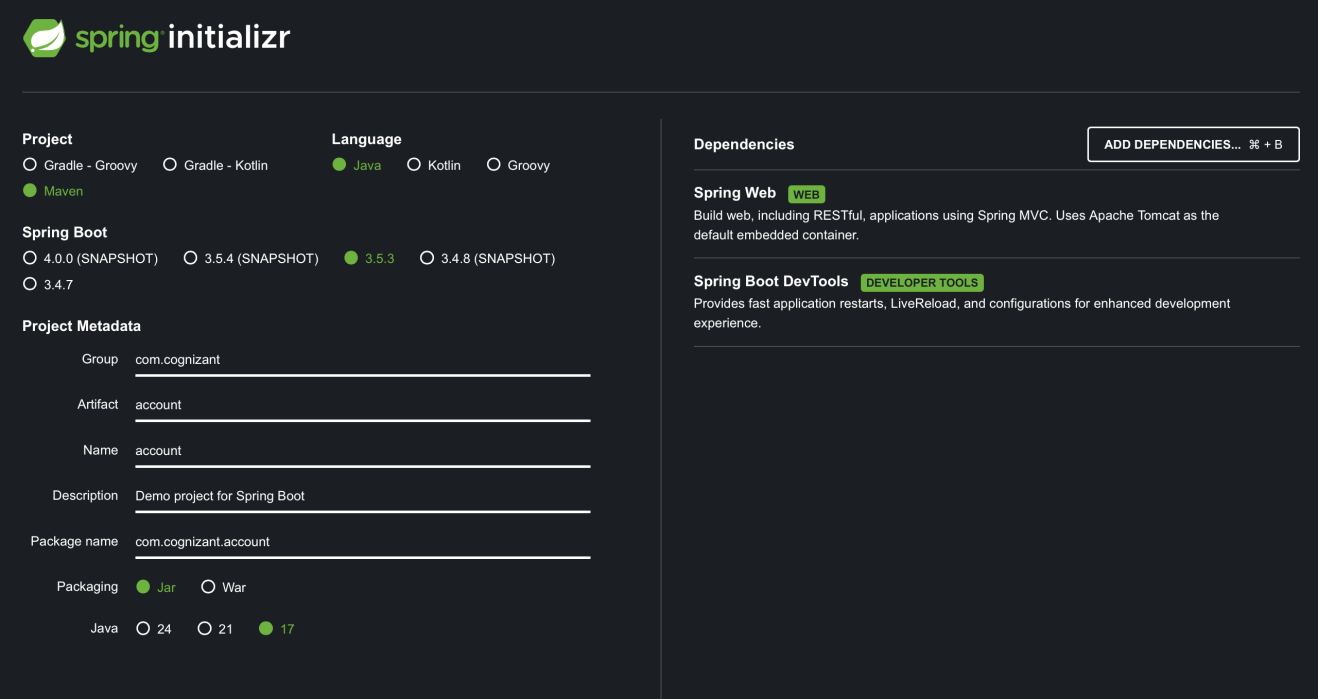
**HANDSON EXERCISES - WEEK 5**

**Skill : Microservices with Spring Boot 3 and Spring Cloud**

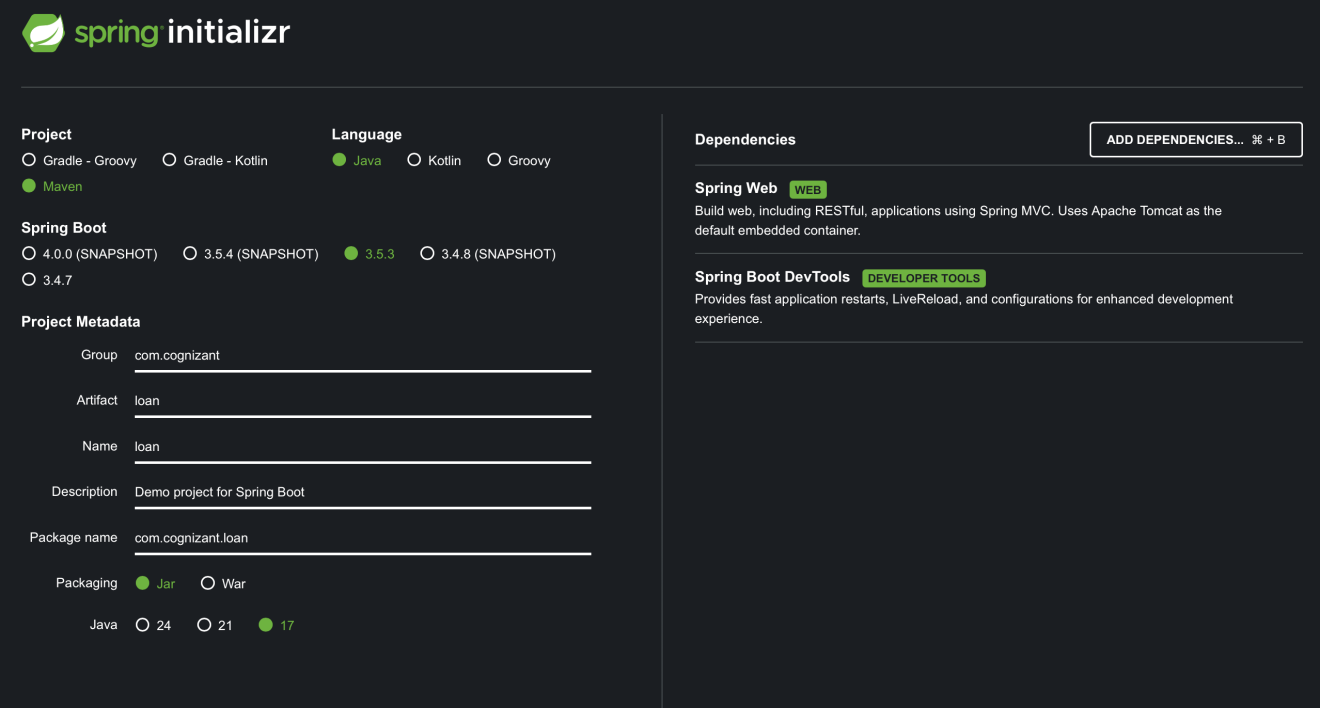
**File : 2. Microservices with API gateway**

**Exercise : Creating Microservices for account and loan**

**account-service :**



**loan-service :**



**AccountController.java :**

package com.cognizant.account.controller;  
  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.Map;  
  
@RestController  
@RequestMapping("/accounts")  
public class AccountController {  
  
 @GetMapping("/{number}")  
 public Map<String, Object> getAccountDetails(@PathVariable String number) {  
 return Map.of(  
 "number", number,  
 "type", "savings",  
 "balance", 234343  
 );  
 }  
}

**LoanController.java :**

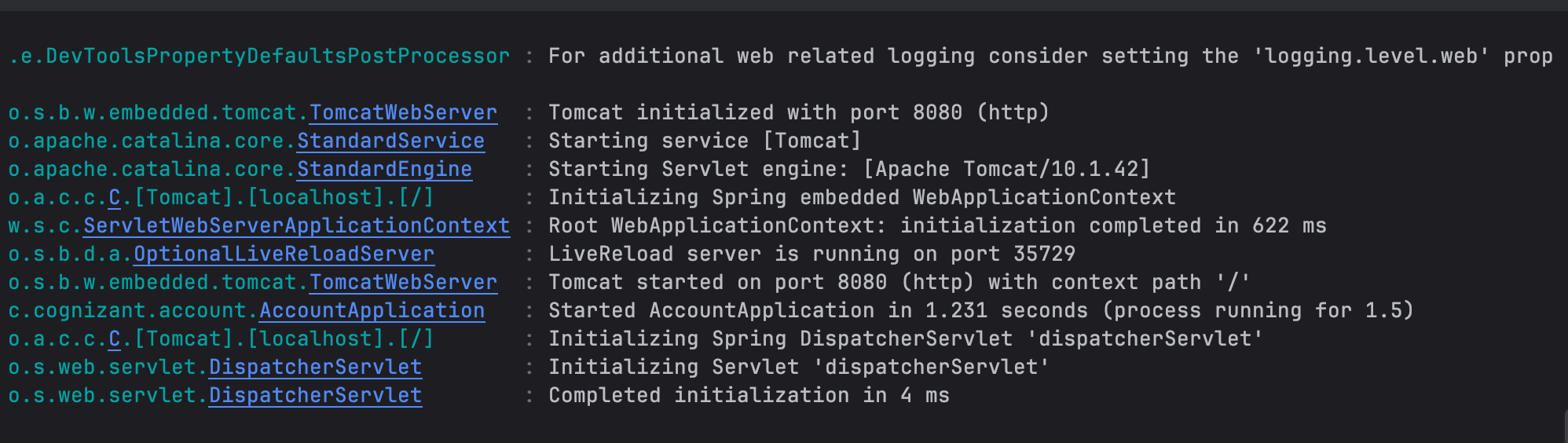
package com.cognizant.loan.controller;  
  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.Map;  
  
@RestController  
@RequestMapping("/loans")  
public class LoanController {  
  
 @GetMapping("/{number}")  
 public Map<String, Object> getLoanDetails(@PathVariable String number) {  
 return Map.of(  
 "number", number,  
 "type", "car",  
 "loan", 400000,  
 "emi", 3258,  
 "tenure", 18  
 );  
 }  
}

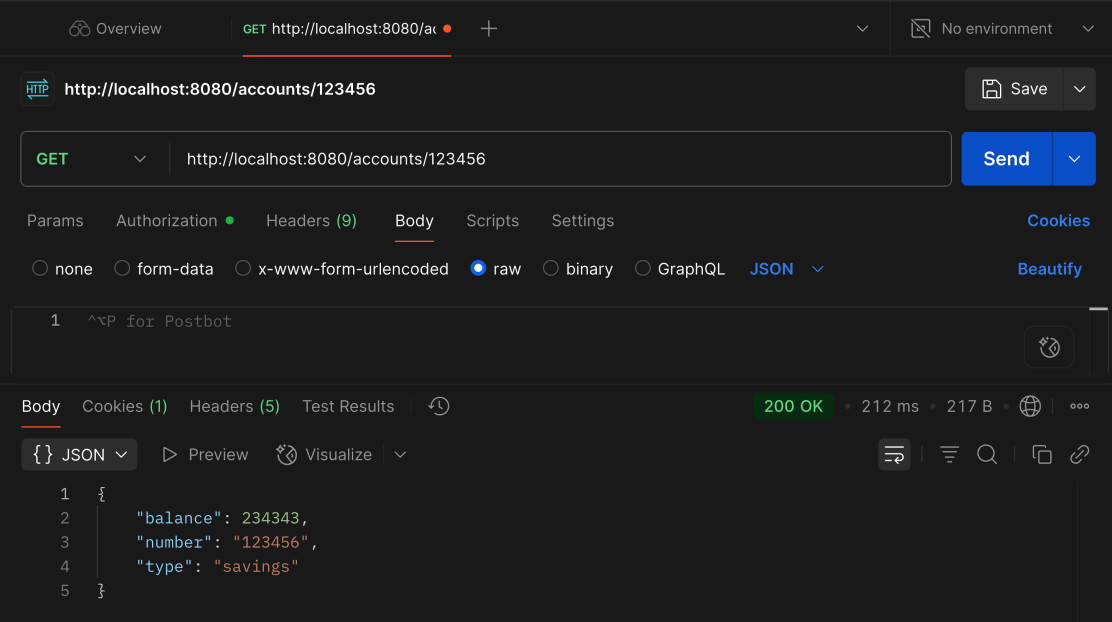
**Loan - application.properties :**

spring.application.name=loan  
server.port=8081

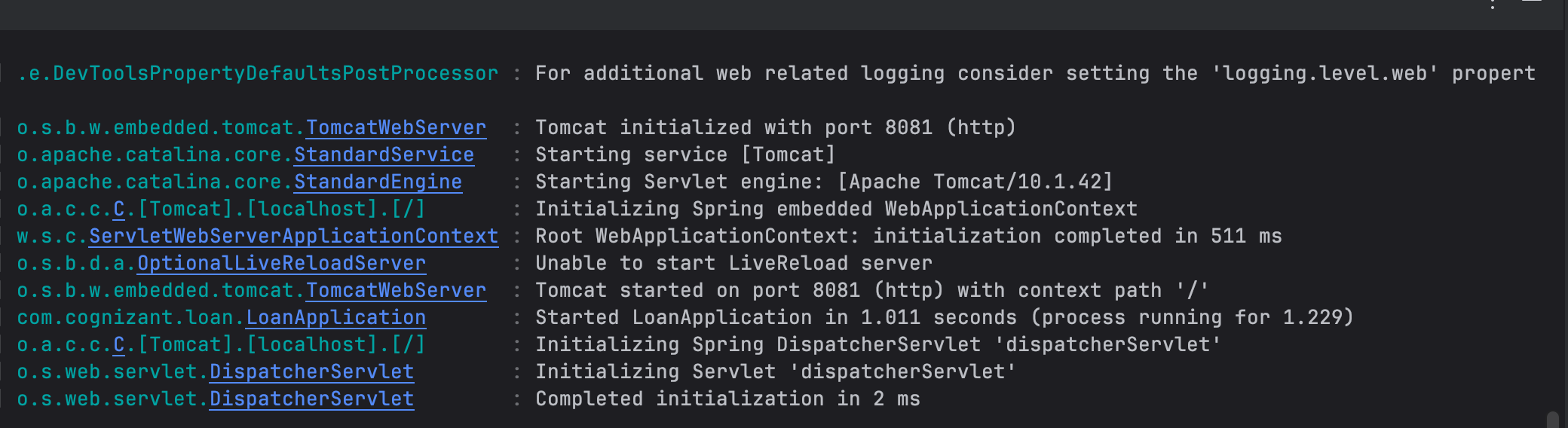
**OUTPUT :**

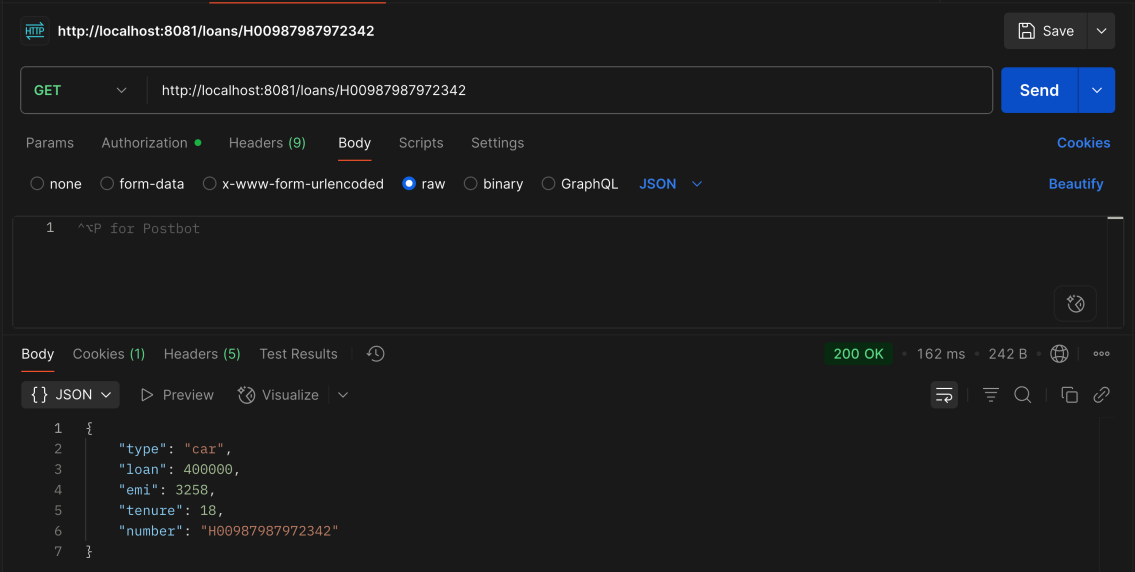
**Account : ( on port 8080)**



****

**Loan : ( on port 8081)**

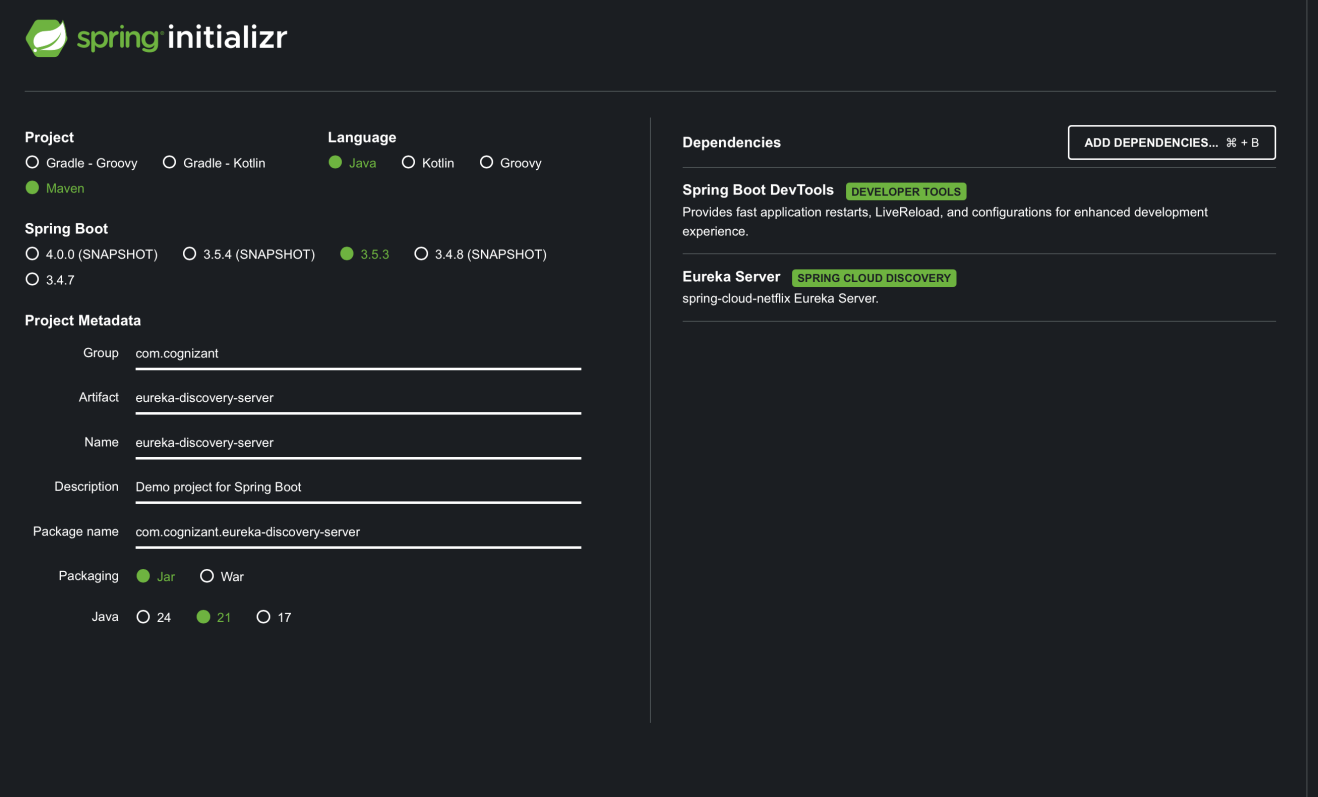


****

**Exercise : Create Eureka Discovery Server and register**

**microservices**

**eureka-discovery-server :**



**pom.xml :**

<parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>3.1.5</version>  
 <relativePath/>  
</parent>  
  
<properties>  
 <java.version>17</java.version>  
 <spring-cloud.version>2022.0.4</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-devtools</artifactId>  
 <scope>runtime</scope>  
 <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>

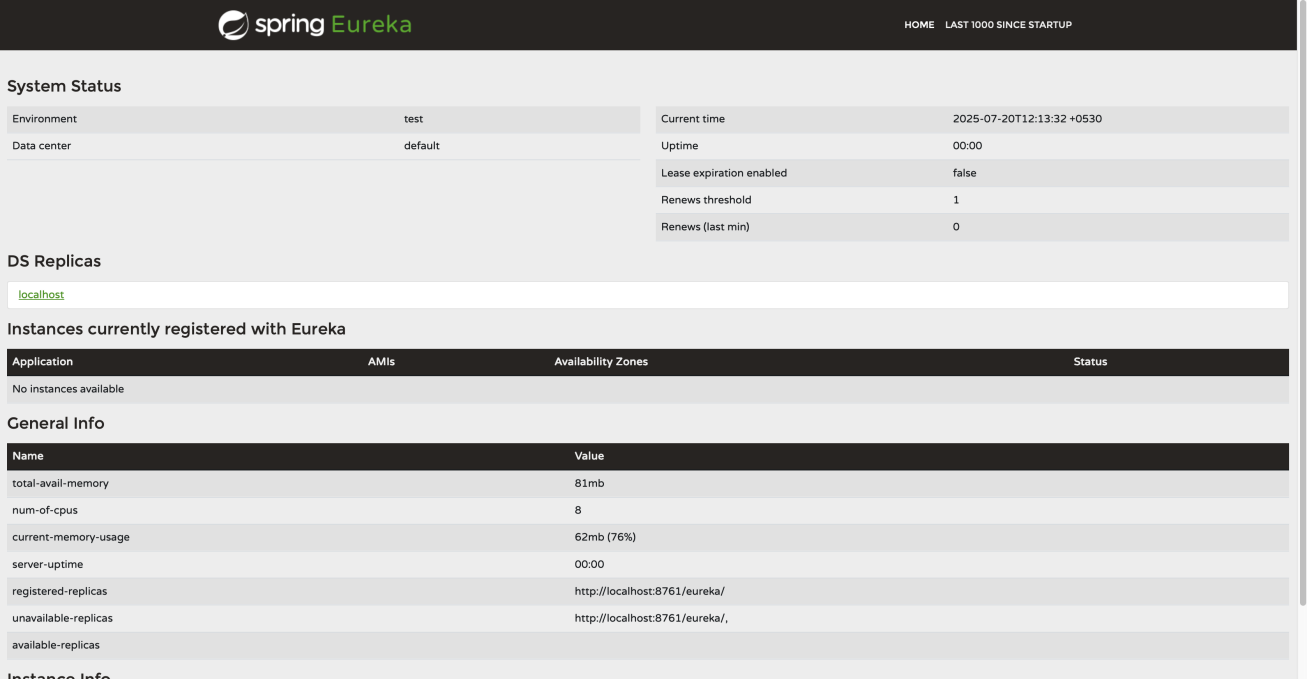
**EurekaDiscoveryServerApplication.java :**

package com.cognizant.eureka\_discovery\_server;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;  
  
@SpringBootApplication  
@EnableEurekaServer  
public class EurekaDiscoveryServerApplication {  
 public static void main(String[] args) {  
 SpringApplication.run(EurekaDiscoveryServerApplication.class, args);  
 }  
}

**resources/application.properties :**

spring.application.name=eureka-discovery-server  
server.port=8761  
eureka.client.register-with-eureka=false  
eureka.client.fetch-registry=false  
logging.level.com.netflix.eureka=OFF  
logging.level.com.netflix.discovery=OFF

**OUTPUT ~ No services are registered yet**

****

**pom.xml ( in both account and loan microservices )**

<dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>  
</dependency>

**AccountServiceApplication.java :**

package com.cognizant.account;  
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@EnableDiscoveryClient  
@SpringBootApplication  
public class AccountApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.run(AccountApplication.class, args);  
 }  
  
}

**resources/applciation.properties :**

server.port=8080  
spring.application.name=account-service  
eureka.client.service-url.defaultZone=http://localhost:8761/eureka

**LoanServiceApplication.java :**

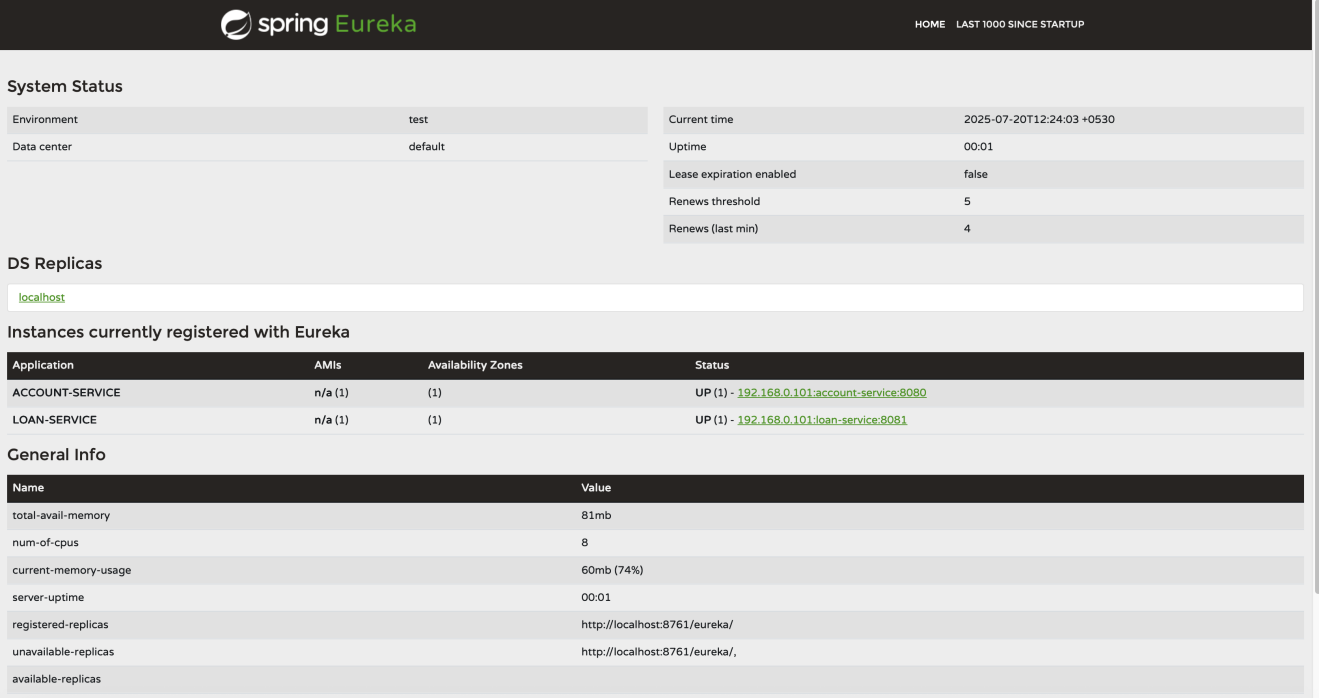
package com.cognizant.loan;  
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@EnableDiscoveryClient  
@SpringBootApplication  
public class LoanApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.run(LoanApplication.class, args);  
 }  
  
}

**resources/applciation.properties :**

server.port=8081  
spring.application.name=loan-service  
eureka.client.service-url.defaultZone=http://localhost:8761/eureka

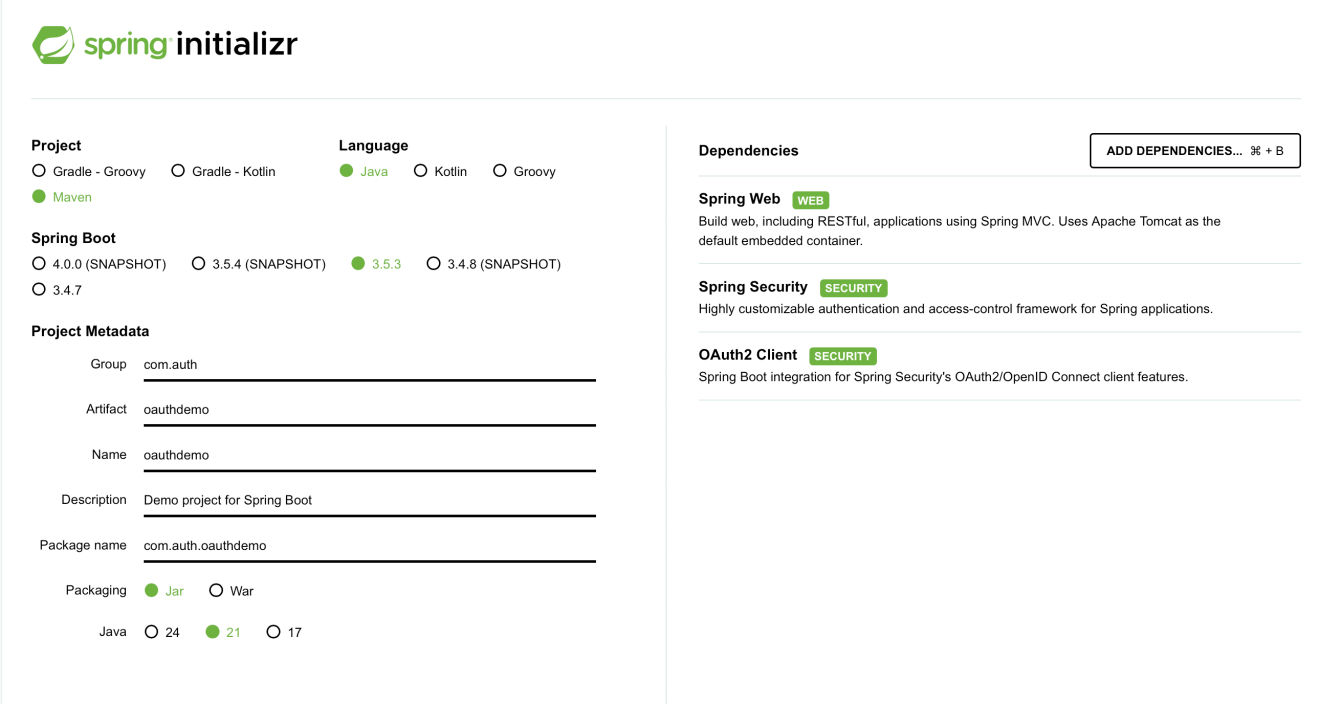
**OUTPUT :**

**Eureka Dashboard ~** http://localhost:8761

****

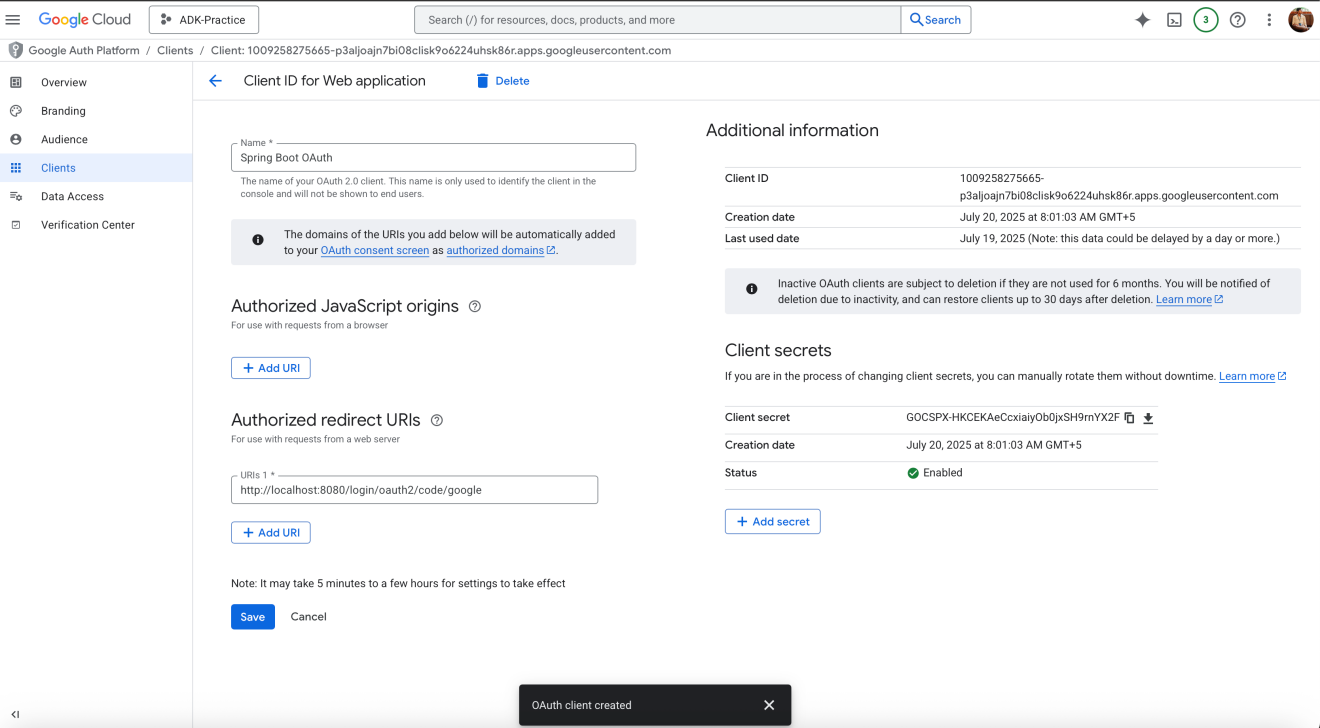
**File : 0. Sample Microservices exercises**

**Exercise 1 : Implementing Centralized Authentication with OAuth 2.1/OIDC**



**Google Cloud Console Setup :**

* Enabled OAuth Content Screen and set user type to External
* Created OAuth 2.o Credentials.



**resources/application.yml :**

spring:  
 security:  
 oauth2:  
 client:  
 registration:  
 google:  
 client-id: 1009258275665-p3aljoajn7bi08clisk9o6224uhsk86r.apps.googleusercontent.com  
 client-secret: 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  
 scope: openid, profile, email  
 redirect-uri: "{baseUrl}/login/oauth2/code/{registrationId}"  
 client-name: Google  
 provider:  
 google:  
 authorization-uri: https://accounts.google.com/o/oauth2/auth  
 token-uri: https://oauth2.googleapis.com/token  
 user-info-uri: https://openidconnect.googleapis.com/v1/userinfo  
 user-name-attribute: sub

**com.auth.oauthdemo.config/SecurityConfig.java :**

package com.example.oauthdemo.config;  
  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.security.config.annotation.web.builders.HttpSecurity;  
import org.springframework.security.web.SecurityFilterChain;  
  
@Configuration  
public class SecurityConfig {  
  
 @Bean  
 public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {  
 http  
 .authorizeHttpRequests(authorize ->  
 authorize.anyRequest().authenticated()  
 )  
 .oauth2Login(); // enables OAuth2 login  
  
 return http.build();  
 }  
}

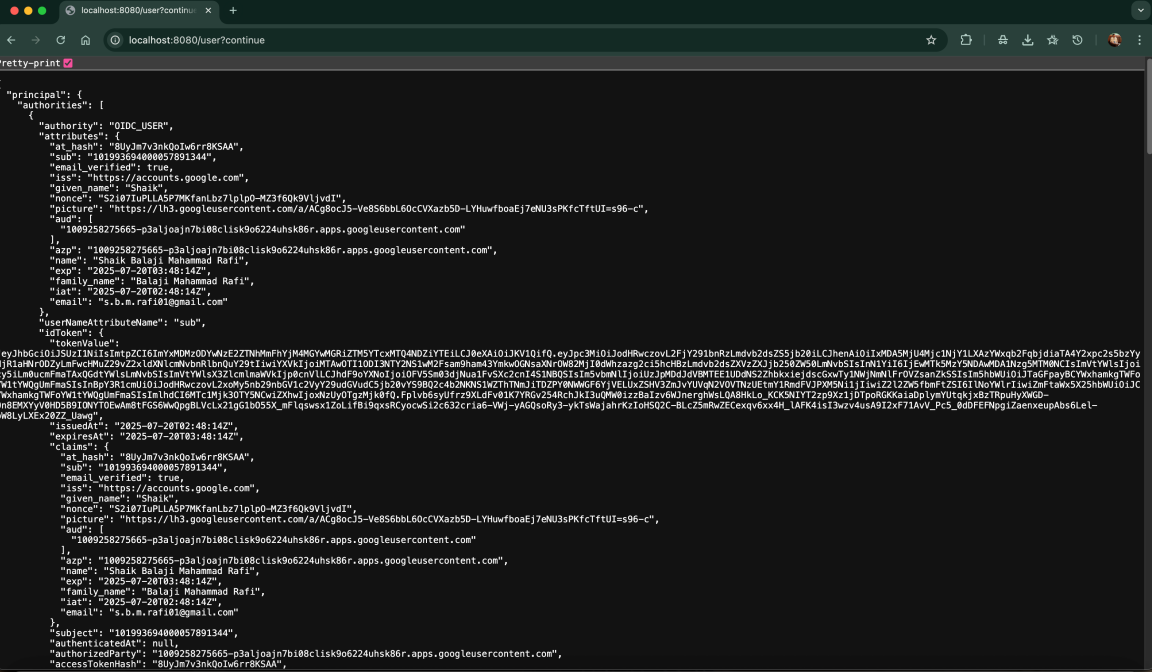
**com.auth.oauthdemo.controller/UserController.java :**

package com.auth.oauthdemo.controller;  
  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
import java.security.Principal;  
  
@RestController  
public class UserController {  
 @GetMapping("/user")  
 public Principal user(Principal principal) {  
 return principal;  
 }  
}

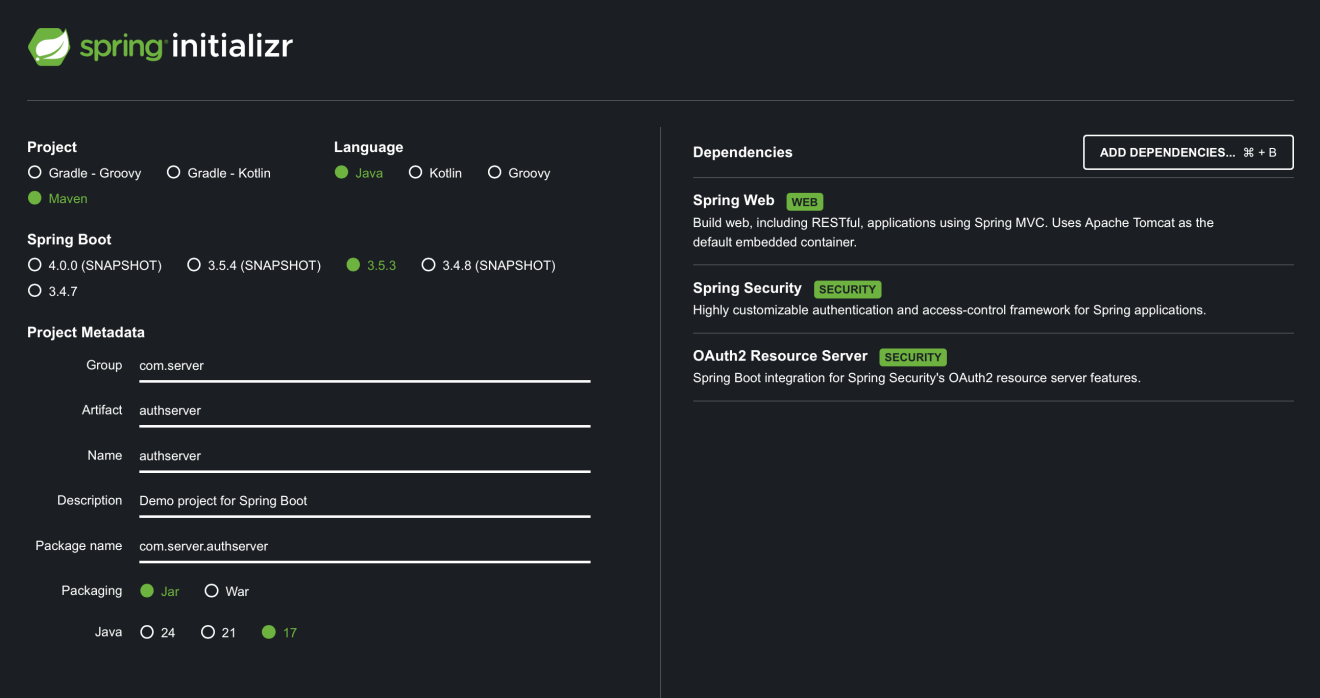
**OUTPUT : <http://localhost:8080/user>**

|  |  |
| --- | --- |
|  |  |

**JSON Data :**



**Exercise 2 : Configuring Authorization Servers and Resource Servers**



**resources/application.yml :**

spring:  
 security:  
 oauth2:  
 resourceserver:  
 jwt:  
 issuer-uri: https://issuer.example.com

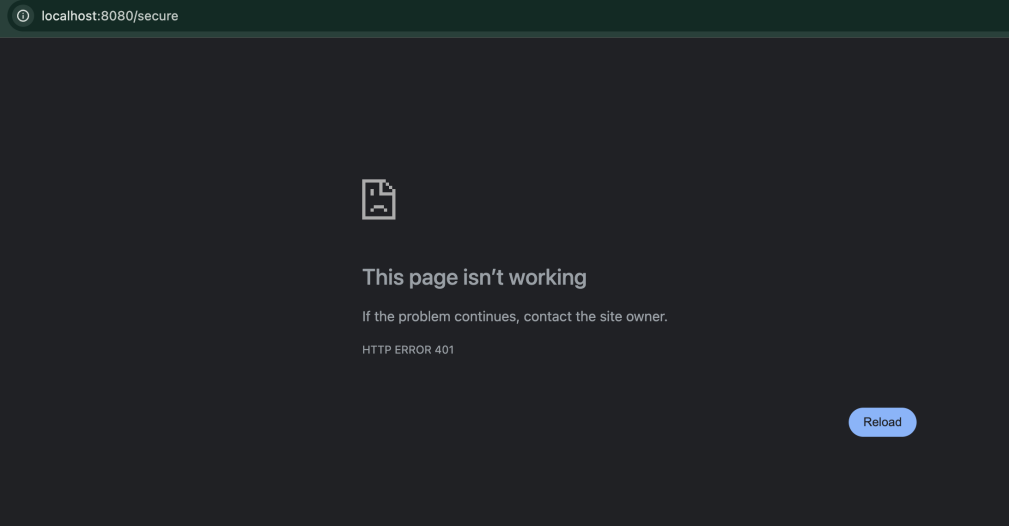
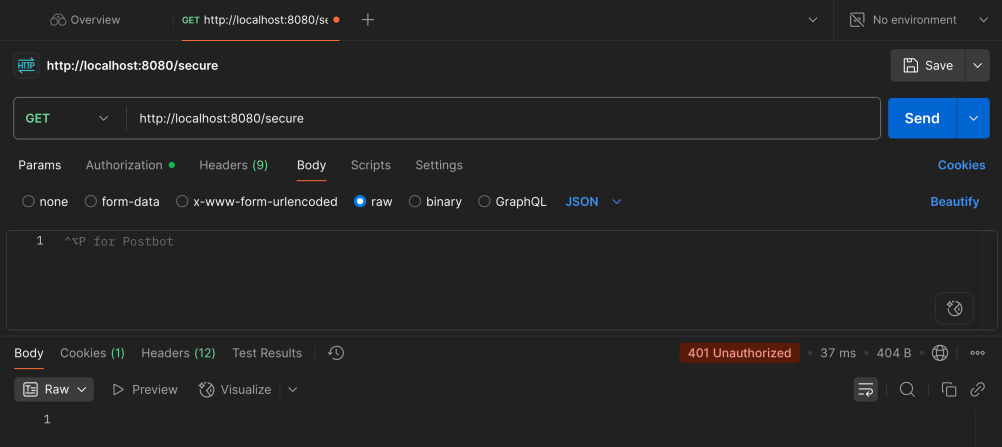
**com.server.authserver.config/ResourceServerConfig.java :**

package com.server.authserver.config;  
  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.security.config.annotation.web.builders.HttpSecurity;  
import org.springframework.security.web.SecurityFilterChain;  
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;  
import org.springframework.security.oauth2.server.resource.authentication.JwtAuthenticationConverter;  
  
@Configuration  
@EnableWebSecurity  
public class ResourceServerConfig {  
  
 @Bean  
 public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {  
 http  
 .authorizeHttpRequests(auth -> auth  
 .anyRequest().authenticated()  
 )  
 .oauth2ResourceServer(oauth2 -> oauth2  
 .jwt()  
 );  
  
 return http.build();  
 }  
}

**com.server.authserver.controller/SecureController.java :**

package com.example.demo.controller;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RestController;  
@RestController  
public class SecureController {  
 @GetMapping("/secure")  
 public String secure() {  
 return "This is a secure endpoint";  
 }  
}

**OUTPUT:**



Due to my preparation for the upcoming Agentic AI Hackathon by Hack2Skill, I was only able to dedicate this amount of time to practice. Thank you for your understanding.