

Lab 02 RSMortgage Spring Boot Person Service 🡪 REST Microservice

**RSMortgage Spring Boot Person Service**

Udemy Spring Cloud Course

Presented By

Binit Datta

**Rolling Stone Technology**

**Formatted: February, 2017**

**Table of Content**

1.0 - Introduction 5

1.1 – Create a new Spring Starter Project 6

1.2 – Fill initial values 7

1.3 – Do not choose anything 8

1.4 – Click Finish Now 29

1.5 – Let Spring Tool Suite Prepare the Project 30

1.6 – Verify the project identification in the pom.xml 31

1.7 – Add Parent project 31

1.8 – Add Properties 31

1.9– Add web and actuator Dependencies 32

1.10 – Add security and tomcat dependencies 32

1.11 – Add jpa orm and h2 db dependencies 32

1.12 – Add spring test dependencies 33

1.13 – Add Jackson databind dependencies 33

1.14 – Add HAL Browser dependencies 33

1.15 – Add Swagger dependencies 34

1.16 – Add MySQL Driver dependencies 34

1.17 – Add maven build section 34

1.18 –Add properties for application.properties resource directory 35

1.19 –Create application.yml under resource directory 35

1.20 –Create domain package 37

1.21 –Create api.rest package 38

1.22 –Create exception package 39

1.23 –Create dao.jpa package 40

1.24 –Create service package 41

1.25 –Create Person domain class in domain package 42

1.26 –Do the following for the Person domain class 43

1.27 –Create RESTAPIErrorInfo domain class 43

1.28 –Create HTTP400Exception class in the exception pckage 44

1.29 –Create HTTP404Exception class in the exception pckage 45

1.30 –Create PersonRepository interface in the dao.jpa package 46

1.31 –Create ServiceProperties class in the service package 47

1.32 –Create PersonServiceHealth class in the service package 48

1.33 –Create PersonServiceEvent class in the service package 49

1.34 –Create PersonServiceListener class in the service package 50

1.35 –Create PersonServiceListener class in the service package 51

1.36 –Create AbstractRestHandler class in the rest.api package 53

1.37 –Add PersonController in the rest.api package 55

1.38 –Add PersonController in the com.rollingstone package 58

1.39 –Add Application class in the com.rollingstone package 60

1.40 –Open Git Bash in project folder 61

1.41 –Run build 61

1.42 –Build successful 62

1.43 –Now Run 63

1.44 –Verify Database 64

1.45 –Verify Database Data 65

1.46 –Verify Data Retrieval API is working 66

1.47 –Verify Data AOP for @Before and @AfterReturning 67

1.47 –Verify Single Data Retrieval API is working 68

1.48 –Verify Not Found is working 69

1.49 –Verify Person Creation is working 70

1.49 –Verify Person Updation is working 72

1.50 –Verify Person Deletion is working 75

1.51 –Custom Health 76

1.54 – Metrics and Custom Metrics 77

1.55 – /env endpoint 78

1.56 – /beans endpoint 79

1.57 – /actuator endpoint 80

1.58 – /dump endpoint 82

1.59 – /logfile endpoint 83

1.60 –Conclusion 83

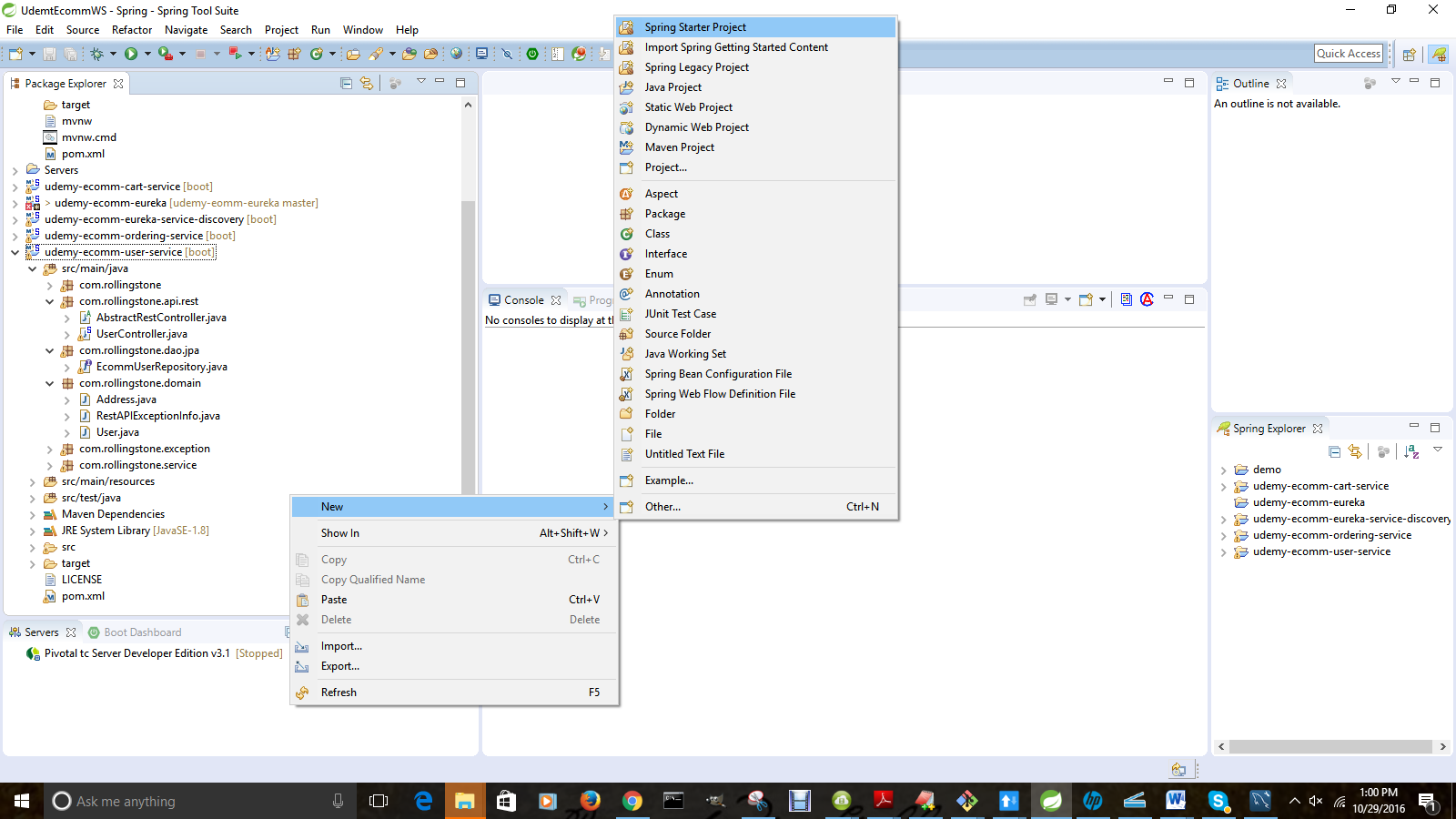
Chapter 1

*Spring Boot Person Service Project Creation*

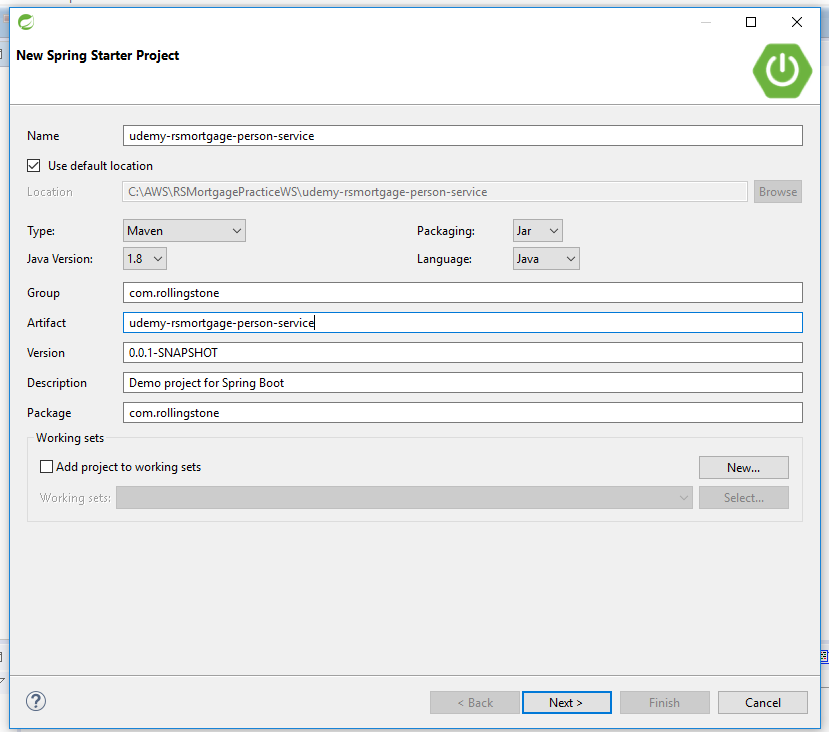
1.0 - Introduction

The purpose of this guide is to help walk through the process of creating a Spring Boot based Person Service REST Application so that the reader / learner becomes completely familiar with Spring Boot and its associated features such as actuator etc.

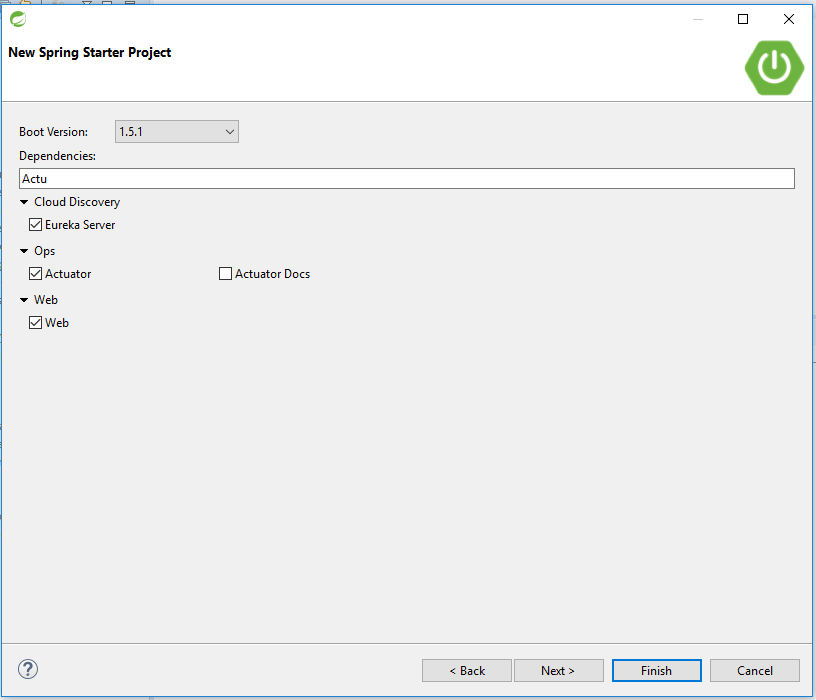
1.1 – Create a new Spring Starter Project



1.2 – Fill initial values

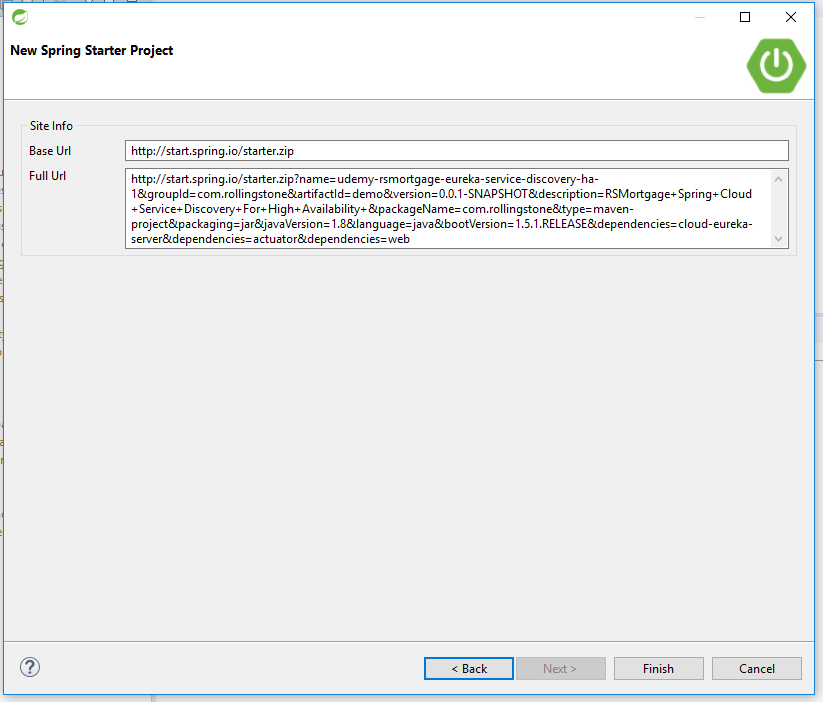


1.3 – Do not choose anything

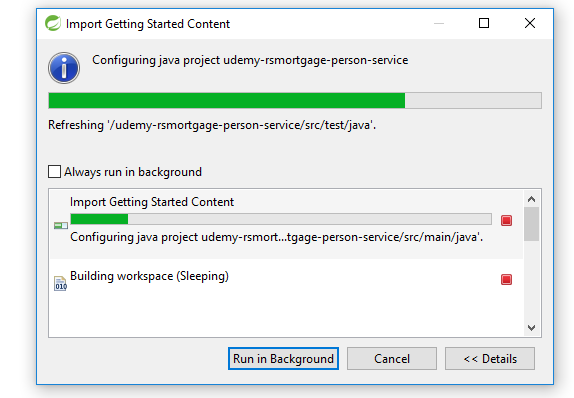


.

1.4 – Click Finish Now



1.5 – Let Spring Tool Suite Prepare the Project



1.6 – Verify the project identification in the pom.xml

<groupId>com.rollingstone</groupId>

<artifactId>udemy-rsmortgage-person-service</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>udemy-rsmortgage-person-service</name>

<description>Demo project for Spring Boot</description>

1.7 – Add Parent project

<parent>

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

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

<version>1.4.2.RELEASE</version>

</parent>

1.8 – Add Properties

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<java.version>1.8</java.version>

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

<!-- <slf4j.version>1.6.4</slf4j.version>

<logback.version>1.0.1</logback.version> -->

</properties>

1.9– Add web and actuator 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>

1.10 – Add security and tomcat dependencies

<!-- 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>

1.11 – Add jpa orm and h2 db dependencies

<!-- 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>

<dependency>

<groupId>org.hsqldb</groupId>

<artifactId>hsqldb</artifactId>

<scope>runtime</scope>

</dependency>

1.12 – Add spring test dependencies

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

<dependency>

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

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

<scope>test</scope>

</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>

1.13 – Add Jackson databind dependencies

<dependency>

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

<artifactId>jackson-databind</artifactId>

</dependency>

1.14 – Add HAL Browser dependencies

<dependency>

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

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

</dependency>

1.15 – Add Swagger dependencies

<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>

1.16 – Add MySQL Driver dependencies

<dependency>

<groupId>mysql</groupId>

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

<version>5.1.40</version>

</dependency>

1.17 – Add maven build section

<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>

1.18 –Add properties for application.properties resource directory

logging.level.org.springframework.web=INFO

logging.level.guru.springframework.blogs.controllers=INFO

logging.level.org.hibernate=ERROR

logging.file=logs/spring-boot-person-service-logging.log

1.19 –Create application.yml under resource directory

spring.jmx:

enabled: **false**

spring.datasource:

driverClassName: org.h2.Driver

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

server:

port: 8090

#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: udemy-rsmortgage-person-service

version: 0.1

description: udemy-rsmortgage-person-service

info:

build:

artifact: ${project.artifactId}

name: ${project.name}

description: ${project.description}

version: ${project.version}

spring.jpa:

hibernate.ddl-auto: create-drop

---

spring:

profiles: mysql

datasource:

driverClassName: com.mysql.jdbc.Driver

url: jdbc:mysql://localhost/rsmortgage

username: root

password: root

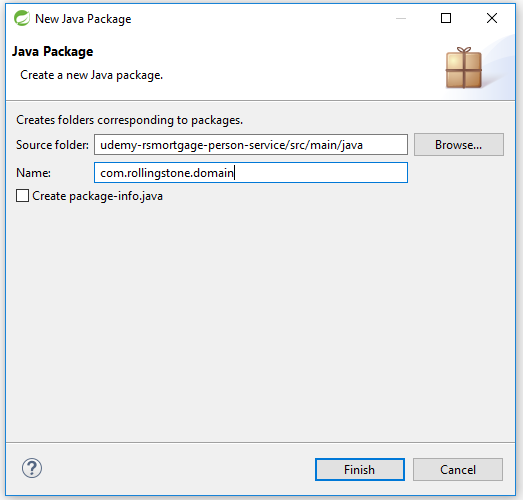
jpa:

hibernate:

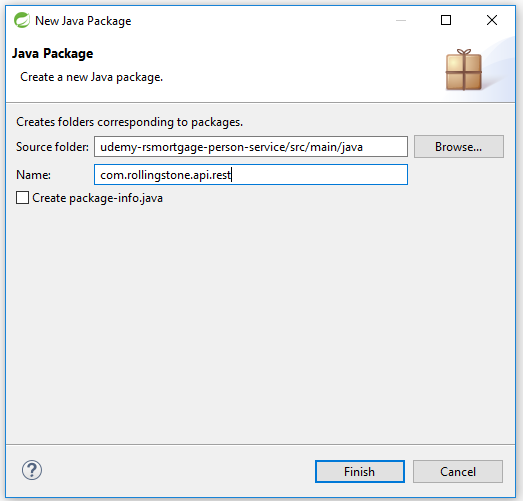
dialect: org.hibernate.dialect.MySQLInnoDBDialect

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

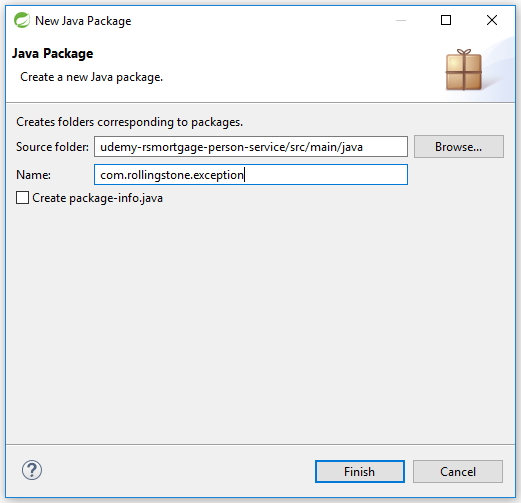
1.20 –Create domain package



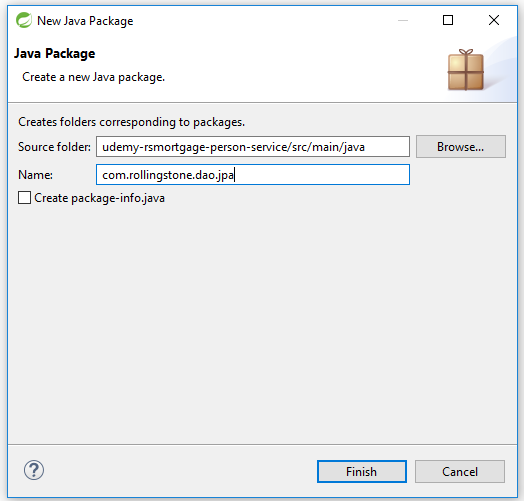
1.21 –Create api.rest package



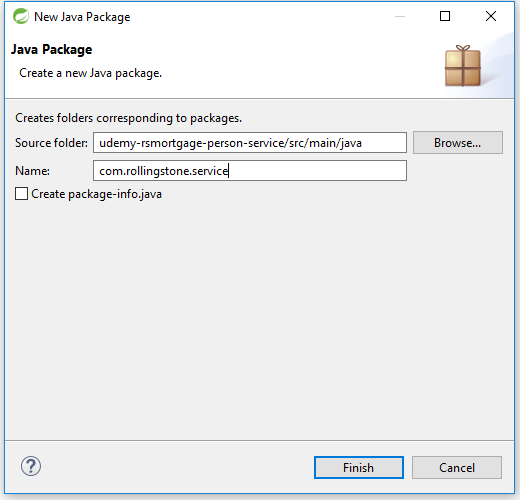
1.22 –Create exception package



1.23 –Create dao.jpa package



1.24 –Create service package



1.25 –Create Person domain class in domain package

**package** com.rollingstone.domain;

**import** java.util.Date;

**import** javax.persistence.\*;

**import** javax.xml.bind.annotation.\*;

/\*

\* a simple domain entity doubling as a DTO

\*/

@Entity

@Table(name = "rsmortgage\_person")

@XmlRootElement

@XmlAccessorType(XmlAccessType.***FIELD***)

**public** **class** Person {

@Id

@GeneratedValue()

**private** **long** id;

@Column(nullable = **false**)

**private** String firstName;

@Column(nullable = **false**)

**private** String lastName;

@Temporal(TemporalType.***DATE***)

@Column(name = "dob", unique = **true**, nullable = **false**, length = 10)

**private** Date dateOfBirth;

@Column(nullable = **false**)

**private** **int** age;

@Column(nullable = **false**)

**private** String socialSecurityNumber;

**public** Person() {

}

}

1.26 –Do the following for the Person domain class

* Generate a non-default constructor using all fields
* Generate a hashCode methos
* Generate an equals method
* Generate a toString method using all fields
* Generate getter and setters for all fields

1.27 –Create RESTAPIErrorInfo domain class

**package** com.rollingstone.domain;

**import** javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement

**public** **class** RESTAPIErrorInfo {

**public** **final** String errorDetail;

**public** **final** String errorMessage;

**public** RESTAPIErrorInfo(Exception ex, String detail) {

**this**.errorMessage = ex.getLocalizedMessage();

**this**.errorDetail = detail;

}

}

1.28 –Create HTTP400Exception class in the exception pckage

**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);

}

}

1.29 –Create HTTP404Exception class in the exception pckage

**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);

}

}

1.30 –Create PersonRepository interface in the dao.jpa package

package com.rollingstone.dao.jpa;

import java.util.List;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.Pageable;

import org.springframework.data.repository.PagingAndSortingRepository;

import com.rollingstone.domain.Person;

public interface PersonRepository extends PagingAndSortingRepository<Person, Long> {

Person findPersopnByage(int age);

List<Person> findPersonByLastName(String lastName);

Page findAll(Pageable pageable);

}

1.31 –Create ServiceProperties 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 = "person.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;

}

}

1.32 –Create PersonServiceHealth 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 PersonServiceHealth implements HealthIndicator {

@Autowired

private ServiceProperties configuration;

// extend this to create an application-specific health check according to http://goo.gl/vt8I7O

@Override

public Health health() {

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

}

}

1.33 –Create PersonServiceEvent class in the service package

package com.rollingstone.service;

import org.springframework.context.ApplicationEvent;

import com.rollingstone.domain.Person;

/\*\*

\* 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 PersonServiceEvent extends ApplicationEvent {

Person eventPerson;

String eventType;

public PersonServiceEvent(Object source, String eventType, Person person) {

super(source);

this.eventType = eventType;

this.eventPerson = person;

}

public String toString() {

return "My PErsonService Event";

}

public Person getEventPerson() {

return eventPerson;

}

public void setEventPerson(Person eventPerson) {

this.eventPerson = eventPerson;

}

public String getEventType() {

return eventType;

}

public void setEventType(String eventType) {

this.eventType = eventType;

}

}

1.34 –Create PersonServiceListener class in the service package

package com.rollingstone.service;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.context.ApplicationListener;

import org.springframework.stereotype.Component;

@Component

public class PersonEventListener implements ApplicationListener<PersonServiceEvent>

{

private final Logger logger = LoggerFactory.getLogger(this.getClass());

public void onApplicationEvent(PersonServiceEvent event)

{

PersonServiceEvent personEvent = (PersonServiceEvent) event;

logger.info("Person " + event.getEventType() + " with details : " + event.getEventPerson());

System.out.println("Person " + event.getEventType() + " with details : " + event.getEventPerson());

}

}

1.35 –Create PersonServiceListener class in the service package

package com.rollingstone.service;

import com.rollingstone.dao.jpa.PersonRepository;

import com.rollingstone.domain.Person;

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;

@Service

public class PersonService {

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

@Autowired

private PersonRepository personRepository;

@Autowired

CounterService counterService;

@Autowired

GaugeService gaugeService;

public PersonService() {

}

public Person createPerson(Person person) {

return personRepository.save(person);

}

public Person getPerson(long id) {

return personRepository.findOne(id);

}

public void updatePerson(Person person) {

personRepository.save(person);

}

public void deletePerson(Long id) {

personRepository.delete(id);

}

public Page<Person> getAllPersons(Integer page, Integer size) {

Page pageOfPersons = personRepository.findAll(new PageRequest(page, size));

// example of adding to the /metrics

if (size > 50) {

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

}

return pageOfPersons;

}

}

1.36 –Create AbstractRestHandler class in the rest.api package

package com.rollingstone.api.rest;

import javax.servlet.http.HttpServletResponse;

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.context.ApplicationEventPublisher;

import org.springframework.context.ApplicationEventPublisherAware;

import org.springframework.http.HttpStatus;

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

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

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

import org.springframework.web.context.request.WebRequest;

import com.rollingstone.domain.RESTAPIErrorInfo;

import com.rollingstone.exception.HTTP400Exception;

import com.rollingstone.exception.HTTP404Exception;

/\*\*

\* This class is used as a super class to be extended by all REST API "controllers".

\*/

public abstract class AbstractRestHandler implements ApplicationEventPublisherAware {

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

protected ApplicationEventPublisher eventPublisher;

protected static final String DEFAULT\_PAGE\_SIZE = "100";

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

@Autowired

CounterService counterService;

@ResponseStatus(HttpStatus.BAD\_REQUEST)

@ExceptionHandler(HTTP400Exception.class)

public

@ResponseBody

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

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

counterService.increment("com.rollingstone.http.400.count");

return new RESTAPIErrorInfo(ex, "You messed up.");

}

@ResponseStatus(HttpStatus.NOT\_FOUND)

@ExceptionHandler(HTTP404Exception.class)

public

@ResponseBody

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

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

counterService.increment("com.rollingstone.http.404.count");

return new RESTAPIErrorInfo(ex, "Sorry I couldn't find it.");

}

@Override

public void setApplicationEventPublisher(ApplicationEventPublisher applicationEventPublisher) {

this.eventPublisher = applicationEventPublisher;

}

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

if (resource == null) {

throw new HTTP404Exception("resource not found");

}

return resource;

}

}

1.37 –Add PersonController in the rest.api package

package com.rollingstone.api.rest;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

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.Person;

import com.rollingstone.exception.HTTP404Exception;

import com.rollingstone.service.PersonService;

import com.rollingstone.service.PersonServiceEvent;

@RestController

@RequestMapping(value = "/rsmortgage-personservice/v1/person")

public class PersonController extends AbstractRestHandler {

@Autowired

private PersonService personService;

@RequestMapping(value = "",

method = RequestMethod.POST,

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

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

@ResponseStatus(HttpStatus.CREATED)

public void createPerson(@RequestBody Person person,

HttpServletRequest request, HttpServletResponse response) {

Person createdPerson = this.personService.createPerson(person);

eventPublisher.publishEvent(new PersonServiceEvent(this,"PersonCreated",createdPerson));

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

}

@RequestMapping(value = "",

method = RequestMethod.GET,

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

@ResponseStatus(HttpStatus.OK)

public

@ResponseBody

Page<Person> getAllPerson(@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.personService.getAllPersons(page, size);

}

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

method = RequestMethod.GET,

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

@ResponseStatus(HttpStatus.OK)

public

@ResponseBody

Person getPerson(@PathVariable("id") Long id,

HttpServletRequest request, HttpServletResponse response) throws Exception {

Person person = this.personService.getPerson(id);

checkResourceFound(person);

return person;

}

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

method = RequestMethod.PUT,

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

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

@ResponseStatus(HttpStatus.NO\_CONTENT)

public void updatePerson(@PathVariable("id") Long id, @RequestBody Person person,

HttpServletRequest request, HttpServletResponse response) {

checkResourceFound(this.personService.getPerson(id));

if (id != person.getId()) throw new HTTP404Exception("ID doesn't match!");

this.personService.updatePerson(person);

}

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

method = RequestMethod.DELETE,

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

@ResponseStatus(HttpStatus.NO\_CONTENT)

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

HttpServletResponse response) {

checkResourceFound(this.personService.getPerson(id));

this.personService.deletePerson(id);

}

}

1.38 –Add PersonController in the com.rollingstone package

package com.rollingstone;

import java.util.NoSuchElementException;

import org.aspectj.lang.JoinPoint;

import org.aspectj.lang.annotation.AfterReturning;

import org.aspectj.lang.annotation.AfterThrowing;

import org.aspectj.lang.annotation.Aspect;

import org.aspectj.lang.annotation.Before;

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.stereotype.Component;

@Aspect

@Component

public class RestControllerAspect {

private final Logger logger = LoggerFactory.getLogger(this.getClass());

@Autowired

CounterService counterService;

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

public void logBeforeRestCall(JoinPoint pjp) throws Throwable {

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

}

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

public void afterCallingCreatePerson(JoinPoint pjp) {

logger.info(":::::AOP @AfterReturning Create REST call:::::" + pjp);

counterService.increment("com.rollingstone.api.rest.PersonController.createPerson");

}

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

public void afterCallinggetAllPerson(JoinPoint pjp) {

logger.info(":::::AOP @AfterReturning getAllPerson REST call:::::" + pjp);

counterService.increment("com.rollingstone.api.rest.PersonController.getAllPerson");

}

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

public void afterCallinggetPerson(JoinPoint pjp) {

logger.info(":::::AOP @AfterReturning getPErson REST call:::::" + pjp);

counterService.increment("com.rollingstone.api.rest.PersonController.getPerson");

}

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

public void afterCallingupdatePerson(JoinPoint pjp) {

logger.info(":::::AOP @AfterReturning update PErson REST call:::::" + pjp);

counterService.increment("com.rollingstone.api.rest.PersonController.updatePerson");

}

@AfterThrowing(pointcut = "execution(public \* com.rollingstone.api.rest.\*Controller.\*(..))", throwing = "e")

public void afterGetGreetingThrowsException(NoSuchElementException e) {

counterService.increment("counter.errors.person.controller");

}

}

1.39 –Add Application class in the com.rollingstone package

package com.rollingstone;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.EnableAutoConfiguration;

import org.springframework.boot.builder.SpringApplicationBuilder;

import org.springframework.boot.context.web.SpringBootServletInitializer;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.data.jpa.repository.config.EnableJpaRepositories;

/\*

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

\*/

@EnableAutoConfiguration // Sprint Boot Auto Configuration

@ComponentScan(basePackages = "com.rollingstone")

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

public class Application extends SpringBootServletInitializer {

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

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

public static void main(String[] args) {

SpringApplication.run(applicationClass, args);

}

@Override

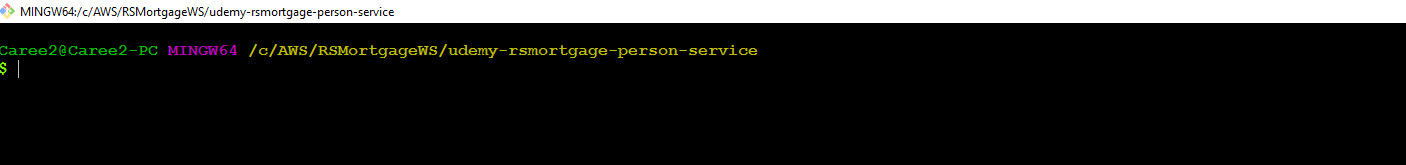
protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {

return application.sources(applicationClass);

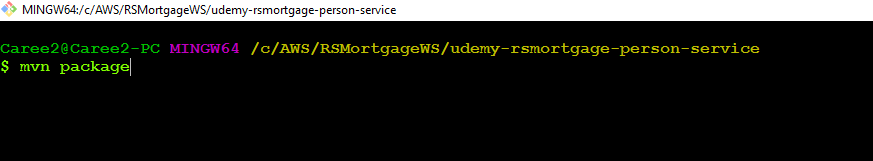
}

}

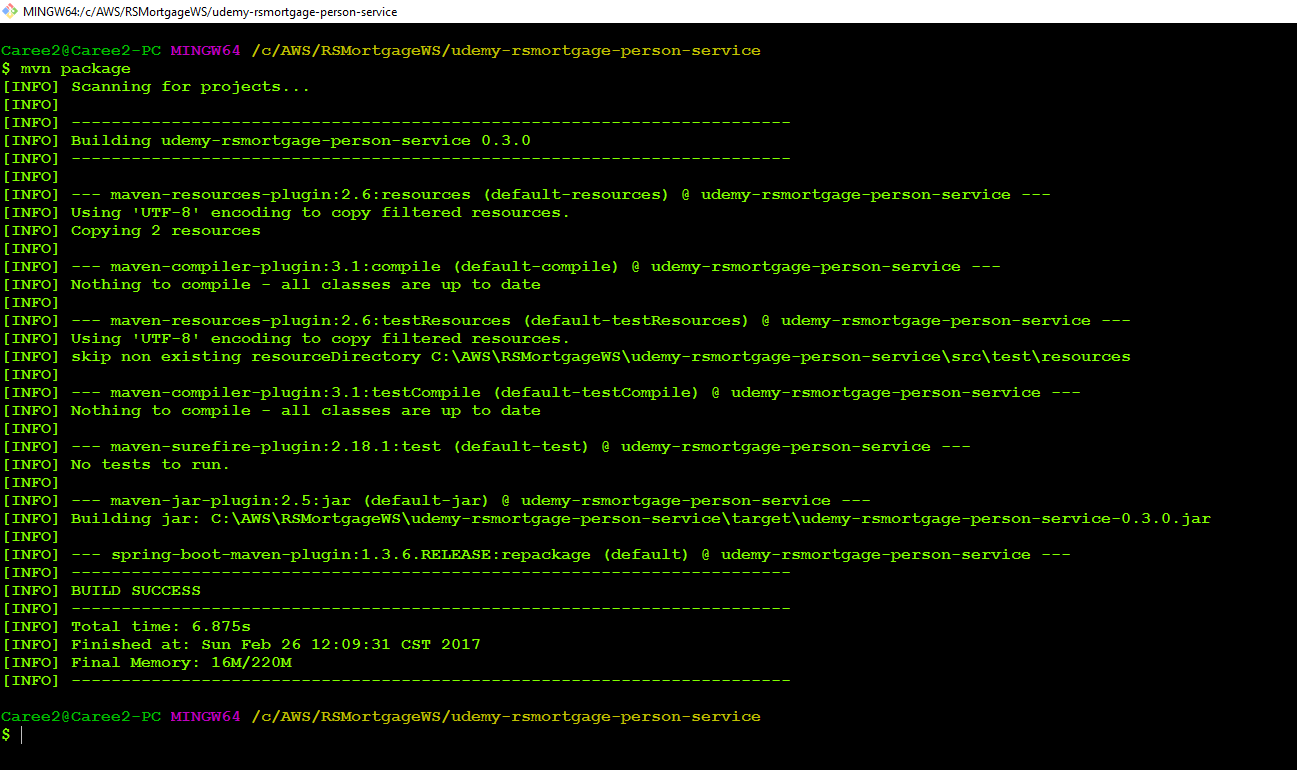
1.40 –Open Git Bash in project folder



1.41 –Run build

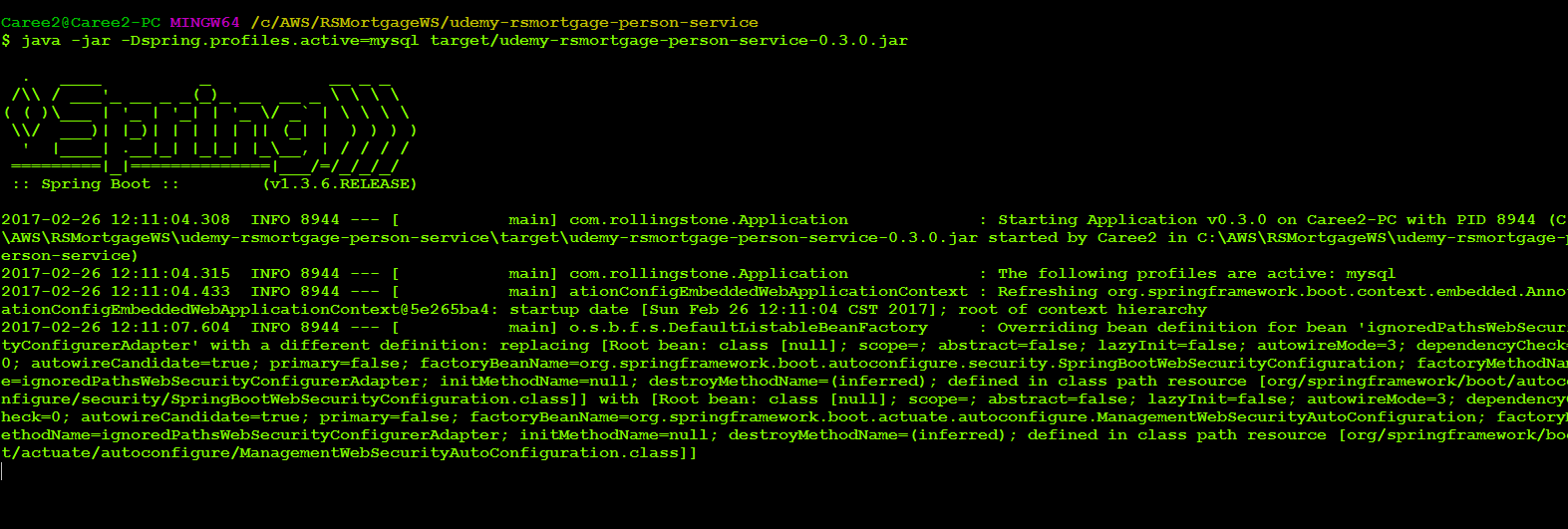


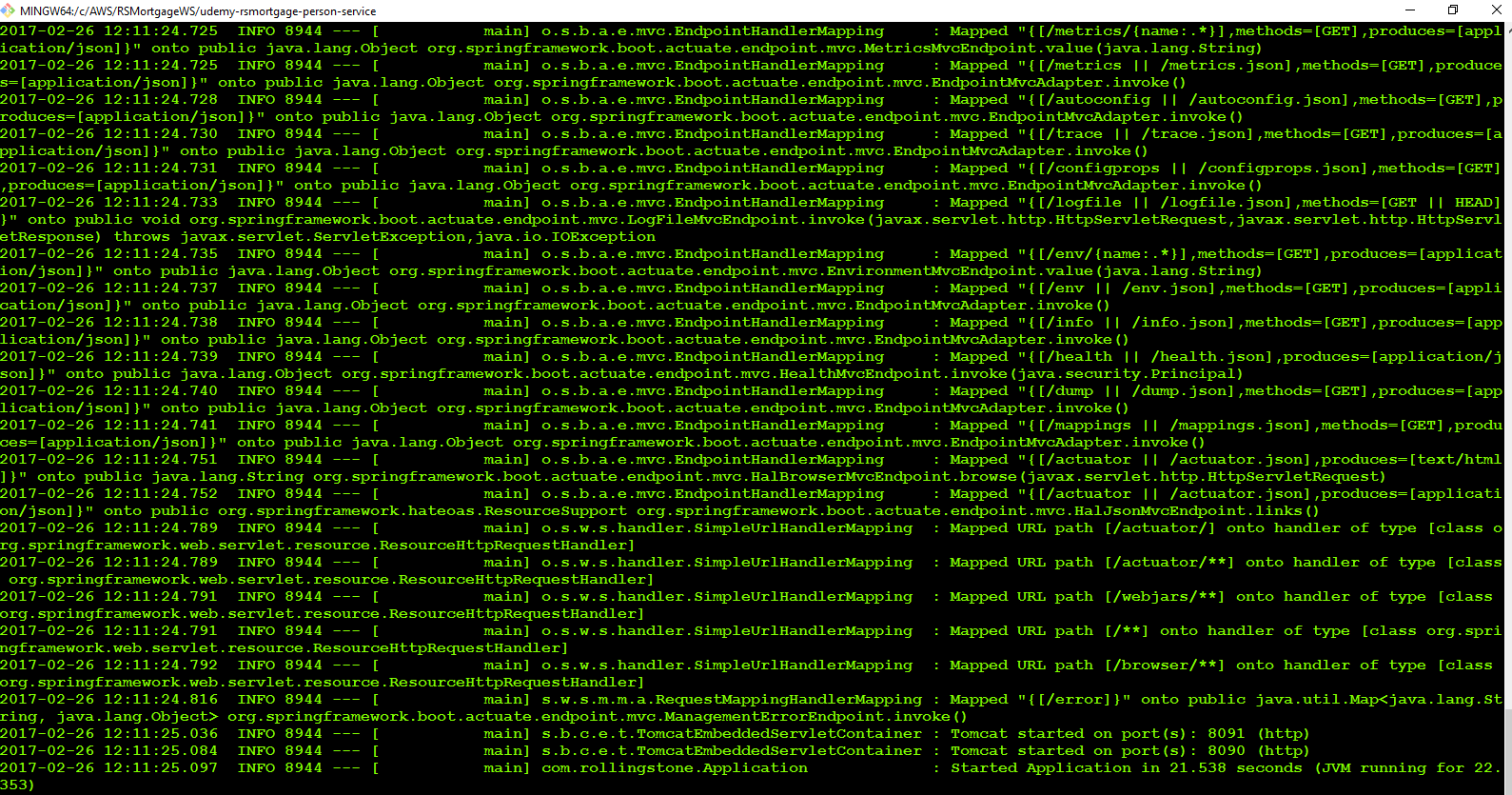
1.42 –Build successful



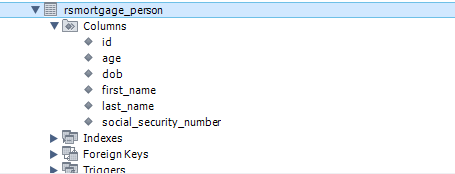
1.43 –Now Run

java -jar -Dspring.profiles.active=mysql target/udemy-rsmortgage-person-service-0.3.0.jar

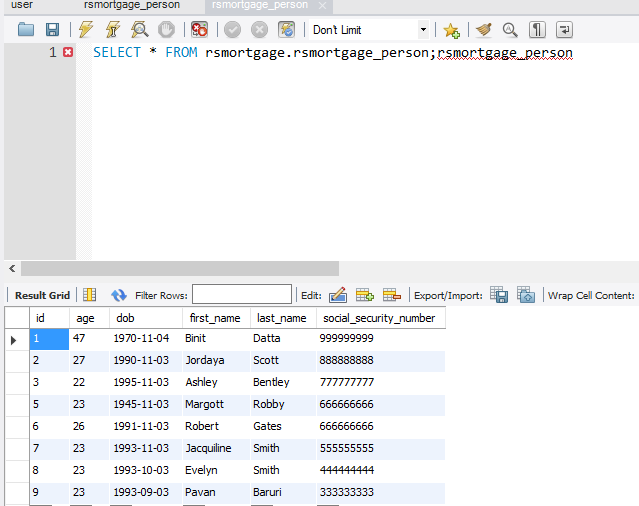




1.44 –Verify Database

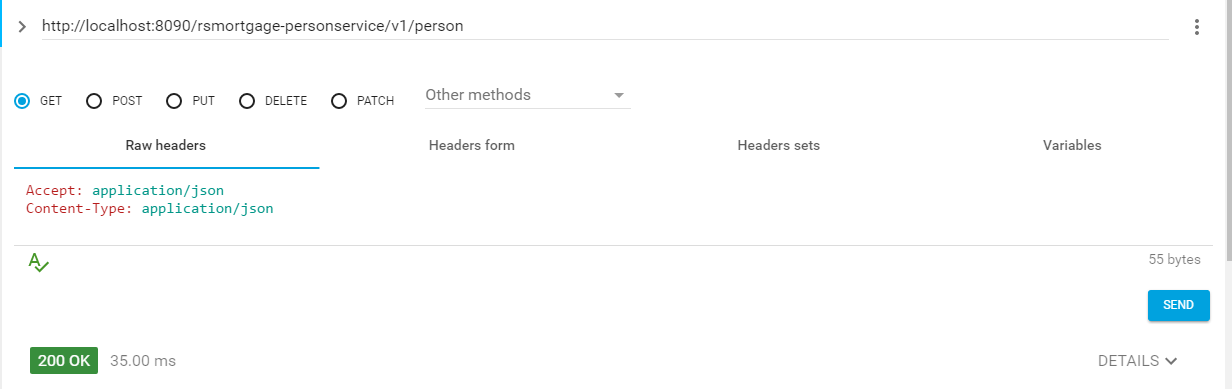


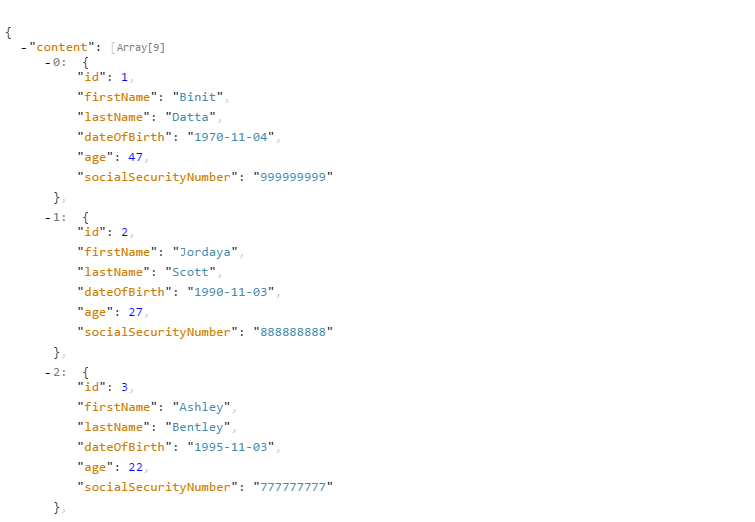
1.45 –Verify Database Data



1.46 –Verify Data Retrieval API is working

<http://localhost:8090/rsmortgage-personservice/v1/person>

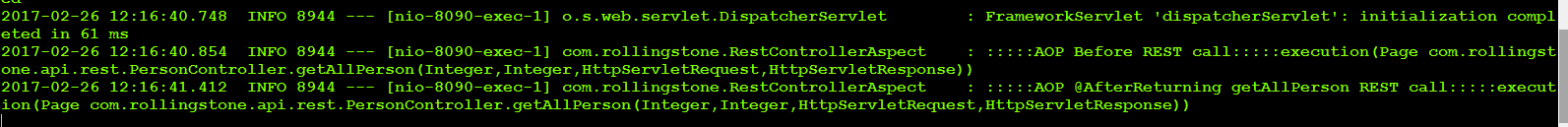




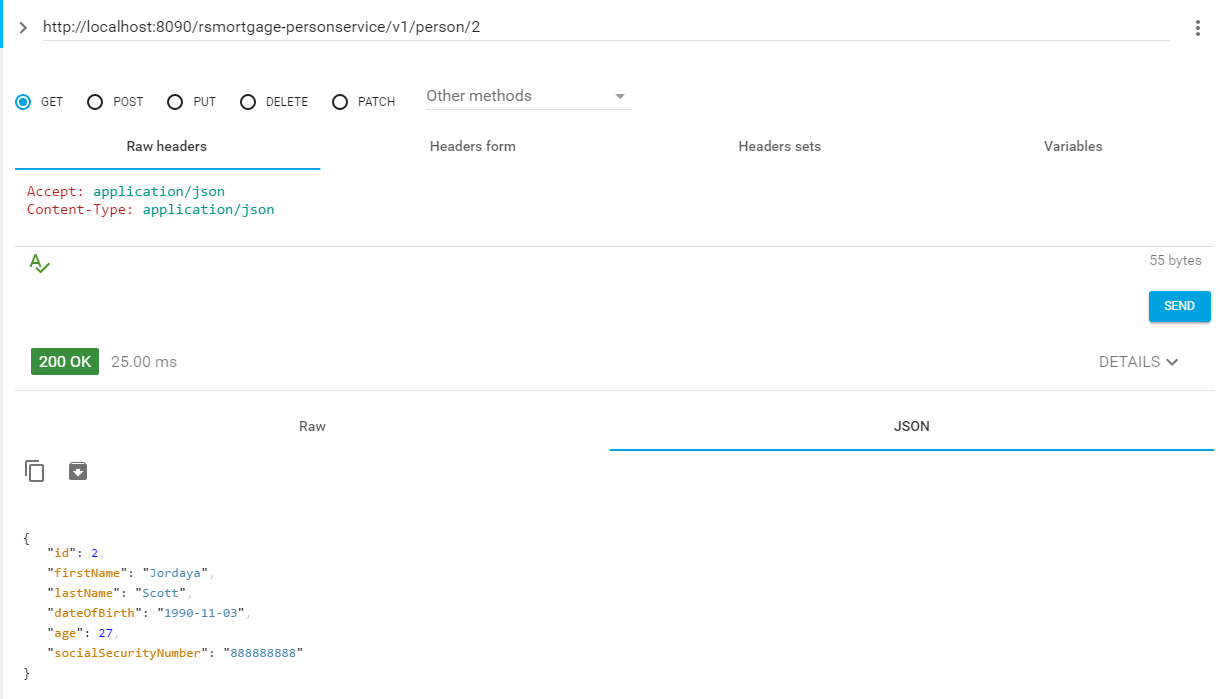
1.47 –Verify Data AOP for @Before and @AfterReturning

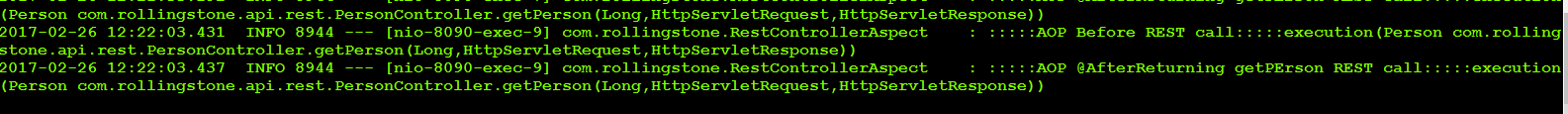
2017-02-26 12:16:40.854 INFO 8944 --- [nio-8090-exec-1] com.rollingstone.RestControllerAspect : :::::AOP Before REST call:::::execution(Page com.rollingstone.api.rest.PersonController.getAllPerson(Integer,Integer,HttpServletRequest,HttpServletResponse))

2017-02-26 12:16:41.412 INFO 8944 --- [nio-8090-exec-1] com.rollingstone.RestControllerAspect : :::::AOP @AfterReturning getAllPerson REST call:::::execution(Page com.rollingstone.api.rest.PersonController.getAllPerson(Integer,Integer,HttpServletRequest,HttpServletResponse))



1.47 –Verify Single Data Retrieval API is working





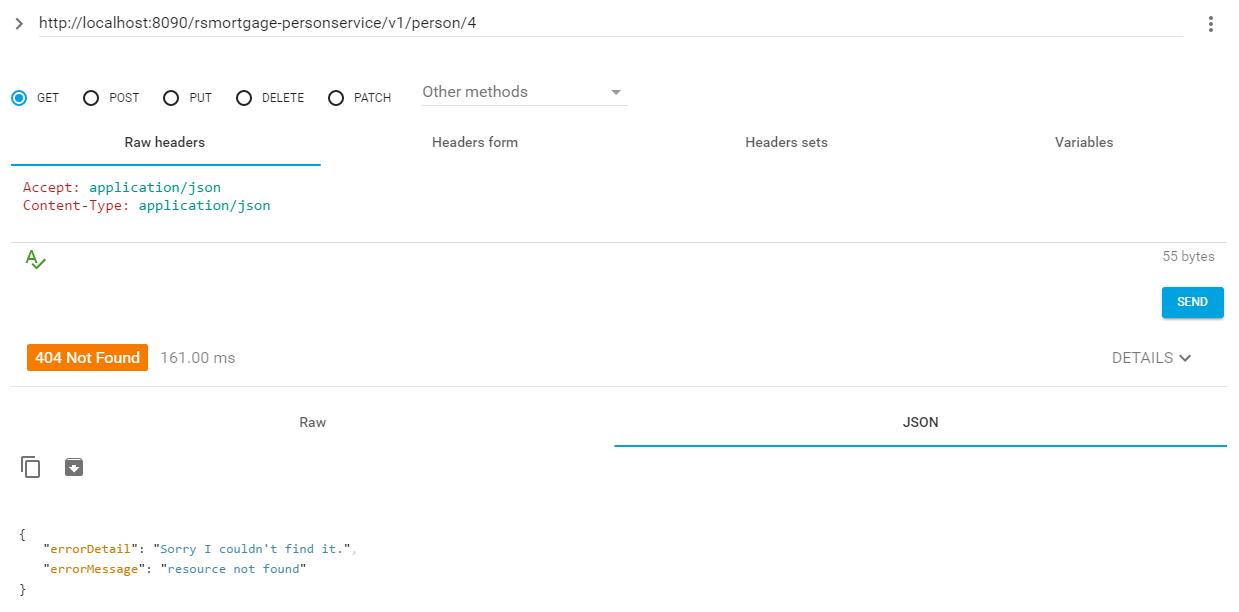
2017-02-26 12:22:03.431 INFO 8944 --- [nio-8090-exec-9] com.rollingstone.RestControllerAspect : :::::AOP Before REST call:::::execution(Person com.rollingstone.api.rest.PersonController.getPerson(Long,HttpServletRequest,HttpServletResponse))

2017-02-26 12:22:03.437 INFO 8944 --- [nio-8090-exec-9] com.rollingstone.RestControllerAspect : :::::AOP @AfterReturning getPErson REST call:::::execution(Person com.rollingstone.api.rest.PersonController.getPerson(Long,HttpServletRequest,HttpServletResponse))

1.48 –Verify Not Found is working

2017-02-26 12:19:33.435 INFO 8944 --- [nio-8090-exec-3] com.rollingstone.RestControllerAspect : :::::AOP Before REST call:::::execution(Person com.rollingstone.api.rest.PersonController.getPerson(Long,HttpServletRequest,HttpServletResponse))

2017-02-26 12:19:33.517 INFO 8944 --- [nio-8090-exec-3] c.r.api.rest.PersonController : ResourceNotFoundException handler:resource not found



1.49 –Verify Person Creation is working

[**http://localhost:8090/rsmortgage-personservice/v1/person**](http://localhost:8090/rsmortgage-personservice/v1/person)

{

"firstName": "Ashley",

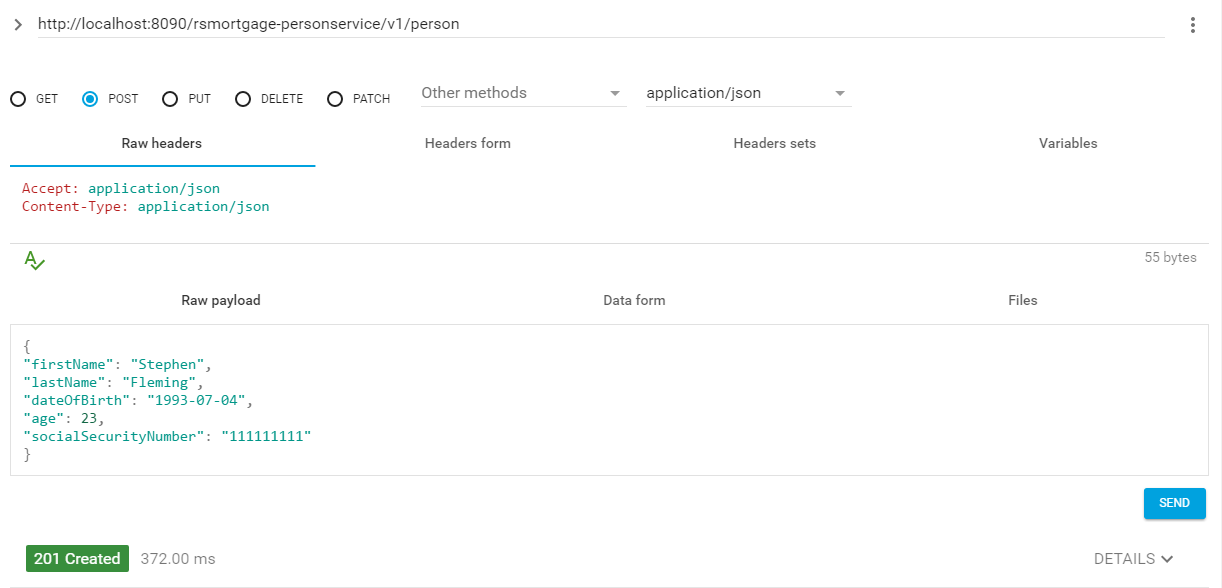
"lastName": "Bentley",

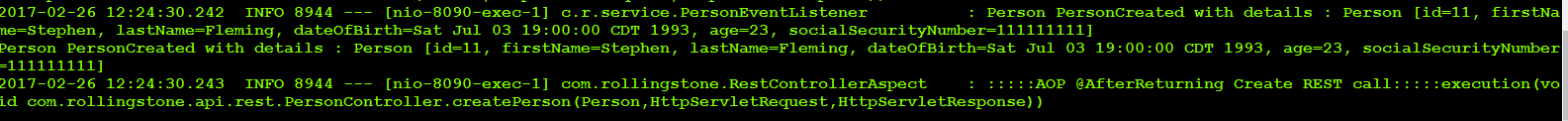
"dateOfBirth": "1995-11-04",

"age": 22,

"socialSecurityNumber": "777777777"

}





2017-02-26 12:24:30.242 INFO 8944 --- [nio-8090-exec-1] c.r.service.PersonEventListener : Person PersonCreated with details : Person [id=11, firstName=Stephen, lastName=Fleming, dateOfBirth=Sat Jul 03 19:00:00 CDT 1993, age=23, socialSecurityNumber=111111111]

Person PersonCreated with details : Person [id=11, firstName=Stephen, lastName=Fleming, dateOfBirth=Sat Jul 03 19:00:00 CDT 1993, age=23, socialSecurityNumber=111111111]

2017-02-26 12:24:30.243 INFO 8944 --- [nio-8090-exec-1] com.rollingstone.RestControllerAspect : :::::AOP @AfterReturning Create REST call:::::execution(void com.rollingstone.api.rest.PersonController.createPerson(Person,HttpServletRequest,HttpServletResponse))

1.49 –Verify Person Updation is working

<http://localhost:8090/rsmortgage-personservice/v1/person/11>

{

"id": 11,

"firstName": "Stephen",

"lastName": "Fleming",

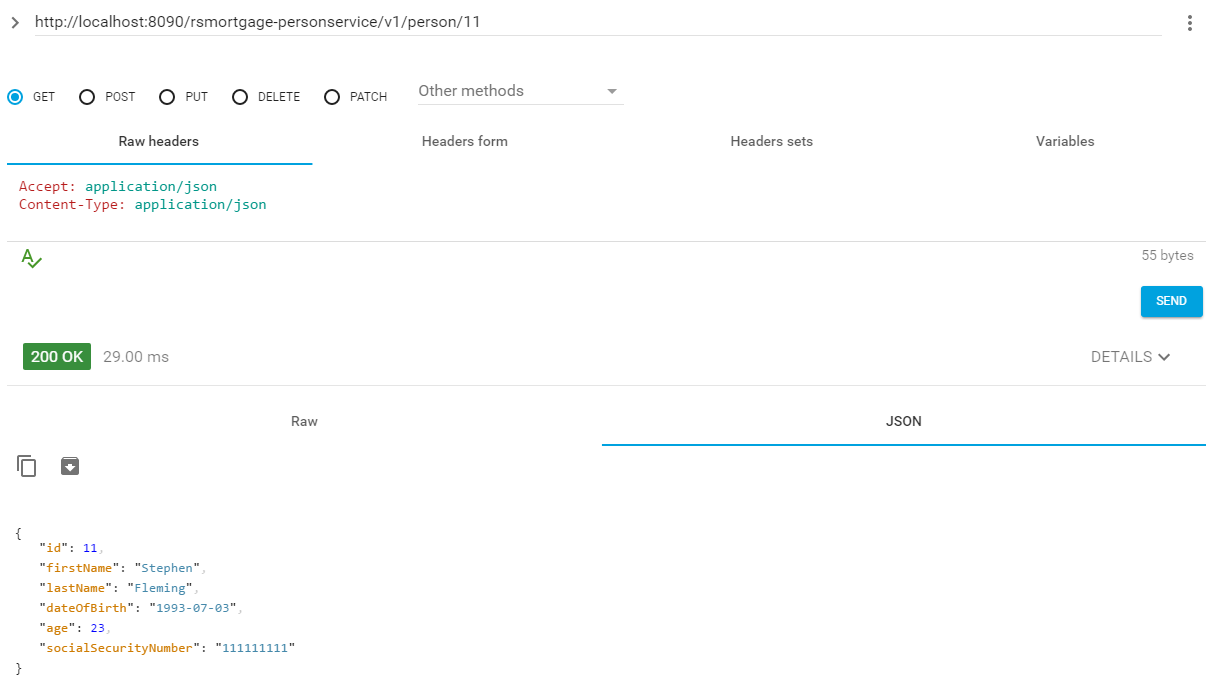
"dateOfBirth": "1993-07-04",

"age": 23,

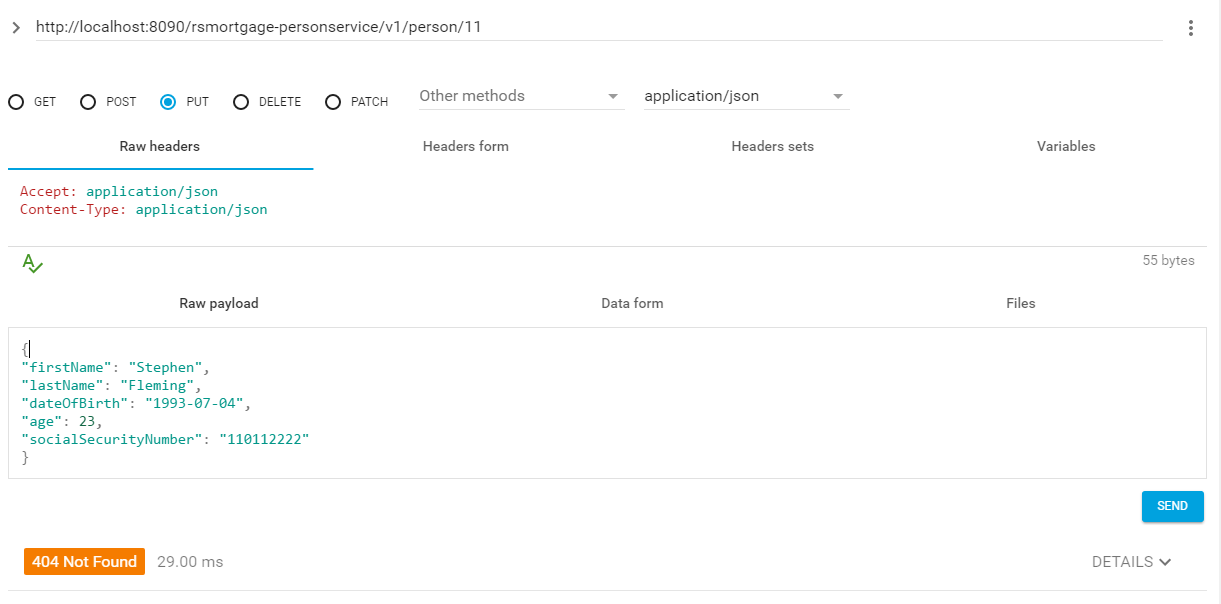
"socialSecurityNumber": "110112222"

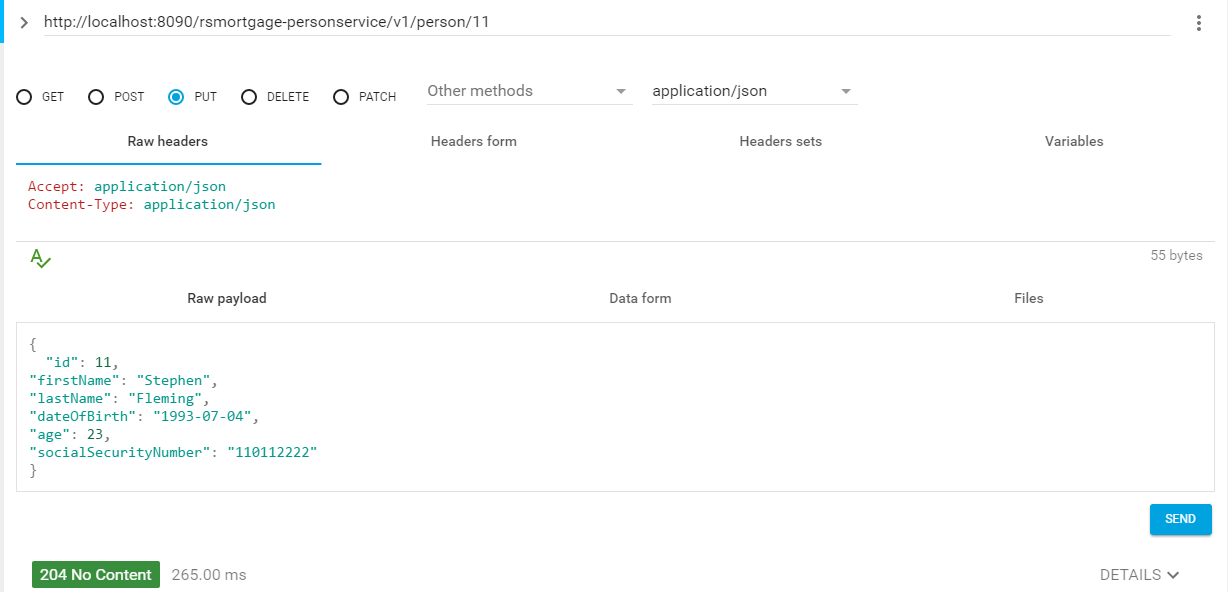
}

Before

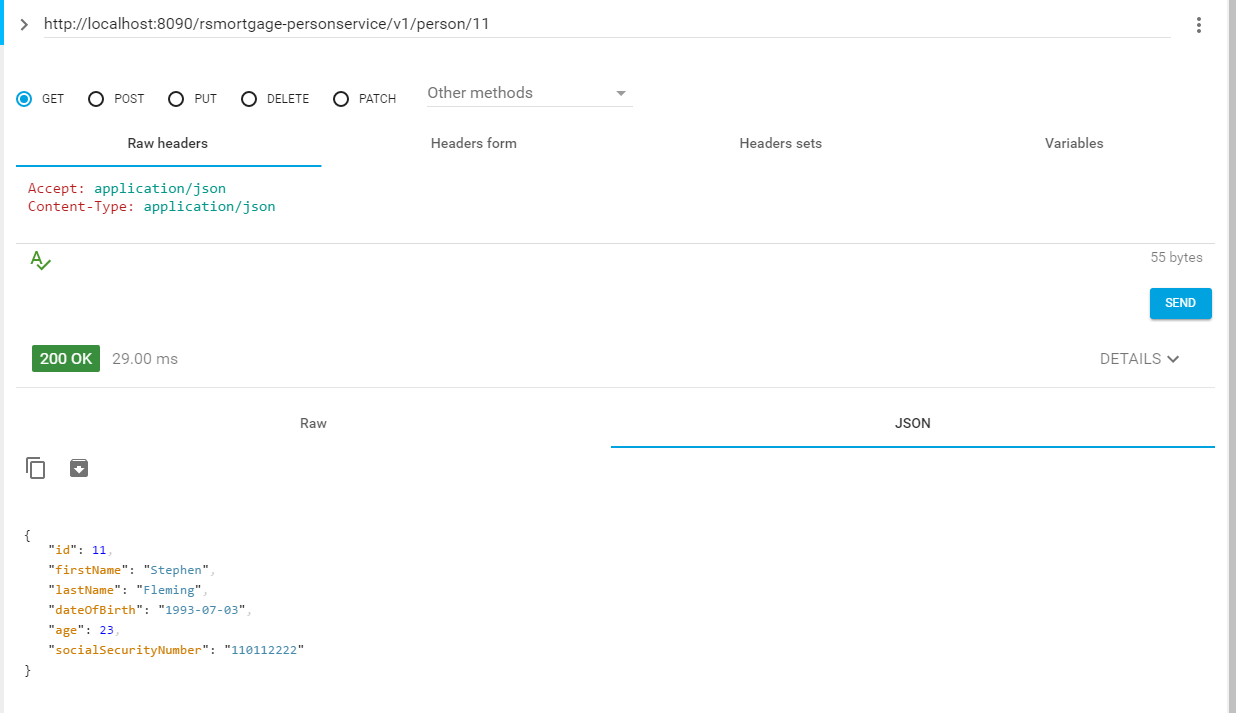


**ID missing in payload**





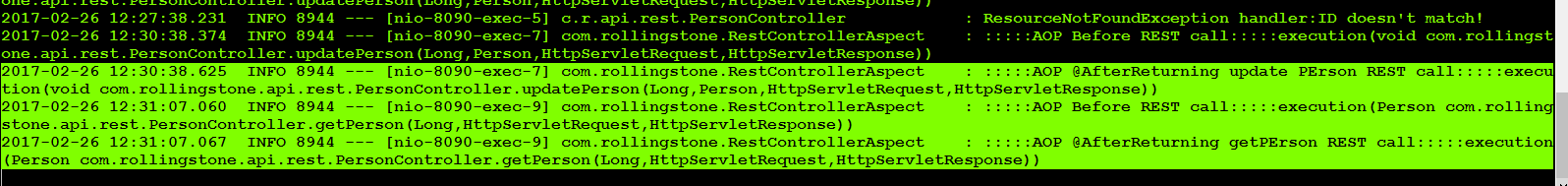
After



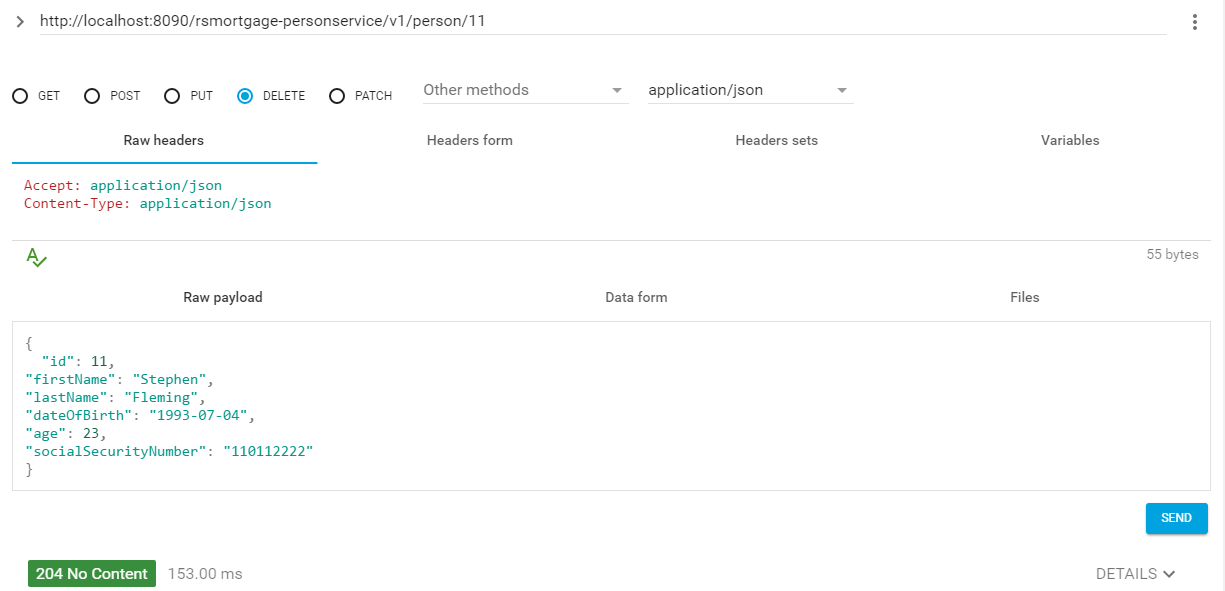
2017-02-26 12:30:38.625 INFO 8944 --- [nio-8090-exec-7] com.rollingstone.RestControllerAspect : :::::AOP @AfterReturning update PErson REST call:::::execution(void com.rollingstone.api.rest.PersonController.updatePerson(Long,Person,HttpServletRequest,HttpServletResponse))

2017-02-26 12:31:07.060 INFO 8944 --- [nio-8090-exec-9] com.rollingstone.RestControllerAspect : :::::AOP Before REST call:::::execution(Person com.rollingstone.api.rest.PersonController.getPerson(Long,HttpServletRequest,HttpServletResponse))

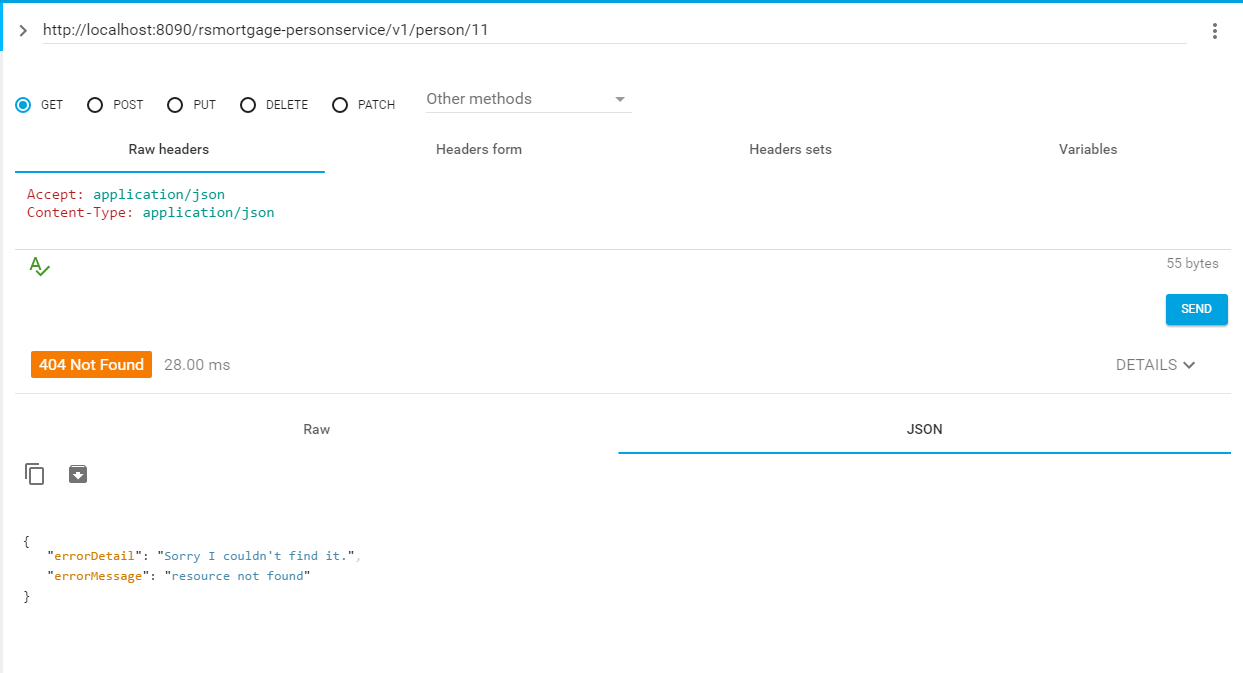
2017-02-26 12:31:07.067 INFO 8944 --- [nio-8090-exec-9] com.rollingstone.RestControllerAspect : :::::AOP @AfterReturning getPErson REST call:::::execution(Person com.rollingstone.api.rest.PersonController.getPerson(Long,HttpServletRequest,HttpServletResponse))



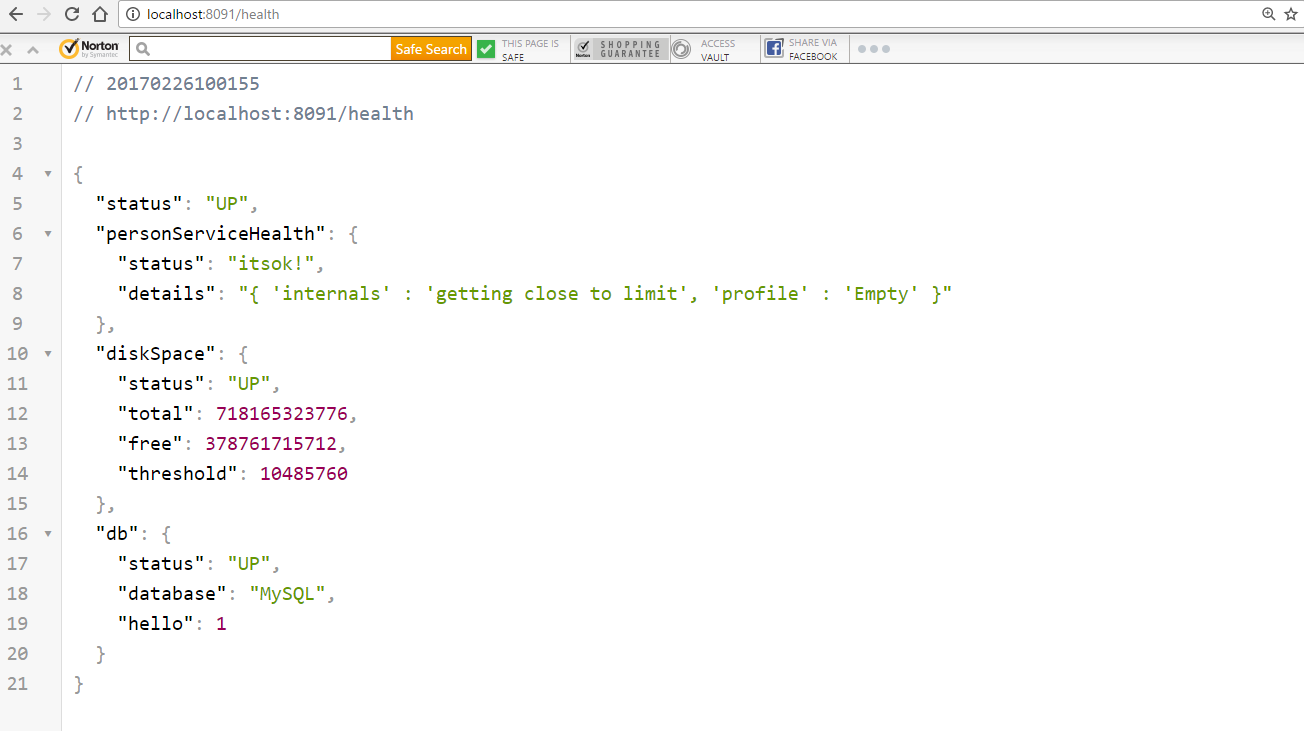
1.50 –Verify Person Deletion is working



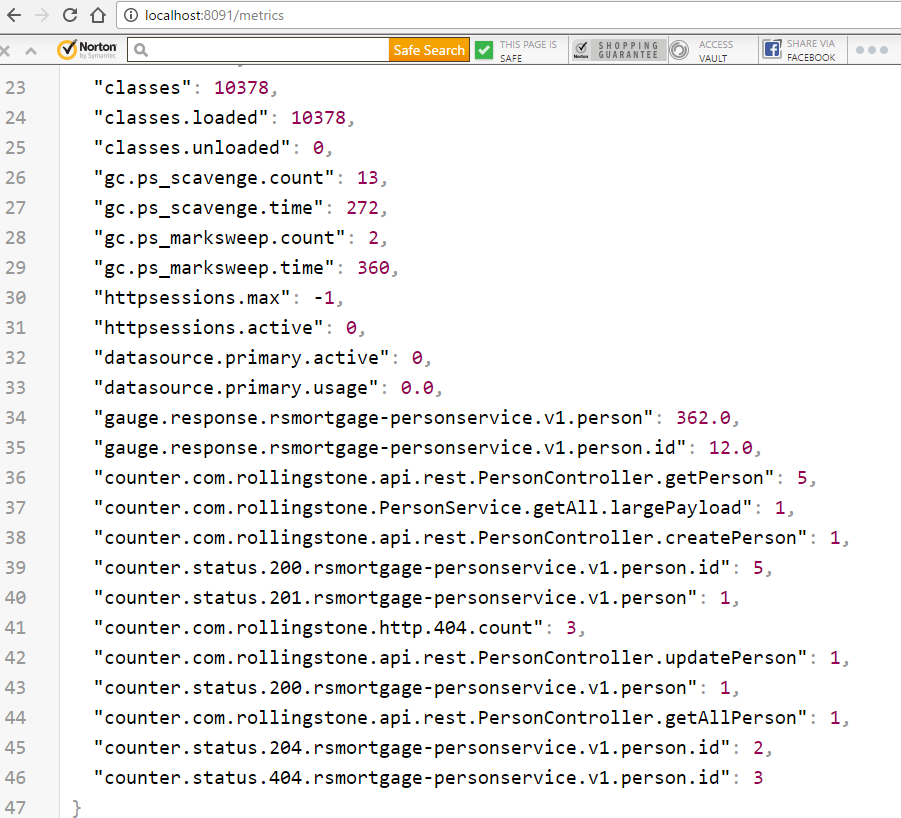
Not Found



1.51 –Custom Health

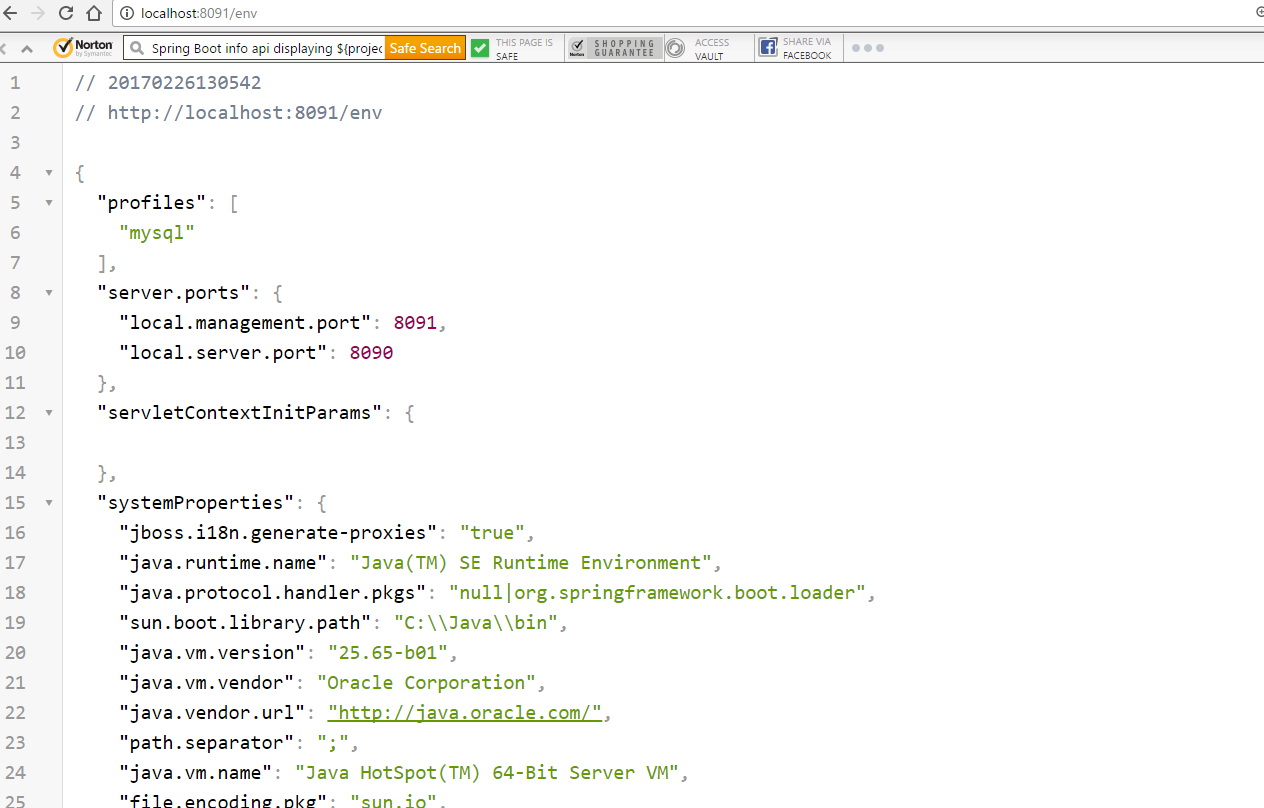


1.54 – Metrics and Custom Metrics



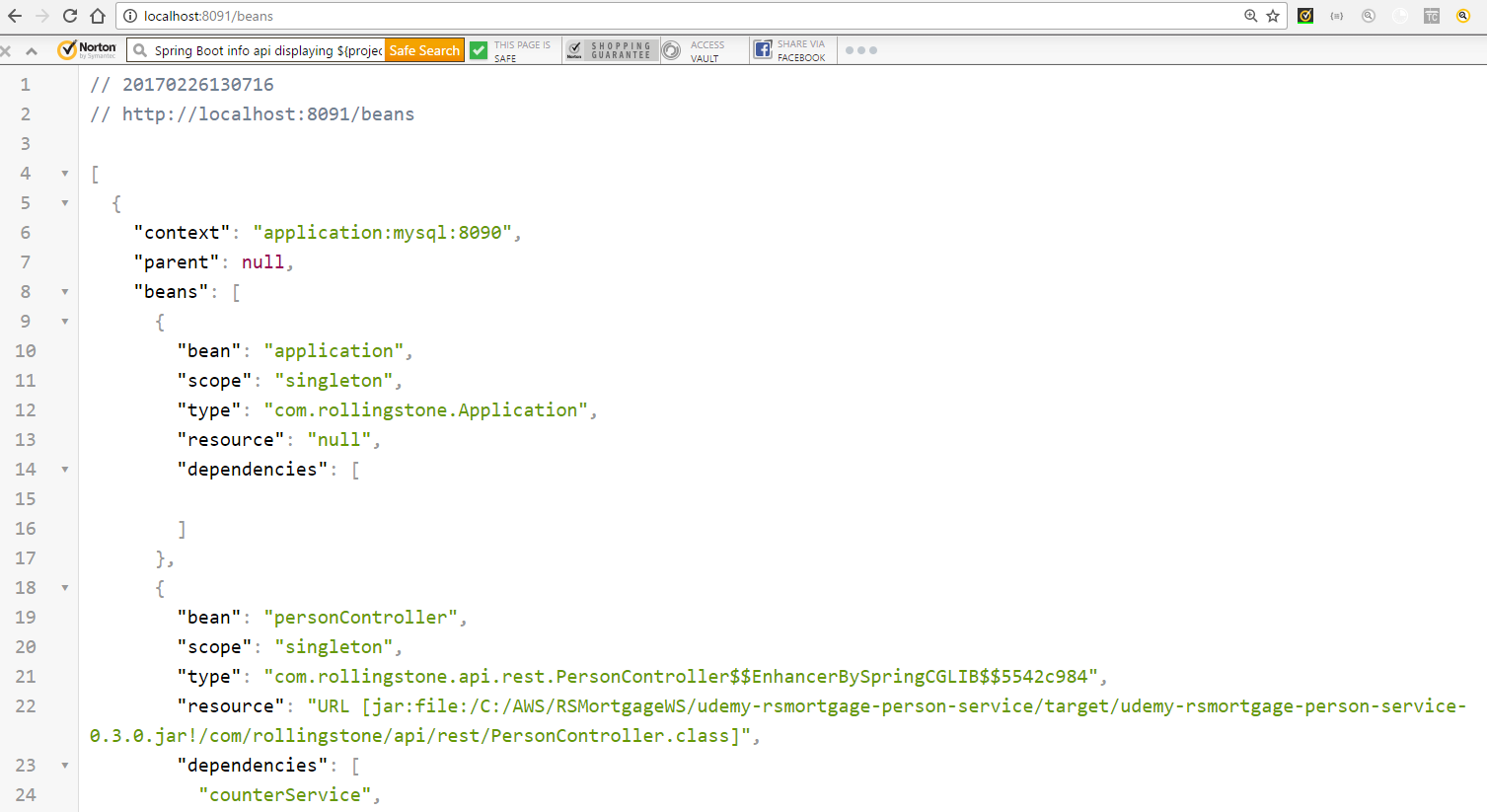
1.55 – /env endpoint

<http://localhost:8091/env>



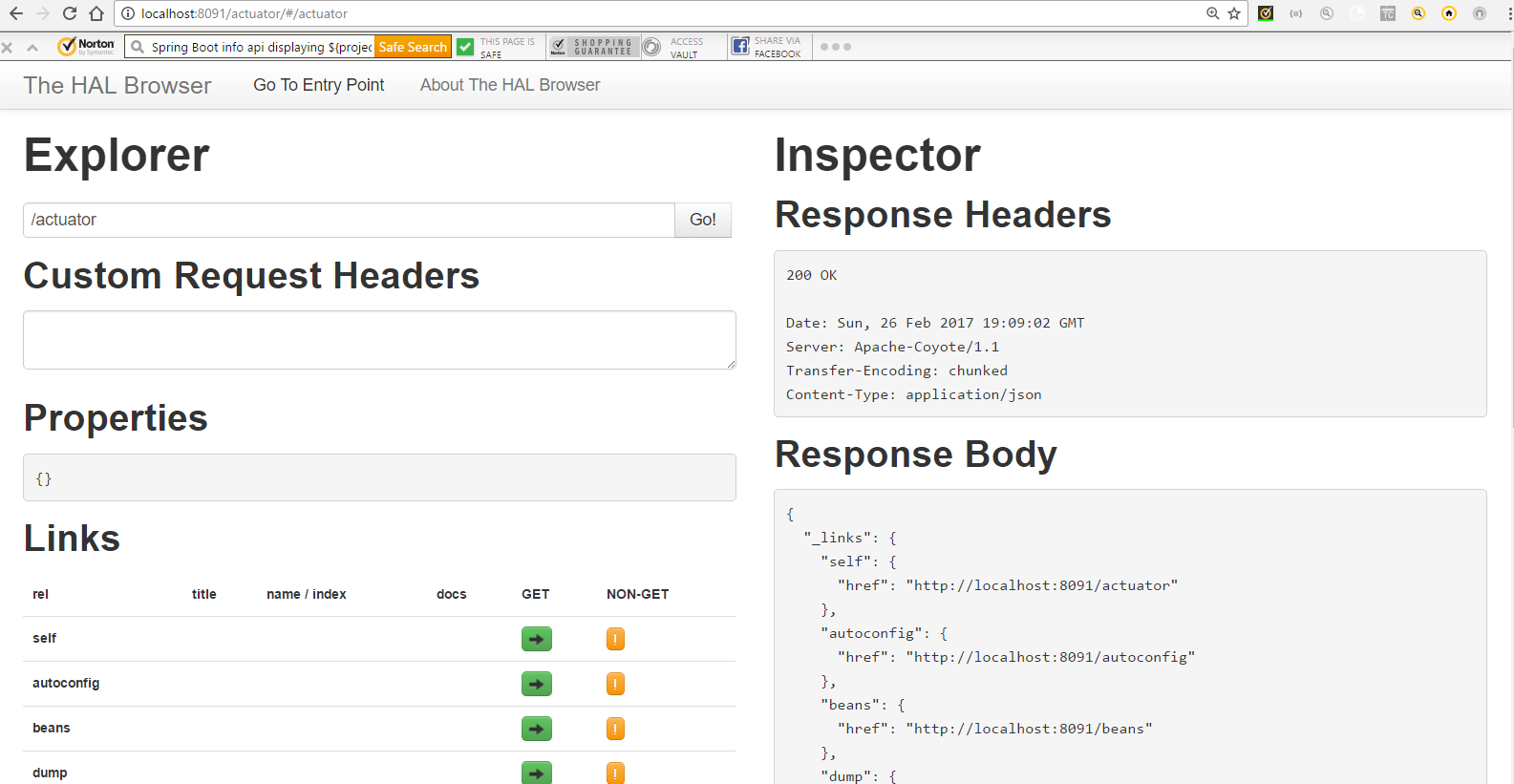
1.56 – /beans endpoint

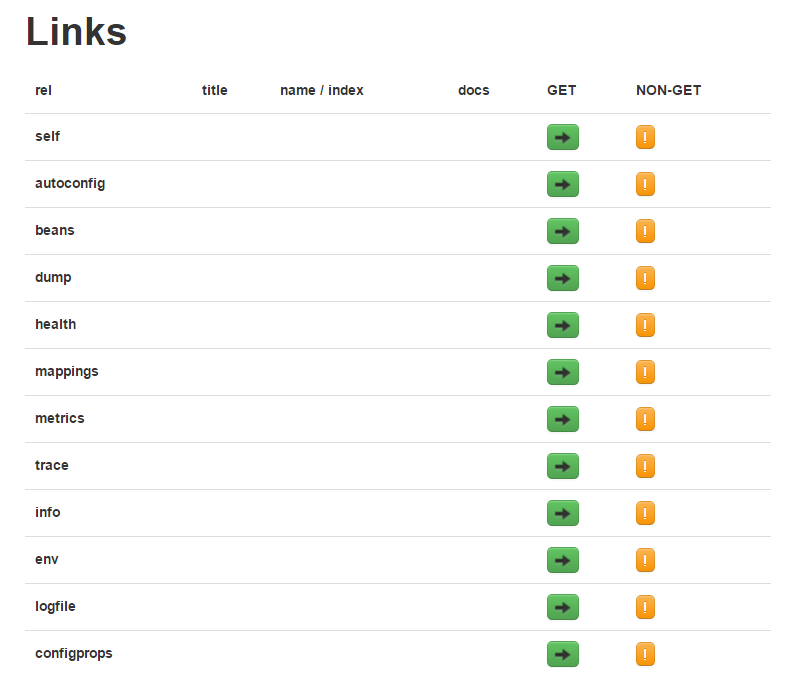
<http://localhost:8091/beans>



1.57 – /actuator endpoint

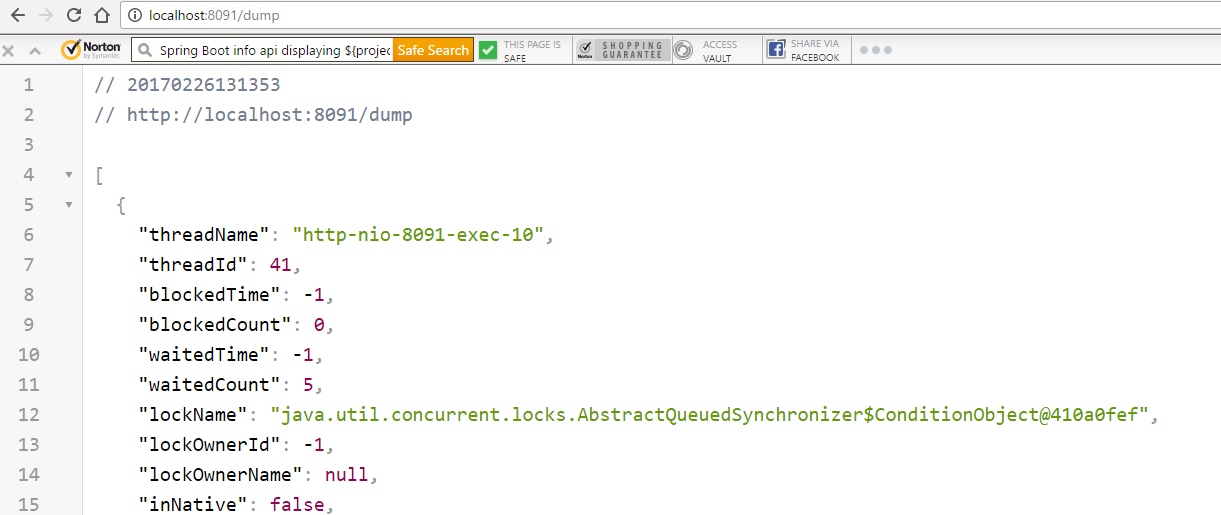
http://localhost:8091/actuator/#/actuator





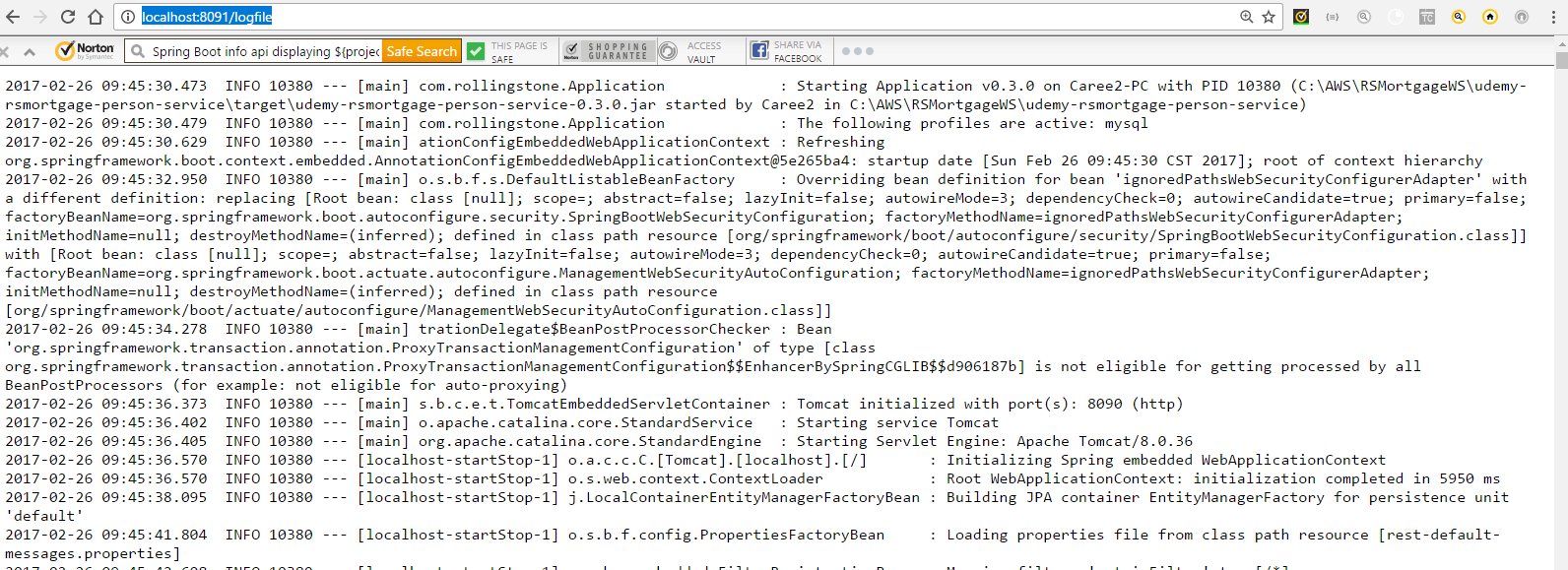
1.58 – /dump endpoint

http://localhost:8091/dump



1.59 – /logfile endpoint

http://localhost:8091/logfile



1.60 –Conclusion

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