**Microservices in Java**

*Simple example for creating microservice architecture using Spring Boot, JPA.*

Service Registry



Department

/users

API

Gateway/

Hystrix

Config

Server

User

User

/departments

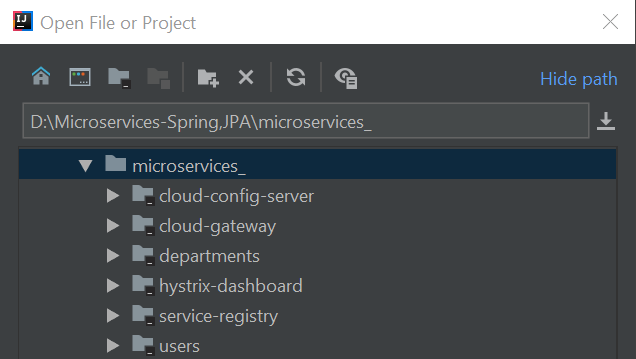
For this purpose, we need:

**IDE** - IntelliJ IDEA (or Eclipse)

**Spring initializr** - to generate Spring Boot project with just what we need to start quickly - <https://start.spring.io/>

**Postman** – for testing our work - <https://www.postman.com/downloads/>

The picture below shows 6 Spring Boot projects we should create.



1. **Create departments and users microservices -**Spring boot restful web services implementing the JPA for the interaction with the databases(H2)

In Spring Initialz we select needed settings and dependencies and create the project:

Maven Project

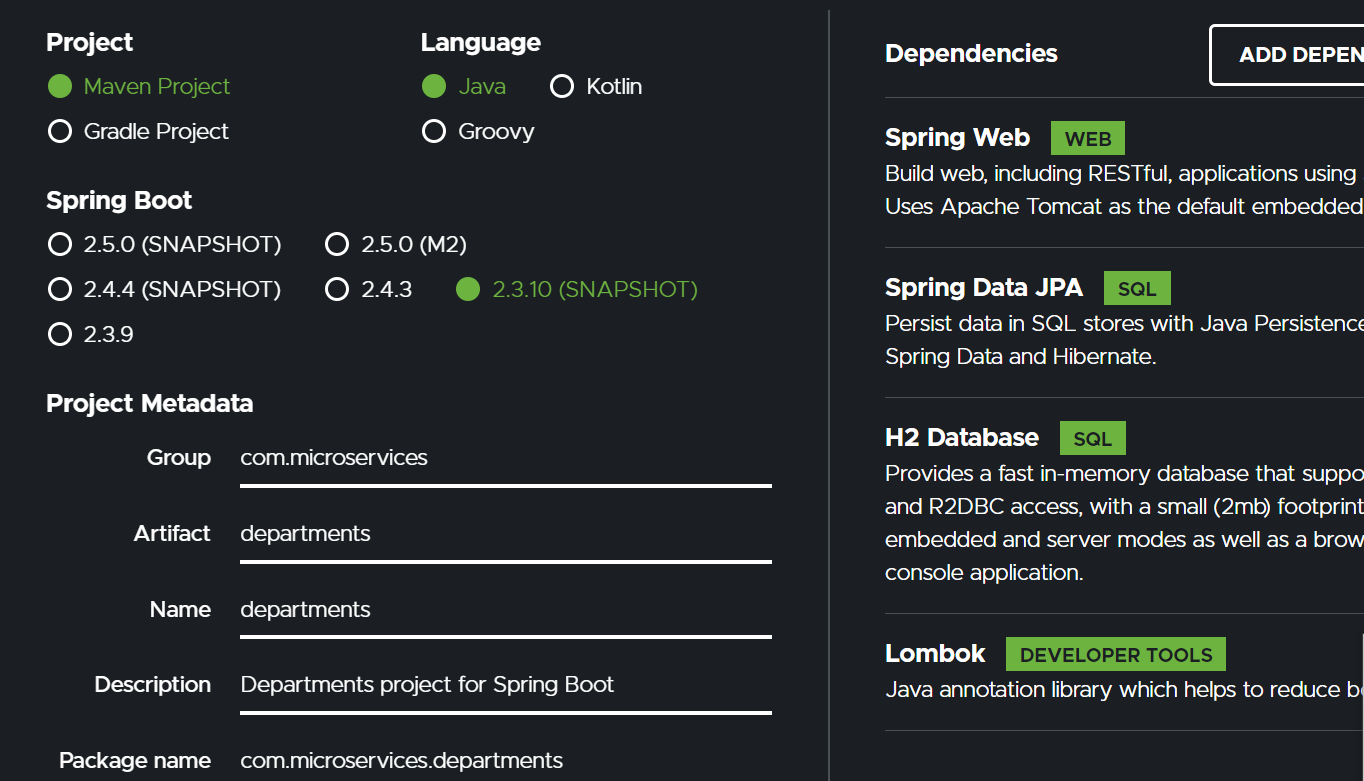
Java – 8 , package- jar

Spring Boot 2.3.10

Dependancies:

Web, JPA, H2, Lombok

1.1 In Spring Initializr we create departments application:



Project structure of departments application:

We create the following packages under src/main/java/com/microservices/departments/:

controllers

entities

repositories

services

Under package entities we create our @Entity class Department:

package com.microservices.departments.entities;  
  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
import javax.persistence.Entity;  
import javax.persistence.GeneratedValue;  
import javax.persistence.GenerationType;  
import javax.persistence.Id;  
  
@Entity  
@NoArgsConstructor  
@AllArgsConstructor  
@Data  
public class Department {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*AUTO*)  
 private Long departmentId;  
 private String name;  
 private String address;  
 private String code;  
}

Under package repositories we create our @Repository interface DepartmentRepository:

package com.microservices.departments.repositories;  
  
import com.microservices.departments.entities.Department;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
  
@Repository  
public interface DepartmentRepository extends JpaRepository<Department, Long> {  
}

Under package services we create our @Service class DepartmentService:

package com.microservices.departments.services;  
  
import com.microservices.departments.entities.Department;  
import com.microservices.departments.repositories.DepartmentRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
  
import java.util.List;

@Service

public class DepartmentService {  
  
 @Autowired  
 DepartmentRepository departmentRepository;  
  
 public Department addDepartment(Department department) {  
  
 return departmentRepository.save(department);  
 }  
  
 public Department getDepartmentById(Long departmentId) {  
  
 return departmentRepository.findByDepartmentId(departmentId);  
 }  
  
 public List<Department> getAllDepartments() {  
  
 return departmentRepository.findAll();  
 }  
}

and add method findByDepartmentId(long departmentId) to DepartmentRepository:

@Repository  
public interface DepartmentRepository extends JpaRepository<Department, Long> {  
 Department findByDepartmentId(Long departmentId);  
}

Under package controllers we create our @RestController class DepartmentController:

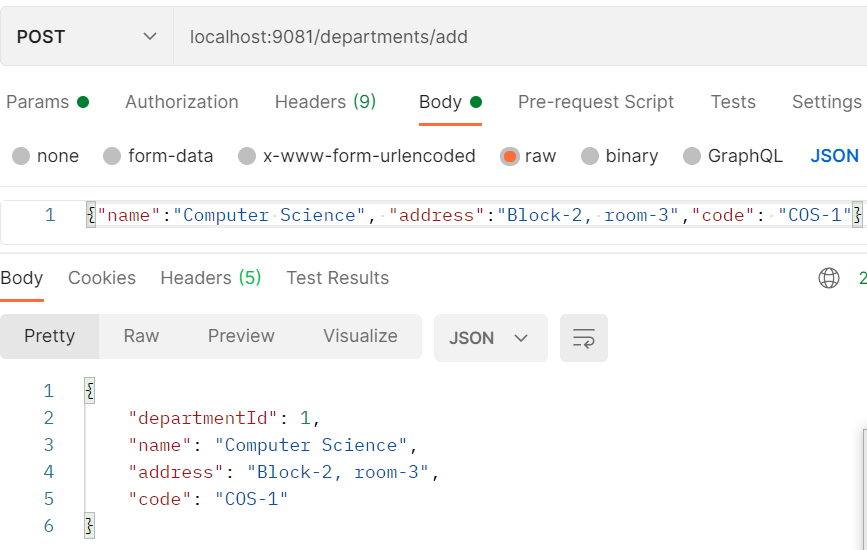
package com.microservices.departments.controllers;  
  
import com.microservices.departments.entities.Department;  
import com.microservices.departments.services.DepartmentService;  
import lombok.extern.java.Log;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
  
@RestController  
@Log  
@RequestMapping("/departments")  
public class DepartmentController {  
  
 @Autowired  
 DepartmentService departmentService;  
  
 @PostMapping("/add")  
 public Department addDepartment(@RequestBody Department department){  
 *log*.info("addDepartment...");  
 return departmentService.addDepartment(department);  
 }  
  
 @GetMapping("/{id}")  
 public Department getDepartmentById(@PathVariable("id") Long id){  
 //log.info("getDepartment by id=" + departmentId);  
 return departmentService.getDepartmentById(id);  
 }  
  
 @GetMapping("/all")  
 public List<Department> getAllDepartments(){  
 *log*.info("getAllDepartments...");  
 return departmentService.getAllDepartments();  
 }  
}

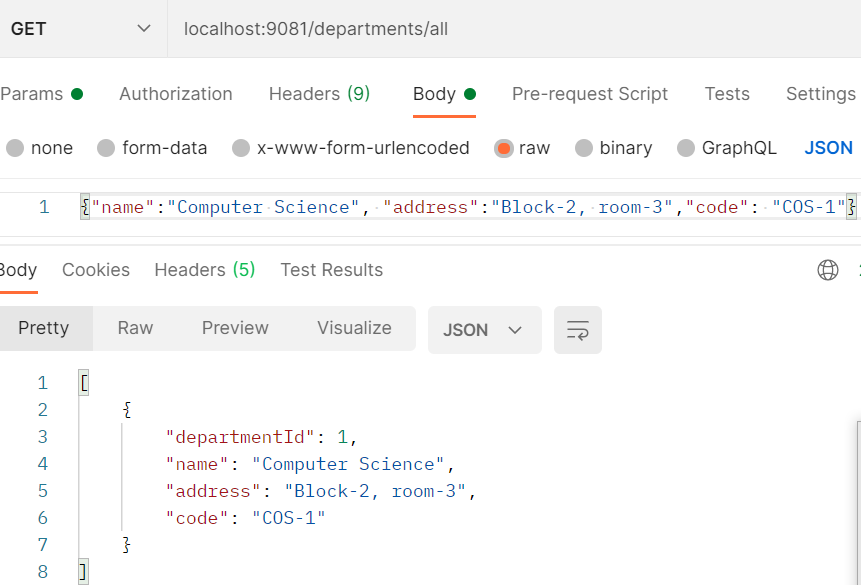
Under folder resources create application.yml file with content:

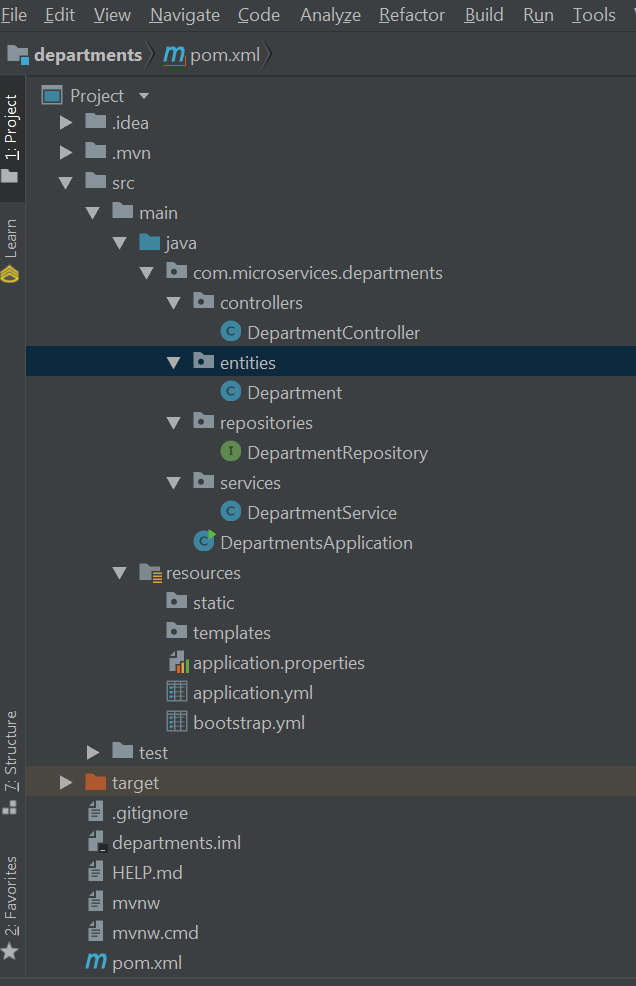
**server**:  
 **port**: 8081  
**spring**:  
 **application**:  
 **name**: DEPARTMENT-SERVICE

Run DepartmentsApplication class.

Test our service in Postman:

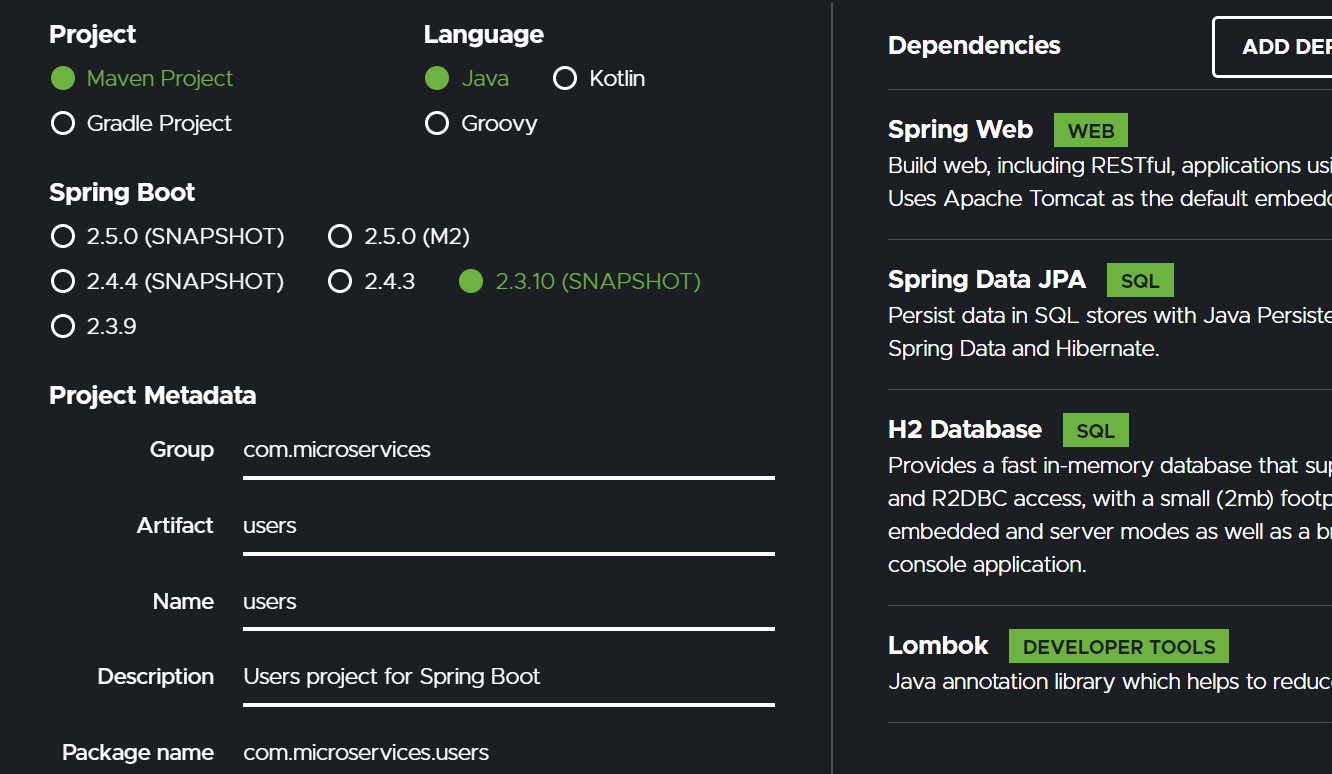






* 1. Create User application:

Create users application from Spring initializr:



Open project in IntellyJ and create packages under src/main/java/com/microservices/users/:

controllers

entities

repositories

services

vo (value object package)

Create entity class User in package entities:

package com.microservices.users.entities;  
  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
import javax.persistence.Entity;  
import javax.persistence.GeneratedValue;  
import javax.persistence.GenerationType;  
import javax.persistence.Id;  
  
@Entity  
@NoArgsConstructor  
@AllArgsConstructor  
@Data  
public class User {  
   
 @Id  
 @GeneratedValue(strategy = GenerationType.*AUTO*)  
 private Long id;  
 private String name;  
 private String email;  
 private Long departmentId;  
  
 @Override  
 public String toString() {  
 return "User{" +  
 "id=" + id +  
 ", name='" + name + '\'' +  
 ", email='" + email + '\'' +  
 ", departmentId=" + departmentId +  
 '}';  
 }  
}

Create @Repository interface in repositories package:

package com.microservices.users.repositories;  
  
import com.microservices.users.entities.User;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
  
@Repository  
public interface UserRepository extends JpaRepository<User, Long> {  
}

Create in package vo two classes Departments:

package com.microservices.users.vo;  
  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
@Data  
@AllArgsConstructor  
@NoArgsConstructor  
public class Department {  
 private Long departmentId;  
 private String name;  
 private String address;  
 private String code;  
}

and wrapper class ResponseTemplateVO:

package com.microservices.users.vo;  
  
import com.microservices.users.entities.User;  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
//wrapper class to facilitate our task  
@Data  
@AllArgsConstructor  
@NoArgsConstructor  
public class ResponseTemplateVO {  
 private Department department;  
 private User user;  
}

We use WebClient( for asynchronous communication) to call remote REST services (http:9081/departments/{id}).

If we what synchronouse communication, we should use RestTemplate.

Add @Bean WebClient to UserApplication class for connecting to department application:

package com.microservices.users;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.context.annotation.Bean;  
import org.springframework.web.reactive.function.client.WebClient;  
  
@SpringBootApplication  
public class UsersApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(UsersApplication.class, args);  
 }  
  
 @Bean  
 /\* For synchronous communication  
 public RestTemplate restTemplate(){  
 return new RestTemplate();  
 }\*/  
 // for asynchronous communication  
 public WebClient webClient(){  
 return WebClient.*create*();  
 }  
}

Create @Service class UserService:

package com.microservices.users.services;  
  
import org.springframework.stereotype.Service;  
  
@Service  
public class UserService {

}

Create @Controller class UserController in package controllers and add methods to UserService class:

package com.microservices.users.controllers;  
  
import com.microservices.users.entities.User;  
import com.microservices.users.services.UserService;  
import com.microservices.users.vo.ResponseTemplateVO;  
import lombok.extern.java.Log;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
  
@RestController  
@RequestMapping("/users")  
@Log  
public class UserController {  
  
 @Autowired  
 UserService userService;  
  
 @PostMapping("/add")  
 public User addUser(@RequestBody User user){  
 *log*.info("Add user....");  
 return userService.addUser(user);  
 }  
  
 @GetMapping("/all")  
 public List<User> getAllUsers(){  
 *log*.info("Get all users....");  
 return userService.getAllUsers();  
 }  
  
 @GetMapping("/{id}")  
 public ResponseTemplateVO getUserWithDepartment(@PathVariable("id") Long id) {  
 *log*.info("getUserWithDepartment");  
 return userService.getUserWithDepartment(id);  
 }  
}

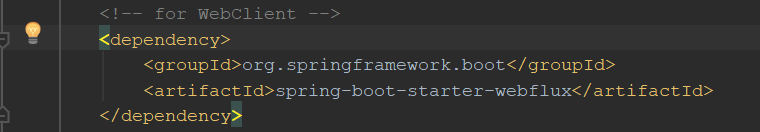
UserService looks like:

package com.microservices.users.services;  
  
import com.microservices.users.entities.User;  
import com.microservices.users.repositories.UserRepository;  
import com.microservices.users.vo.Department;  
import com.microservices.users.vo.ResponseTemplateVO;  
import lombok.extern.java.Log;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.MediaType;  
import org.springframework.stereotype.Service;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.PathVariable;  
import org.springframework.web.reactive.function.client.WebClient;  
import reactor.core.publisher.Mono;  
  
import java.util.List;  
  
@Service  
@Log  
public class UserService {  
  
 @Autowired  
 private UserRepository userRepository;  
  
 @Autowired  
 /\*  
 @LoadBalanced  
 private RestTemplate restTemplate; \*/  
 private WebClient webClient;  
  
 public User addUser(User user) {  
 return userRepository.save(user);  
 }  
  
 public List<User> getAllUsers() {  
 return userRepository.findAll();  
 }  
  
 public ResponseTemplateVO getUserWithDepartment(@PathVariable("id") Long id) {  
 User user = userRepository.getUserById(id);  
 *log*.info("User=" + user.toString());  
 ResponseTemplateVO responseTemplateVO = new ResponseTemplateVO();  
 //Department department = restTemplate.getForObject("http://DEPARTMENT-SERVICE/departments/" + user.getDepartmentId(), Department.class );  
 Mono<Department> department = webClient.get().uri("http://localhost:9081/departments/" + user.getDepartmentId()).accept(MediaType.*ALL*).retrieve().bodyToMono(Department.class );  
 *log*.info("Department from mono="+department.block().toString());  
 responseTemplateVO.setUser(user);  
 responseTemplateVO.setDepartment(department.block());  
 return responseTemplateVO;  
 }  
  
}

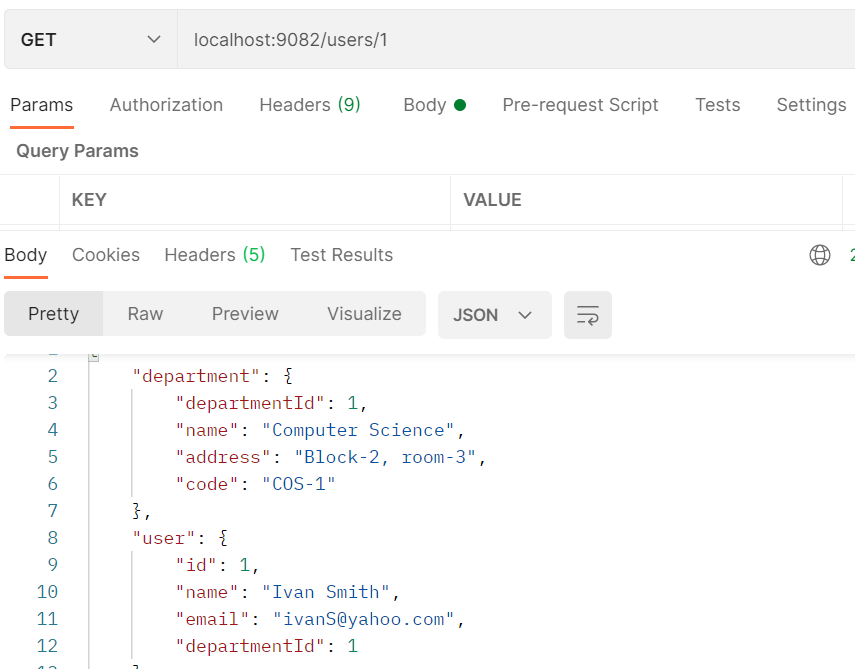
Add method to UserRepository interface:

package com.microservices.users.repositories;  
  
import com.microservices.users.entities.User;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
  
@Repository  
public interface UserRepository extends JpaRepository<User, Long> {  
 User getUserById(Long id);  
}

Add to pom.xml if we want to use WebClient:



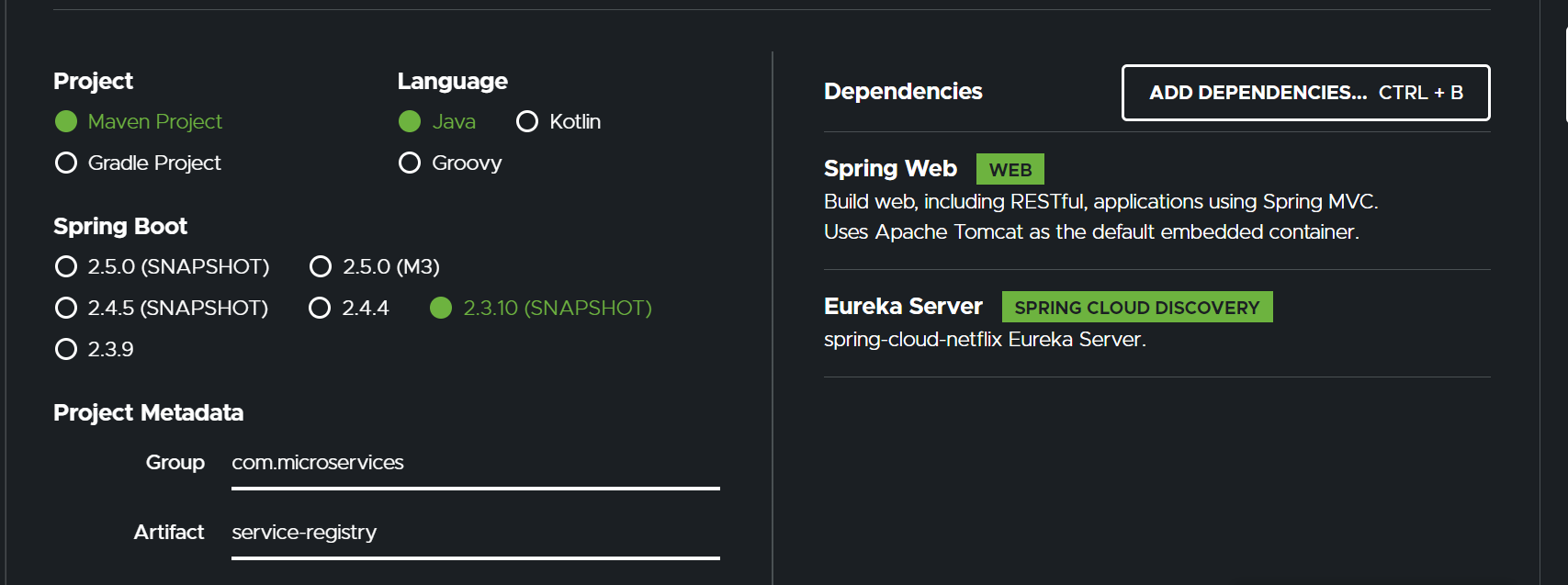
Run UserApplication and test it with postman:



2.Create Eureka service registry:

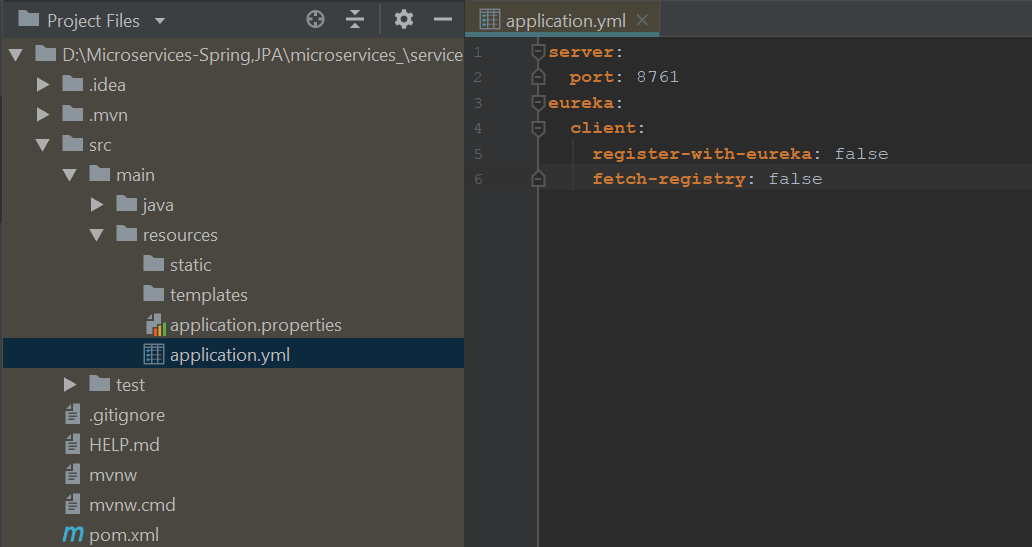
Eureka server (known as Discovery Server) is an application that holds information about all client-server applications. Each one of these applications should register into the Eureka server which will know the ports on which they run and their IP address.

2.1 Create Spring Boot project from Spring Initializr:



package com.microservices.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);  
 }  
  
}

2.2 Create into package resources file application. yml



2.3 Add this lines to pom.xml of out server-client applications users and dependancies:

<properties>  
 <java.version>1.8</java.version>  
 <spring-cloud.version>Hoxton.BUILD-SNAPSHOT</spring-cloud.version>  
</properties>  
<dependencies>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>  
 </dependency>

And between </dependancies> and <build> tags:

</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>

Add @EnableEurekaClient to DepartmentsApplication class and UserApplication class:

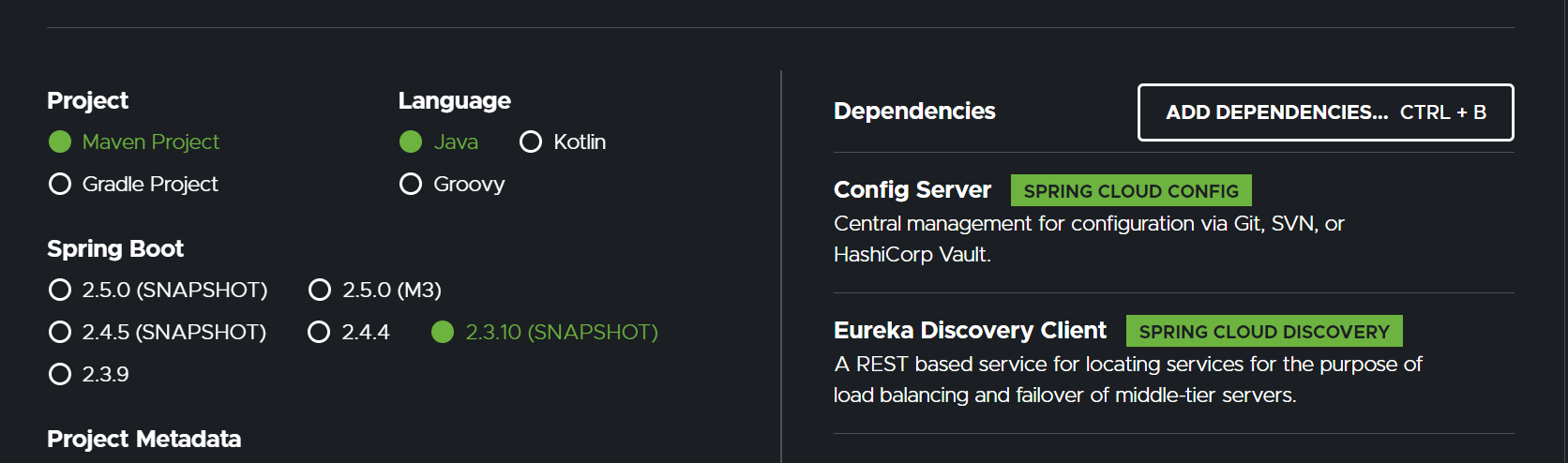
package com.microservices.departments;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;  
  
@SpringBootApplication  
@EnableEurekaClient  
public class DepartmentsApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(DepartmentsApplication.class, args);  
 }  
  
}

Run Discovery server on: localhost: 8761

3.Create Cloud Config server:

Spring Cloud Config provides server-side and client-side support for externalized configuration in a distributed system. With the Cloud Config Server, we have a central place to manage external properties for applications across all environments.

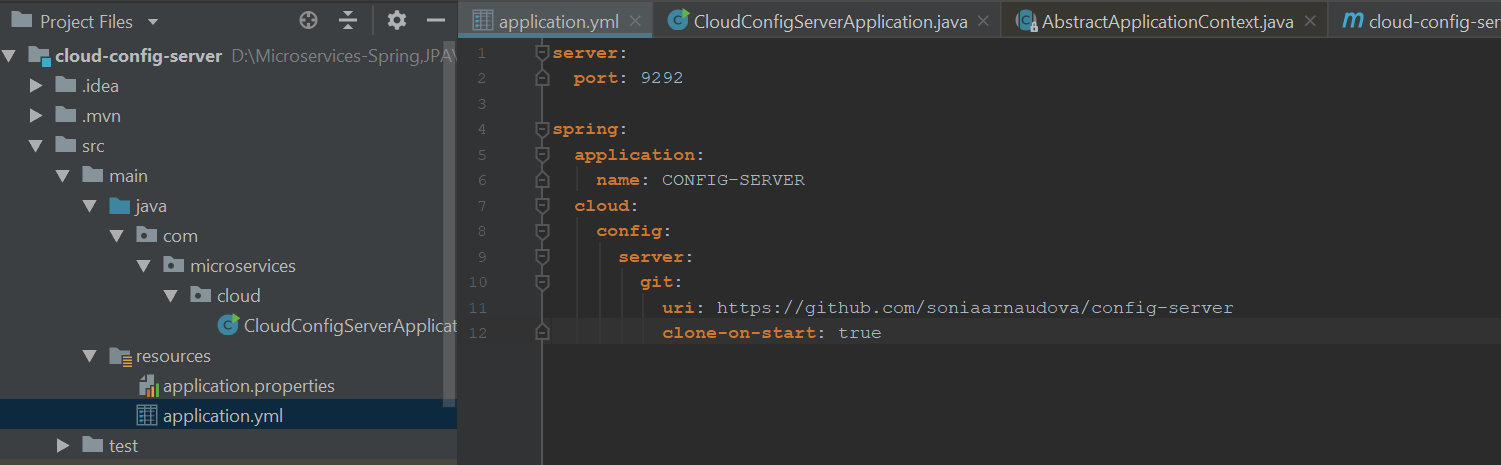
3.1 Create cloud-config server through Spring Boot Initializr:



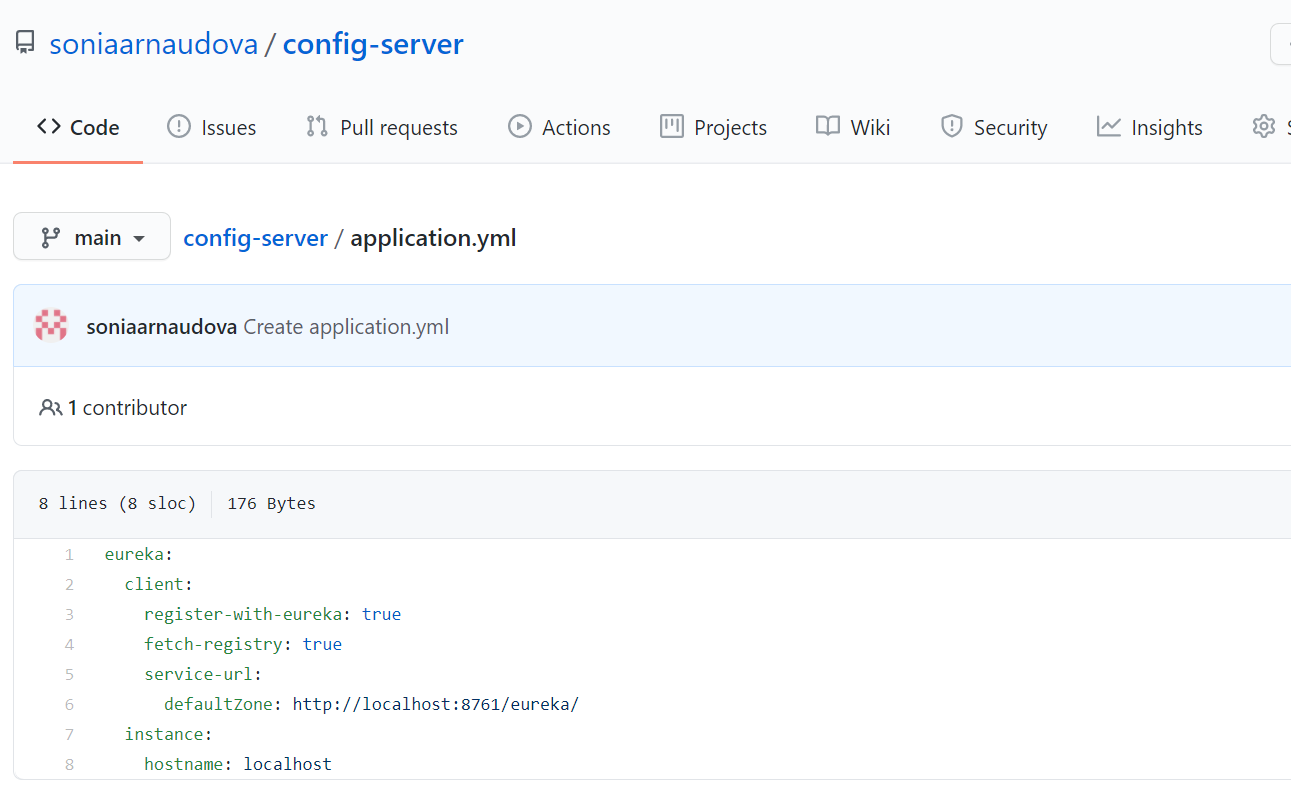
Add @EnableEurekaClient to CloudConfigServerAppllication class:

package com.microservices.cloud;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;  
  
@SpringBootApplication  
@EnableEurekaClient  
public class CloudConfigServerApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(CloudConfigServerApplication.class, args);  
 }  
  
}

* 1. Create application.yml file:



* 1. Go to github and create repository config-server and application.yml file:



And add to application.yml this code:

**server**:  
 **port**: 9292  
  
**spring**:  
 **application**:  
 **name**: CONFIG-SERVER  
 **cloud**:  
 **config**:  
 **server**:  
 **git**:  
 **uri**: https://github.com/soniaarnaudova/config-server  
 **clone-on-start**: true

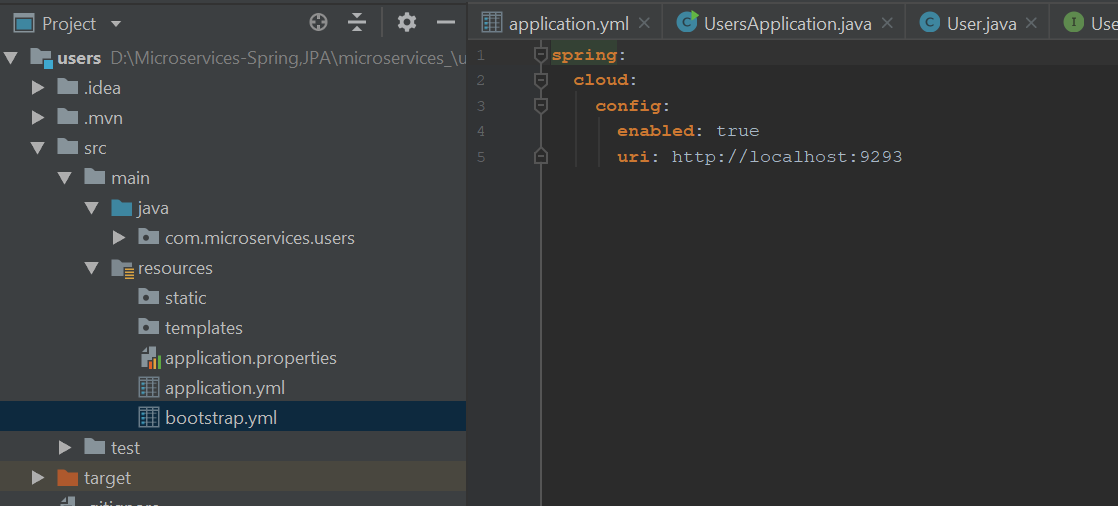
* 1. Go to pom.xml of users and departments applications and add:

1. <dependency>  
    <groupId>org.springframework.cloud</groupId>  
    <artifactId>spring-cloud-starter-config</artifactId>  
   </dependency>

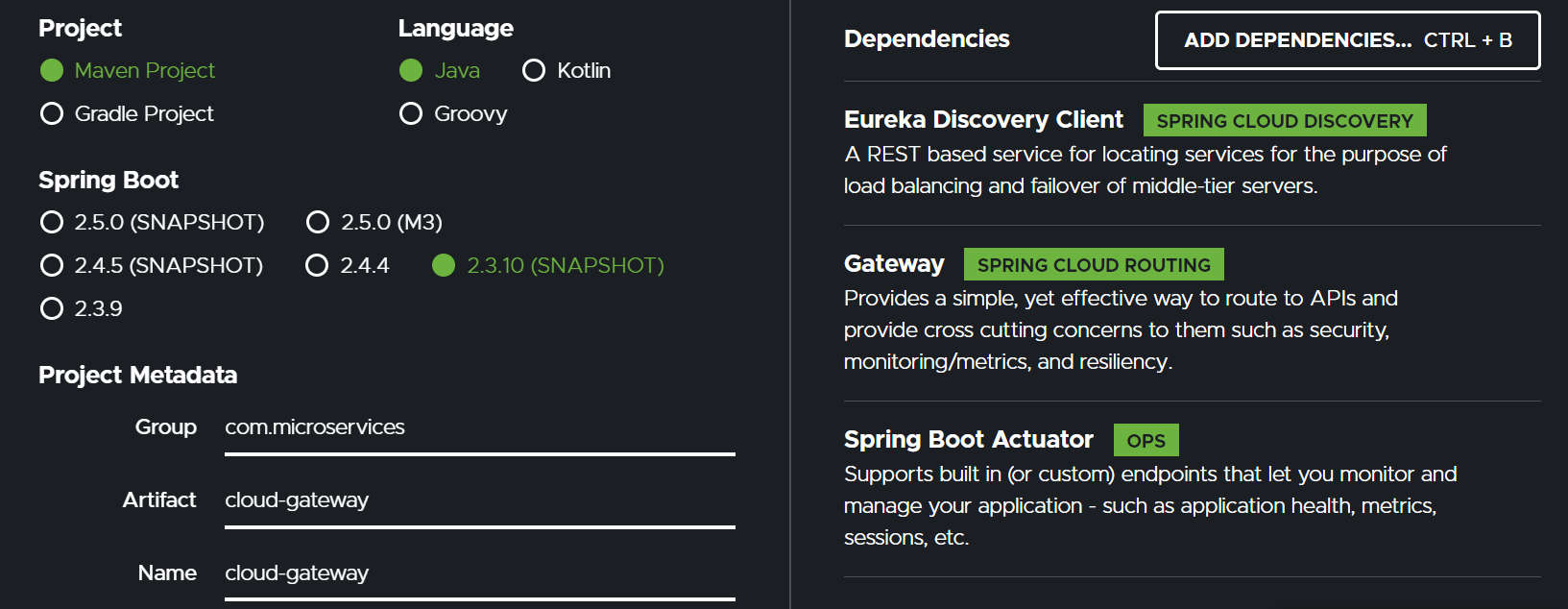
And

<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>

* 1. Create bootstrap.yml file into package resources in users and departments applications:



4.Gateway service – provides a simple way to route to APIs and cross cutting concerns to them such as: security, monitoring/metrics, and resiliency.



Add @EnableEurekaClient to application class:

package com.microservices.cloud.gateway;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;  
  
@SpringBootApplication  
@EnableEurekaClient  
public class CloudGatewayApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(CloudGatewayApplication.class, args);  
 }  
  
}

Create into package resources file application.yml:

**server**:  
 **port**: 9191  
  
**spring**:  
 **application**:  
 **name**: GATEWAY-API  
 **cloud**:  
 **gateway**:  
 **routes**:  
 - **id**: USER-SERVICE  
 **uri**: lb:/USER-SERVICE  
 **predicates**:  
 - Path=/users/\*\*  
 - **id**: DEPARTMENT-SERVICE  
 **uri**: lb:/DEPARTMENT-SERVICE  
 **predicates**:  
 - Path=/departments/\*\*

Add dependency to pom.xml:

<dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-config</artifactId>  
</dependency>

And

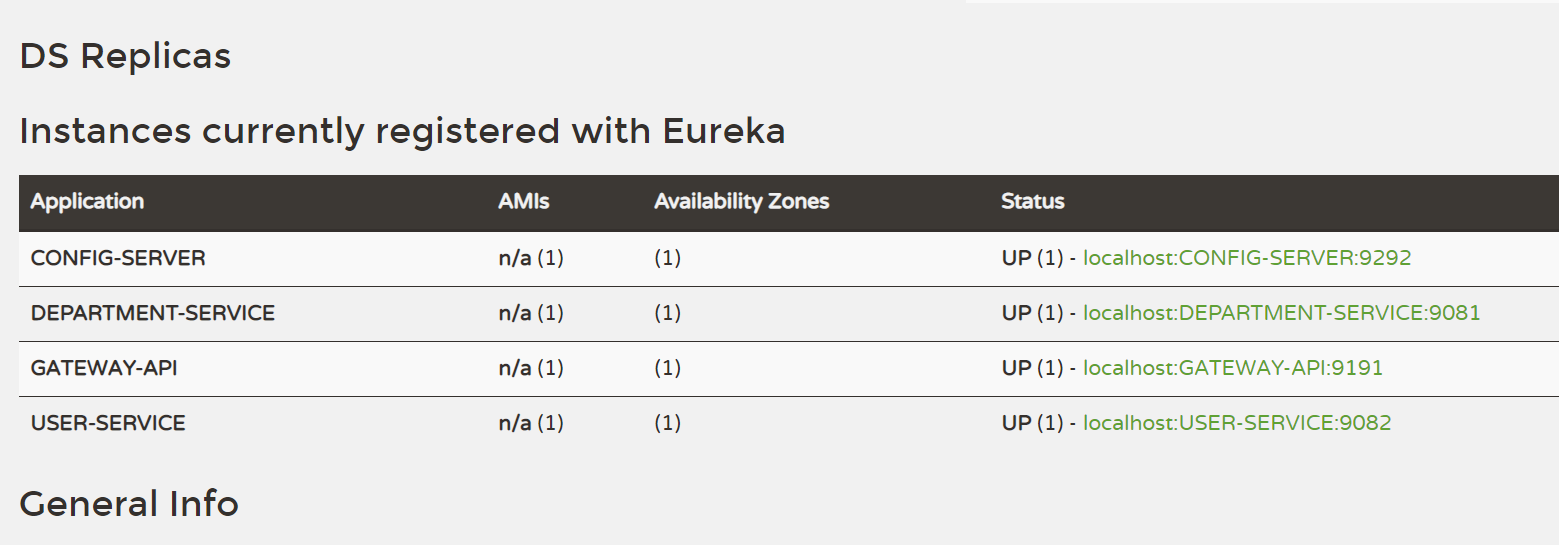
<dependencyManagement>  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-dependencies</artifactId>  
 <version>${spring-cloud.version}</version>  
 <type>pom</type>  
 <scope>import</scope>  
 </dependency>  
 </dependencies>  
</dependencyManagement>

Add to package resources file bootstrap.yml:

**spring**:  
 **cloud**:  
 **config**:  
 **enabled**: true  
 **uri**: http://localhost:9292

First we run service-registry application, second: cloud-config application, and then: gateway, users and departments apps.

Open browser and type: localhost:8761

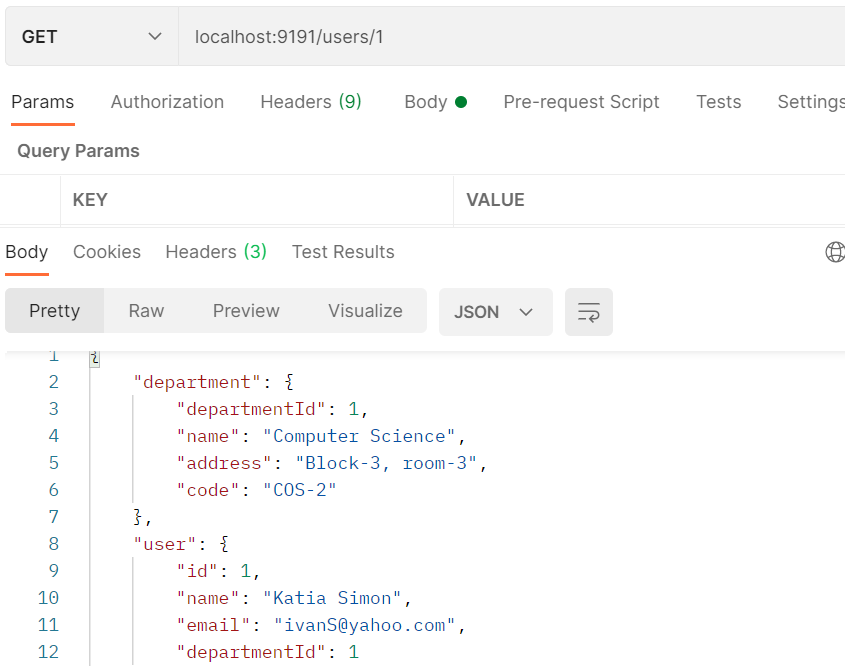


In Postman:

All requests will be done through gateway: <http://localhost:9191/departments/add>

<http://localhost:9191/users/add>

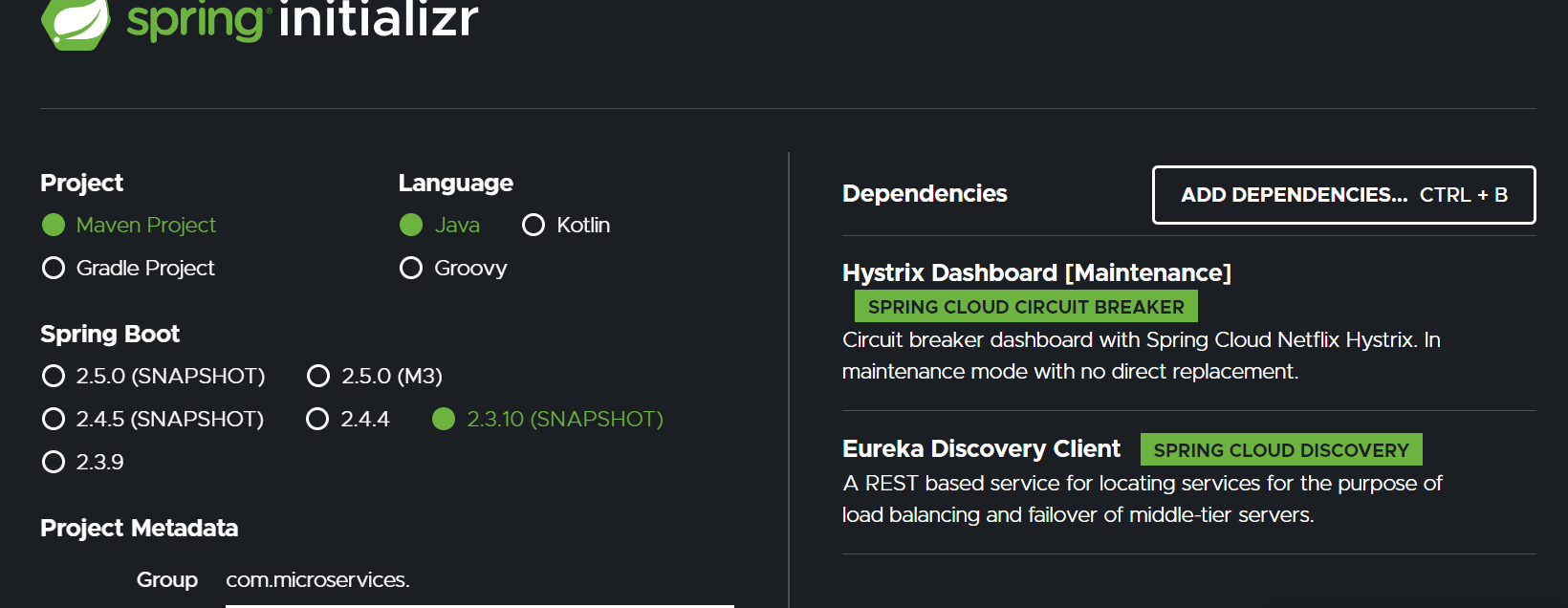
<http://localhost:9191/users/1> in postman application:



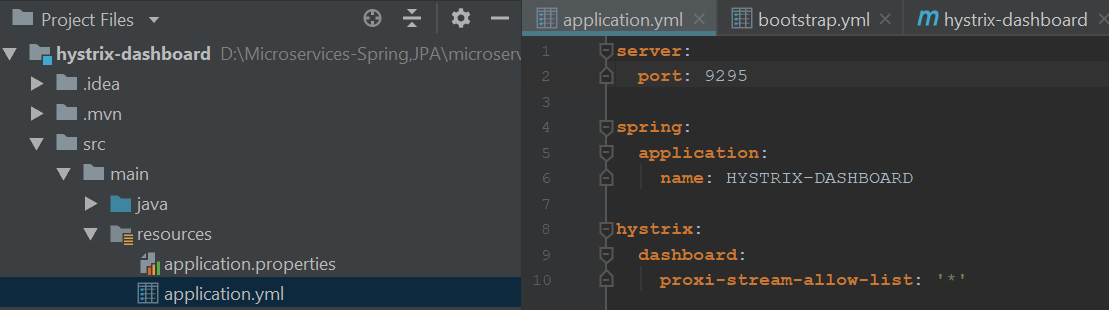
5.Create Hystrix service and dashboard:

Hystrix is a library from Netflix. Hystrix isolates the points of access between the services, stops cascading failures across them and provides the fallback options. (<https://www.tutorialspoint.com/spring_boot/spring_boot_hystrix.htm> )

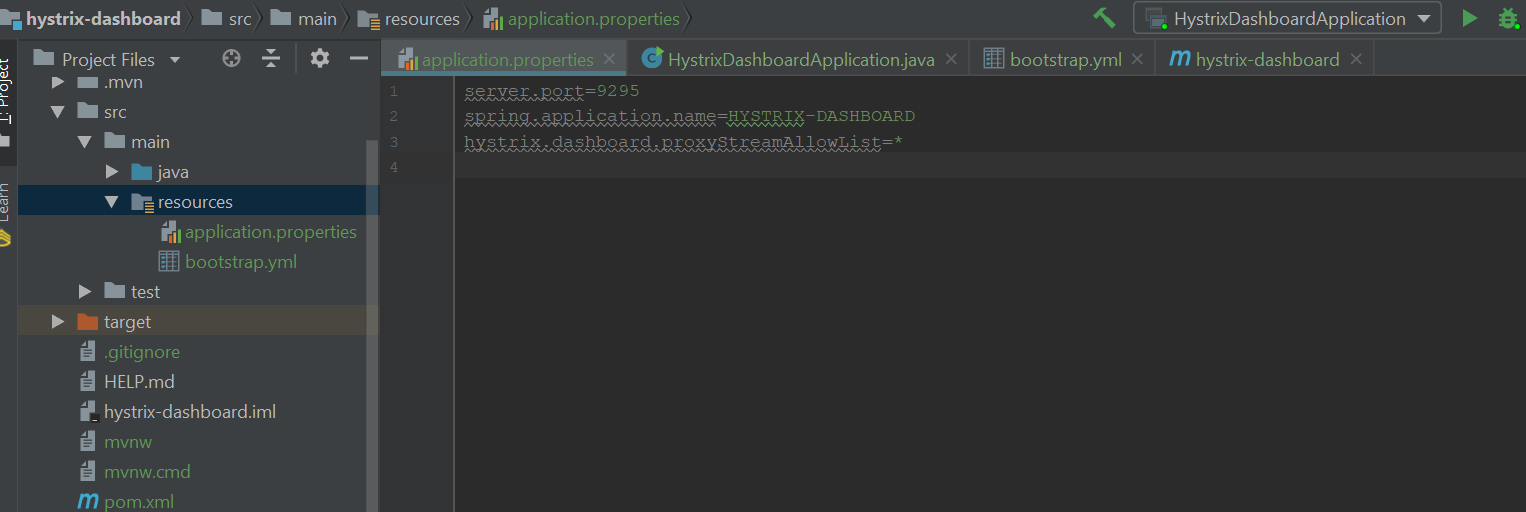
In Spring Initialz create hystrix-dashboard project: Hystrix dashboard allows to identify which services are running in dashboard manner.



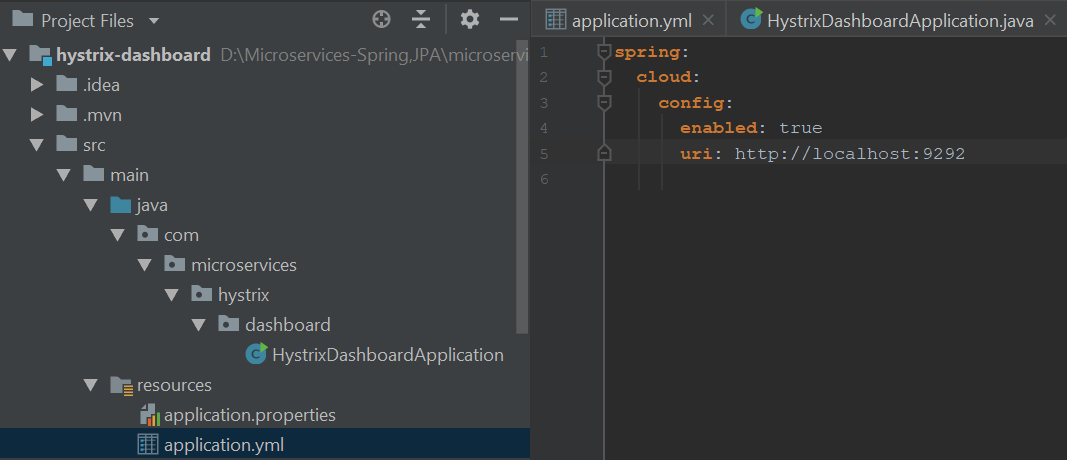
Create application.yml file under resources folder:



! I experienced a problem with displaying a dashboard, so I had to replace application.yml file with application.properties file.



Create bootstrap.yml file under resources folder:



Add dependency into pom.xml:

<dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-config</artifactId>  
</dependency>

And

<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>

Add hystrix functionality to cloud-gateway project. Netflix's Hystrix library provides an implementation of the circuit breaker pattern. When you apply a circuit breaker to a method, Hystrix watches for failing calls to that method, and, if failures build up to a threshold, Hystrix opens the circuit so that subsequent calls automatically fail. (<https://spring.io/guides/gs/circuit-breaker/> )

-add dependency into pom.xml

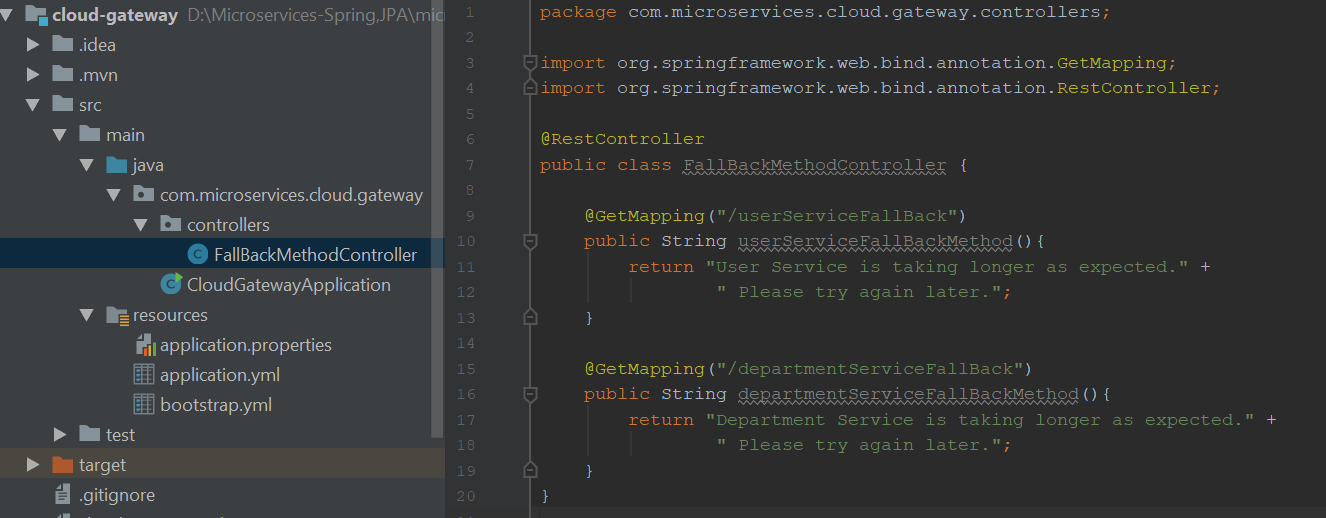
<dependency>

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

<artifactId>spring-cloud-starter-netflix-hystrix</artifactId>

</dependency>

Create new package **controllers** in **cloud-gateway** project and class for handle the failure in users and department services:



Add new lines to application.yml file with regards to hystrix features:

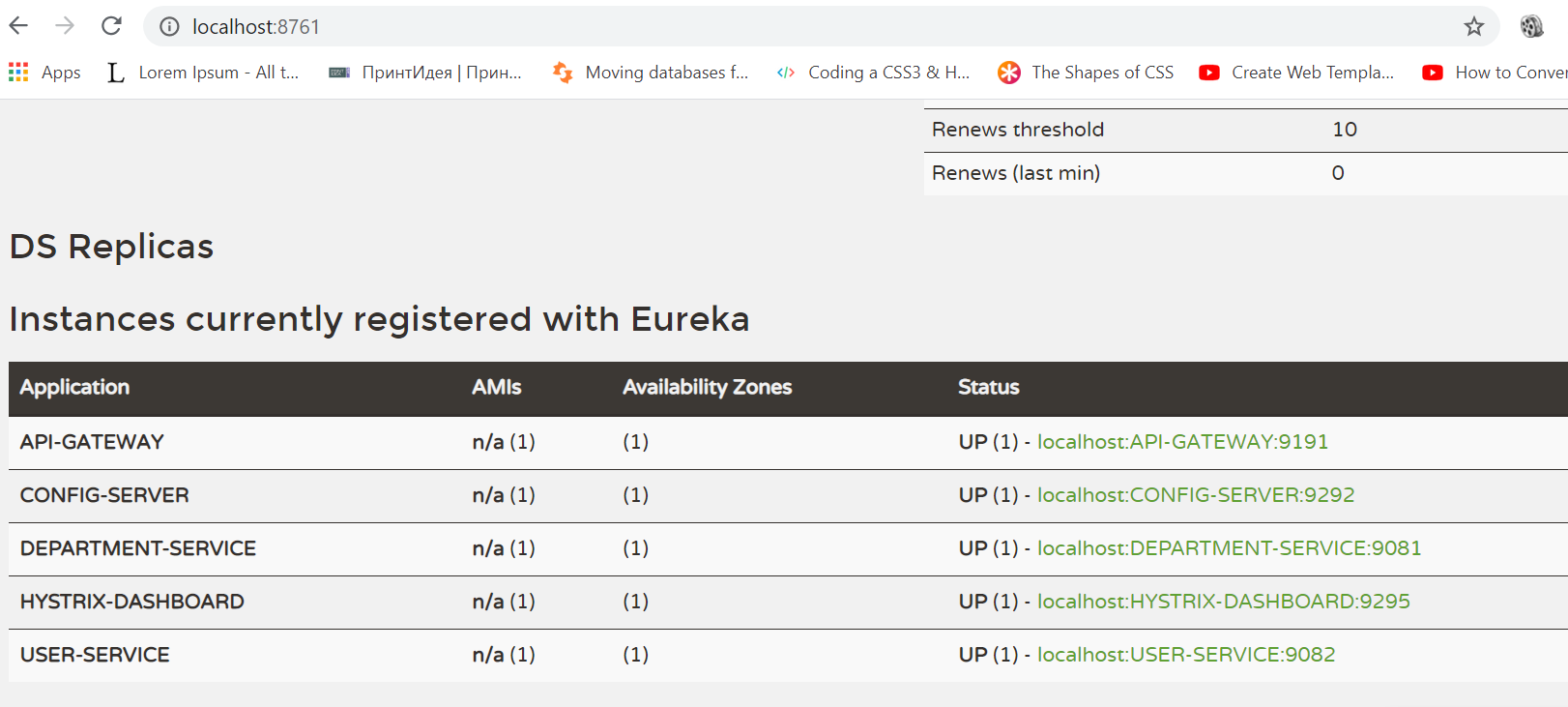
server:

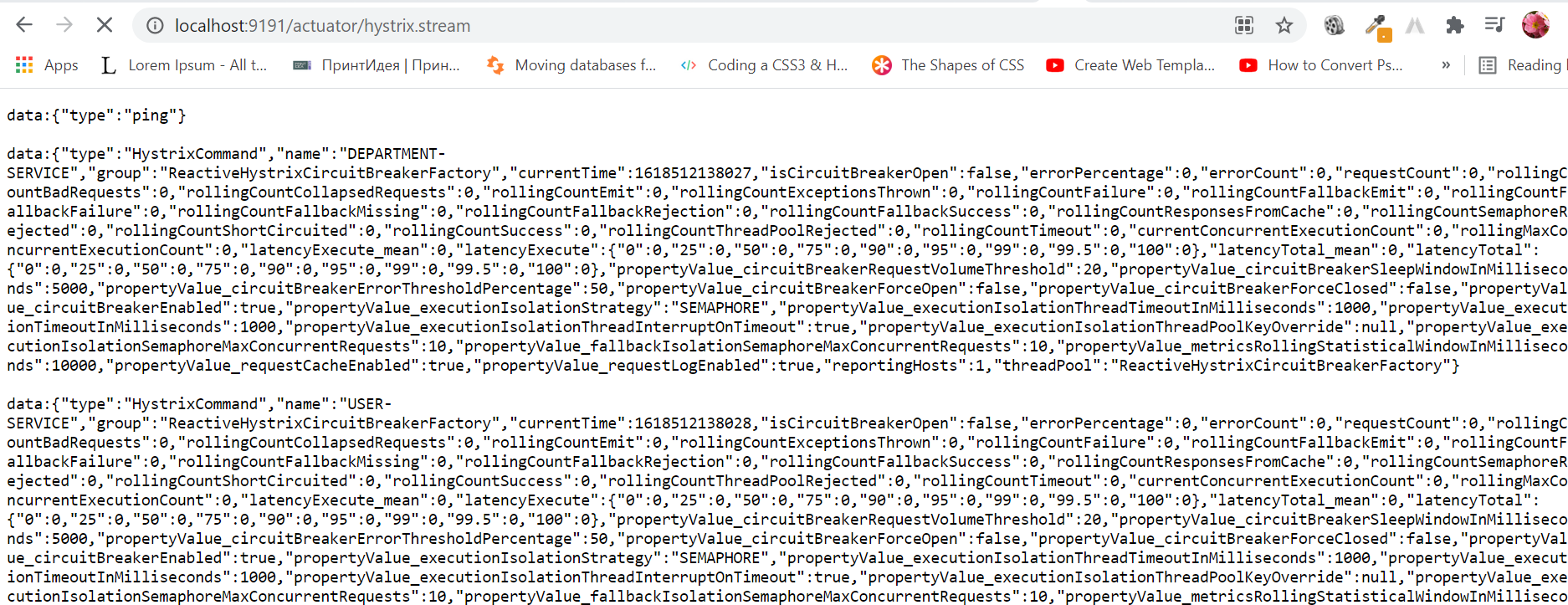
**server**:  
 **port**: 9191  
**spring**:  
 **application**:  
 **name**: API-GATEWAY  
 **cloud**:  
 **gateway**:  
 **routes**:  
 - **id**: USER-SERVICE  
 **uri**: lb://USER-SERVICE  
 **predicates**:  
 - Path=/users/\*\*  
 **filters**:  
 - **name**: CircuitBreaker  
 **args**:  
 **name**: 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  
  
**hystrix**:  
 **command**:  
 **fallbackcmd**:  
 **execution**:  
 **isolation**:  
 **thread**:  
 **timeoutInMilliseconds**: 500  
  
**management**:  
 **endpoints**:  
 **web**:  
 **exposure**:  
 **include**: hystrix.stream

Start all projects and type into browser:

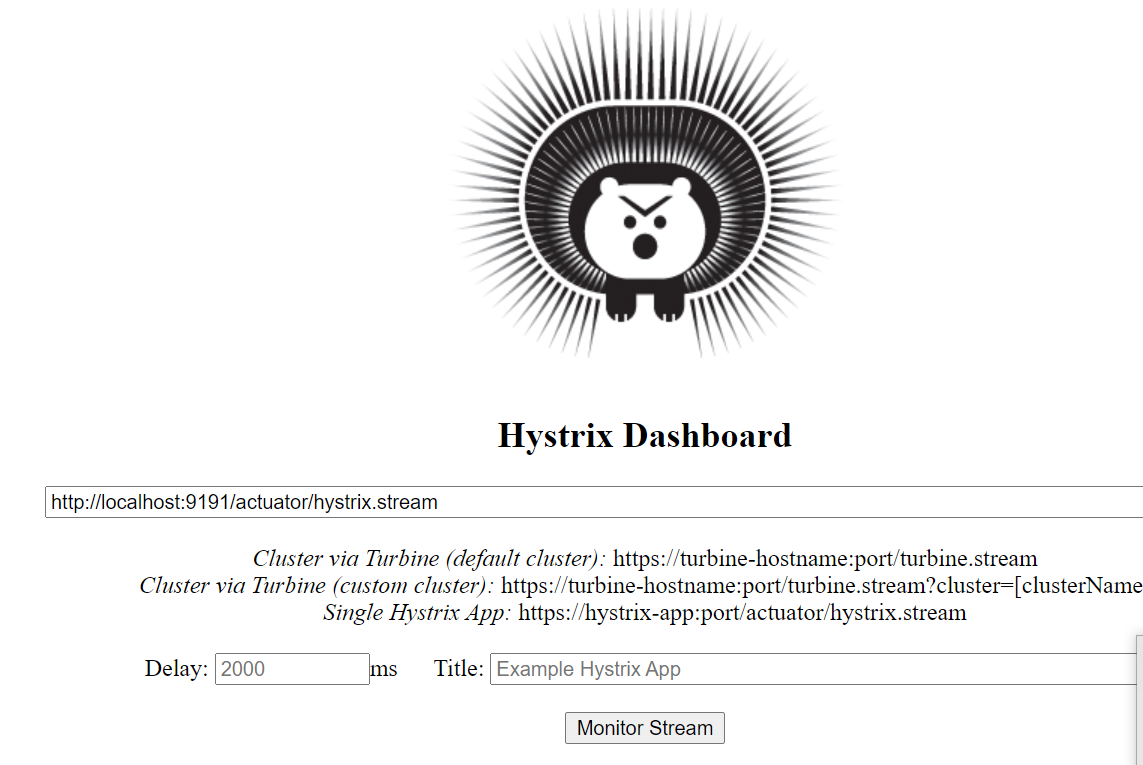
First we run **service-registry** application, second: **cloud-config** application, and then: **gateway**, **users**, **departments** and **hystrix-dashboard** apps.

**Open browser and type: localhost:8761**



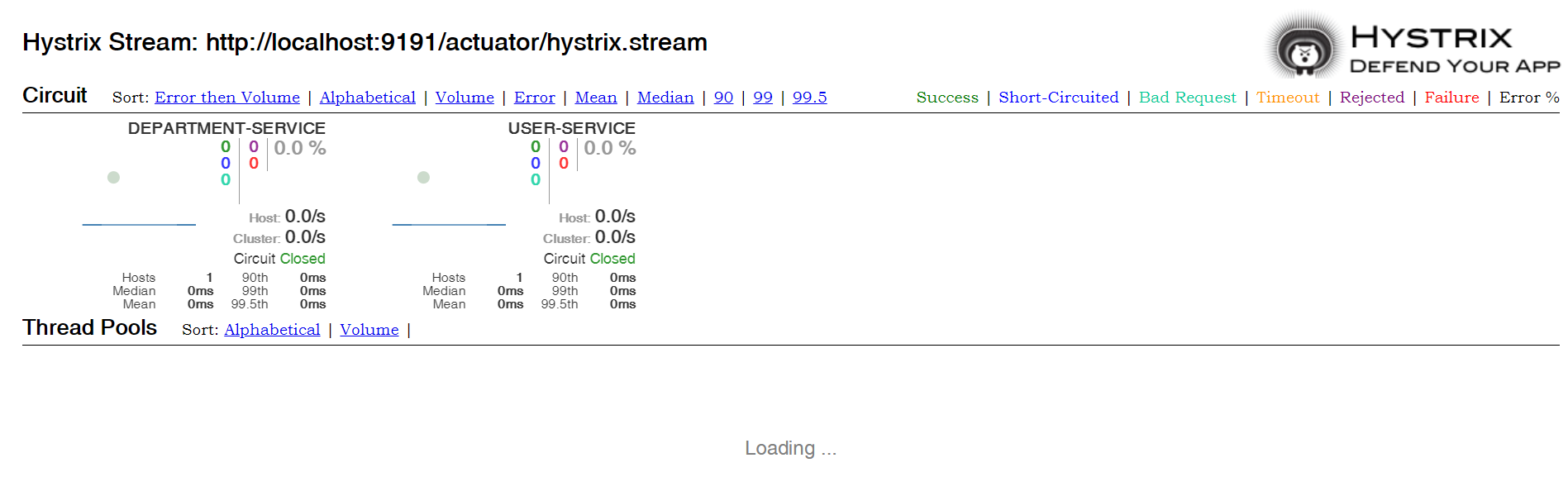
<http://localhost:9191/actuator/hystrix.stream> 

Open new browser: localhost:9295/hystrix and add above link <http://localhost:9191/actuator/hystrix.stream> into first text fild:



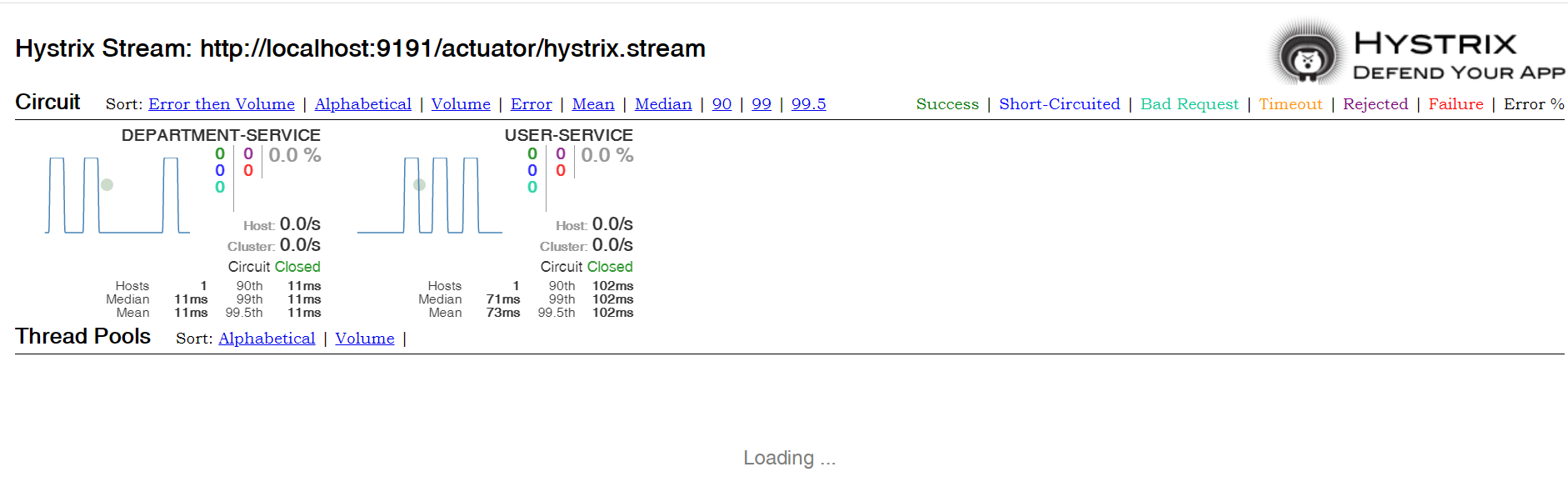
Click on button ‘Monitor Stream’:

http://localhost:9295/hystrix/monitor?stream=http%3A%2F%2Flocalhost%3A9191%2Factuator%2Fhystrix.stream

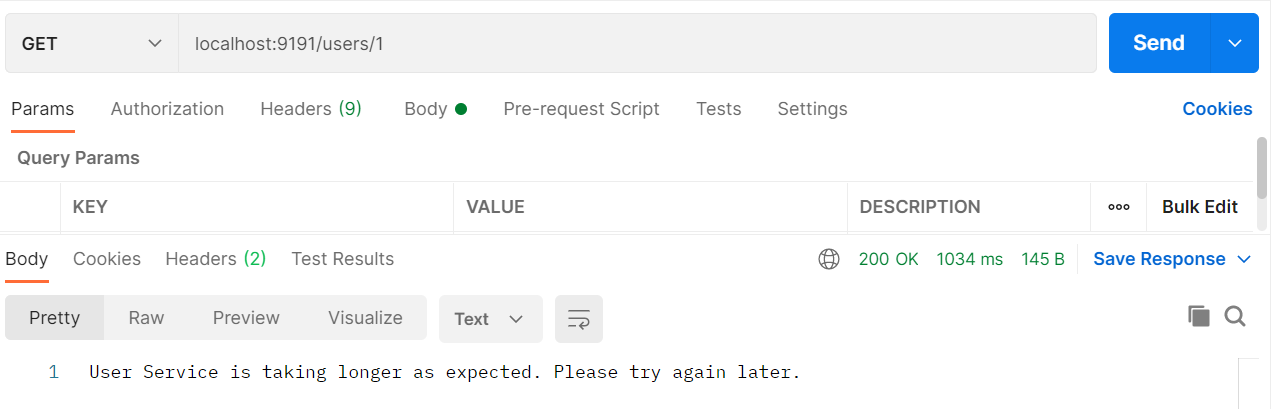


From Postman send requests to our gateway server and see changes into hystrix dashboard:

http://localhost:9295/hystrix/monitor?stream=http%3A%2F%2Flocalhost%3A9191%2Factuator%2Fhystrix.stream



Test method fallback: stop department service and send request to gateway from Postman:



Or

