

Lab 04 Ecomm Product Service 🡪 REST MicroserviceFood Protection Program

**Spring Cloud Product REST Service**

Udemy Spring Cloud Course

Presented By

Binit Datta

**Rolling Stone Technology**

**Formatted: December, 2016**

**Table of Content**

Preface 6

1.0 - Introduction 7

1.1 – Create a new Spring Starter Project 8

1.2 – Fill initial values 9

1.3 – Choose Eureka and Web as starter projects 10

1.4 – Click Finish Now 29

1.5 – Let Spring Tool Suite Prepare the Project 30

1.6 – View the generated pom.xml 31

1.7 – Add this property to the pom.xml 33

1.8 – Add these maven dependencies to the pom.xml 34

1.9 – Replace the Maven build section with this 35

1.10 – Add Eureka Client Discovery Annotation 36

1.11 – Add api.rest package 37

1.12– Add dao.jpa package 38

1.13– Add domain package 39

1.14– Add exception package 40

1.15– Add service package 41

1.16– Create User Domain class in the domain package 42

1.17– Do the following to the User Class 44

1.18– Create the Address Domain class in the domain package 44

1.19– Do the following to the Address Class 45

1.20– Generate RestAPIExceptionInfo in the domain package 45

1.21– Generate HTTP400Exception in the exception package 46

1.22– Generate HTTP404Exception in the exception package 47

1.23– Generate DAOInterface in the dao.jpa package 48

1.24– Generate Service class in the service package 49

1.25– Generate ServiceProperties class in the service package 50

1.26– Generate ServiceHealth class in the service package 51

1.27– Generate ServiceEvent class in the service package 52

1.28– Generate AbstractRestController class in the rest.api package 53

1.29– Generate UserRestController class in the rest.api package 55

1.30– Create Aspect class in com.rollingstone package 57

1.31– Create application.yml file under resources 57

1.32– Add bootstrap.yml file in the resources folder 59

1.33– Run the Eureka Discovery Server 59

1.34– Open Git bash in the project directory 60

1.35– Run the project 61

1.36 – Navigate to http://localhost:8761 61

1.37– Create a User 62

1.38– Create another user 63

1.39– Verify in MySQL Workbench 64

1.40—Try GET Request 65

1.41—Try to Update a Record 66

1.42 – Try to get a single user 67

1.43 – Try to delete a single user 68

1.44 – Conclusion 69

Chapter 1

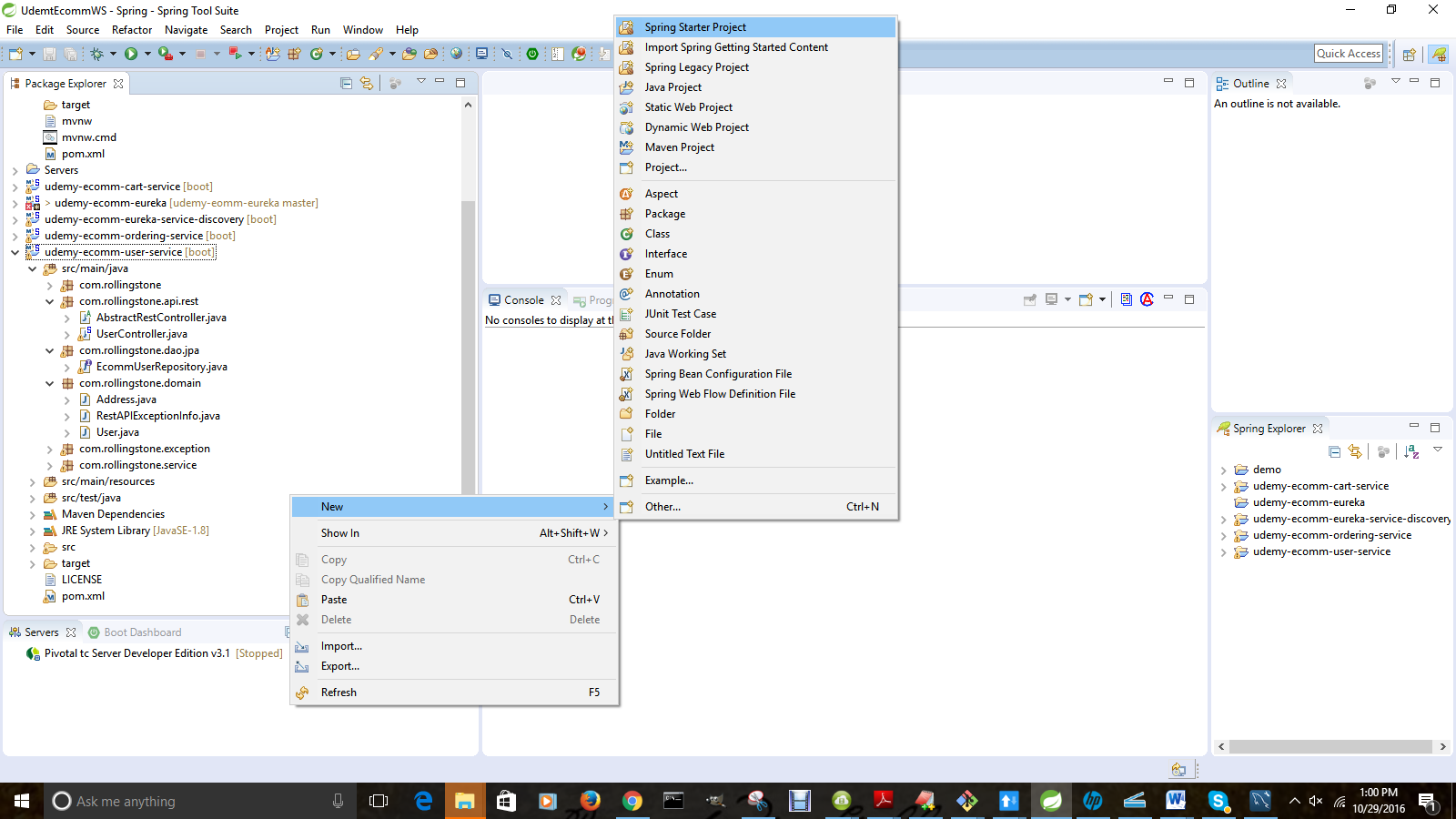
*Spring Cloud Eureka Service Discovery Project Creation*

1.0 - Introduction

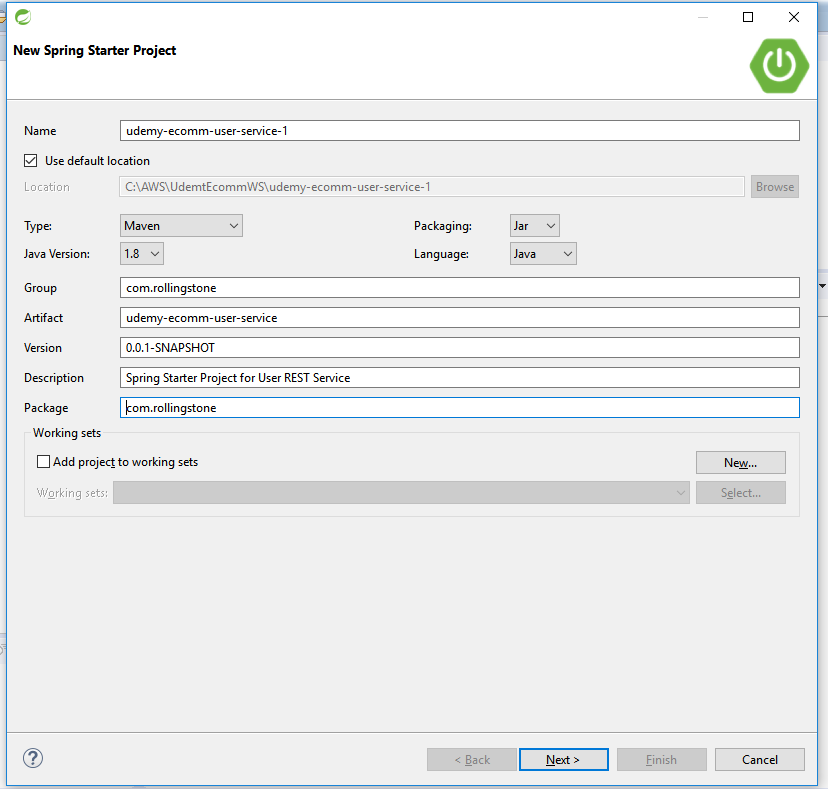
The purpose of the *MPPU Safe Food-Handling Plan* is to insure that the products – whole raw poultry and giblets – offered for sale by Massachusetts’ smallest-scale poultry producers using a Massachusetts-approved MPPU – are wholesome and processed under clean and sanitary conditions, and that the operation meets Massachusetts Department of Environmental Protection (DEP) guidelines for waste disposal, and, therefore, does not contribute to environmental harm.



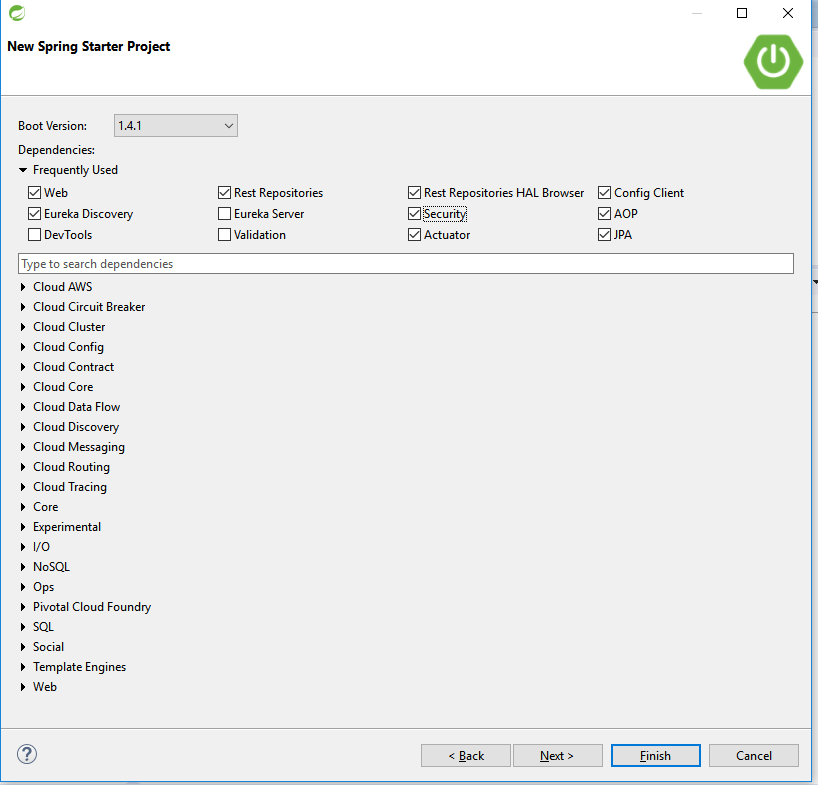
1.1 – Create a new Spring Starter Project



1.2 – Fill initial values

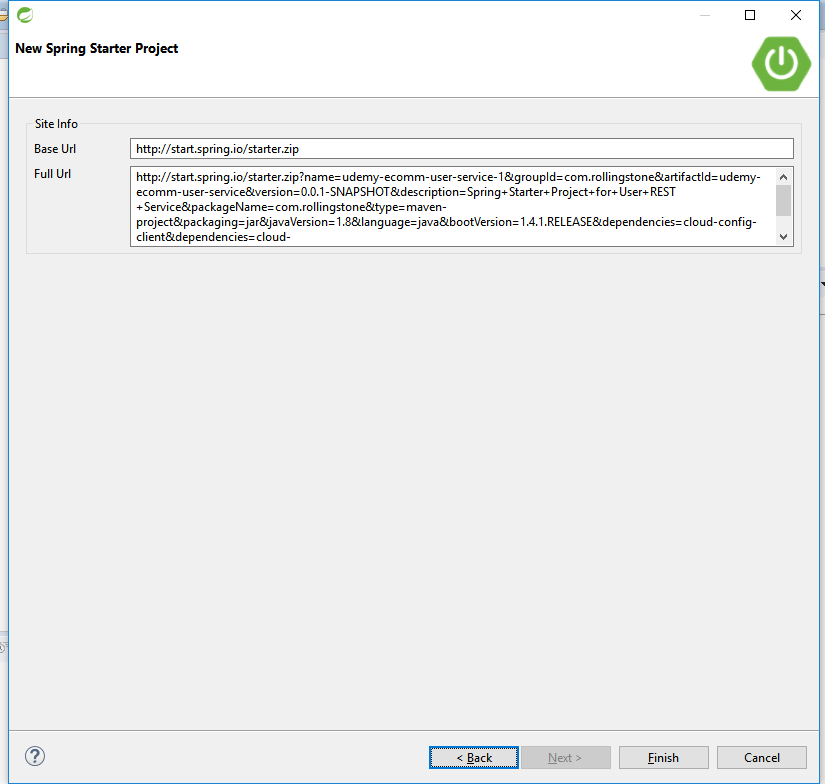


1.3 – Choose Eureka and Web as starter projects

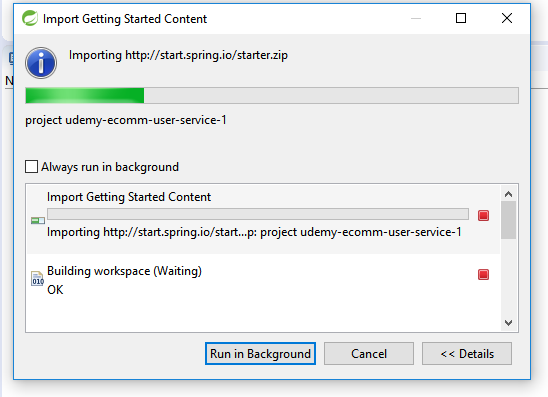


.

1.4 – Click Finish Now



1.5 – Let Spring Tool Suite Prepare the Project



1.6 – View the generated 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 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.rollingstone</groupId>

<artifactId>udemy-ecomm-product-service</artifactId>

<version>1.0</version>

<packaging>jar</packaging>

<description>Example project demonstrating Spring Cloud based Product Microservice as a REST API</description>

<parent>

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

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

<version>1.3.6.RELEASE</version>

</parent>

<properties>

<start-class>com.rollingstone.ProductRestAPIApplication</start-class>

</properties>

<dependencyManagement>

<dependencies>

<dependency>

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

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

<version>Angel.SR6</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<dependencies>

<dependency>

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

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

</dependency>

<!-- web development, including Tomcat and spring-webmvc -->

<dependency>

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

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

</dependency>

<!-- Spring security -->

<dependency>

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

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

</dependency>

<dependency>

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

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

<scope>provided</scope>

</dependency>

<!-- spring-data-jpa, spring-orm and Hibernate -->

<dependency>

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

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<version>1.4.181</version>

</dependency>

<!-- spring-test, hamcrest, ... -->

<dependency>

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

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

<scope>test</scope>

</dependency>

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.data</groupId>

<artifactId>spring-data-rest-hal-browser</artifactId>

</dependency>

<!-- attribute level json comparisons -->

<dependency>

<groupId>com.jayway.jsonpath</groupId>

<artifactId>json-path</artifactId>

<version>0.9.1</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>com.jayway.jsonpath</groupId>

<artifactId>json-path-assert</artifactId>

<version>0.9.1</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>io.springfox</groupId>

<artifactId>springfox-swagger2</artifactId>

<version>2.3.1</version>

</dependency>

<dependency>

<groupId>io.springfox</groupId>

<artifactId>springfox-swagger-ui</artifactId>

<version>2.3.1</version>

</dependency>

<dependency>

<groupId>org.hsqldb</groupId>

<artifactId>hsqldb</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.40</version>

</dependency>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

</dependencies>

<build>

<resources>

<resource>

<directory>src/main/resources</directory>

<filtering>true</filtering>

</resource>

</resources>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

<!-- Spring boot support -->

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

<configuration>

<addResources>false</addResources>

</configuration>

</plugin>

</plugins>

</build>

</project>

1.7 – Maven POM Elements Explanation

1. Starter parent
2. Dependency Management
3. Packaging
4. POM Hierarchy
5. Spring Boot Actuator
6. Starter Web
7. Starter Security
8. Starter Tomcat
9. Starter JPA
10. Starter Test
11. Jackson Databind
12. HAL Browser
13. Swagger
14. Eureka
15. Feign
16. Resources Plugin
17. Compiler Plugin
18. Spring Boot Plugin

1.8 – Add this property to the pom.xml

<start-class>com.rollingstone.ProductRestAPIApplication</start-class>

1.9 – Add these maven dependencies to the pom.xml

<dependency>

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

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

</dependency>

<!-- web development, including Tomcat and spring-webmvc -->

<dependency>

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

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

</dependency>

<!-- Spring security -->

<dependency>

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

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

</dependency>

<dependency>

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

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

<scope>provided</scope>

</dependency>

<!-- spring-data-jpa, spring-orm and Hibernate -->

<dependency>

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

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<version>1.4.181</version>

</dependency>

<!-- spring-test, hamcrest, ... -->

<dependency>

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

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

<scope>test</scope>

</dependency>

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.data</groupId>

<artifactId>spring-data-rest-hal-browser</artifactId>

</dependency>

<!-- attribute level json comparisons -->

<dependency>

<groupId>com.jayway.jsonpath</groupId>

<artifactId>json-path</artifactId>

<version>0.9.1</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>com.jayway.jsonpath</groupId>

<artifactId>json-path-assert</artifactId>

<version>0.9.1</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>io.springfox</groupId>

<artifactId>springfox-swagger2</artifactId>

<version>2.3.1</version>

</dependency>

<dependency>

<groupId>io.springfox</groupId>

<artifactId>springfox-swagger-ui</artifactId>

<version>2.3.1</version>

</dependency>

<dependency>

<groupId>org.hsqldb</groupId>

<artifactId>hsqldb</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.40</version>

</dependency>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

1.10 – Replace the Maven build section with this

<build>

<resources>

<resource>

<directory>src/main/resources</directory>

<filtering>true</filtering>

</resource>

</resources>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

<!-- Spring boot support -->

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

<configuration>

<addResources>false</addResources>

</configuration>

</plugin>

</plugins>

</build>

1.11 – Add Eureka Client Discovery Annotation

package com.rollingstone;

/\*

\* This is the primary Spring Boot application class. It configures Spring Boot, JPA, Swagger and

\* other dependent Spring modules.

\*/

@SuppressWarnings("deprecation")

@EnableAutoConfiguration // Sprint Boot Automatic Configuration

@ComponentScan(basePackages = "com.rollingstone")

@EnableJpaRepositories("com.rollingstone.dao.jpa") // To segregate MongoDB and JPA repositories. Otherwise not needed.

@EnableSwagger2

@EnableDiscoveryClient

@EnableFeignClients

public class ProductRestAPIApplication extends SpringBootServletInitializer {

private static final Class<ProductRestAPIApplication> applicationClass = ProductRestAPIApplication.class;

private static final Logger log = LoggerFactory.getLogger(applicationClass);

public static void main(String[] args) {

SpringApplication.run(applicationClass, args);

}

@Override

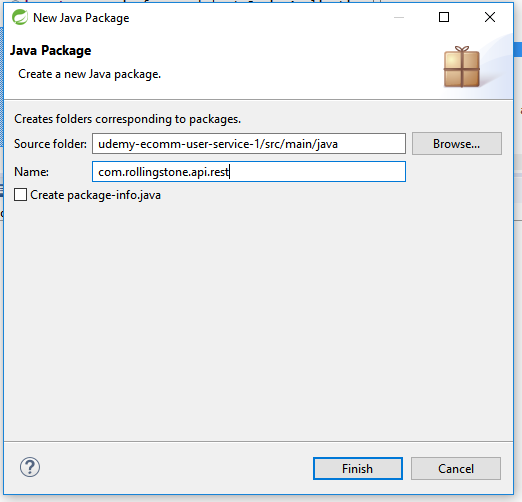
protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {

return application.sources(applicationClass);

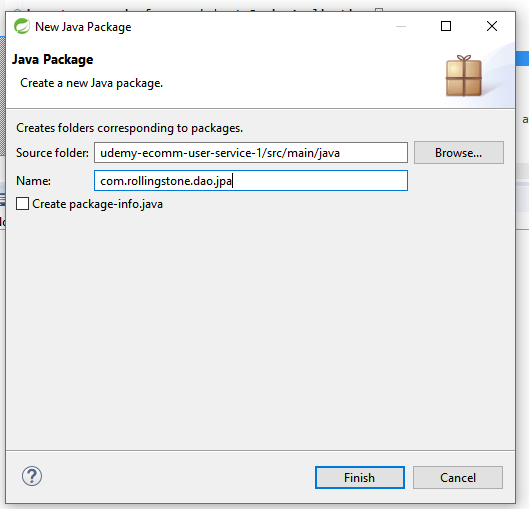
}

}

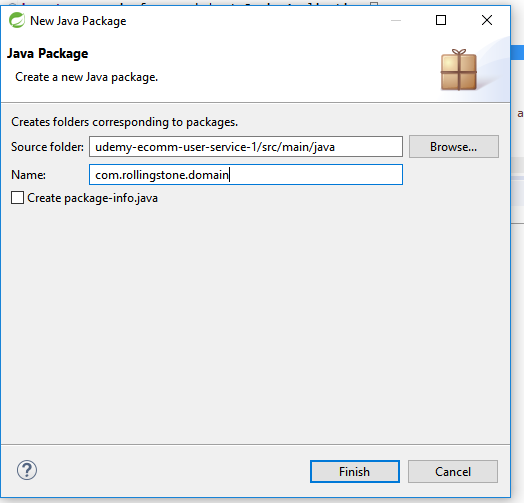
1.12 – Add api.rest package



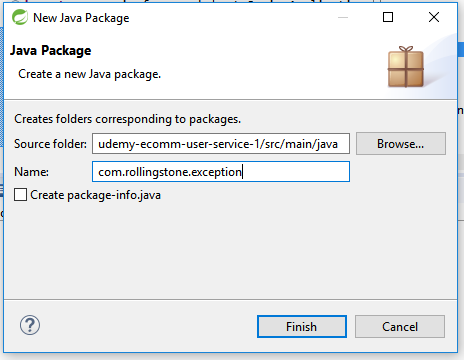
1.13– Add dao.jpa package



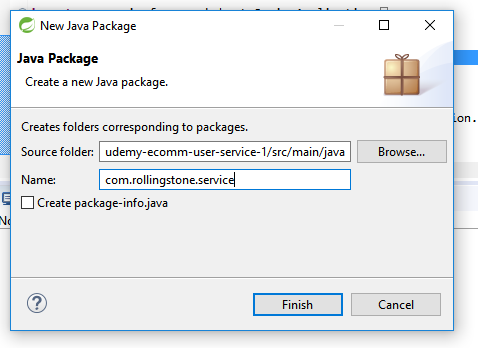
1.14– Add domain package



1.15– Add exception package



1.16– Add service package



1.17– Create User Domain class in the domain package

package com.rollingstone;

import org.aspectj.lang.JoinPoint;

import org.aspectj.lang.ProceedingJoinPoint;

import org.aspectj.lang.annotation.Around;

import org.aspectj.lang.annotation.Aspect;

import org.aspectj.lang.annotation.Before;

import org.aspectj.lang.annotation.Pointcut;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.stereotype.Component;

@Aspect

@Component

public class RestControllerAspect {

private static final Logger log = LoggerFactory.getLogger(RestControllerAspect.class);

@Before("execution(public \* com.rollingstone.api.rest.\*Controller.\*(..))")

public void logBeforeRestCall(JoinPoint pjp) throws Throwable {

log.info(":::::AOP Before REST call:::::" + pjp);

}

}

1.18– Create User Domain class in the domain package

package com.rollingstone.domain;

@Entity

@Table(name = "ecomm\_product")

@XmlRootElement

@XmlAccessorType(XmlAccessType.FIELD)

public class Product {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private long id;

@Column(nullable = false)

private String productName;

@Column(nullable = false)

private String productShortDescription;

@Column(nullable = false)

private String productLongDescription;

@Column(nullable = false)

private String imageUrl;

@Column(nullable = false)

private String productType;

@Column(nullable = false)

private double productPrice;

public Product() {

}

public Product(String productName, String productShortDescription, String productLongDescription, String imageUrl,

String productType, double productPrice) {

super();

this.productName = productName;

this.productShortDescription = productShortDescription;

this.productLongDescription = productLongDescription;

this.imageUrl = imageUrl;

this.productType = productType;

this.productPrice = productPrice;

}

public double getProductPrice() {

return productPrice;

}

public void setProductPrice(double productPrice) {

this.productPrice = productPrice;

}

public long getId() {

return id;

}

public void setId(long id) {

this.id = id;

}

public String getProductName() {

return productName;

}

public void setProductName(String productName) {

this.productName = productName;

}

public String getProductShortDescription() {

return productShortDescription;

}

public void setProductShortDescription(String productShortDescription) {

this.productShortDescription = productShortDescription;

}

public String getProductLongDescription() {

return productLongDescription;

}

public void setProductLongDescription(String productLongDescription) {

this.productLongDescription = productLongDescription;

}

public String getImageUrl() {

return imageUrl;

}

public void setImageUrl(String imageUrl) {

this.imageUrl = imageUrl;

}

public String getProductType() {

return productType;

}

public void setProductType(String productType) {

this.productType = productType;

}

@Override

public String toString() {

return "Product [id=" + id + ", productName=" + productName + ", productShortDescription="

+ productShortDescription + ", productLongDescription=" + productLongDescription + ", imageUrl="

+ imageUrl + ", productType=" + productType + ", productPrice=" + productPrice + ", getProductPrice()="

+ getProductPrice() + ", getId()=" + getId() + ", getProductName()=" + getProductName()

+ ", getProductShortDescription()=" + getProductShortDescription() + ", getProductLongDescription()="

+ getProductLongDescription() + ", getImageUrl()=" + getImageUrl() + ", getProductType()="

+ getProductType() + ", getClass()=" + getClass() + ", hashCode()=" + hashCode() + ", toString()="

+ super.toString() + "]";

}

}

}

1.18– Do the following to the Product Class

* Create Product Class
* Paste Code
* Press CTRL+Shift+O [Command + Shift + O in Mac] to import
* Choose java.persistence package
* Generate Getter Setter
* Generate Constructor using Fields
* Generate toString

1.19– Generate RestAPIExceptionInfo in the domain package

package com.rollingstone.domain;

import javax.xml.bind.annotation.XmlRootElement;

/\*

\* A sample class for adding error information in the response

\*/

@XmlRootElement

public class RestAPIExceptionInfo {

public final String detail;

public final String message;

public RestAPIExceptionInfo(Exception ex, String detail) {

this.message = ex.getLocalizedMessage();

this.detail = detail;

}

}

1.20– Generate HTTP400Exception in the exception package

package com.rollingstone.exception;

/\*\*

\* for HTTP 400 Bad Request errors

\*/

public final class HTTP400Exception extends RuntimeException {

public HTTP400Exception() {

super();

}

public HTTP400Exception(String message, Throwable cause) {

super(message, cause);

}

public HTTP400Exception(String message) {

super(message);

}

public HTTP400Exception(Throwable cause) {

super(cause);

}

}

1.21– Generate HTTP404Exception in the exception package

package com.rollingstone.exception;

/\*\*

\* For HTTP 404 Not Found errros

\*/

public class HTTP404Exception extends RuntimeException {

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

public HTTP404Exception() {

super();

}

public HTTP404Exception(String message, Throwable cause) {

super(message, cause);

}

public HTTP404Exception(String message) {

super(message);

}

public HTTP404Exception(Throwable cause) {

super(cause);

}

}

1.22– Generate DAOInterface in the dao.jpa package

package com.rollingstone.dao.jpa;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.Pageable;

import org.springframework.data.repository.PagingAndSortingRepository;

import com.rollingstone.domain.Product;

public interface EcommProductRepository extends PagingAndSortingRepository<Product, Long> {

Product findUserByProductType(String productType);

Page findAll(Pageable pageable);

}

1.23– Generate Service class in the service package

package com.rollingstone.service;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.actuate.metrics.CounterService;

import org.springframework.boot.actuate.metrics.GaugeService;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.PageRequest;

import org.springframework.stereotype.Service;

import com.rollingstone.dao.jpa.EcommProductRepository;

import com.rollingstone.domain.Product;

/\*

\* Service class to do CRUD for Product through JPA Repository

\*/

@Service

public class EcommProductService {

private static final Logger log = LoggerFactory.getLogger(EcommProductService.class);

@Autowired

private EcommProductRepository productRepository;

@Autowired

CounterService counterService;

@Autowired

GaugeService gaugeService;

public EcommProductService() {

}

public Product createProduct(Product product) {

return productRepository.save(product);

}

public Product getProduct(long id) {

return productRepository.findOne(id);

}

public void updateProduct(Product product) {

productRepository.save(product);

}

public void deleteProduct(Long id) {

productRepository.delete(id);

}

public Page<Product> getAllProducts(Integer page, Integer size) {

Page pageOfProducts = productRepository.findAll(new PageRequest(page, size));

// example of adding to the /metrics

if (size > 50) {

counterService.increment("com.rollingstone.getAll.largePayload");

}

return pageOfProducts;

}

}

1.24– Generate Service Properties class in the service package

package com.rollingstone.service;

import org.springframework.boot.context.properties.ConfigurationProperties;

import org.springframework.stereotype.Component;

import javax.validation.constraints.NotNull;

/\*

\* demonstrates how service-specific properties can be injected

\*/

@ConfigurationProperties(prefix = "product.service", ignoreUnknownFields = false)

@Component

public class ServiceProperties {

@NotNull // you can also create configurationPropertiesValidator

private String name = "Empty";

public String getName() {

return this.name;

}

public void setName(String name) {

this.name = name;

}

}

1.25– Generate ServiceHealth class in the service package

package com.rollingstone.service;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.actuate.health.Health;

import org.springframework.boot.actuate.health.HealthIndicator;

import org.springframework.stereotype.Component;

/\*\*

\* This is an optional class used to inject application specific health check

\* into the Spring Boot health management endpoint.

\*/

@Component

public class ProductServiceHealth implements HealthIndicator {

@Autowired

private ServiceProperties configuration;

@Override

public Health health() {

return Health.up().withDetail("details", "{ 'internals' : 'getting close to limit', 'profile' : '" + this.configuration.getName() + "' }").status("itsok!").build();

}

}

1.27– Generate ServiceEvent class in the service package

package com.rollingstone.service;

import org.springframework.context.ApplicationEvent;

/\*\*

\* This is an optional class used in publishing application events.

\* This can be used to inject events into the Spring Boot audit management endpoint.

\*/

public class ProductServiceEvent extends ApplicationEvent {

public ProductServiceEvent(Object source) {

super(source);

}

public String toString() {

return "My Product Service Event";

}

}

1.28– Generate AbstractRestController class in the rest.api package

package com.rollingstone.api.rest;

/\*\*

\* This class is meant to be the backbone of all other REst controllers. It contains common functionality such as exception handling etc.

\*/

//@ControllerAdvice?

public abstract class AbstractRestController implements ApplicationEventPublisherAware {

protected final Logger log = LoggerFactory.getLogger(this.getClass());

protected ApplicationEventPublisher eventPublisher;

protected static final String DEFAULT\_PAGE\_SIZE = "30";

protected static final String DEFAULT\_PAGE\_NUM = "0";

@ResponseStatus(HttpStatus.BAD\_REQUEST)

@ExceptionHandler(HTTP400Exception.class)

public

@ResponseBody

RestAPIExceptionInfo handleDataStoreException(HTTP400Exception ex, WebRequest request, HttpServletResponse response) {

log.info("Converting Data Store exception to RestResponse : " + ex.getMessage());

return new RestAPIExceptionInfo(ex, "The Request did not have correct parameters / body etc. Please check");

}

@ResponseStatus(HttpStatus.NOT\_FOUND)

@ExceptionHandler(HTTP404Exception.class)

public

@ResponseBody

RestAPIExceptionInfo handleResourceNotFoundException(HTTP404Exception ex, WebRequest request, HttpServletResponse response) {

log.info("ResourceNotFoundException handler:" + ex.getMessage());

return new RestAPIExceptionInfo(ex, "The Endpoint was not found.");

}

@Override

public void setApplicationEventPublisher(ApplicationEventPublisher applicationEventPublisher) {

this.eventPublisher = applicationEventPublisher;

}

//todo: replace with exception mapping

public static <T> T checkResourceFound(final T resource) {

if (resource == null) {

throw new HTTP404Exception("resource not found");

}

return resource;

}

}

1.29– Generate ProductRestController class in the rest.api package

package com.rollingstone.api.rest;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.data.domain.Page;

import org.springframework.http.HttpStatus;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.ResponseBody;

import org.springframework.web.bind.annotation.ResponseStatus;

import org.springframework.web.bind.annotation.RestController;

import com.rollingstone.domain.Product;

import com.rollingstone.exception.HTTP400Exception;

import com.rollingstone.service.EcommProductService;

/\*

\* Demonstrates how to set up RESTful API endpoints using Spring MVC

\*/

@RestController

@RequestMapping(value = "/productsservice/v1/products")

public class ProductController extends AbstractRestController {

private static final Logger log = LoggerFactory.getLogger(ProductController.class);

@Autowired

private EcommProductService productService;

@RequestMapping(value = "",

method = RequestMethod.POST,

consumes = {"application/json", "application/xml"},

produces = {"application/json", "application/xml"})

@ResponseStatus(HttpStatus.CREATED)

public void createProduct(@RequestBody Product product,

HttpServletRequest request, HttpServletResponse response) {

Product createdProduct = this.productService.createProduct(product);

response.setHeader("Location", request.getRequestURL().append("/").append(createdProduct.getId()).toString());

}

@RequestMapping(value = "",

method = RequestMethod.GET,

produces = {"application/json", "application/xml"})

@ResponseStatus(HttpStatus.OK)

public

@ResponseBody

Page<Product> getAllProducts(@RequestParam(value = "page", required = true, defaultValue = DEFAULT\_PAGE\_NUM) Integer page,

@RequestParam(value = "size", required = true, defaultValue = DEFAULT\_PAGE\_SIZE) Integer size,

HttpServletRequest request, HttpServletResponse response) {

return this.productService.getAllProducts(page, size);

}

@RequestMapping(value = "/{id}",

method = RequestMethod.GET,

produces = {"application/json", "application/xml"})

@ResponseStatus(HttpStatus.OK)

public

@ResponseBody

Product getProduct(@PathVariable("id") Long id,

HttpServletRequest request, HttpServletResponse response) throws Exception {

Product product = this.productService.getProduct(id);

checkResourceFound(product);

return product;

}

@RequestMapping("/simple/{id}")

public Product getSimpleProduct(@PathVariable("id") Long id) {

log.info("In Real ProductController getProduct 11111 getSimpleUser Kaku 000" + System.currentTimeMillis());

Product product = this.productService.getProduct(id);

checkResourceFound(product);

return product;

}

@RequestMapping(value = "/{id}",

method = RequestMethod.PUT,

consumes = {"application/json", "application/xml"},

produces = {"application/json", "application/xml"})

@ResponseStatus(HttpStatus.NO\_CONTENT)

public void updateProduct(@PathVariable("id") Long id, @RequestBody Product product,

HttpServletRequest request, HttpServletResponse response) {

checkResourceFound(this.productService.getProduct(id));

log.info("received id :"+id);

if (id != product.getId()) throw new HTTP400Exception("ID doesn't match!");

this.productService.updateProduct(product);

}

@RequestMapping(value = "/{id}",

method = RequestMethod.DELETE,

produces = {"application/json", "application/xml"})

@ResponseStatus(HttpStatus.NO\_CONTENT)

public void deleteProduct(@PathVariable("id") Long id, HttpServletRequest request,

HttpServletResponse response) {

checkResourceFound(this.productService.getProduct(id));

this.productService.deleteProduct(id);

}

}

1.30– Create application.yml file under resources

### This is the main way to configure the application (other than annotations).

### This fils is in Yaml format but you can also do this using the traditional

### Java properties file.

spring.jmx:

enabled: false

spring.datasource:

driverClassName: org.h2.Driver

url: jdbc:h2:mem:bootexample;MODE=MySQL

server:

port: 8284

#todo: make sure to always enable security in production

security:

basic:

enabled: false

#management endpoints on a separate port

management:

port: 8091

security:

enabled: false # management port is internal only. no need to secure it.

#default project info followed by actual injected pom-specified values.

project:

name: spring-boot-rest-example

version: 0.1

description: boot-example default description

info:

build:

artifact: ${project.artifactId}

name: ${project.name}

description: ${project.description}

version: ${project.version}

user.service:

name: 'default profile:'

---

spring:

profiles: test

spring.jpa:

hibernate.ddl-auto: create-drop

user.service:

name: 'test profile:'

---

spring:

profiles: mysql

datasource:

driverClassName: com.mysql.jdbc.Driver

url: jdbc:mysql://localhost/ecommerce

username: root

password: root

jpa:

hibernate:

dialect: org.hibernate.dialect.MySQLInnoDBDialect

ddl-auto: update # todo: in non-dev environments, comment this out:

user.service:

name: 'test profile:'

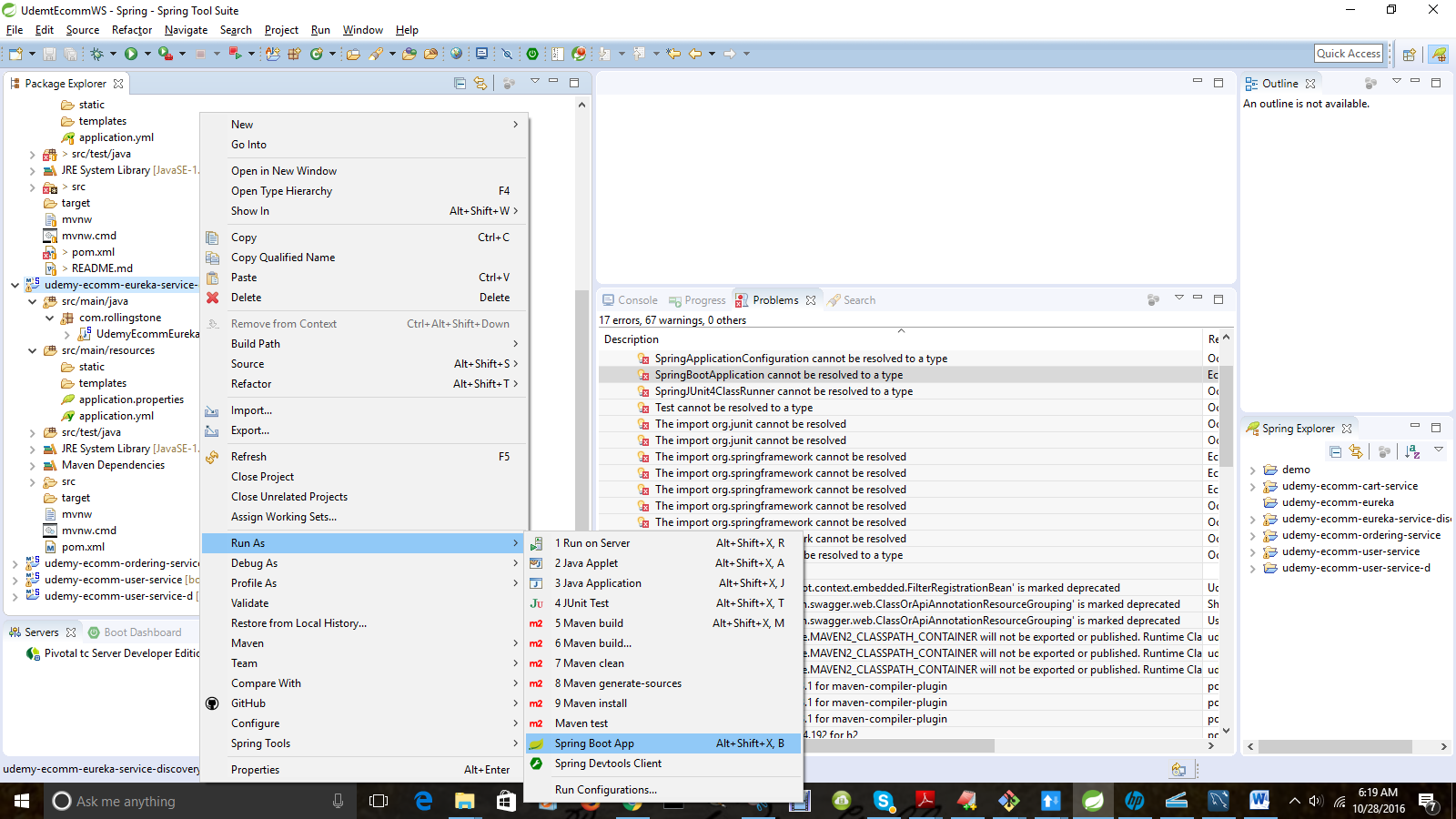
1.33– Add bootstrap.yml file in the resources folder

spring:

application:

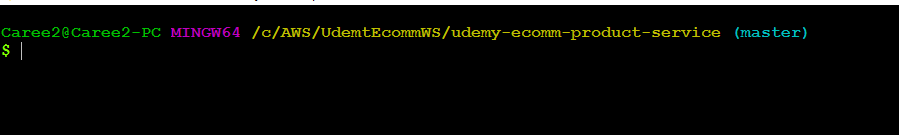
name: product-service

1.34– Run the Eureka Discovery Server



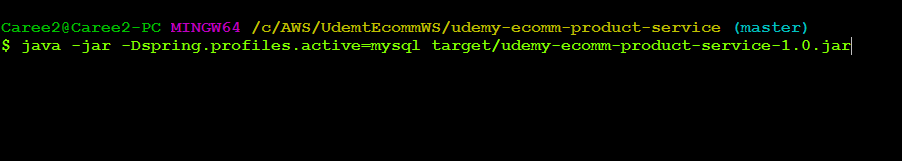


1.35– Open Git bash in the project directory

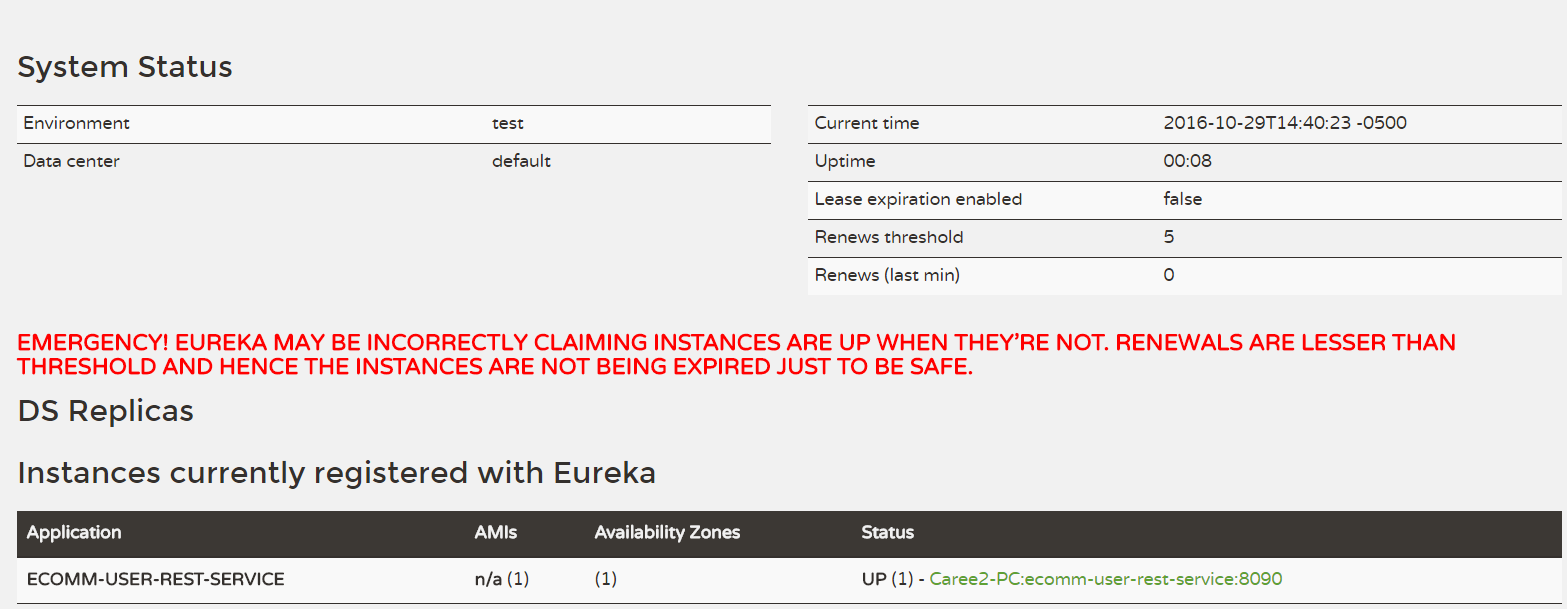


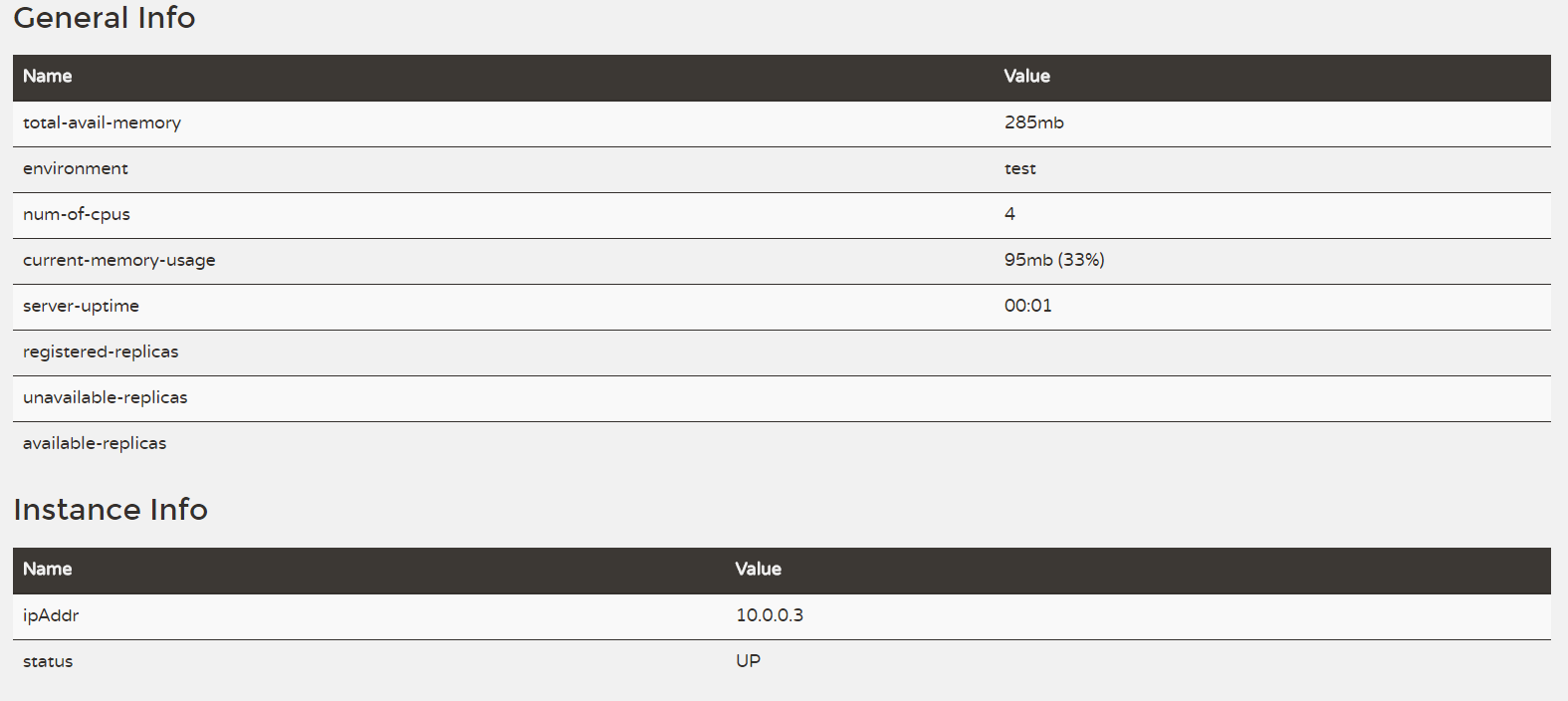
1.36– Run the project

NOTE : First run mvn clean package

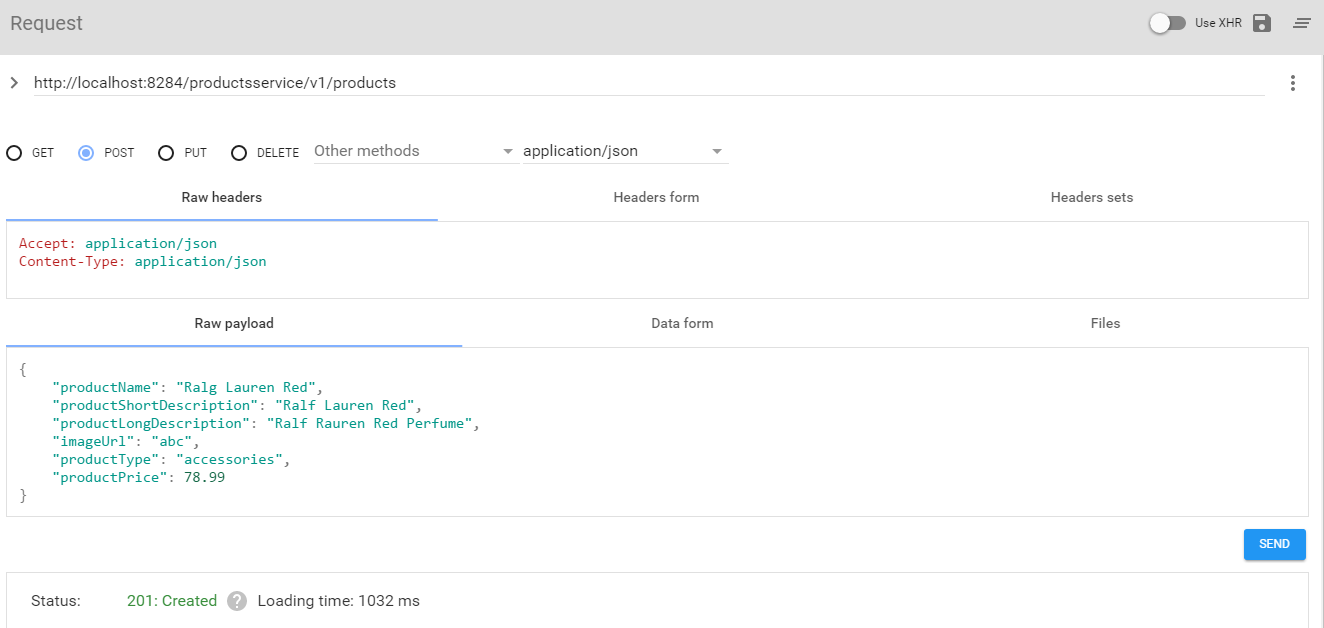


1.37 – Navigate to <http://localhost:8761>

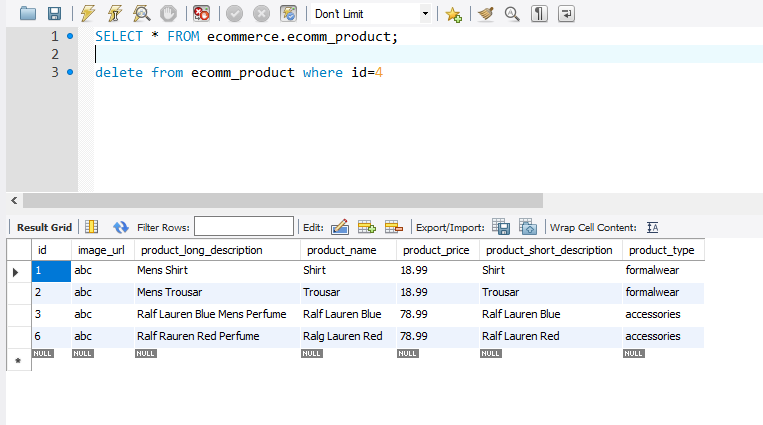




1.38– Create a Product



1.39– Verify in MySQL Workbench

















































































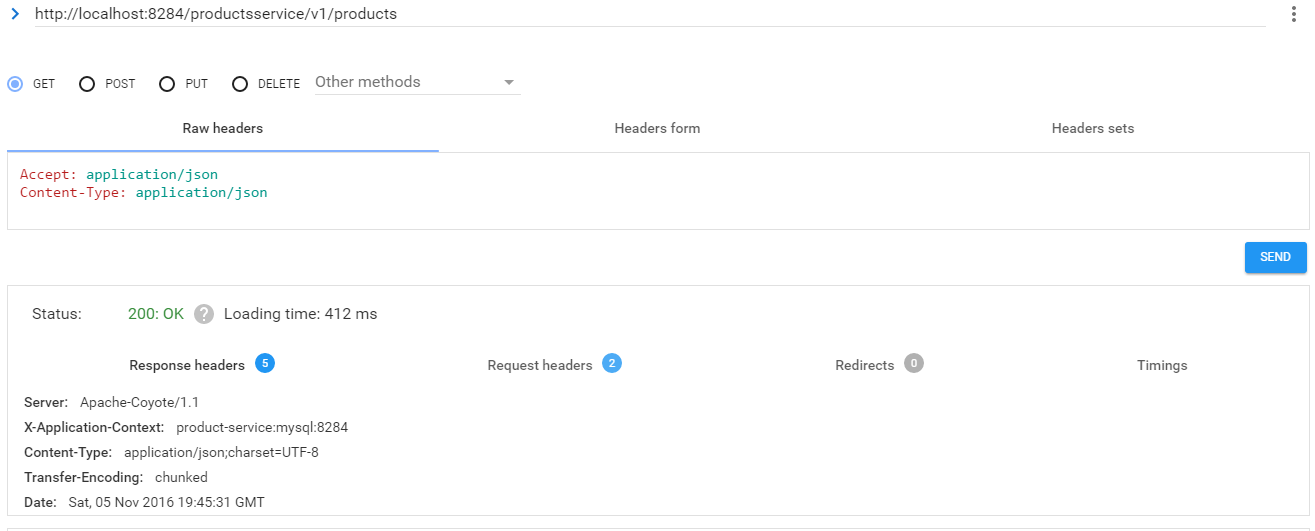


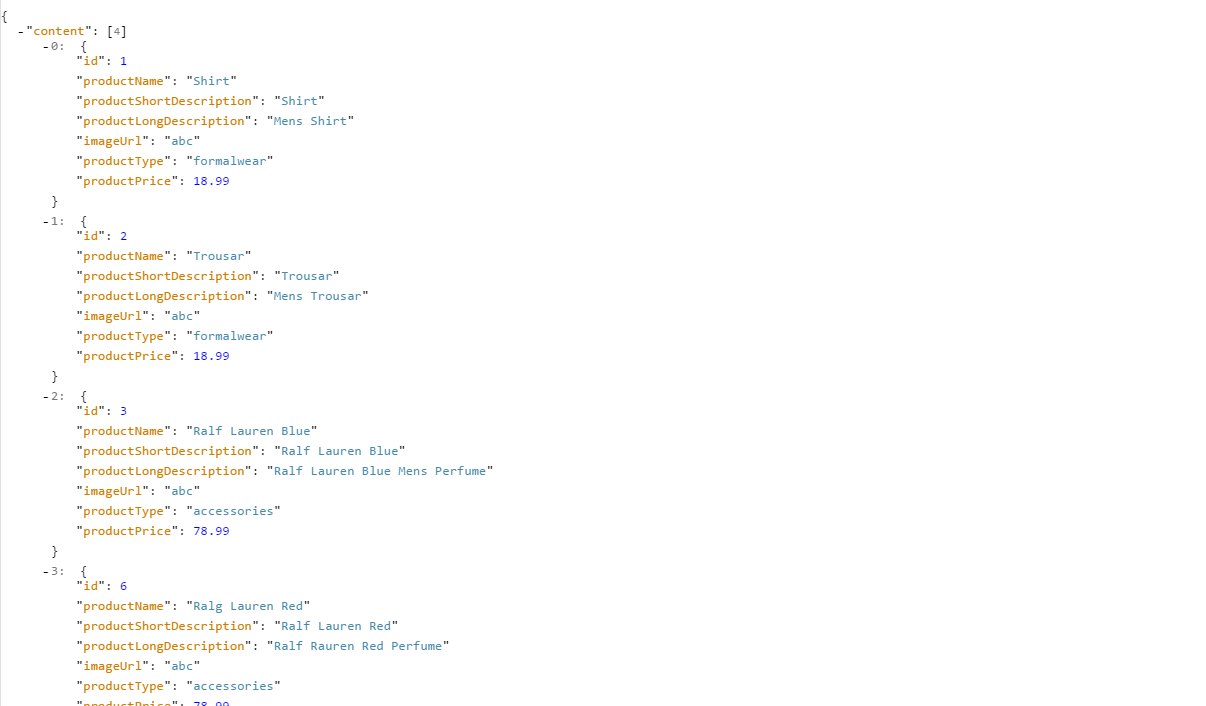




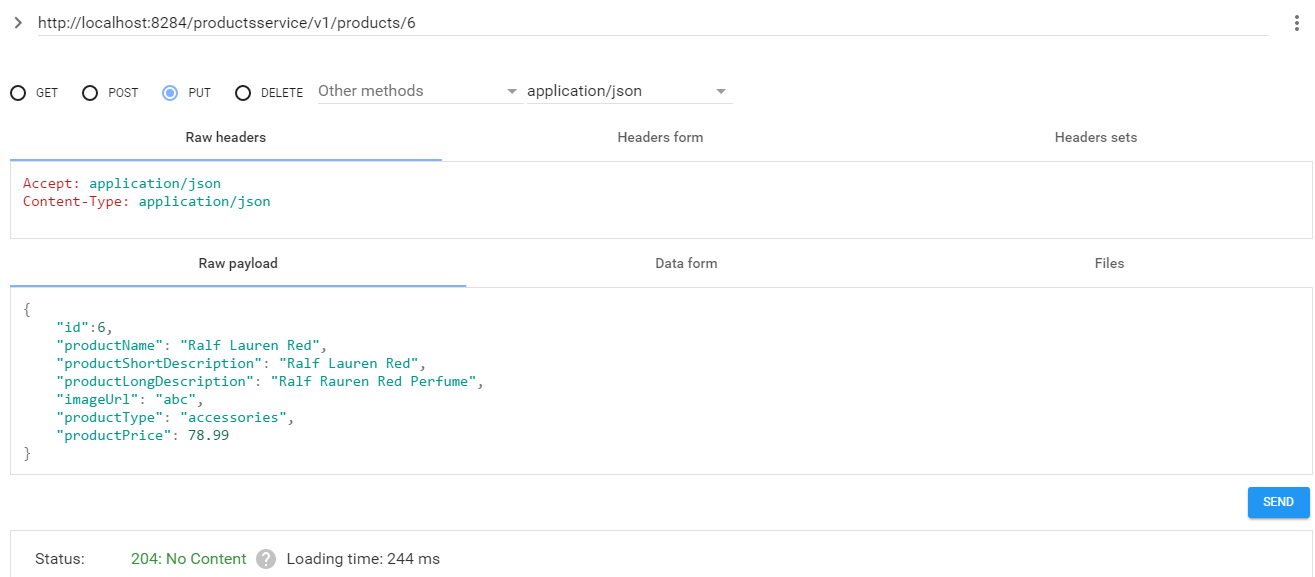


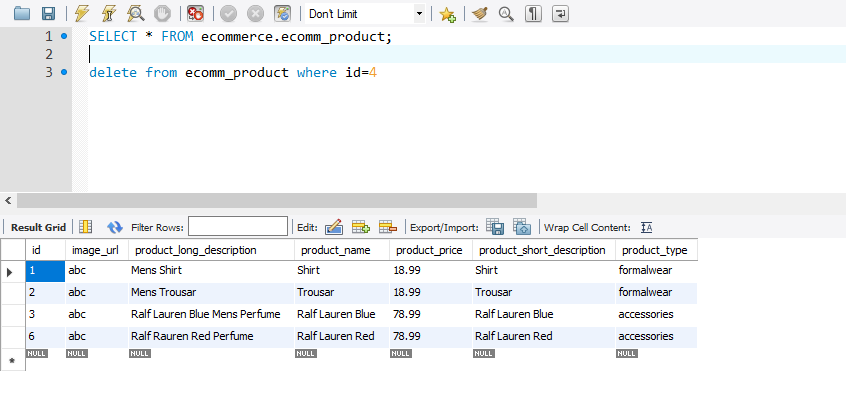
1.41—Try GET Request



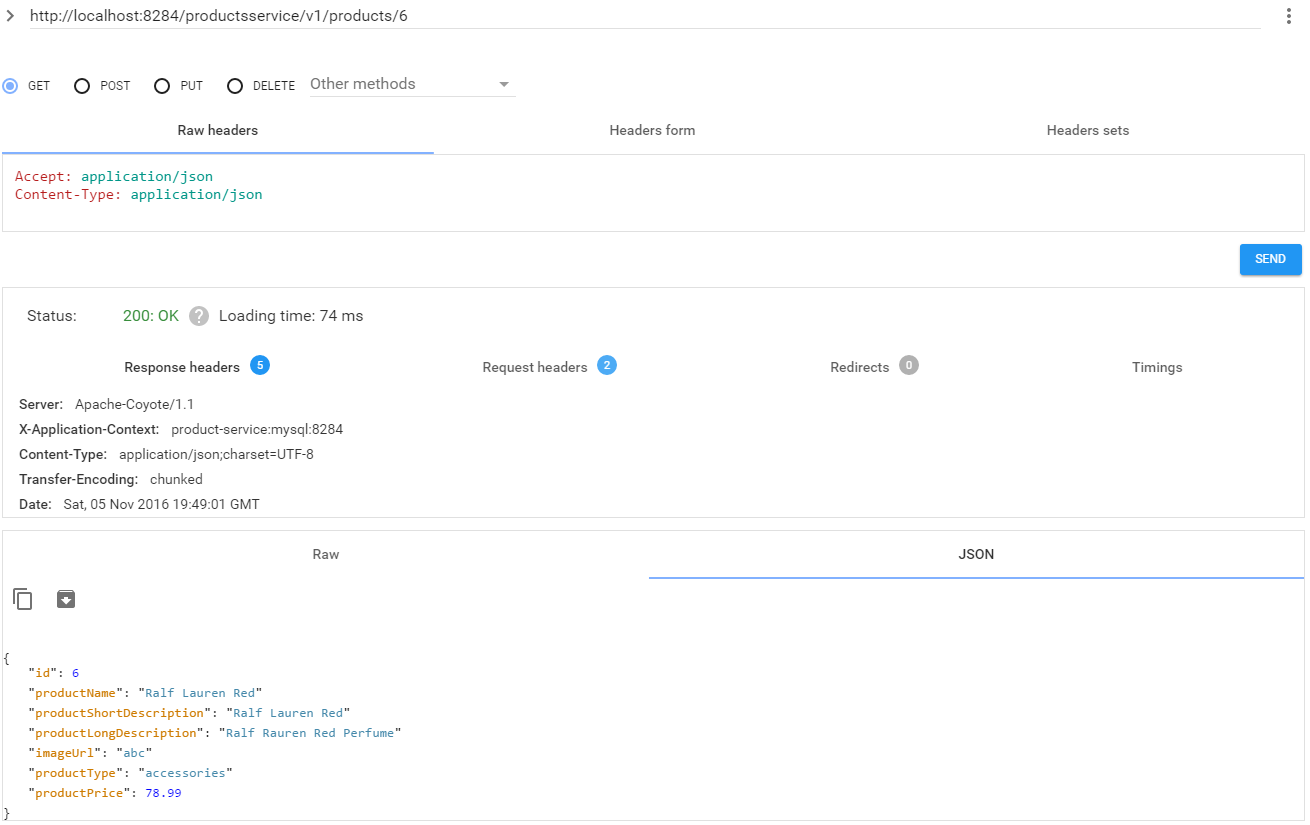


1.42—Try to Update a Record

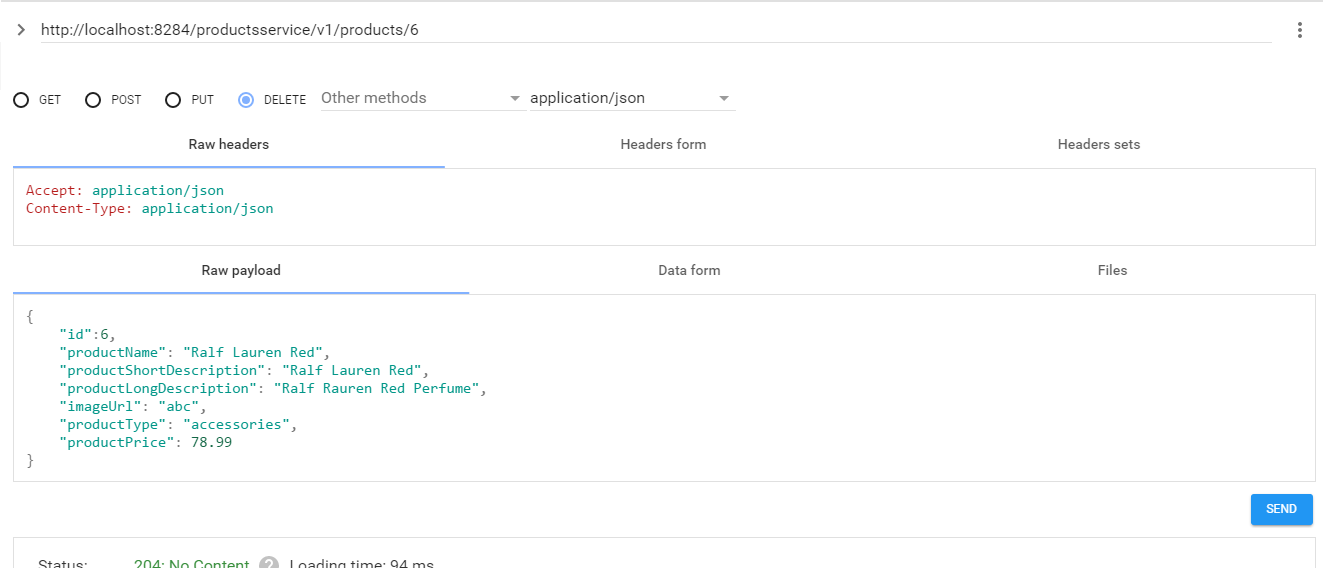


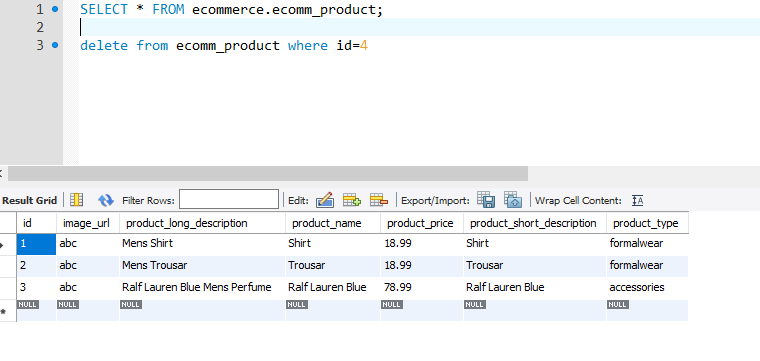


1.43 – Try to get a single product

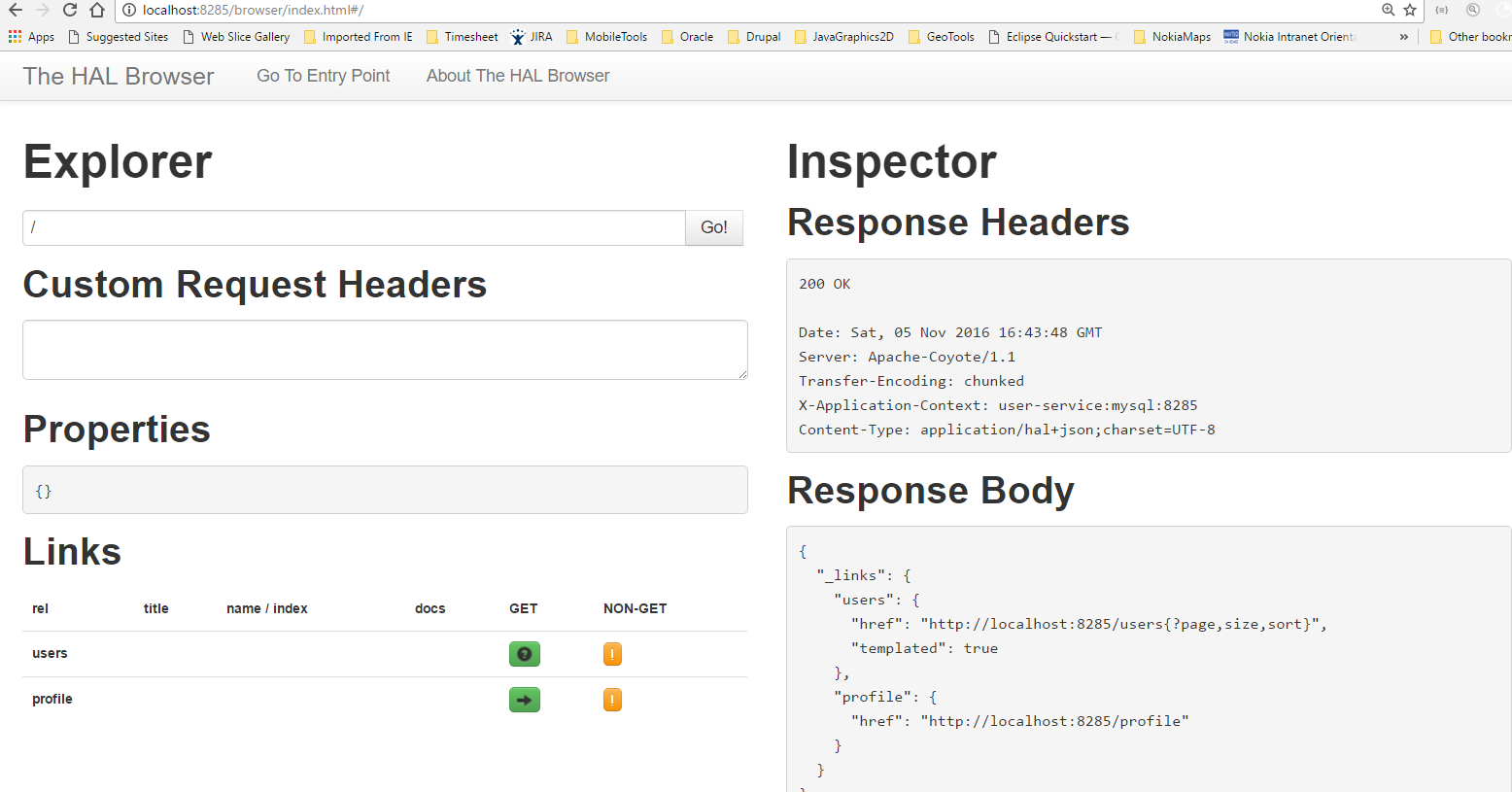


1.44 – Try to delete a single user

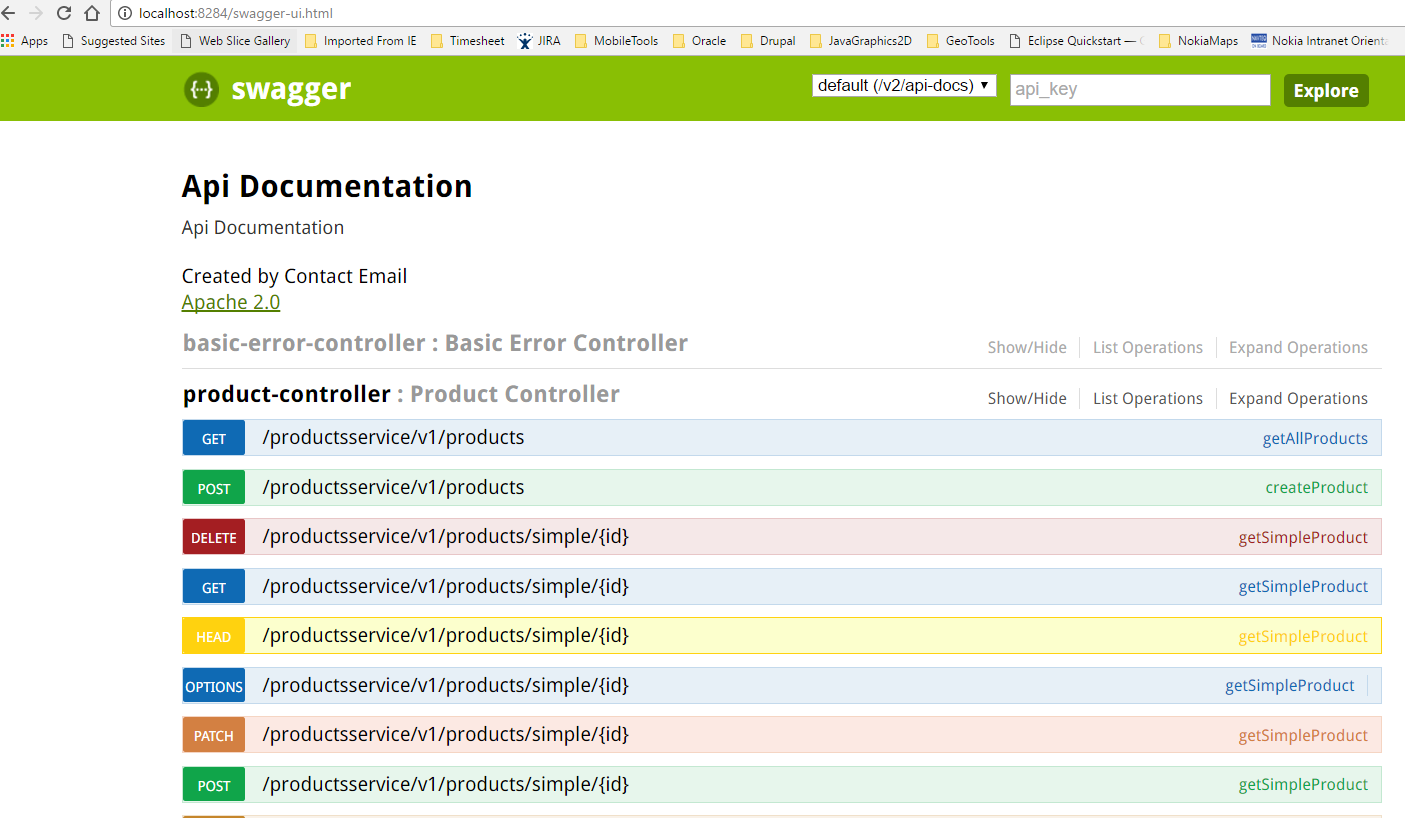




1.45 –HAL Browser



1.47 – Swagger UI



1.48 – Conclusion

This document listed the steps as well as provided the explanation of creating a Spring Boot application based on Spring Cloud Service Discovery as well as JPA.