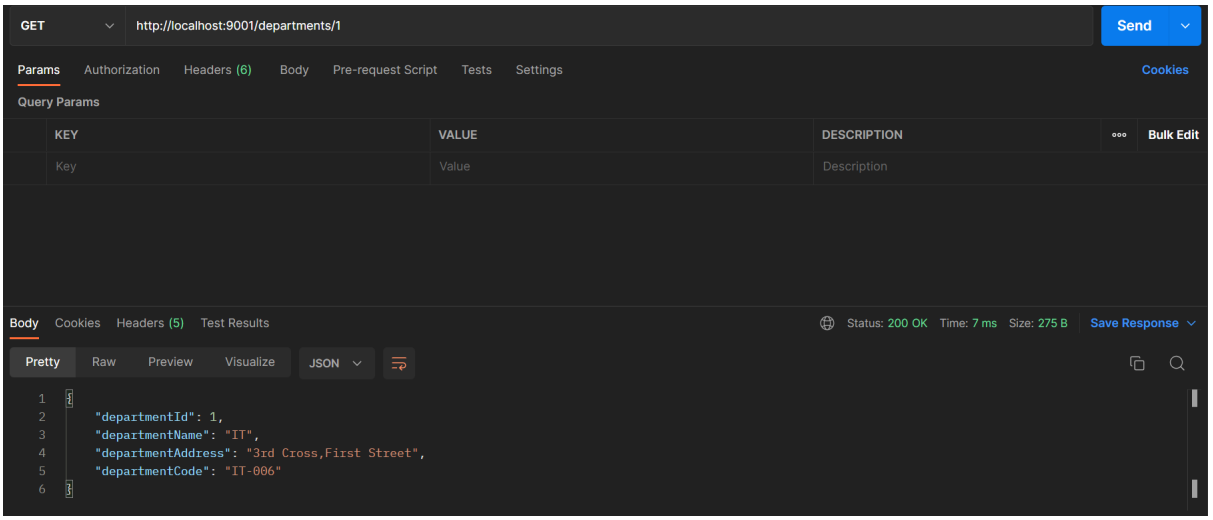
response -JSON body

```
{
  "departmentId": 1,
  "departmentName": "IT",
  "departmentAddress": "3rd Cross,First Street",
  "departmentCode": "IT-006"
}
```

method – POST

2)FindDepartmentById

departmentId is set to be auto-incremented



Description: API hits @RequestMapping("/departments") and thereafter @GetMapping("/") in DepartmentController

url – http://localhost:9001/departments/{id}

end point - /departments/{id}

id=1,2,3…etc

request – {id}

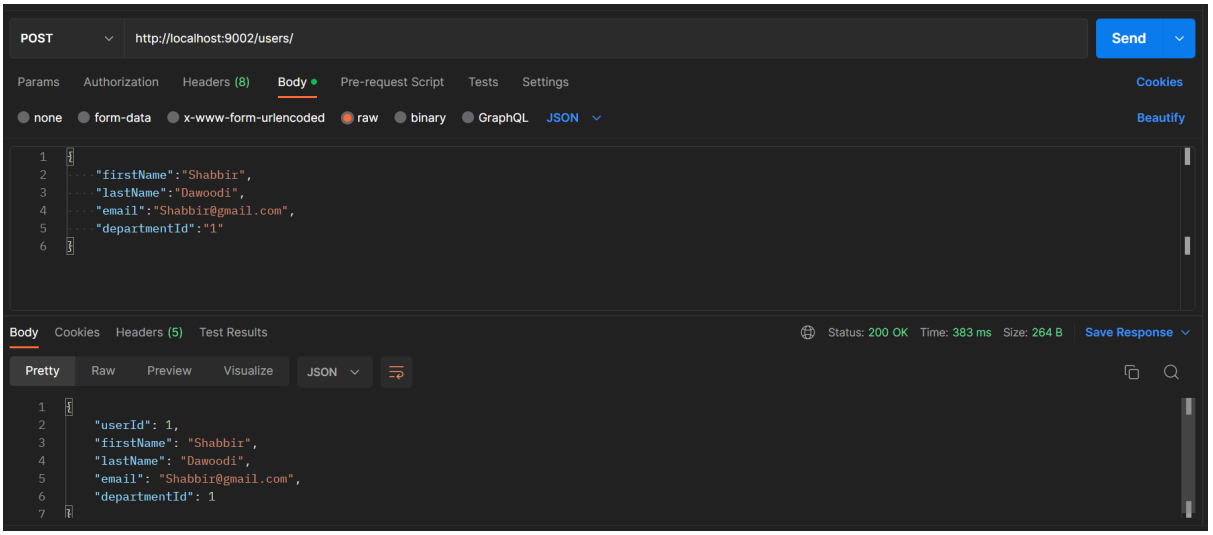
response -JSON body


```
{
  "departmentId": 1,
  "departmentName": "IT",
  "departmentAddress": "3rd Cross,First Street",
  "departmentCode": "IT-006"
}
```


method – GET

3)SaveUser

userId is set to be auto-incremented



Description: API hits @RequestMapping("/users") and thereafter @GetMapping("/") in UserController

url – http://localhost:9002/users/

end point - /users/

response -JSON body


```
{
  "firstName":"Shabbir",
  "lastName":"Dawoodi",
  "email":"Shabbir@gmail.com",
  "departmentId":"1"
}
```

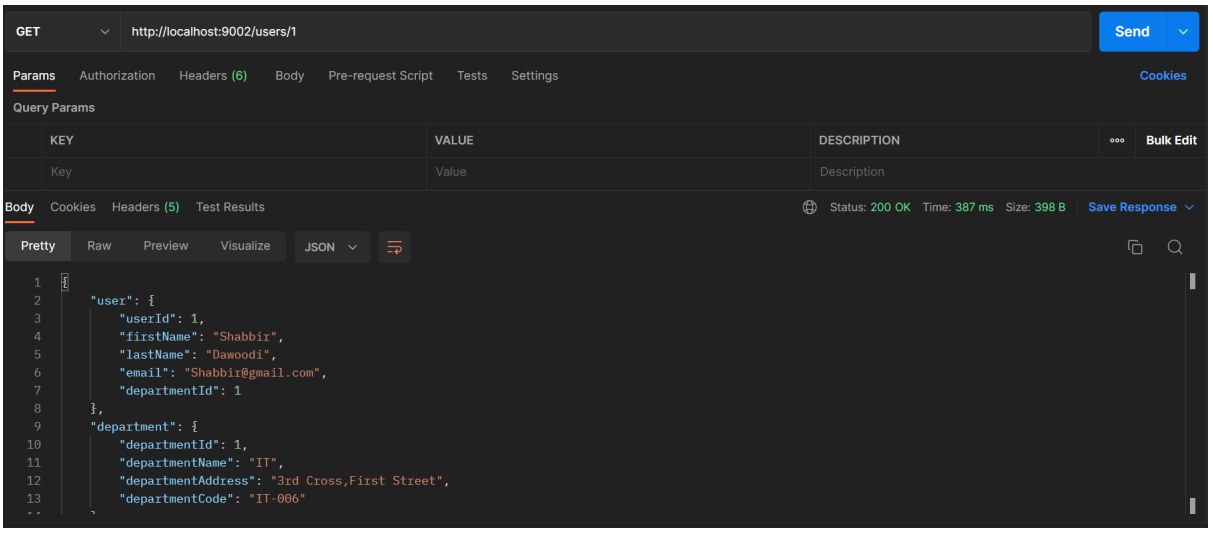

response-JSON body


```
{
  "userId": 1,
  "firstName": "Shabbir",
  "lastName": "Dawoodi",
```

```
  "email": "Shabbir@gmail.com",

  "departmentId": 1

}
```


method – POST

4)GetUserWithDepartment(http://localhost:9002/users/1 **)**



Description: API hits @RequestMapping("/userss") and thereafter @GetMapping("/") in UserController

url – http://localhost:9001/users/{id}

end point - /users/{id}

id=1,2,3…etc

request – {id}

response -JSON body


```
{

  "user": {

    "userId": 2,
```

```
    "firstName": "Shabbir",
    "lastName": "Dawoodi",
    "email": "Shabbir@gmail.com",
    "departmentId": 1
  },
  "department": {
    "departmentId": 1,
    "departmentName": "IT",
    "departmentAddress": "3rd Cross,First Street",
    "departmentCode": "IT-006"
  }
}
```


method – GET

"WHITELABEL ERROR" OCCURS IF WE STOP DEPARTMENT SERVICE AND MAKE GET REQUEST USING (http://localhost:9002/users/1)

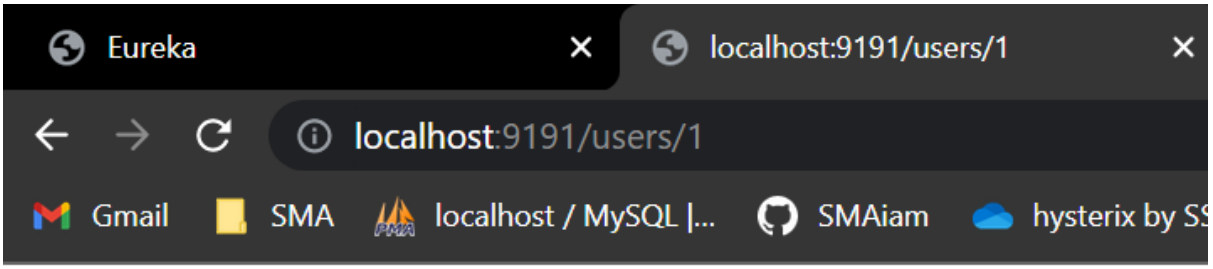
Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

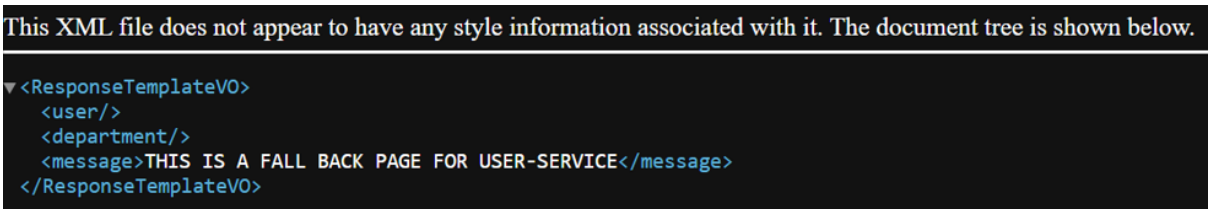
Tue Feb 21 00:59:10 IST 2023
There was an unexpected error (type=Internal Server Error, status=500).

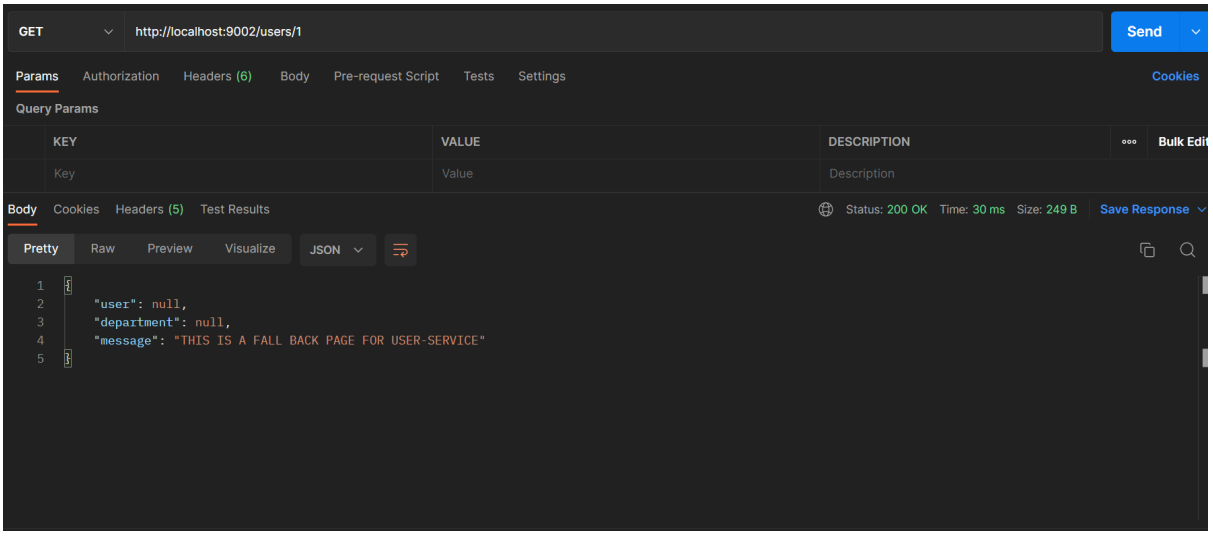
SOLUTION IS BY USING "RESILLIENCE4J" WE CAN SHOW A SIMPLE ERROR MESSAGE TO CLIENT

OUTPUT ON BROWSER:

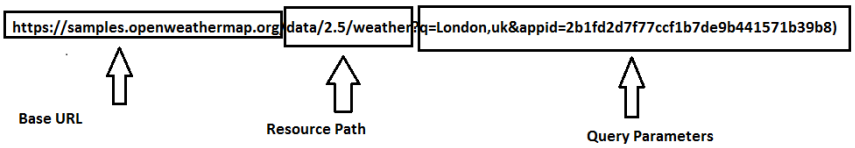


User Service is taking longer than expected. Please Try again later





Endpoint vs URL vs BaseURL:



Endpoint=resourcepath + query parameters

BaseURL=protocol+serveraddress

URL=Endpoint+BaseURL

Query Params vs JSON body:

These two are different way of sending and receiving information through API Calls.

Query Param is visible in URL whereas JSON Body is hidden from client.

Conclusion:

The microservices architecture has become increasingly popular in recent years due to its scalability, modularity, and fault tolerance. In this project, we have demonstrated how microservices can be used to separate the concerns of different functionalities of the application. Each microservice is responsible for a specific functionality, allowing developers to work independently on their respective services. This approach also makes it easier to add new functionalities to the application without affecting the other services.

The API gateway acts as a single entry point for all the microservices, providing a unified interface to the clients. It also provides an additional layer of security and traffic management. The Eureka server, on the other hand, acts as a service registry, allowing the services to register themselves and discover other services in the network. This helps in the dynamic scaling of the services and provides fault tolerance.

The project also demonstrated the importance of good software design and architecture. The use of design patterns, such as the repository pattern and the facade pattern, helped in creating modular and maintainable code. The use of Spring Boot and Spring Cloud also helped in reducing boilerplate code and providing a rich set of tools for building microservices.

Throughout the project, the project trainer, Siddharth Sharma, provided valuable guidance and support. His expertise in microservices architecture and Spring Boot helped in making this project a success. The project also provided an opportunity to learn new skills and technologies.

In conclusion, this project has successfully implemented a distributed architecture using microservices, API gateway, and Eureka server. It has demonstrated the benefits of a modular and scalable architecture, and the importance of good software design and architecture. The project has provided valuable learning opportunities and has equipped us with the necessary skills and knowledge to build robust and scalable applications using microservices.