

Lab RSMortgage Customer Service

**Spring Cloud RSMortgage Customer REST Service**

Udemy Exploring Spring Cloud Course

Presented By

Binit Datta

**Rolling Stone Technology**

**Formatted: December, 2016**

**Table of Content**

1.0 - Introduction 6

1.1 – Create a new Spring Starter Project 7

1.2 – Fill initial values 8

1.3 – Choose Eureka and Web as starter projects 9

1.4 – Click Finish Now 10

1.5 – Let Spring Tool Suite Prepare the Project 11

1.6 – Make sure the following looks like below 12

1.7 – Spring Boot Maven Parent Section 12

1.8 – Maven Properties Section 12

1.9 – Maven Dependency Management Section 12

1.10 – Spring Boot Actuator Dependency 13

1.11 – Spring Boot Web Dependency 13

1.12 – Spring Boot JPA Dependency 13

1.13 – Spring Boot H2 Dependency 13

1.14 – Spring Boot Test Dependency 14

1.15 – Spring Boot Jackson DataBind Dependency 14

1.16 – Spring Boot Jackson HAL Browser Dependency 15

1.17 – Spring Boot Jackson JSON Test Dependency 15

1.18 – Spring Boot Jackson Swagger Dependency 16

1.19 – Spring Boot HSQL Dependency 16

1.20 – Spring Boot HSQL Dependency 17

1.21 – Spring Cloud Eureka Dependency 17

1.22 – Spring Cloud Feign Dependency 17

1.23 – Spring Cloud Config Dependency 18

1.24 – Maven Build configuration 18

1.25 – Add api.rest package 19

1.26– Add dao.jpa package 21

1.27– Add domain package 22

1.15– Add exception package 23

1.28– Add service package 24

1.29– Create Account Domain class in the domain package 25

1.30– Generate the following for the Account class 26

1.31– Create AccountType Domain class in the domain package 27

1.32– Generate the following for the AccountType class 27

1.33– Create the Address Domain class in the domain package 28

1.34– Do the following to the Address Class 29

1.35--Create the Address Domain class in the domain package 30

1.36– Do the following to the Contact Class 31

1.37--Create the ContactType Domain class in the domain package 32

1.38– Do the following to the ContactType Class 32

1.39--Create the Customer Domain class in the domain package 33

1.40– Do the following to the Customer Class 34

1.41--Create the DegreeType Domain class in the domain package 35

1.42– Do the following to the DegreeType Class 35

1.43--Create the Education Domain class in the domain package 36

1.44– Do the following to the Education Class 37

1.45--Create the Employment Domain class in the domain package 38

1.46– Do the following to the Employment Class 40

1.47--Create the Investment Domain class in the domain package 41

1.48– Do the following to the Investment Class 42

1.49--Create the InvestmentType class in the domain package 42

1.50– Do the following to the InvestmentType Class 42

1.51--Create the Liability Domain class in the domain package 43

1.52– Do the following to the Liability Class 44

1.53--Create the LiabilityType Domain class in the domain package 44

1.54– Do the following to the LiabilityType Class 44

1.55--Create the RestAPIExceptionInfo class in the domain package 45

1.56– Generate HTTP400Exception in the exception package 46

1.57– Generate HTTP404Exception in the exception package 47

1.58– Generate DAOInterface in the dao.jpa package 48

1.59– Generate Service class in the service package 49

1.60– Generate ServiceProperties class in the service package 51

1.61– Generate ServiceHealth class in the service package 52

1.62– Generate ServiceEvent class in the service package 53

1.63– Generate AbstractRestController class in the rest.api package 54

1.64– Generate CustomerControllerclass in the rest.api package 56

1.65– Generate RestControllerAspect in the rest.api package 59

1.66– Generate RsMortgageCustomerRestAPIApplication in the rest.api package 60

1.67– Create application.yml file under resources 61

1.68– Add bootstrap.yml file in the resources folder 62

1.69 –Open Git Bash in project folder 63

1.70 –Run the first instance 63

1.71 –Run the second instance 64

1.72 – Navigate to http://localhost:8761 65

1.73 – Navigate to http://localhost:8762 66

1.74 – Navigate to http://eureka-host1:8761/ 67

1.75 – Navigate to http://eureka-host2:8762/ 68

1.76– Open Git bash in the config project directory 69

1.77– Run the config service project 69

1.78– Verify Customer Project mysql properties 70

1.79– Open Git bash in the project directory 71

1.80– Build the project 72

1.81 –Run the Project 73

1.82 –Verify Config Property is read and used 73

1.83 – Navigate to http://localhost:8761 74

1.84—Get an existing Customer 76

1.85– Create a Customer 77

1.86– Verify the Database 78

1.87—Try to Update a Record 78

1.87—Verify the Database 79

1.88 – Try to get a single customer 80

1.89 – Try to delete a single customer 81

1.90 – Swagger UI 82

1.91 – Conclusion 83

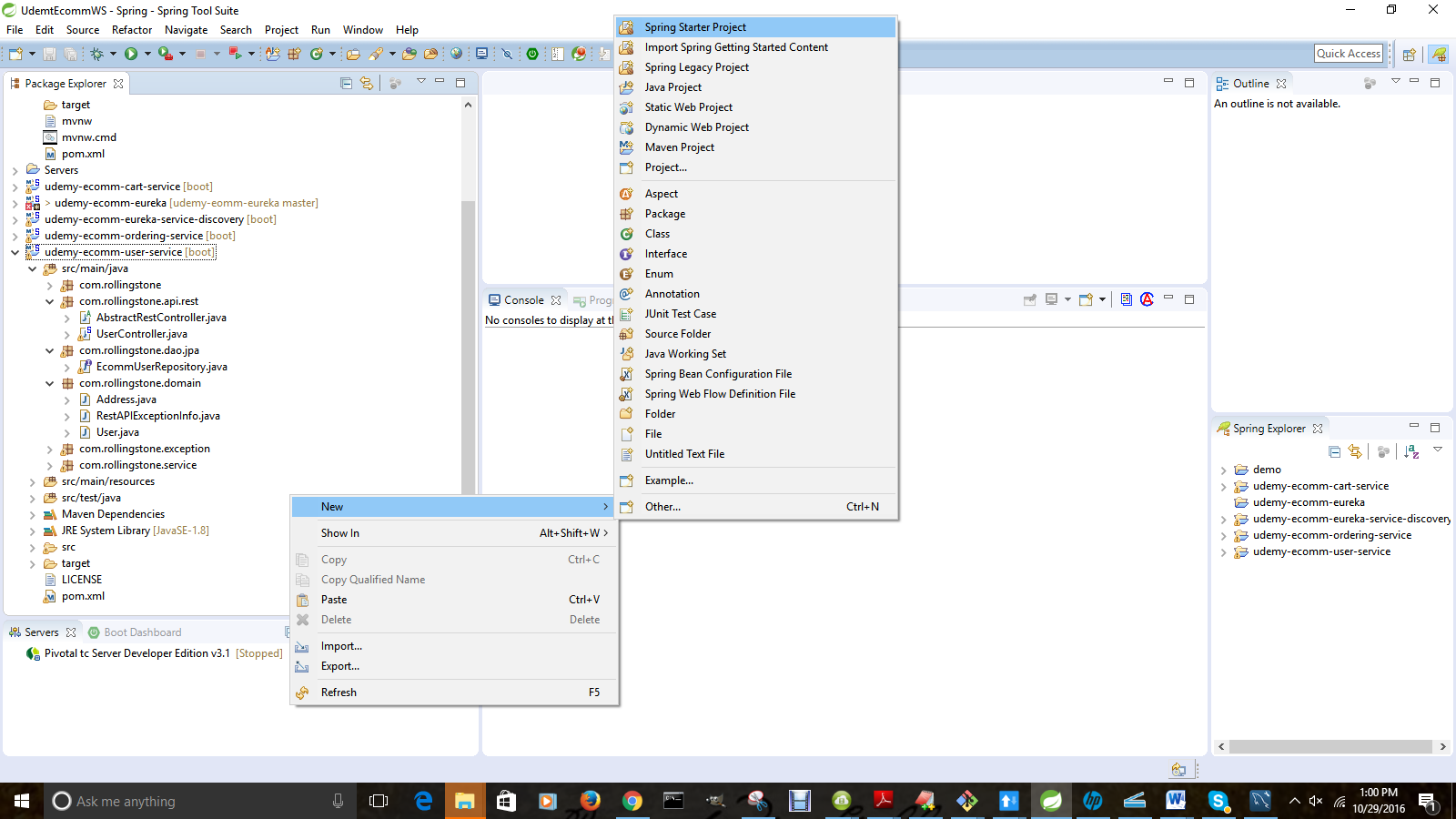
Chapter 1

*Spring Cloud Customer Service Project Creation*

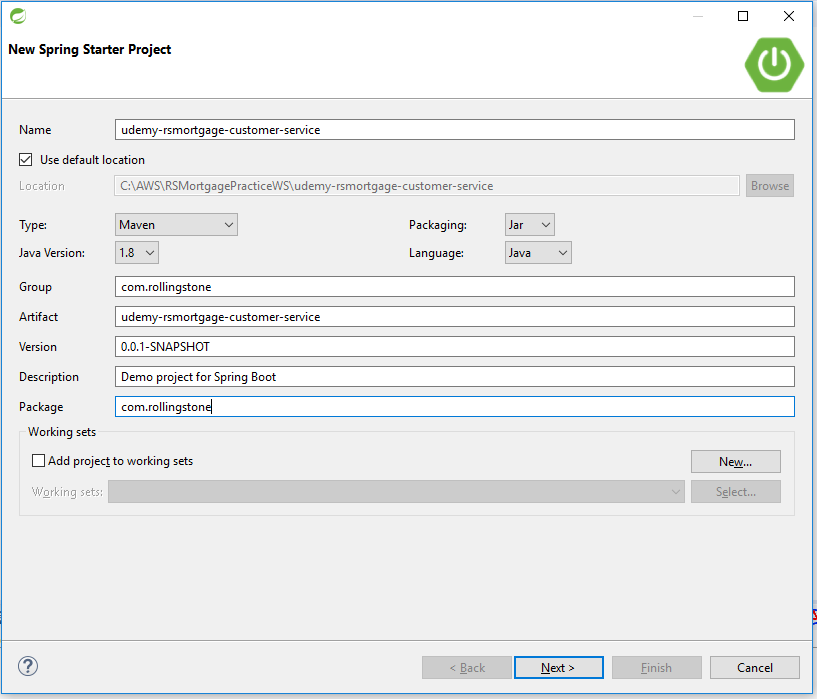
1.0 - Introduction

The following sections will lead us through creating the Spring Cloud Customer Microservice, one step at a time.

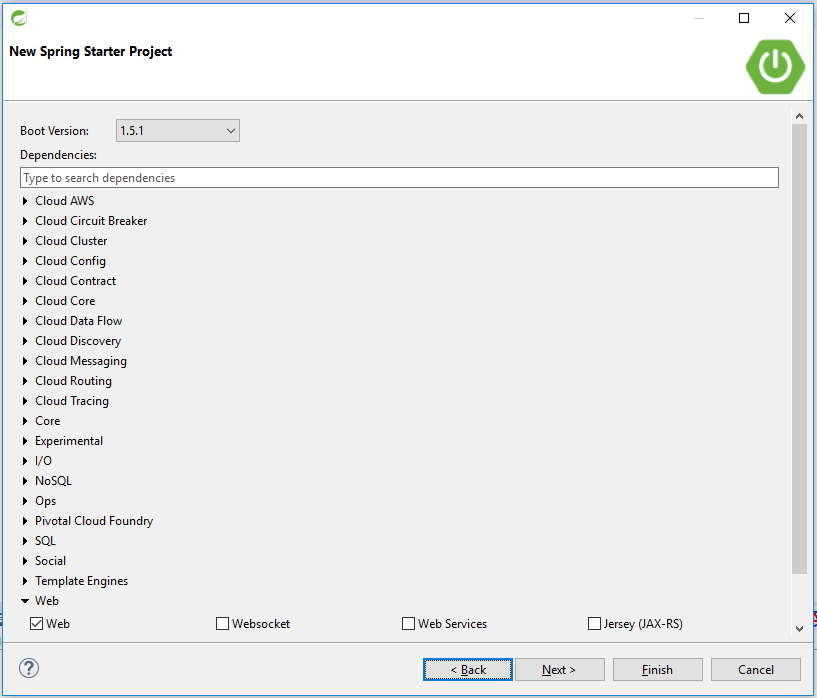
1.1 – Create a new Spring Starter Project



1.2 – Fill initial values

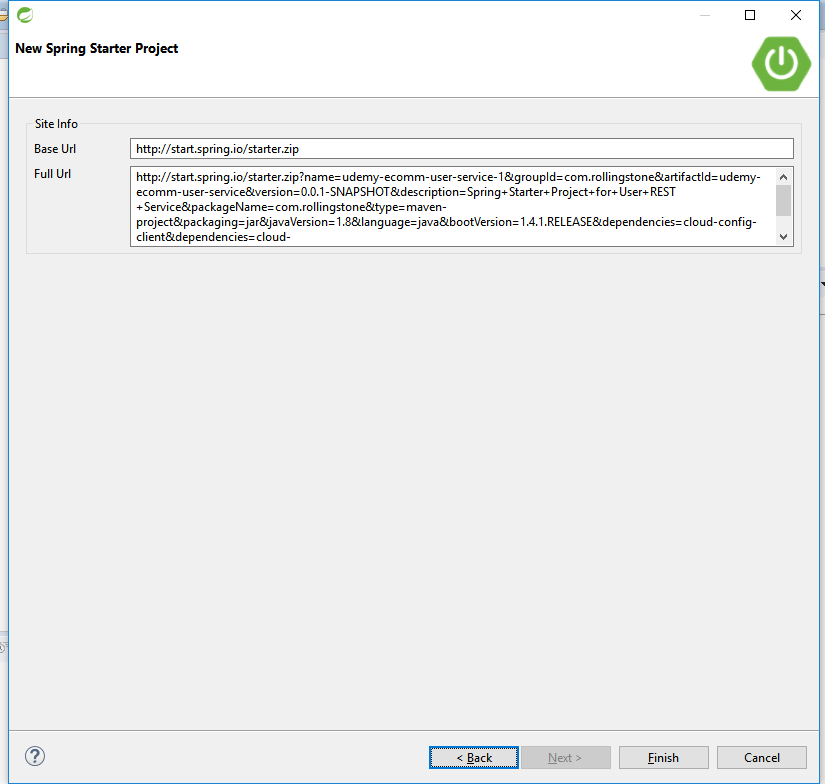


1.3 – Choose Eureka and Web as starter projects

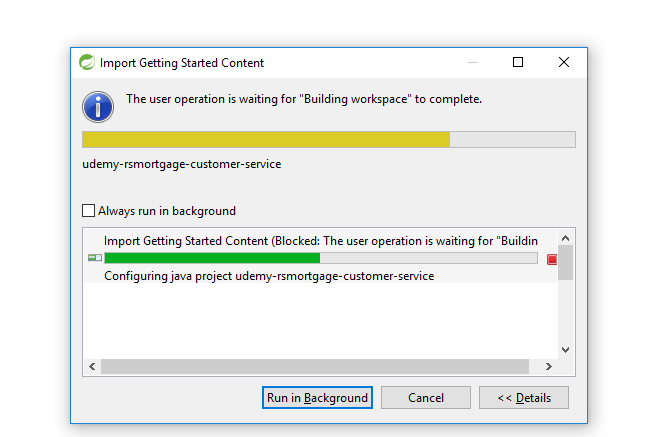


.

1.4 – Click Finish Now



1.5 – Let Spring Tool Suite Prepare the Project



1.6 – Make sure the following looks like below

<groupId>com.rollingstone</groupId>

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

<version>1.0</version>

<packaging>jar</packaging>

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

</description>

1.7 – Spring Boot Maven Parent Section

<parent>

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

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

<version>1.3.6.RELEASE</version>

</parent>

1.8 – Maven Properties Section

<properties>

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

</properties>

1.9 – Maven Dependency Management Section

<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>

1.10 – Spring Boot Actuator Dependency

<dependency>

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

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

</dependency>

1.11 – Spring Boot Web Dependency

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

<dependency>

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

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

</dependency>

1.12 – Spring Boot JPA Dependency

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

<dependency>

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

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

</dependency>

1.13 – Spring Boot H2 Dependency

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<version>1.4.181</version>

</dependency>

1.14 – Spring Boot Test Dependency

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

<dependency>

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

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

<scope>test</scope>

</dependency>

1.15 – Spring Boot Jackson DataBind Dependency

<dependency>

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

<artifactId>jackson-databind</artifactId>

</dependency>

1.16 – Spring Boot Jackson HAL Browser Dependency

<dependency>

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

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

</dependency>

1.17 – Spring Boot Jackson JSON Test 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>

1.18 – Spring Boot Jackson Swagger 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>

1.19 – Spring Boot HSQL Dependency

<dependency>

<groupId>org.hsqldb</groupId>

<artifactId>hsqldb</artifactId>

<scope>runtime</scope>

</dependency>

1.20 – Spring Boot HSQL Dependency

<dependency>

<groupId>mysql</groupId>

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

<version>5.1.40</version>

</dependency>

1.21 – Spring Cloud Eureka Dependency

<dependency>

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

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

</dependency>

1.22 – Spring Cloud Feign Dependency

<dependency>

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

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

</dependency>

1.23 – Spring Cloud Config Dependency

<dependency>

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

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

</dependency>

<dependency>

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

<artifactId>spring-cloud-config-client</artifactId>

</dependency>

1.24 – Maven Build configuration

<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>

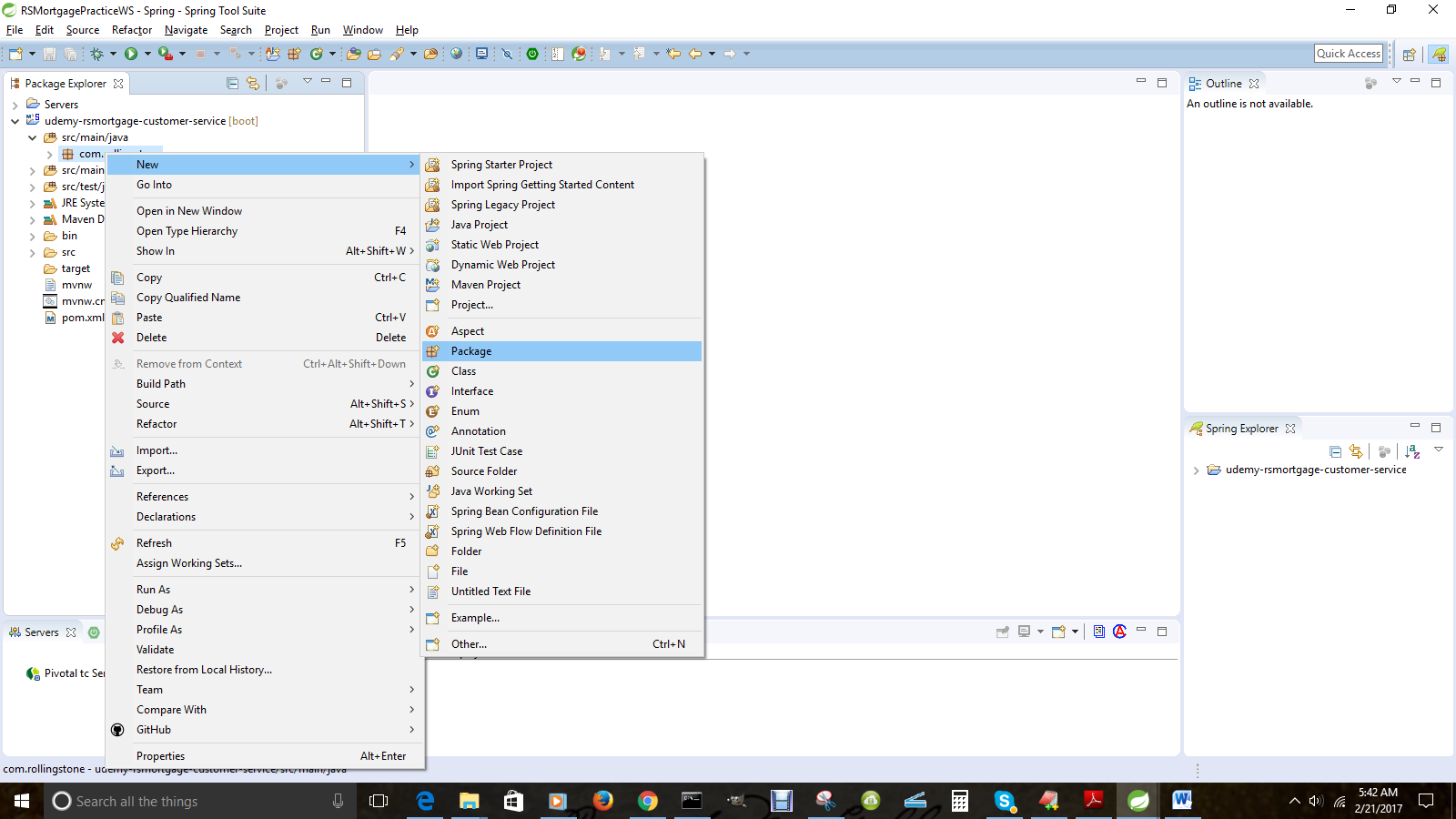
</configuration>

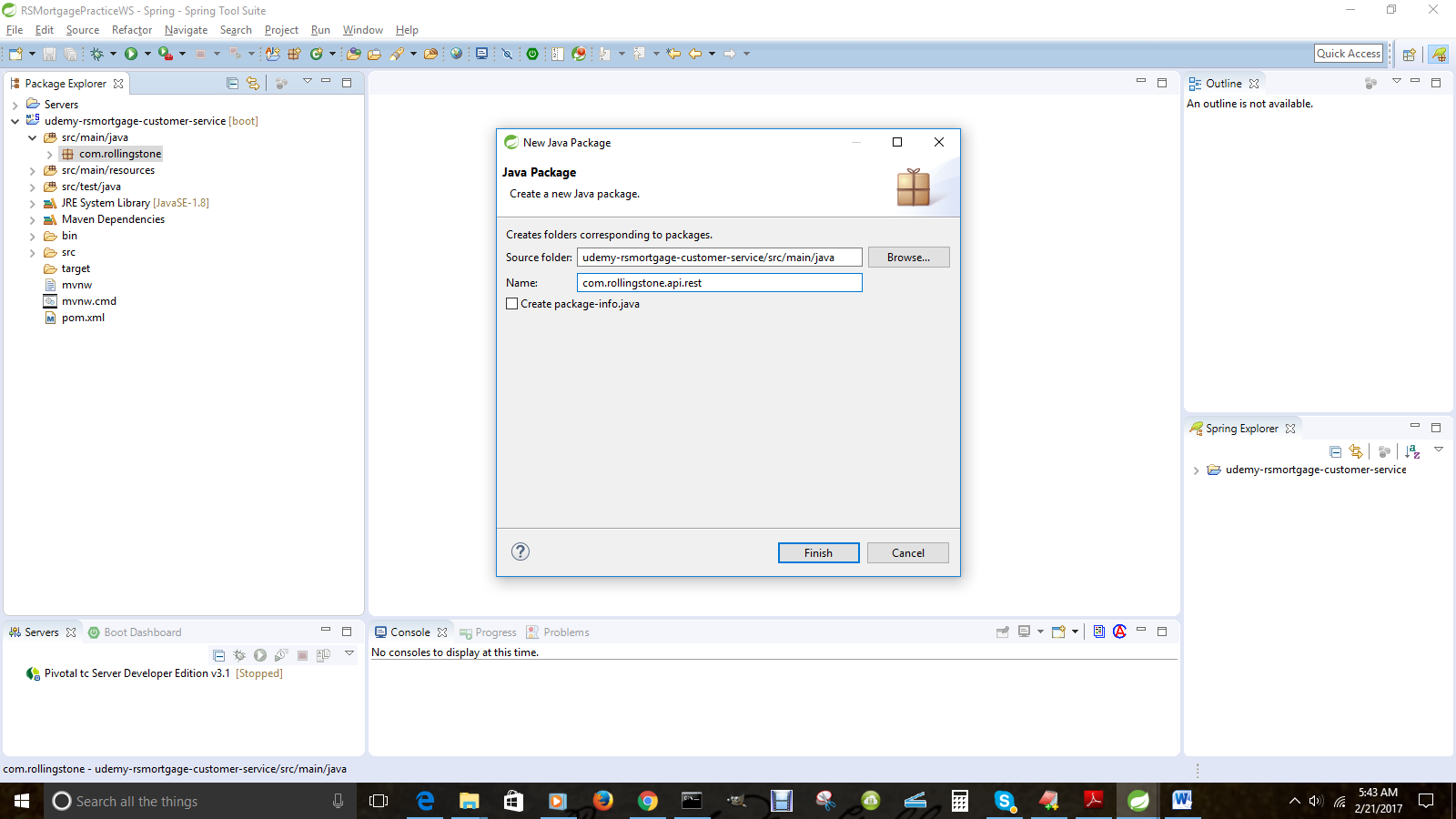
</plugin>

</plugins>

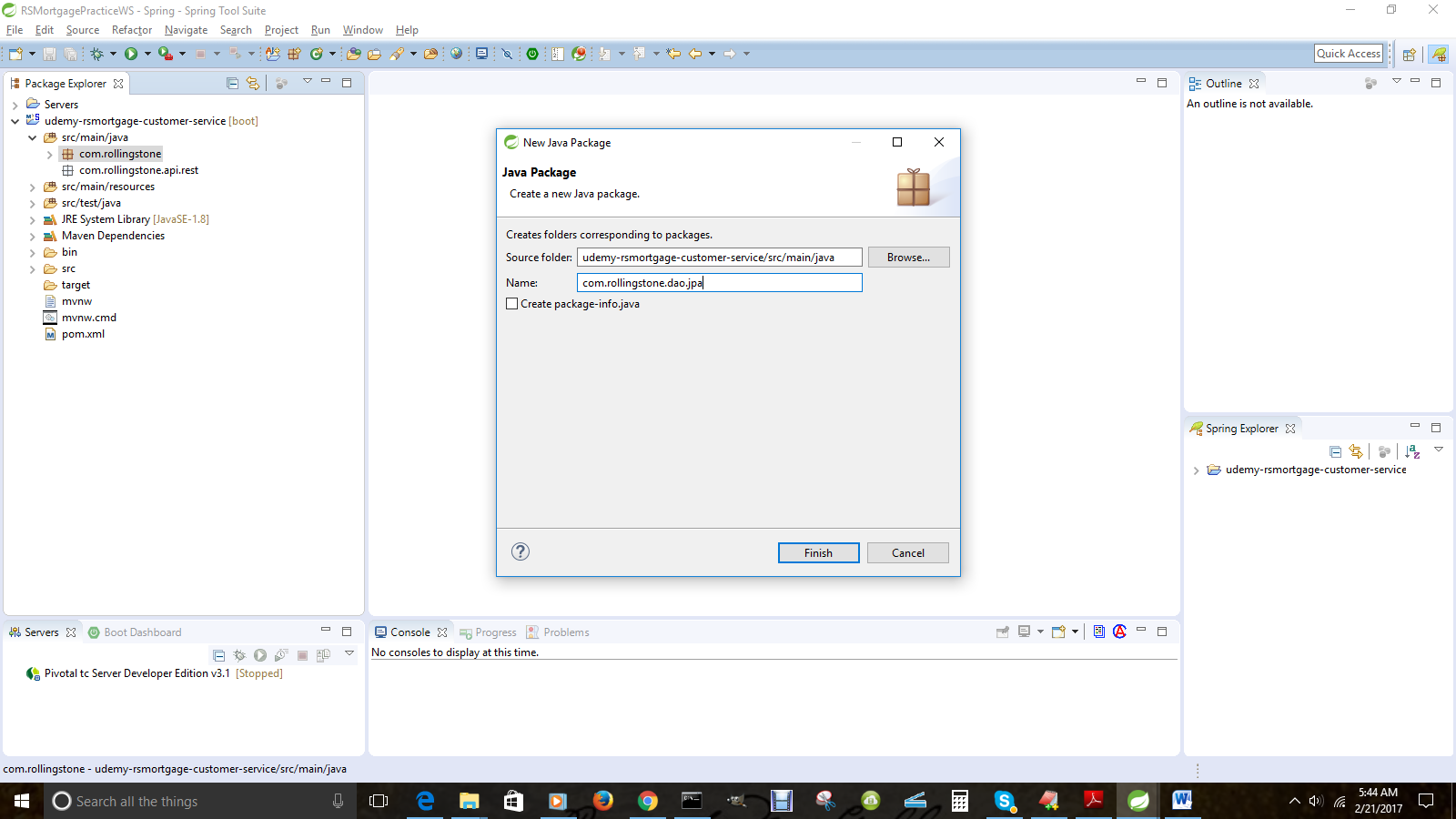
</build>

1.25 – Add api.rest package

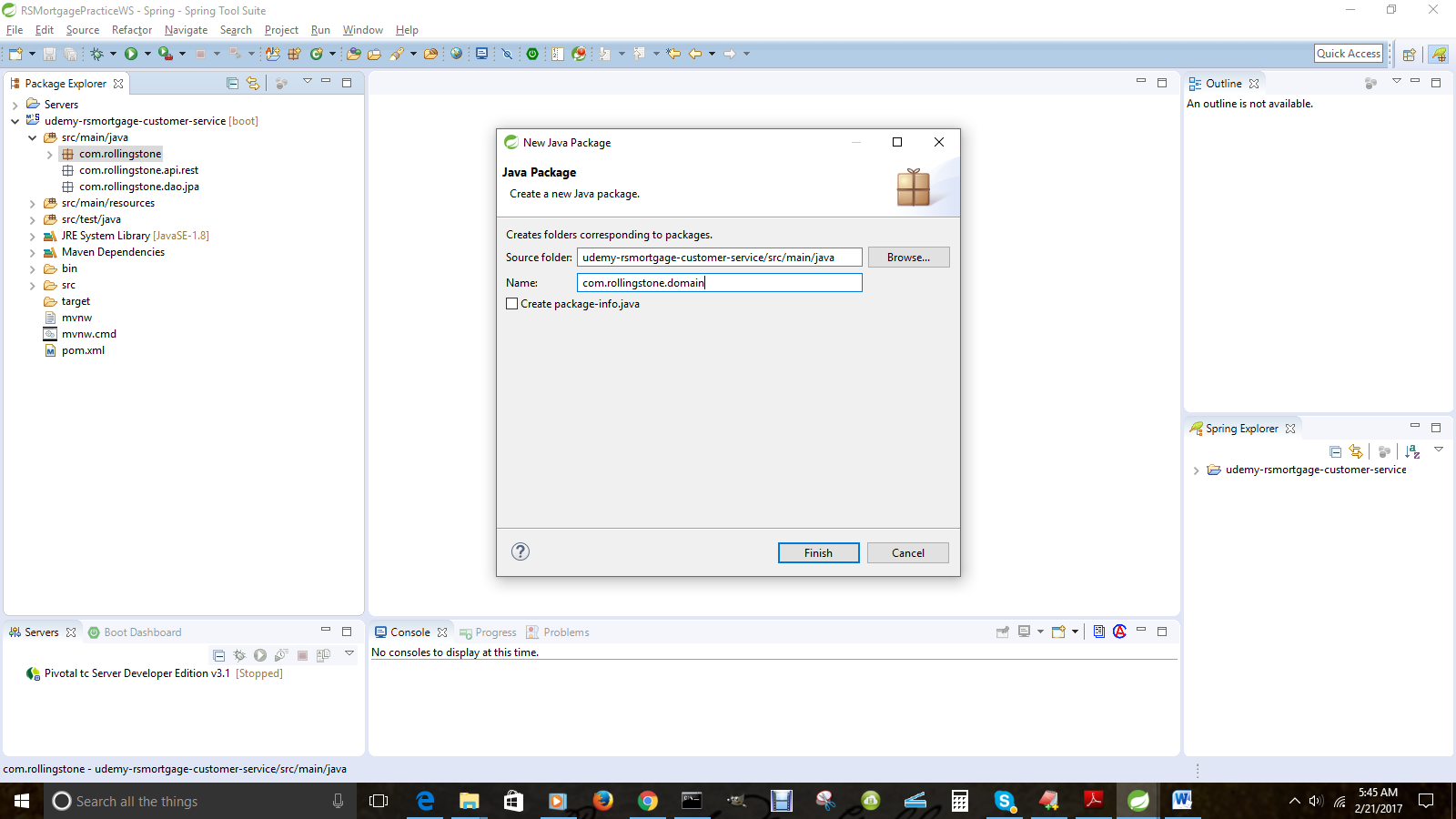




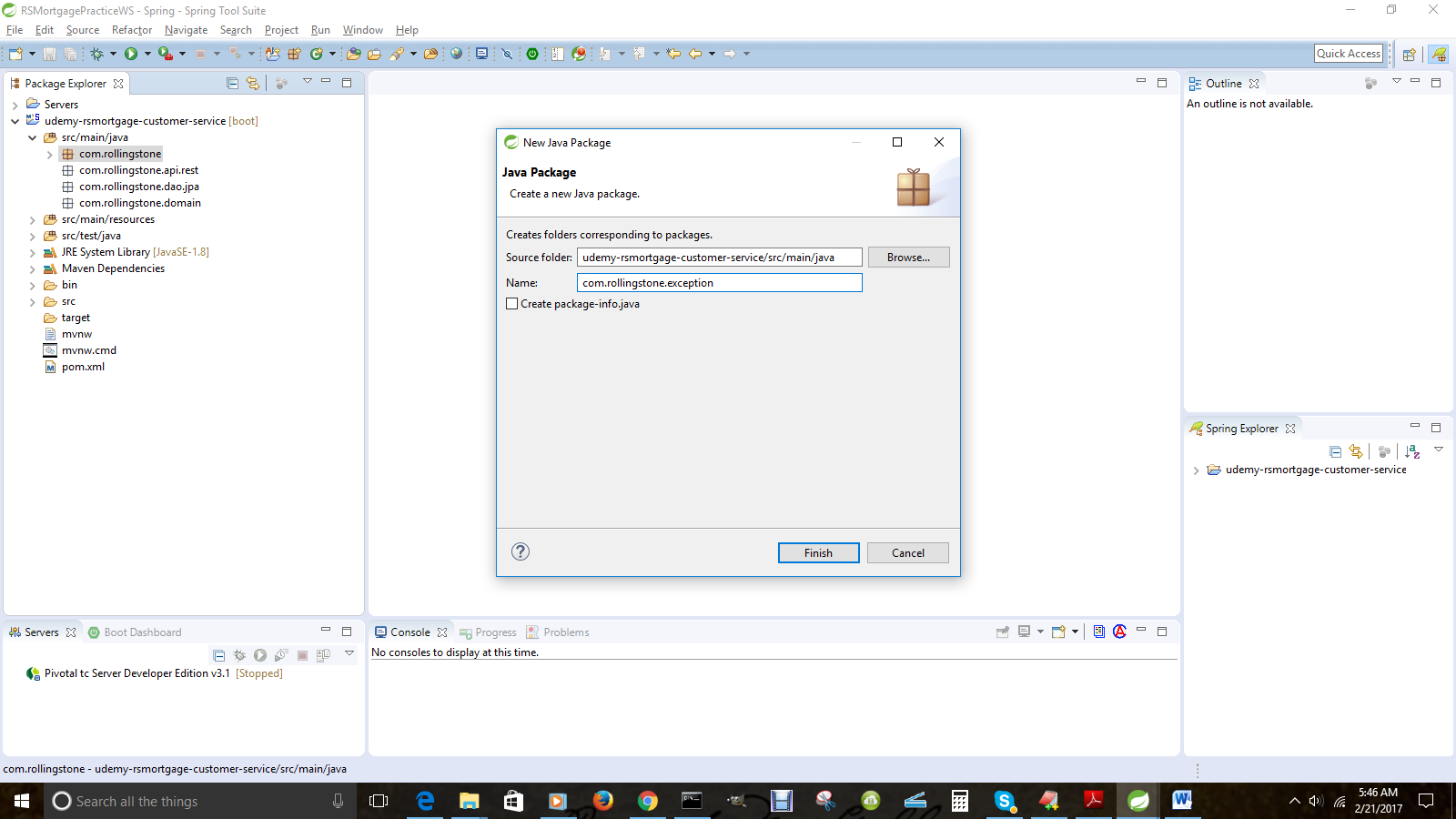
1.26– Add dao.jpa package



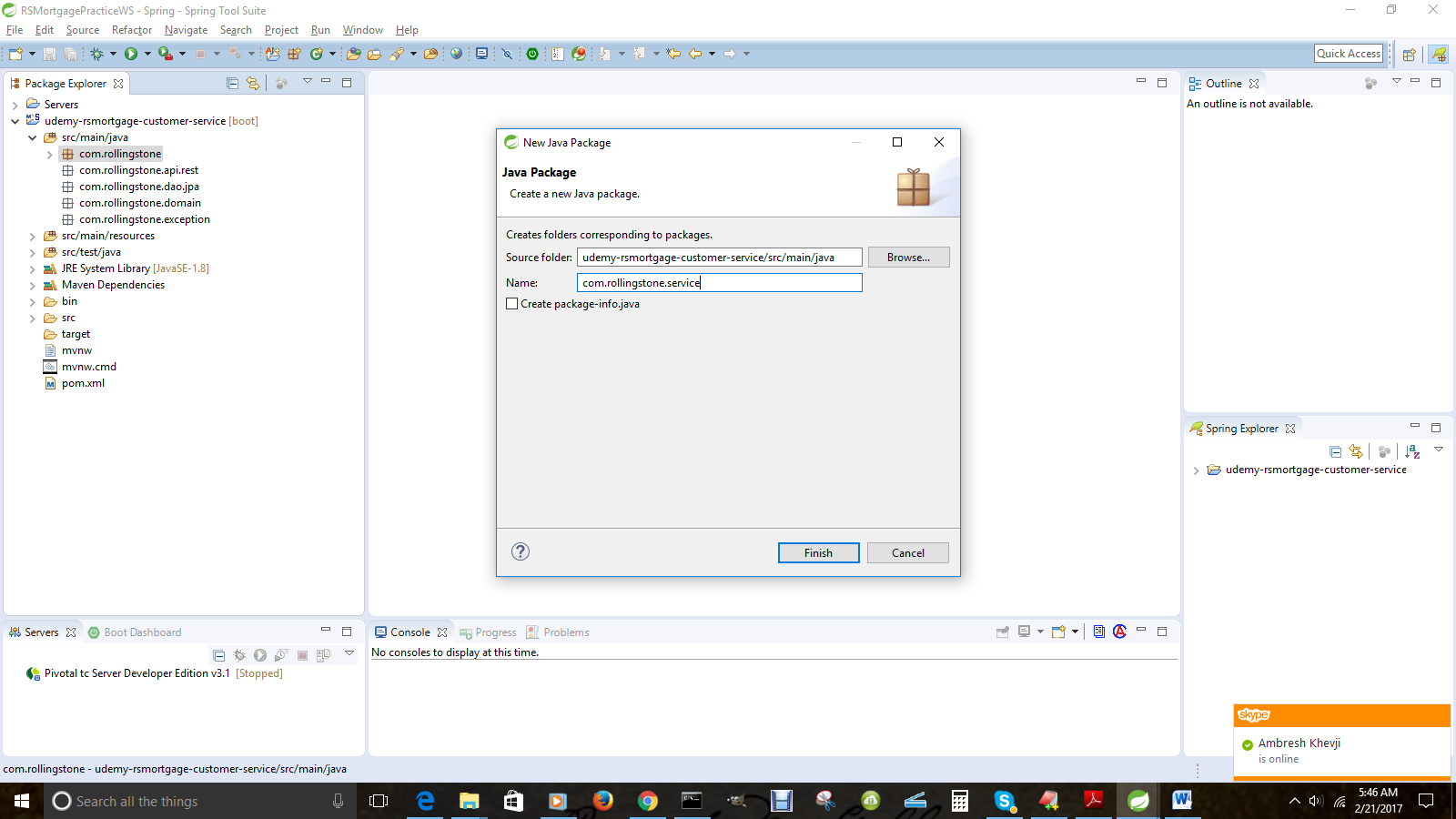
1.27– Add domain package



1.15– Add exception package



1.28– Add service package



1.29– Create Account Domain class in the domain package

package com.rollingstone.domain;

import java.util.Date;

/\*

\* A Customer POJO serving as an Entity as well as a Data Transfer Object i.e DTO

\*/

@Entity

@Table(name = "rsmortgage\_account")

@XmlRootElement

@XmlAccessorType(XmlAccessType.FIELD)

public class Account {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private long id;

@OneToOne

@JoinColumn(name="account\_type\_id")

private AccountType accountType;

@Temporal(TemporalType.DATE)

@Column(name = "date\_created", unique = true, nullable = false, length = 10)

private Date dateCreated;

@Column(nullable = false)

private double originalCreditAmount;

@Column(nullable = false)

private double balanceAmount;

@Column(nullable = false)

private boolean fullyPaid;

@Column(nullable = false)

private int term;

@Column(nullable = false)

private float rateOfInterest;

@Column(nullable = false)

private boolean escrowAttached;

@Column(nullable = false)

private boolean pmiAttached;

@ManyToOne(fetch = FetchType.LAZY)

@JoinColumn(name = "customer\_id", nullable = false)

@JsonBackReference

Customer customer;

}

1.30– Generate the following for the Account class

* Getter and Setters
* hashCode
* equals
* toString
* A non-default constructor

1.31– Create AccountType Domain class in the domain package

@Entity

@Table(name = "rsmortgage\_account\_type")

@XmlRootElement

@XmlAccessorType(XmlAccessType.FIELD)

public class AccountType {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private long id;

@Column(nullable = false)

private String accountTypeName;

@Column(nullable = false)

private String accountTypeDescription;

}

1.32– Generate the following for the AccountType class

* Press CTRL+Shift+O [Command + Shift + O in Mac] to import
* Choose java.persistence package
* Getter and Setters
* hashCode
* equals
* toString
* A non-default constructor

1.33– Create the Address Domain class in the domain package

package com.rollingstone.domain;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.FetchType;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.JoinColumn;

import javax.persistence.ManyToOne;

import javax.persistence.Table;

import javax.xml.bind.annotation.XmlAccessType;

import javax.xml.bind.annotation.XmlAccessorType;

import javax.xml.bind.annotation.XmlRootElement;

import com.fasterxml.jackson.annotation.JsonBackReference;

@Entity

@Table(name = "rsmortgage\_address")

@XmlRootElement

@XmlAccessorType(XmlAccessType.FIELD)

public class Address {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private long id;

@Column(nullable = false)

private String streetAddress;

@Column(nullable = false)

private String state;

@Column(nullable = false)

private String city;

@Column(nullable = false)

private String zipCode;

@Column(nullable = false)

private String country;

@Column(nullable = false)

private boolean isCurrentAddress;

@Column(nullable = false)

private boolean isMailingAddress;

@Column(nullable = false)

private boolean isBillingAddress;

@Column(nullable = false)

private boolean isPermanentResidence;

@Column(nullable = false)

private boolean isInvestmentProperty;

@ManyToOne(fetch = FetchType.LAZY)

@JoinColumn(name = "customer\_id", nullable = false)

@JsonBackReference

Customer customer;

}

1.34– Do the following to the Address Class

* Press CTRL+Shift+O [Command + Shift + O in Mac] to import
* Choose java.persistence package
* Generate Getter Setter
* Generate Constructor using Fields
* Generate toString
* hashCode
* equals
* toString

1.35--Create the Address Domain class in the domain package

package com.rollingstone.domain;

/\*

\* A Contact POJO serving as an Entity as well as a Data Transfer Object i.e DTO

\*/

@Entity

@Table(name = "rsmortgage\_contact")

@XmlRootElement

@XmlAccessorType(XmlAccessType.FIELD)

public class Contact {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private long id;

@OneToOne

@JoinColumn(name="contact\_type\_id")

private ContactType contactType;

@Temporal(TemporalType.DATE)

@Column(name = "date\_created", unique = true, nullable = false, length = 10)

private Date dateCreated;

@ManyToOne(fetch = FetchType.LAZY)

@JoinColumn(name = "customer\_id", nullable = false)

@JsonBackReference

Customer customer;

@Column(nullable = true)

private String emailAddress;

@Column(nullable = true)

private String phoneNumber;

@Column(nullable = true)

private String twitterHandles;

@Column(nullable = true)

private String faceBookId;

1.36– Do the following to the Contact Class

* Press CTRL+Shift+O [Command + Shift + O in Mac] to import
* Choose java.persistence package
* Generate Getter Setter
* Generate Constructor using Fields
* Generate toString
* hashCode
* equals
* toString

1.37--Create the ContactType Domain class in the domain package

@Entity

@Table(name = "rsmortgage\_contact\_type")

@XmlRootElement

@XmlAccessorType(XmlAccessType.FIELD)

public class ContactType {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private long id;

@Column(nullable = false)

private String contactTypeName;

@Column(nullable = false)

private String contactTypeDescription;

}

1.38– Do the following to the ContactType Class

* Press CTRL+Shift+O [Command + Shift + O in Mac] to import
* Choose java.persistence package
* Generate Getter Setter
* Generate Constructor using Fields
* Generate toString
* hashCode
* equals
* toString

1.39--Create the Customer Domain class in the domain package

/\*

\* A Customer POJO serving as an Entity as well as a Data Transfer Object i.e DTO

\*/

@Entity

@Table(name = "rsmortgage\_customer")

@XmlRootElement

@XmlAccessorType(XmlAccessType.FIELD)

public class Customer {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private long id;

@Column(nullable = false)

private String firstName;

@Column(nullable = false)

private String lastName;

@Column(nullable = false)

private String socialSecurityNumber;

@Temporal(TemporalType.DATE)

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

private Date dateOfBirth;

@Column(nullable = false)

private double totalLoanAmount;

@Column(nullable = false)

private int bonusPoints;

@Temporal(TemporalType.DATE)

@Column(name = "customer\_since", unique = true, nullable = false, length = 10)

private Date memberSince;

@OneToMany(fetch = FetchType.LAZY, mappedBy = "customer")

@JsonManagedReference

private Set<Address> addresses = new HashSet<Address>();

@OneToMany(fetch = FetchType.LAZY, mappedBy = "customer")

@JsonManagedReference

private Set<Account> accounts = new HashSet<Account>();

@OneToMany(fetch = FetchType.LAZY, mappedBy = "customer")

@JsonManagedReference

private Set<Contact> contacts = new HashSet<Contact>();

@OneToMany(fetch = FetchType.LAZY, mappedBy = "customer")

@JsonManagedReference

private Set<Education> education = new HashSet<Education>();

@OneToMany(fetch = FetchType.LAZY, mappedBy = "customer")

@JsonManagedReference

private Set<Employment> employment = new HashSet<Employment>();

@OneToMany(fetch = FetchType.LAZY, mappedBy = "customer")

@JsonManagedReference

private Set<Investment> investments = new HashSet<Investment>();

@OneToMany(fetch = FetchType.LAZY, mappedBy = "customer")

@JsonManagedReference

private Set<Liability> liabilities = new HashSet<Liability>();

@Column()

private int rating;

}

1.40– Do the following to the Customer Class

* Press CTRL+Shift+O [Command + Shift + O in Mac] to import
* Choose java.persistence package
* Generate Getter Setter
* Generate Constructor using Fields
* Generate toString
* hashCode
* equals
* toString

1.41--Create the DegreeType Domain class in the domain package

@Entity

@Table(name = "rsmortgage\_degree\_type")

@XmlRootElement

@XmlAccessorType(XmlAccessType.FIELD)

public class DegreeType {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private long id;

@Column(nullable = false)

private String degreeTypeName;

@Column(nullable = false)

**private** String degreeTypeDescription;

}

1.42– Do the following to the DegreeType Class

* Press CTRL+Shift+O [Command + Shift + O in Mac] to import
* Choose java.persistence package
* Generate Getter Setter
* Generate Constructor using Fields
* Generate toString
* hashCode
* equals
* toString

1.43--Create the Education Domain class in the domain package

@Entity

@Table(name = "rsmortgage\_education")

@XmlRootElement

@XmlAccessorType(XmlAccessType.FIELD)

public class Education {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private long id;

@Temporal(TemporalType.DATE)

@Column(name = "date\_from", unique = true, nullable = false, length = 10)

private Date fromDate;

@Temporal(TemporalType.DATE)

@Column(name = "date\_to", unique = true, nullable = false, length = 10)

private Date dateTo;

@Column(nullable = false)

private boolean isCurrentSchool;

@Column(nullable = false)

private boolean didGraduate;

@Column(nullable = false)

private float cumulativeGpa;

@Column(nullable = false)

private String schoolName;

@OneToOne

@JoinColumn(name="degree\_type\_id")

private DegreeType degreeType;

@Column(nullable = false)

private String schoolAdminPerson;

@Column(nullable = false)

private String schoolAdminPhone;

@Column(nullable = false)

private String schoolAdminEmail;

@Column(nullable = false)

private String schoolAdminFax;

@Column(nullable = false)

private String schoolAddressLine1;

@Column(nullable = false)

private String schoolAddressLine2;

@Column(nullable = false)

private String schoolAddressCity;

@Column(nullable = false)

private String schoolAddressState;

@Column(nullable = false)

private String schoolAddressCountry;

@ManyToOne(fetch = FetchType.LAZY)

@JoinColumn(name = "customer\_id", nullable = false)

@JsonBackReference

Customer customer;

}

1.44– Do the following to the Education Class

* Press CTRL+Shift+O [Command + Shift + O in Mac] to import
* Choose java.persistence package
* Generate Getter Setter
* Generate Constructor using Fields
* Generate toString
* hashCode
* equals
* toString

1.45--Create the Employment Domain class in the domain package

@Entity

@Table(name = "rsmortgage\_employment")

@XmlRootElement

@XmlAccessorType(XmlAccessType.FIELD)

public class Employment {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private long id;

@Temporal(TemporalType.DATE)

@Column(name = "date\_from", unique = true, nullable = false, length = 10)

private Date fromDate;

@Temporal(TemporalType.DATE)

@Column(name = "date\_to", unique = true, nullable = false, length = 10)

private Date dateTo;

@Column(nullable = false)

private float numYears;

@Column(nullable = false)

private float grossSalary;

@Column(nullable = false)

private float netSalary;

@Column(nullable = false)

private boolean isCurrentEmployer;

@Column(nullable = false)

private String jobTitle;

@Column(nullable = false)

private String jobDescription;

@Column(nullable = false)

private String employerName;

@Column(nullable = false)

private String employmentType;

@Column(nullable = false)

private String employerHRPerson;

@Column(nullable = false)

private String employerHRPhone;

@Column(nullable = false)

private String employerHREmail;

@Column(nullable = false)

private String employerHRFax;

@Column(nullable = false)

private String employerAddressLine1;

@Column(nullable = false)

private String employerAddressLine2;

@Column(nullable = false)

private String employerAddressCity;

@Column(nullable = false)

private String employerAddressState;

@Column(nullable = false)

private String employerAddressCountry;

@ManyToOne(fetch = FetchType.LAZY)

@JoinColumn(name = "customer\_id", nullable = false)

@JsonBackReference

Customer customer;

}

1.46– Do the following to the Employment Class

* Press CTRL+Shift+O [Command + Shift + O in Mac] to import
* Choose java.persistence package
* Generate Getter Setter
* Generate Constructor using Fields
* Generate toString
* hashCode
* equals
* toString

1.47--Create the Investment Domain class in the domain package

@Entity

@Table(name = "rsmortgage\_investment")

@XmlRootElement

@XmlAccessorType(XmlAccessType.FIELD)

public class Investment {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private long id;

@Temporal(TemporalType.DATE)

@Column(name = "date\_from", unique = true, nullable = false, length = 10)

private Date fromDate;

@Temporal(TemporalType.DATE)

@Column(name = "maturity\_date", unique = true, nullable = false, length = 10)

private Date dateMaturing;

@OneToOne

@JoinColumn(name="investment\_type\_id")

private InvestmentType investmentType;

@Column(nullable = false)

private double currentValue;

@Column(nullable = false)

private double investedValue;

@Column(nullable = false)

private float monthlyIncome;

@ManyToOne(fetch = FetchType.LAZY)

@JoinColumn(name = "customer\_id", nullable = false)

@JsonBackReference

Customer customer;

}

1.48– Do the following to the Investment Class

* Press CTRL+Shift+O [Command + Shift + O in Mac] to import
* Choose java.persistence package
* Generate Getter Setter
* Generate Constructor using Fields
* Generate toString
* hashCode
* equals
* toString

1.49--Create the InvestmentType class in the domain package

@Entity

@Table(name = "rsmortgage\_investment\_type")

@XmlRootElement

@XmlAccessorType(XmlAccessType.FIELD)

public class InvestmentType {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private long id;

@Column(nullable = false)

private String invetmentTypeName;

@Column(nullable = false)

**private** String investmentTypeDescription;

}

1.50– Do the following to the InvestmentType Class

* Press CTRL+Shift+O [Command + Shift + O in Mac] to import
* Choose java.persistence package
* Generate Getter Setter
* Generate Constructor using Fields
* Generate toString
* hashCode
* equals
* toString

1.51--Create the Liability Domain class in the domain package

@Entity

@Table(name = "rsmortgage\_liability")

@XmlRootElement

@XmlAccessorType(XmlAccessType.FIELD)

public class Liability {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private long id;

@Temporal(TemporalType.DATE)

@Column(name = "date\_from", unique = true, nullable = false, length = 10)

private Date fromDate;

@Temporal(TemporalType.DATE)

@Column(name = "maturity\_date", unique = true, nullable = false, length = 10)

private Date dateMaturing;

@OneToOne

@JoinColumn(name="liability\_type\_id")

private LiabilityType investmentType;

@Column(nullable = false)

private double originalTotalLiability;

@Column(nullable = false)

private double currentTotalLiability;

@Column(nullable = false)

private String paymentFrequency;

@Column(nullable = false)

private float periodEMI;

@ManyToOne(fetch = FetchType.LAZY)

@JoinColumn(name = "customer\_id", nullable = false)

@JsonBackReference

Customer customer;

}

1.52– Do the following to the Liability Class

* Press CTRL+Shift+O [Command + Shift + O in Mac] to import
* Choose java.persistence package
* Generate Getter Setter
* Generate Constructor using Fields
* Generate toString
* hashCode
* equals
* toString

1.53--Create the LiabilityType Domain class in the domain package

@Entity

@Table(name = "rsmortgage\_liability\_type")

@XmlRootElement

@XmlAccessorType(XmlAccessType.FIELD)

public class LiabilityType {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private long id;

@Column(nullable = false)

private String liabilityTypeName;

@Column(nullable = false)

private String liabilityTypeDescription;

}

1.54– Do the following to the LiabilityType Class

* Press CTRL+Shift+O [Command + Shift + O in Mac] to import
* Choose java.persistence package
* Generate Getter Setter
* Generate Constructor using Fields
* Generate toString
* hashCode
* equals
* toString

1.55--Create the RestAPIExceptionInfo class 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;

}

}

1.56– 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);

}

}

1.57– 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);

}

}

1.58– 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.Customer;

public interface RsMortgageCustomerRepository extends PagingAndSortingRepository<Customer, Long> {

Customer findCustomerByRating(int rating);

Page findAll(Pageable pageable);

}

1.59– Generate Service class in the service package

package com.rollingstone.service;

import java.util.ArrayList;

import java.util.List;

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.RsMortgageCustomerRepository;

import com.rollingstone.domain.Customer;

/\*

\* Service class to do CRUD for Customer and Address through JPS Repository

\*/

@Service

public class RsMortgageCustomerService {

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

@Autowired

private RsMortgageCustomerRepository customerRepository;

@Autowired

CounterService counterService;

@Autowired

GaugeService gaugeService;

public RsMortgageCustomerService() {

}

public Customer createCustomer(Customer customer) {

if (customer.getDateOfBirth() != null){

log.info("Customer Date of Birth :"+customer.getDateOfBirth());

}else {

log.info("Customer Date of Birth is null :");

}

if (customer.getDateOfBirth() != null){

log.info("Customer Date of Birth :"+customer.getDateOfBirth());

}else {

log.info("Customer Date of Birth is null :");

}

return customerRepository.save(customer);

}

public Customer getCustomer(long id) {

return customerRepository.findOne(id);

}

public void updateCustomer(Customer customer) {

customerRepository.save(customer);

}

public void deleteCustomer(Long id) {

customerRepository.delete(id);

}

//http://goo.gl/7fxvVf

public Page<Customer> getAllCustomersByPage(Integer page, Integer size) {

Page pageOfCustomers = customerRepository.findAll(new PageRequest(page, size));

// example of adding to the /metrics

if (size > 50) {

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

}

return pageOfCustomers;

}

public List<Customer> getAllCustomers() {

Iterable<Customer> pageOfCustomers = customerRepository.findAll();

List<Customer> customers = new ArrayList<Customer>();

for (Customer customer : pageOfCustomers){

customers.add(customer);

}

log.info("In Real Service getAllCustomers size :"+customers.size());

return customers;

}

*}*

1.60– Generate 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 = "customer.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;

}

}

1.61– 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 UserServiceHealth 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();

}

}

1.62– 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 UserServiceEvent extends ApplicationEvent {

public UserServiceEvent(Object source) {

super(source);

}

public String toString() {

return "My UserService Event";

}

}

1.63– 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;

}

}

1.64– Generate CustomerControllerclass in the rest.api package

package com.rollingstone.api.rest;

import java.util.List;

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.Customer;

import com.rollingstone.exception.HTTP400Exception;

import com.rollingstone.service.RsMortgageCustomerService;

/\*

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

\*/

@RestController

@RequestMapping(value = "/rsmortgage-customerservice/v1/customer")

public class CustomerController extends AbstractRestController {

@Autowired

private RsMortgageCustomerService customerService;

@RequestMapping(value = "",

method = RequestMethod.POST,

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

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

@ResponseStatus(HttpStatus.CREATED)

public void createCustomer(@RequestBody Customer customer,

HttpServletRequest request, HttpServletResponse response) {

Customer createdCustomer = this.customerService.createCustomer(customer);

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

}

@RequestMapping(value = "",

method = RequestMethod.GET,

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

@ResponseStatus(HttpStatus.OK)

public

@ResponseBody

Page<Customer> getAllCustomersByPage(@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.customerService.getAllCustomersByPage(page, size);

}

@RequestMapping(value = "/all",

method = RequestMethod.GET,

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

@ResponseStatus(HttpStatus.OK)

public

@ResponseBody

List<Customer> getAllCustomers(@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.customerService.getAllCustomers();

}

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

public Customer getSimpleCustomer(@PathVariable("id") Long id) {

Customer customer = this.customerService.getCustomer(id);

checkResourceFound(customer);

return customer;

}

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

method = RequestMethod.GET,

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

@ResponseStatus(HttpStatus.OK)

public

@ResponseBody

Customer getCustomer(@PathVariable("id") Long id,

HttpServletRequest request, HttpServletResponse response) throws Exception {

Customer customer = this.customerService.getCustomer(id);

checkResourceFound(customer);

return customer;

}

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

method = RequestMethod.PUT,

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

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

@ResponseStatus(HttpStatus.NO\_CONTENT)

public void updateCustomer(@PathVariable("id") Long id, @RequestBody Customer customer,

HttpServletRequest request, HttpServletResponse response) {

checkResourceFound(this.customerService.getCustomer(id));

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

this.customerService.updateCustomer(customer);

}

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

method = RequestMethod.DELETE,

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

@ResponseStatus(HttpStatus.NO\_CONTENT)

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

HttpServletResponse response) {

checkResourceFound(this.customerService.getCustomer(id));

this.customerService.deleteCustomer(id);

}

}

1.65– Generate RestControllerAspect in the rest.api 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);

}

}

1.66– Generate RsMortgageCustomerRestAPIApplication in the rest.api 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.cloud.client.discovery.EnableDiscoveryClient;

import org.springframework.cloud.netflix.feign.EnableFeignClients;

import org.springframework.context.annotation.ComponentScan;

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

import springfox.documentation.swagger2.annotations.EnableSwagger2;

/\*

\* 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 RsMortgageCustomerRestAPIApplication extends SpringBootServletInitializer {

private static final Class<RsMortgageCustomerRestAPIApplication> applicationClass = RsMortgageCustomerRestAPIApplication.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);

}

}

1.67– 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:

profiles:

active:

mysql

cloud:

config:

uri: http://localhost:9000

server:

port: 9001

eureka:

client:

serviceUrl:

defaultZone: http://eureka-host1:8761/eureka/,http://eureka-host2:8762/eureka/

spring.jmx:

enabled: **false**

spring.datasource:

driverClassName: com.mysql.jdbc.Driver

url: jdbc:mysql://localhost/rsmortgage;MODE=MySQL

#todo: make sure to always enable security in production

security:

basic:

enabled: **false**

#management endpoints on a separate port

management:

port: 9002

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}

1.68– Add bootstrap.yml file in the resources folder

spring:

application:

name: udemy-rsmortgage-customer-service

profiles:

active:

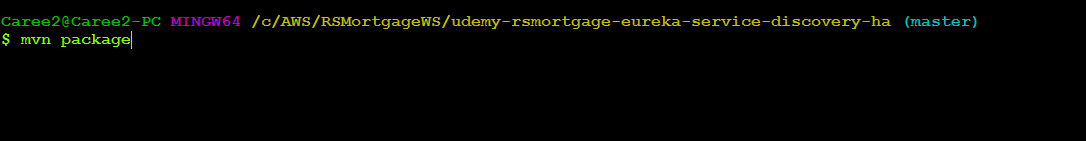
mysql

cloud:

config:

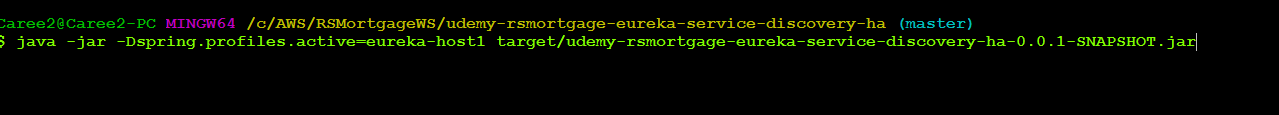
uri: http://localhost:9000

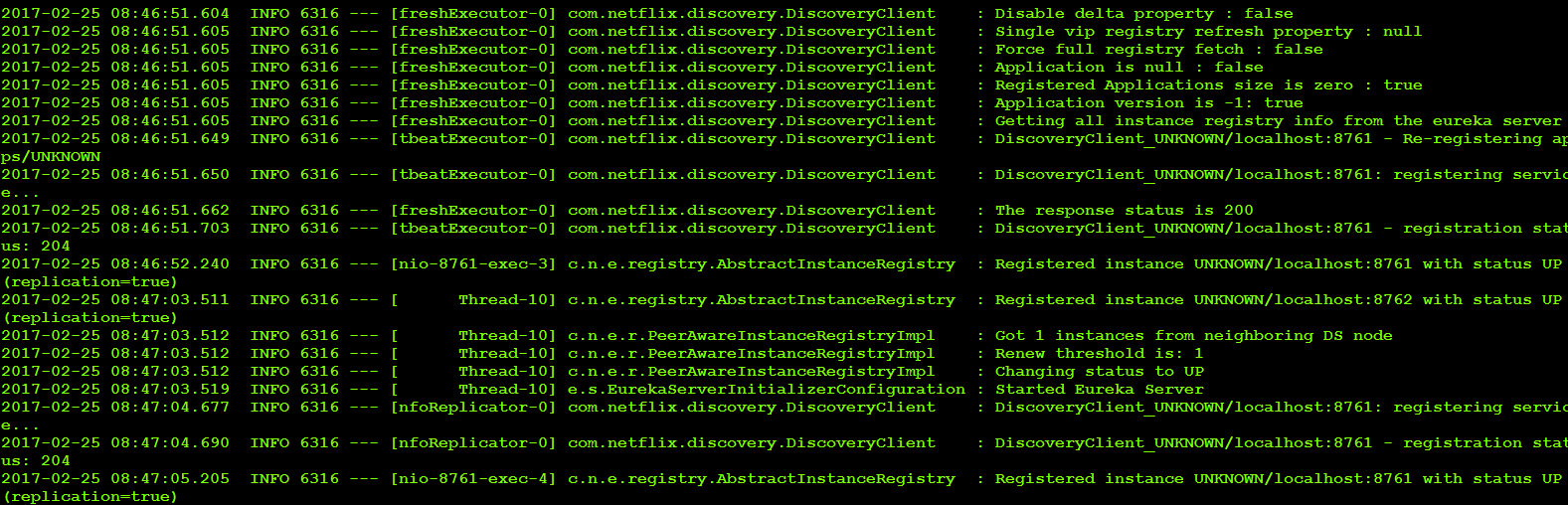
1.69 –Open Git Bash in project folder



1.70 –Run the first instance

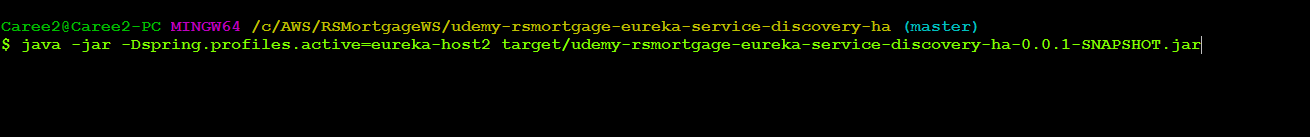
java -jar -Dspring.profiles.active=eureka-host1 target/udemy-rsmortgage-eureka-service-discovery-ha-0.0.1-SNAPSHOT.jar

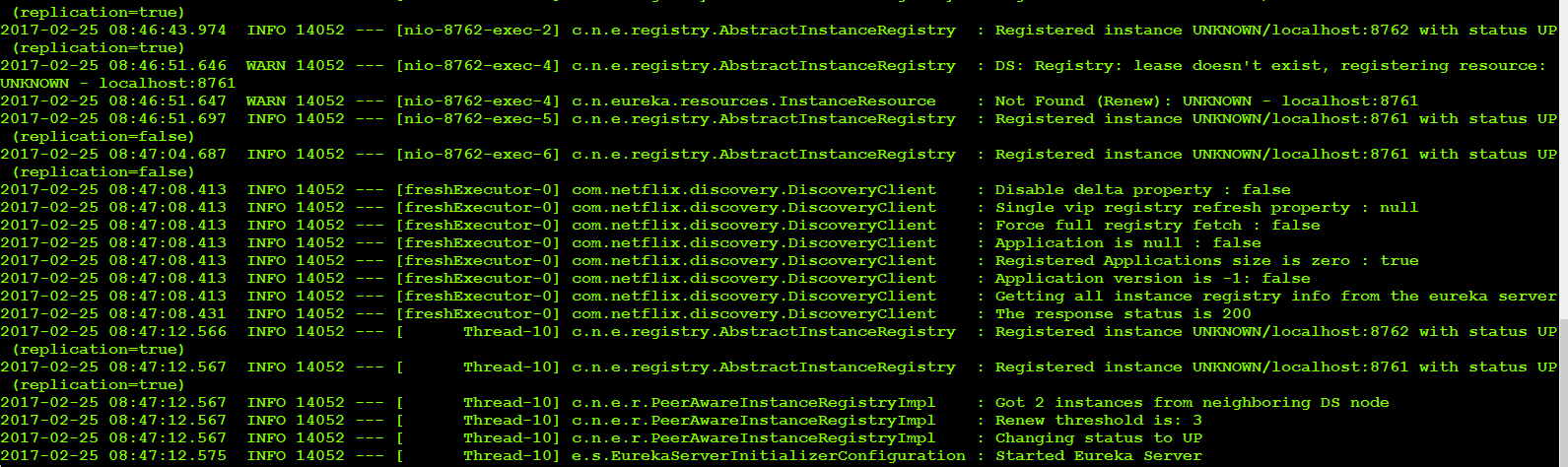




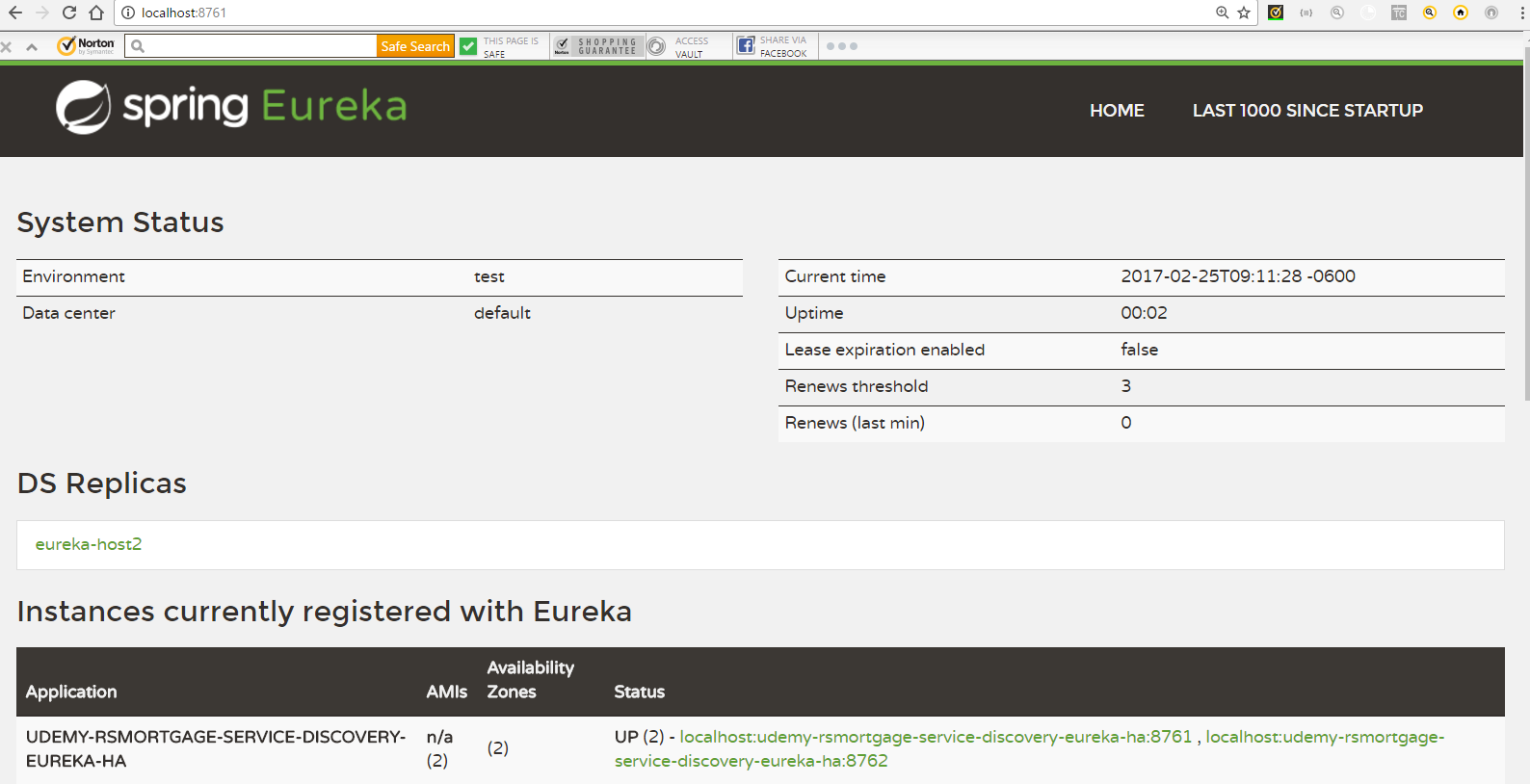
1.71 –Run the second instance

java -jar -Dspring.profiles.active=eureka-host2 target/udemy-rsmortgage-eureka-service-discovery-ha-0.0.1-SNAPSHOT.jar

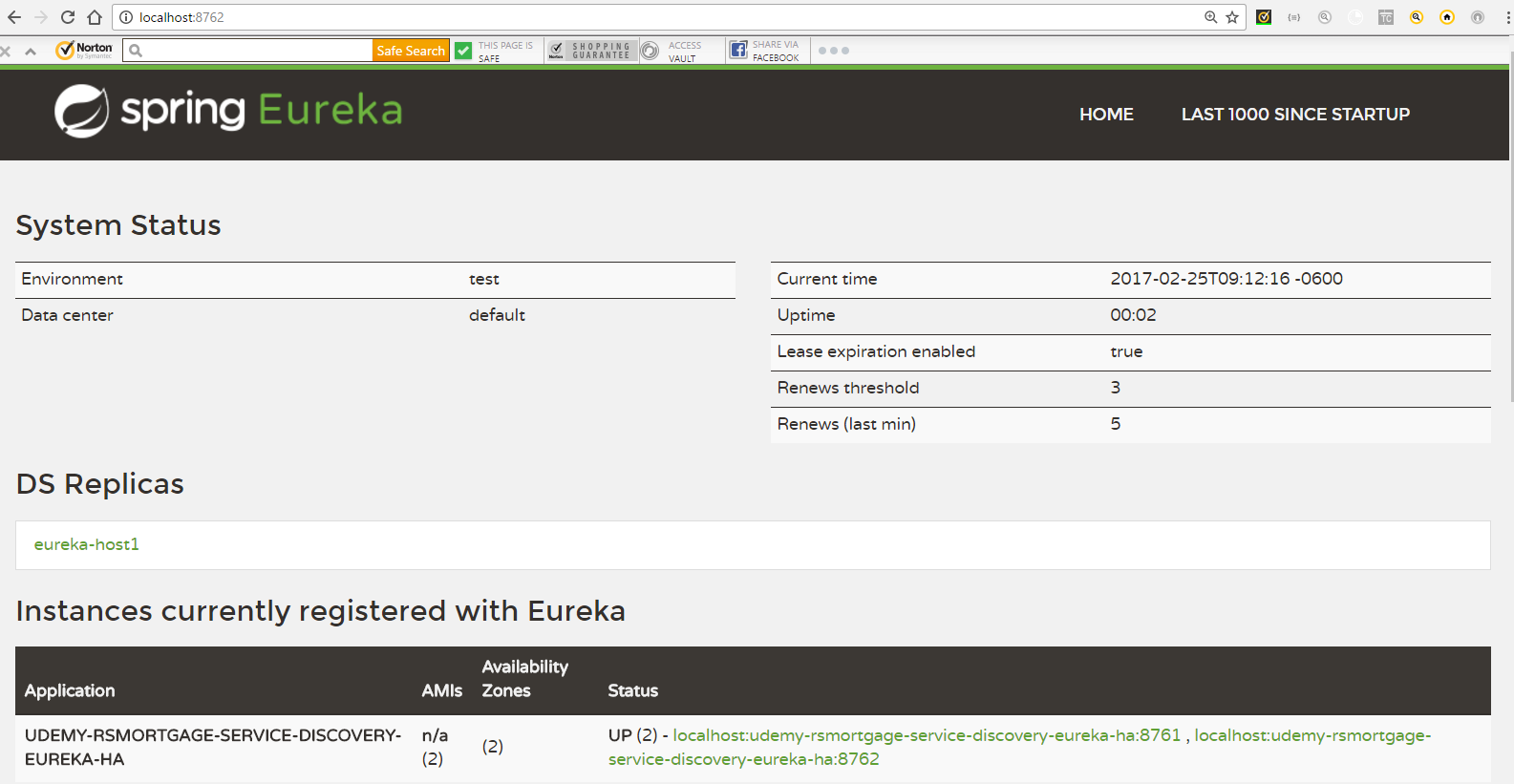




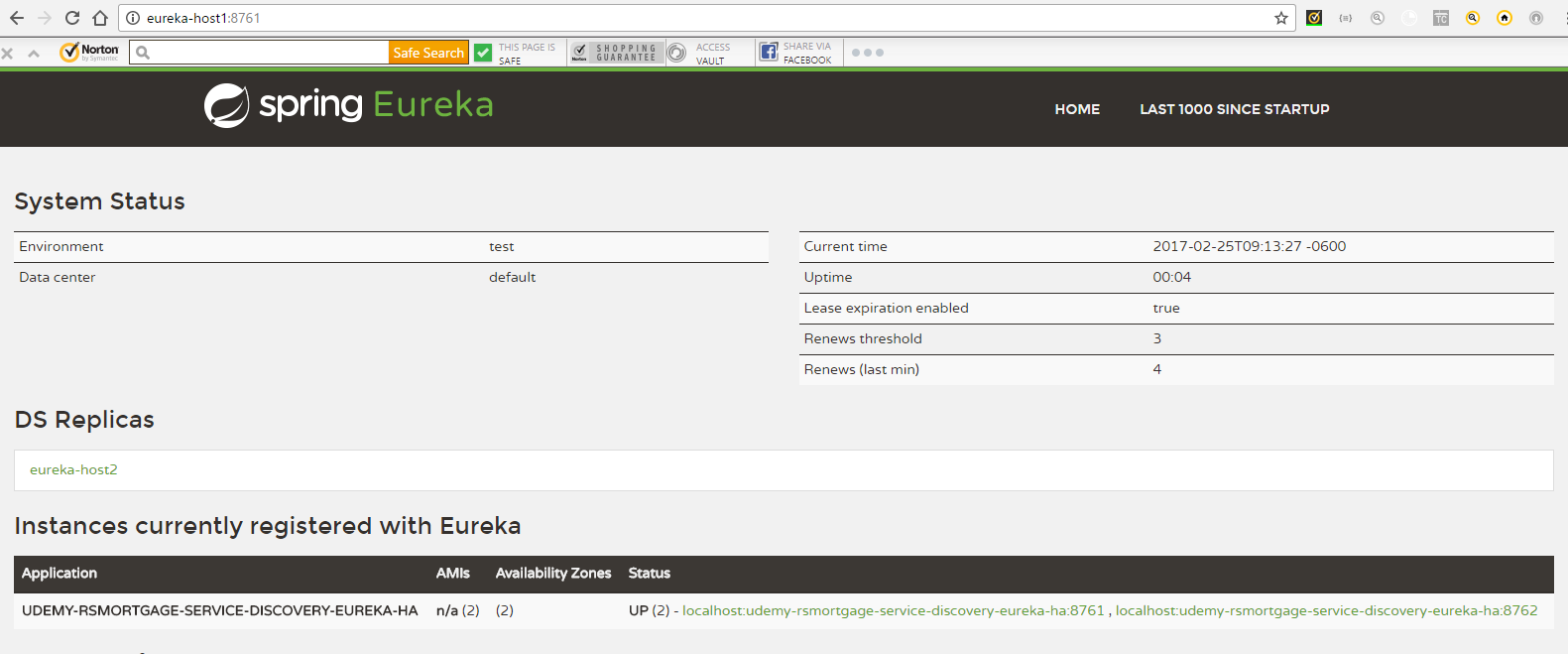
1.72 – Navigate to <http://localhost:8761>



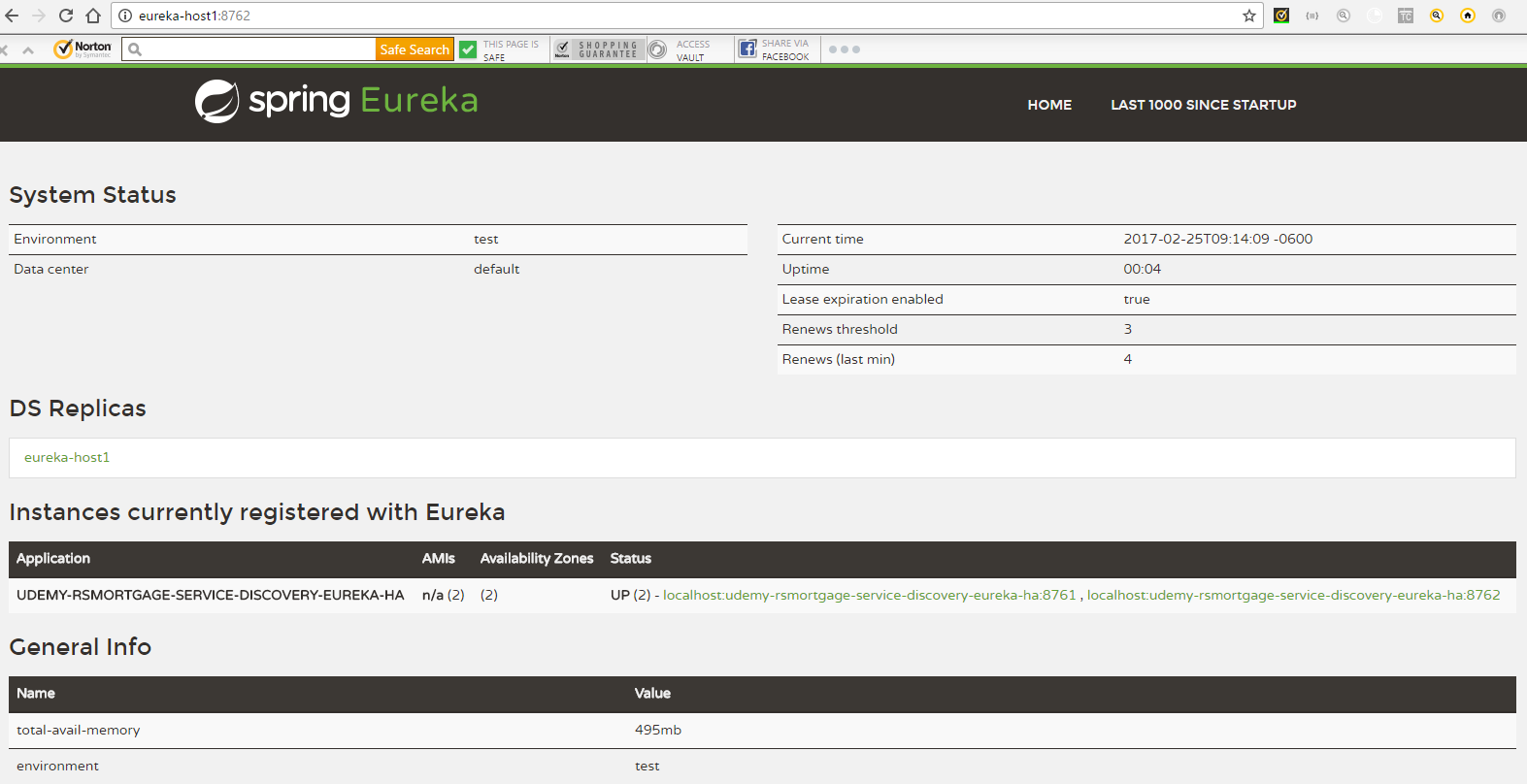
1.73 – Navigate to <http://localhost:8762>

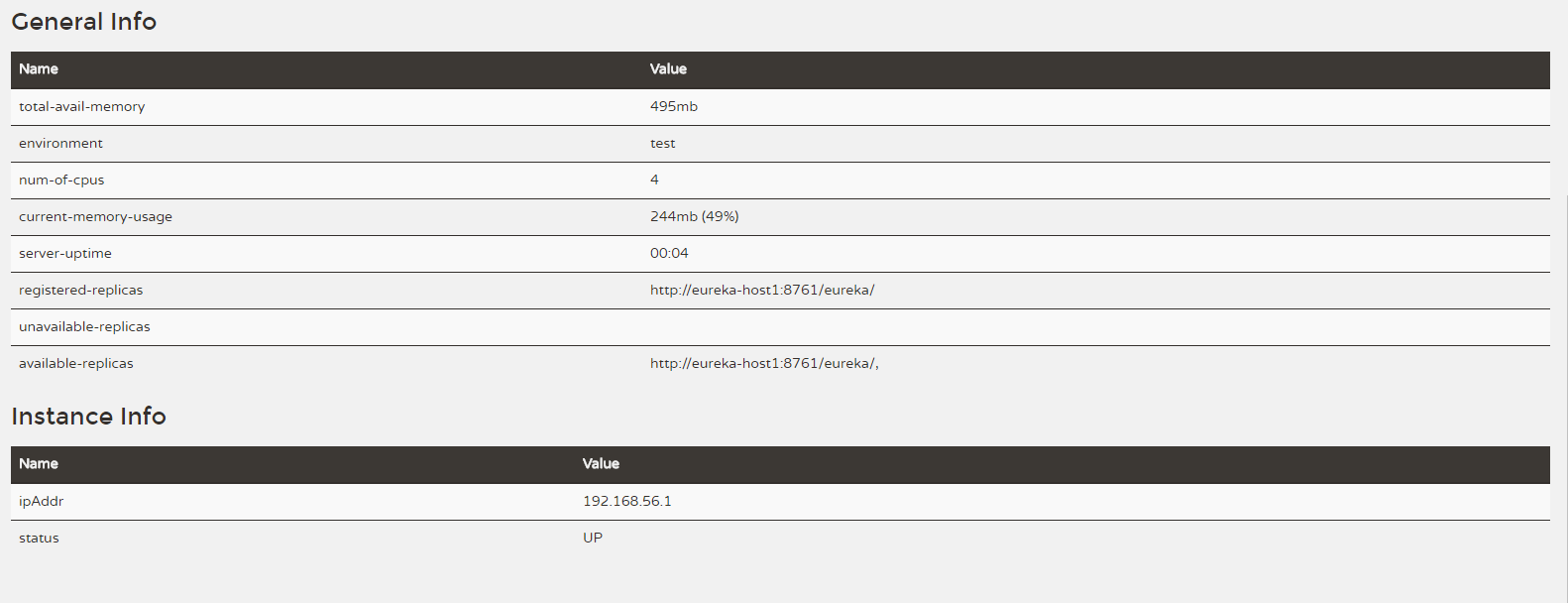


1.74 – Navigate to <http://eureka-host1:8761/>

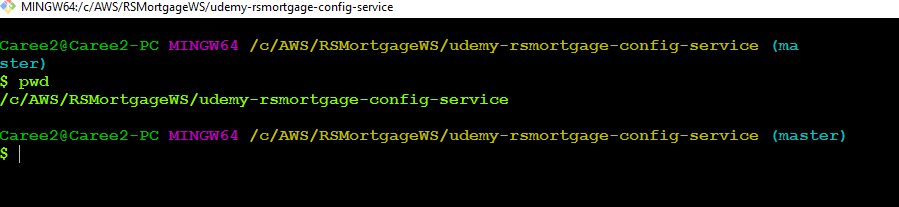


1.75 – Navigate to <http://eureka-host2:8762/>





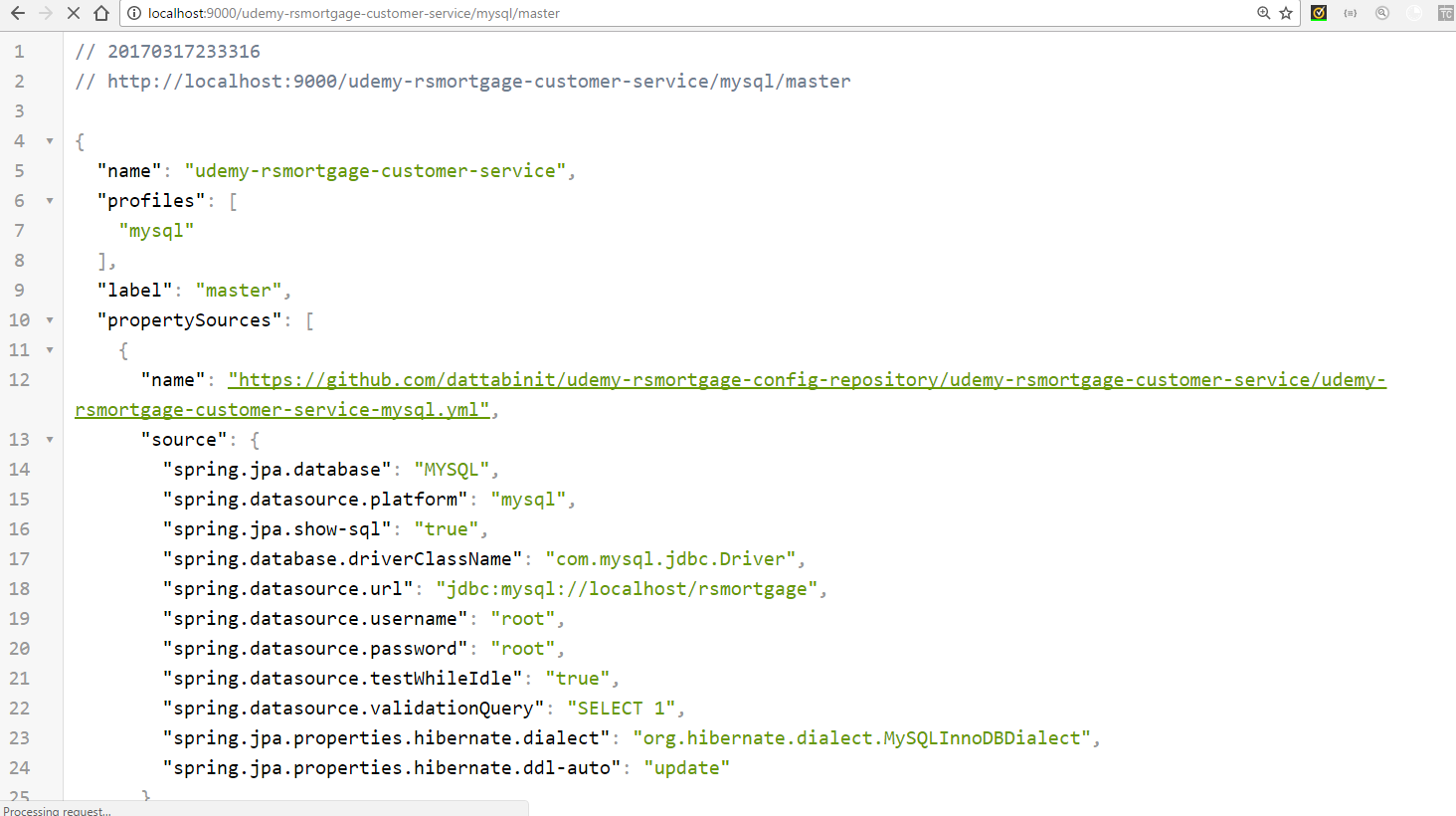
1.76– Open Git bash in the config project directory

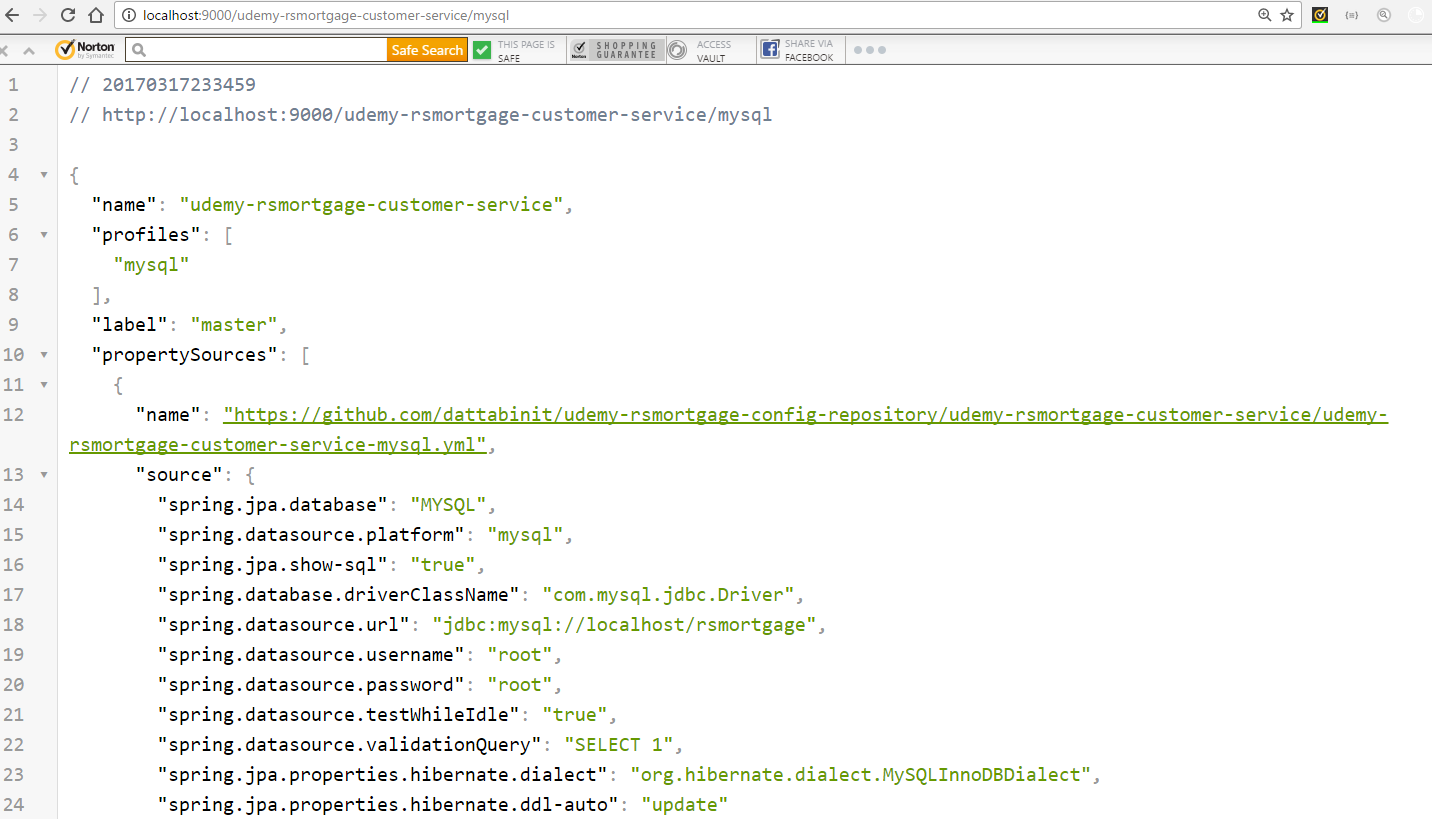


1.77– Run the config service project

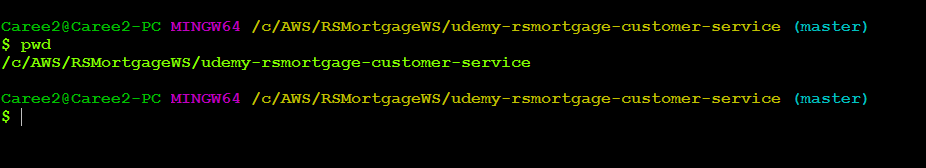
java -jar target/udemy-rsmortgage-config-service-0.0.1-SNAPSHOT.jar

1.78– Verify Customer Project mysql properties

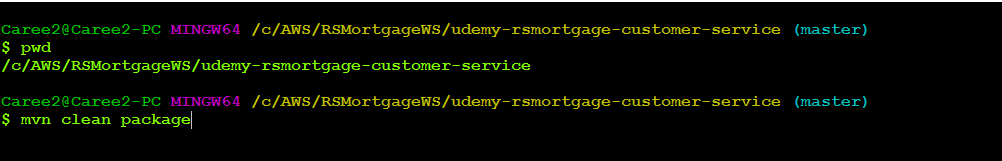


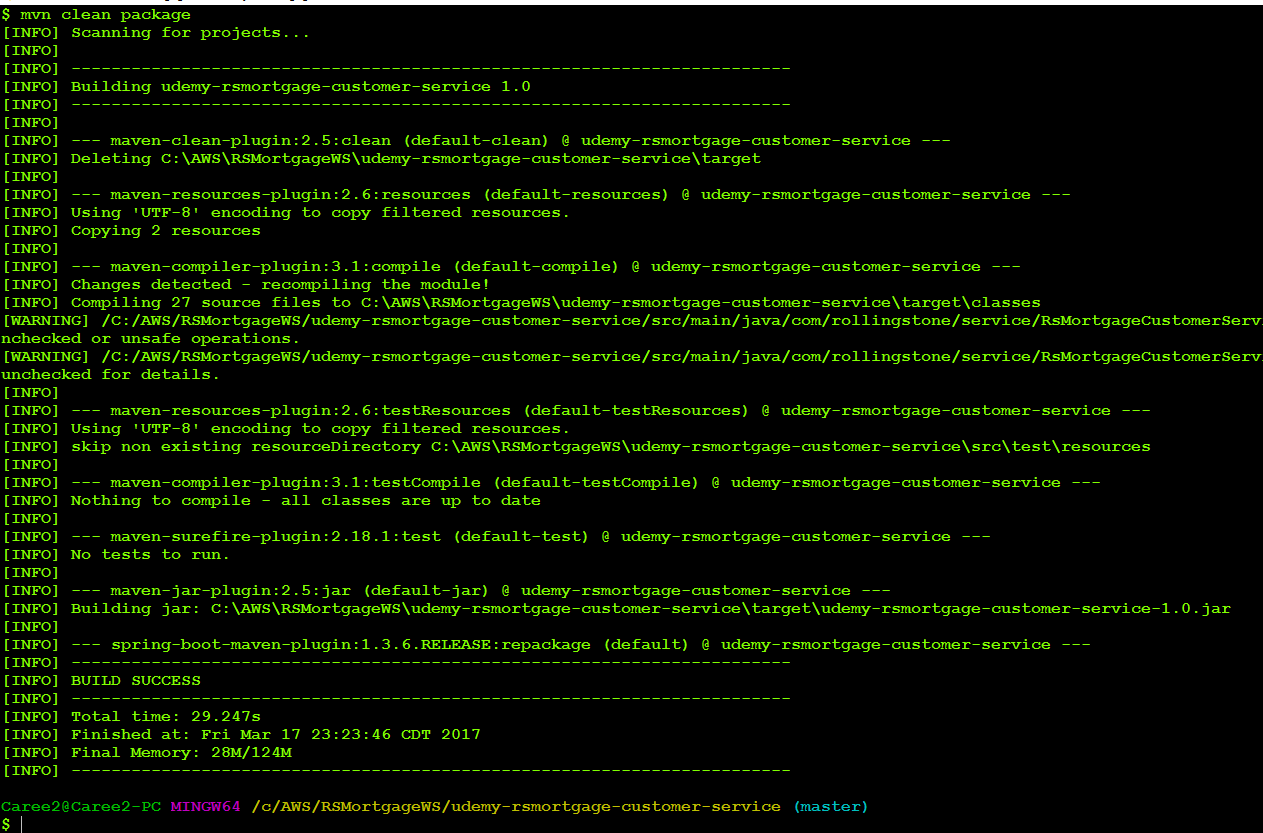


1.79– Open Git bash in the project directory



1.80– Build the project

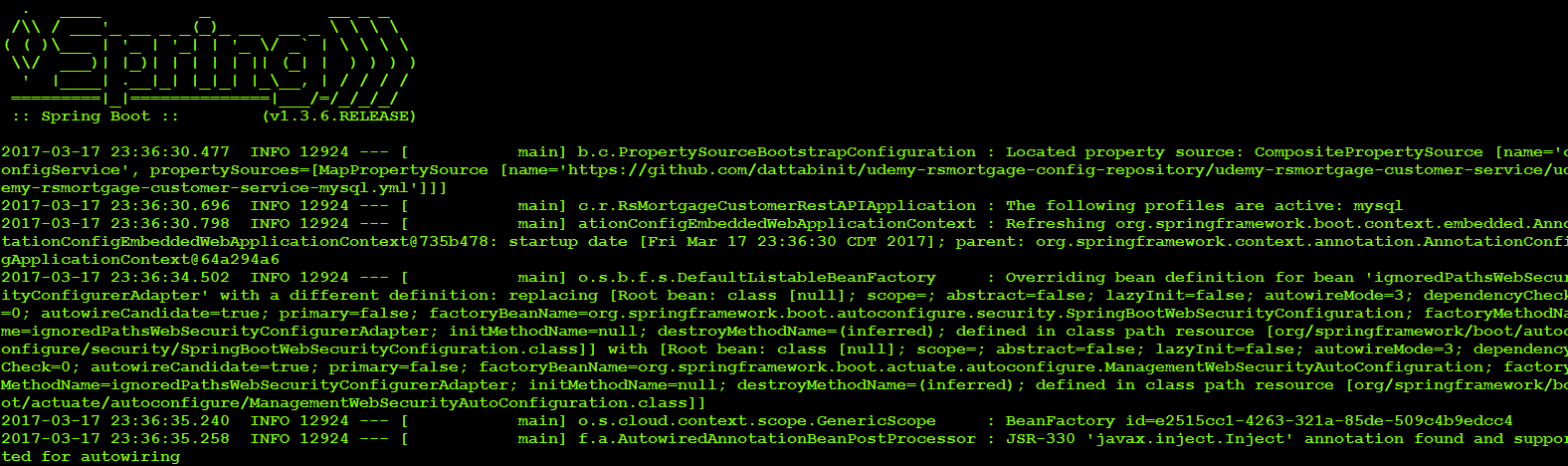




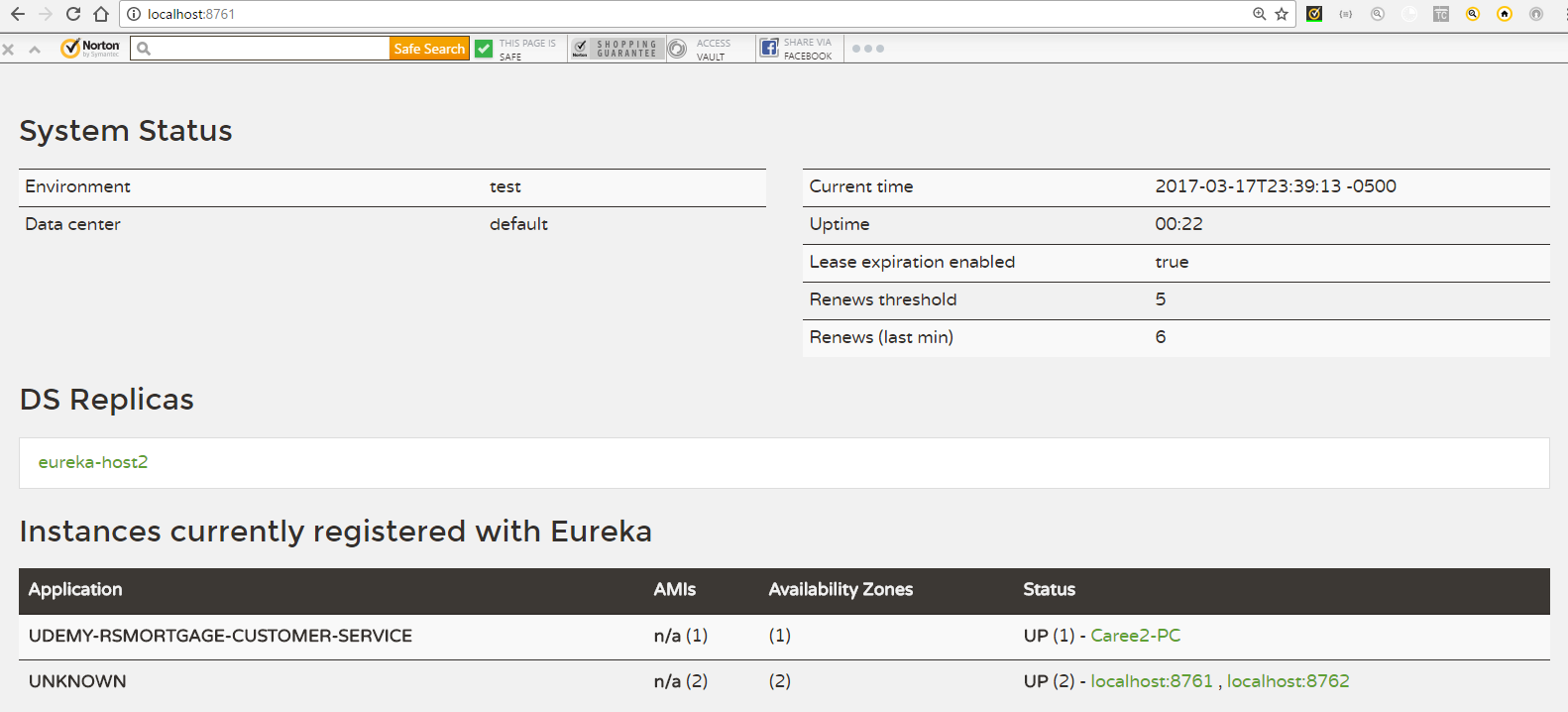
1.81 –Run the Project

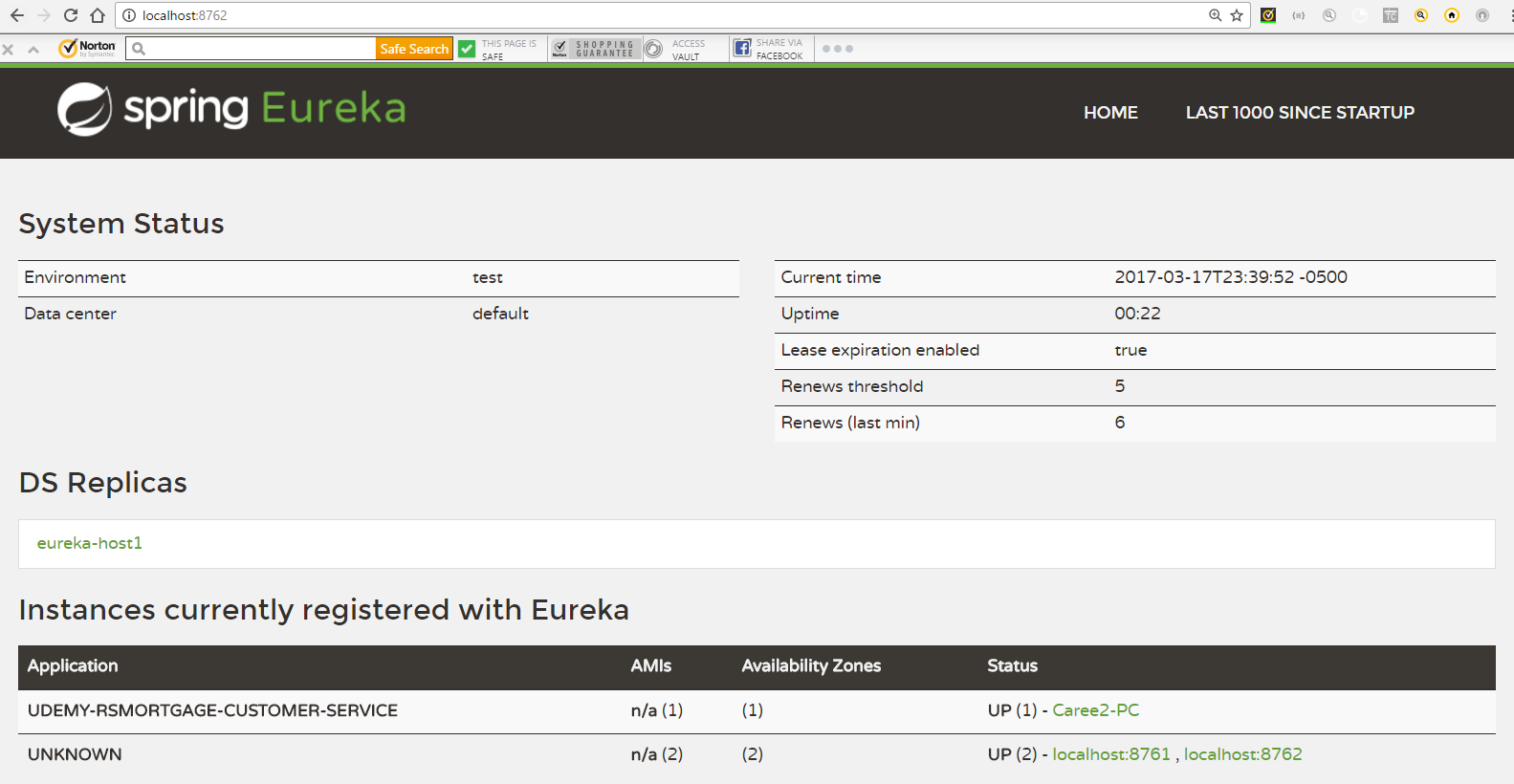
java -jar -Dspring.profiles.active=mysql target/udemy-rsmortgage-customer-service-1.0.jar

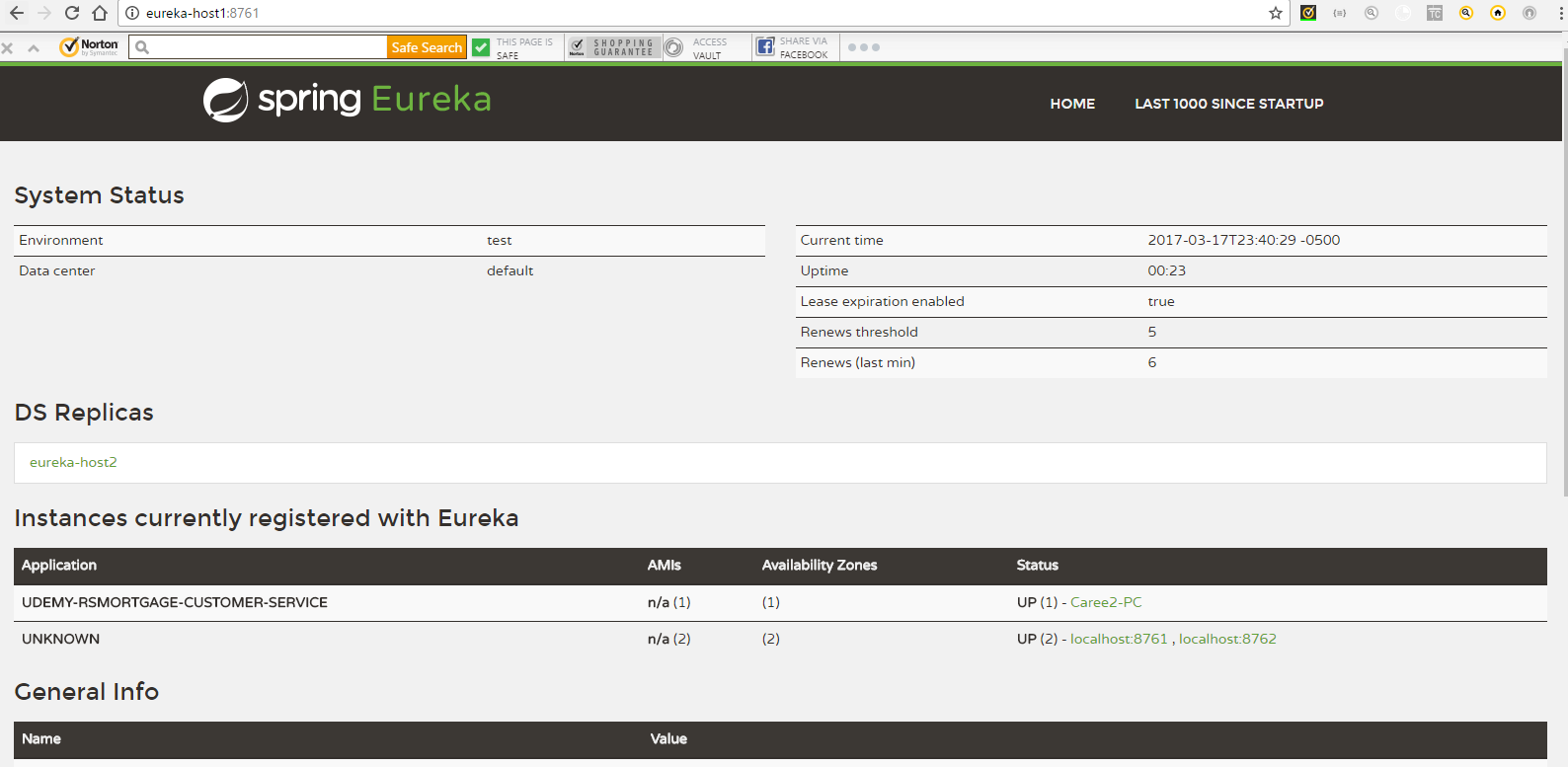
1.82 –Verify Config Property is read and used

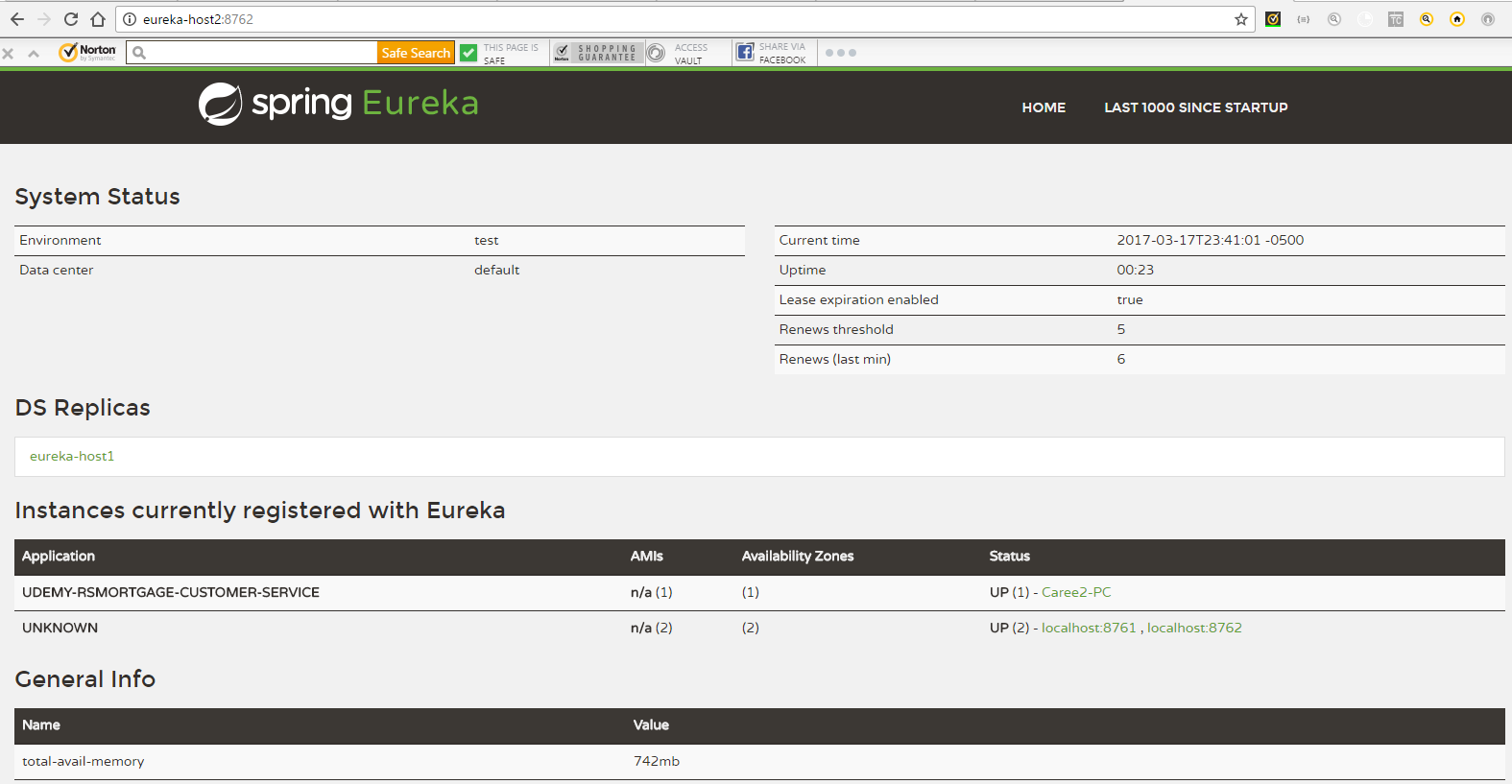


1.83 – Navigate to <http://localhost:8761>



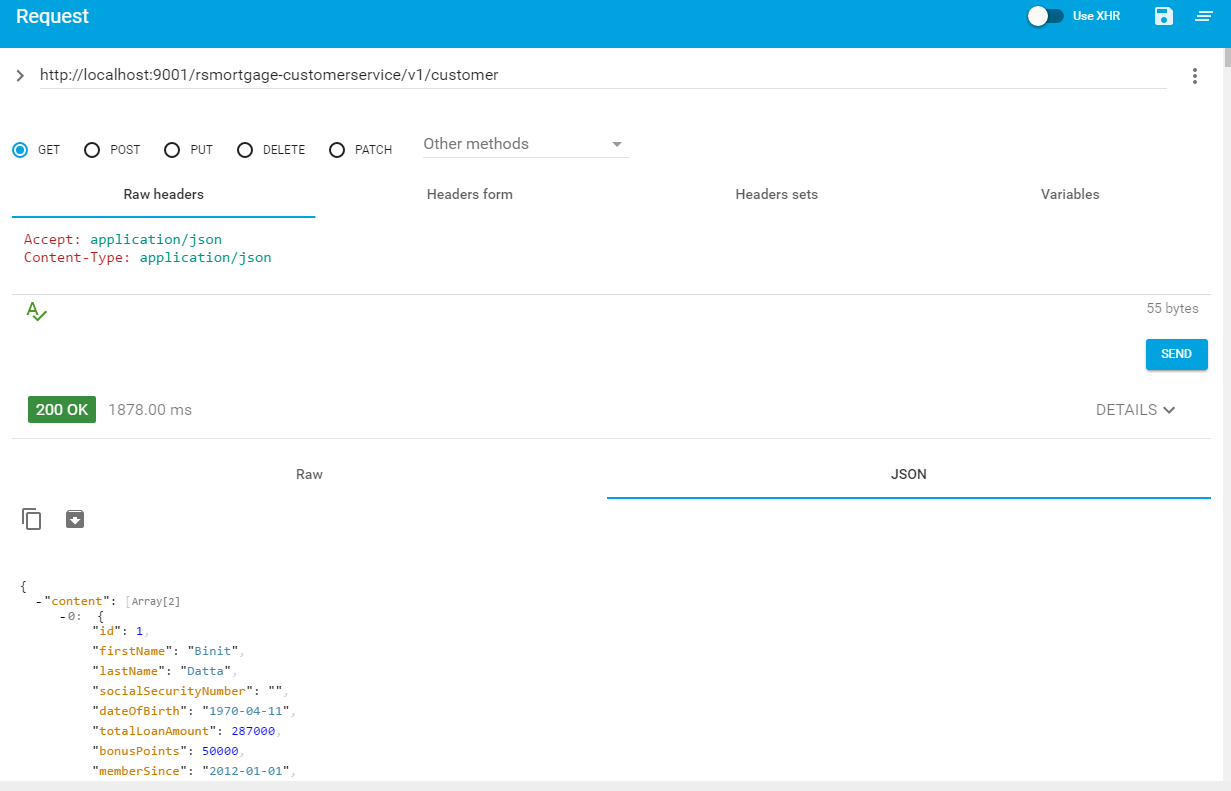






1.84—Get an existing Customer

<http://localhost:9001/rsmortgage-customerservice/v1/customer>



1.85– Create a Customer

<http://localhost:9001/rsmortgage-customerservice/v1/customer>

Payload

{

"firstName": "David",

"lastName": "Jackson",

"dateOfBirth": "1980-04-11",

"totalLoanAmount": 287000,

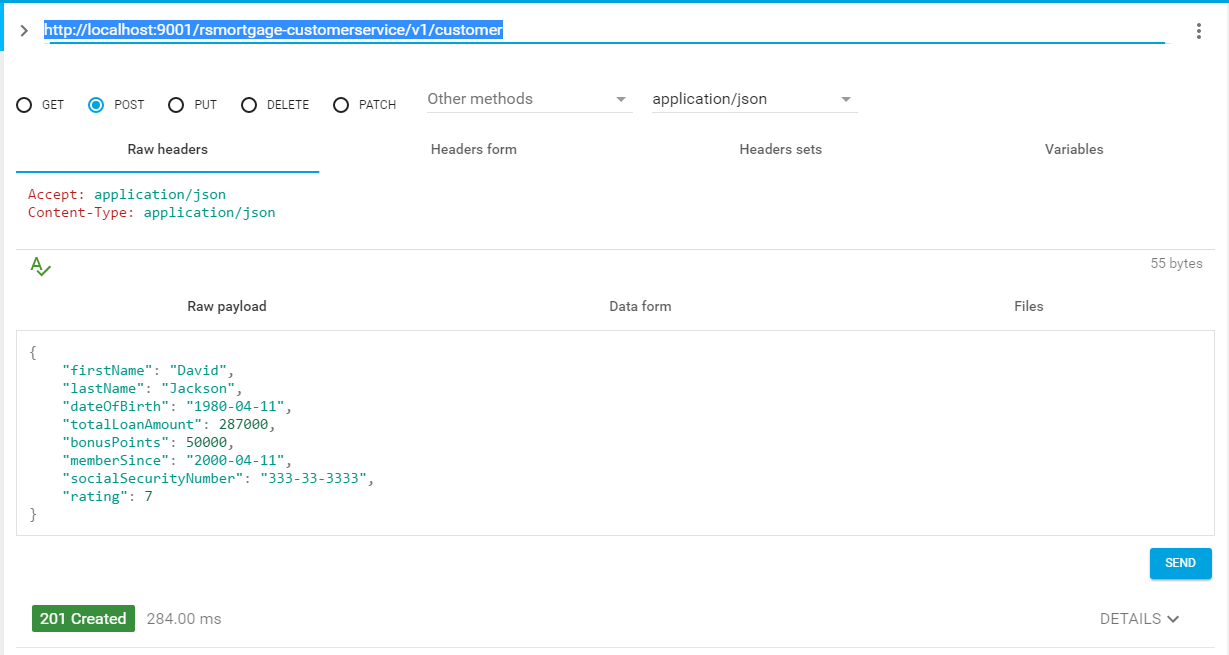
"bonusPoints": 50000,

"memberSince": "2000-04-11",

"socialSecurityNumber": "333-33-3333",

"rating": 7

}



1.86– Verify the Database



1.87—Try to Update a Record

<http://localhost:9001/rsmortgage-customerservice/v1/customer/3>

{

"id":3,

"firstName": "David",

"lastName": "Jackson",

"dateOfBirth": "1980-04-11",

"totalLoanAmount": 290000,

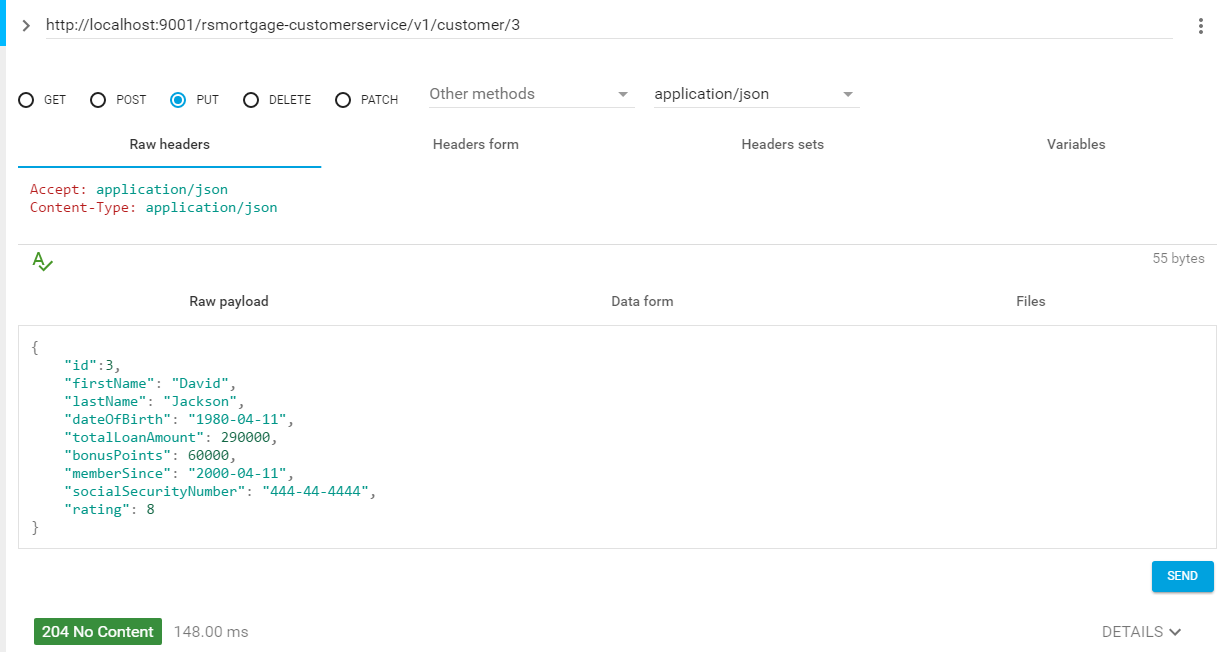
"bonusPoints": 60000,

"memberSince": "2000-04-11",

"socialSecurityNumber": "444-44-4444",

"rating": 8

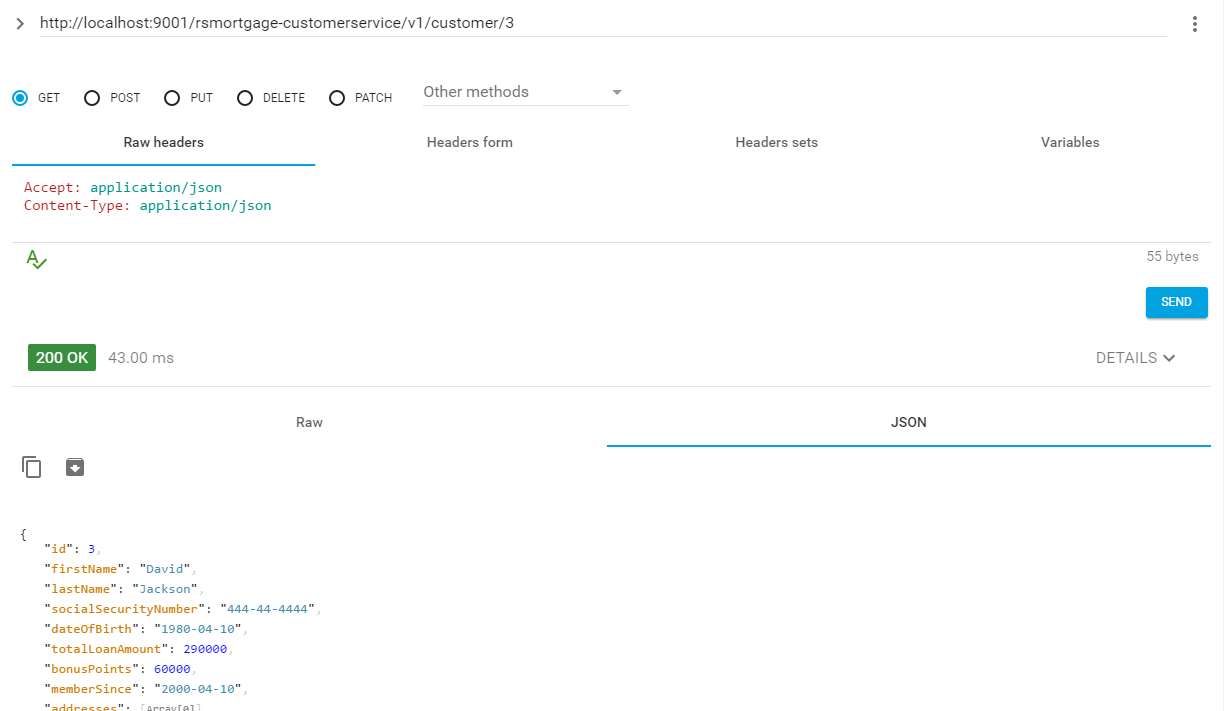
}



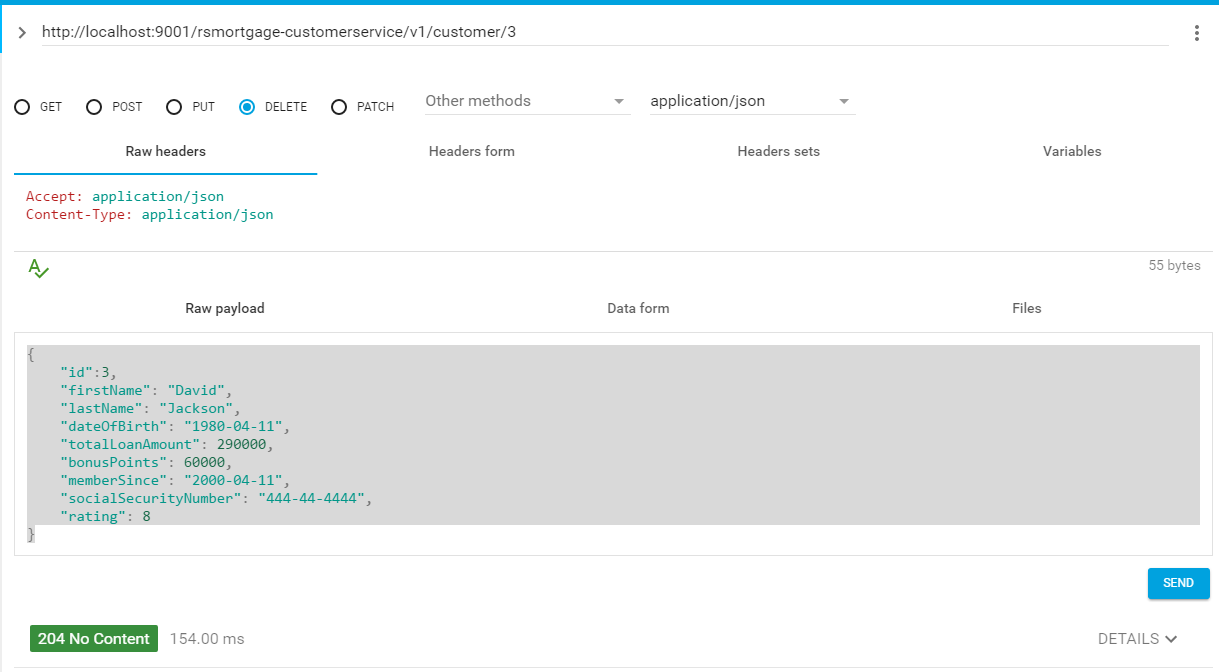
1.87—Verify the Database



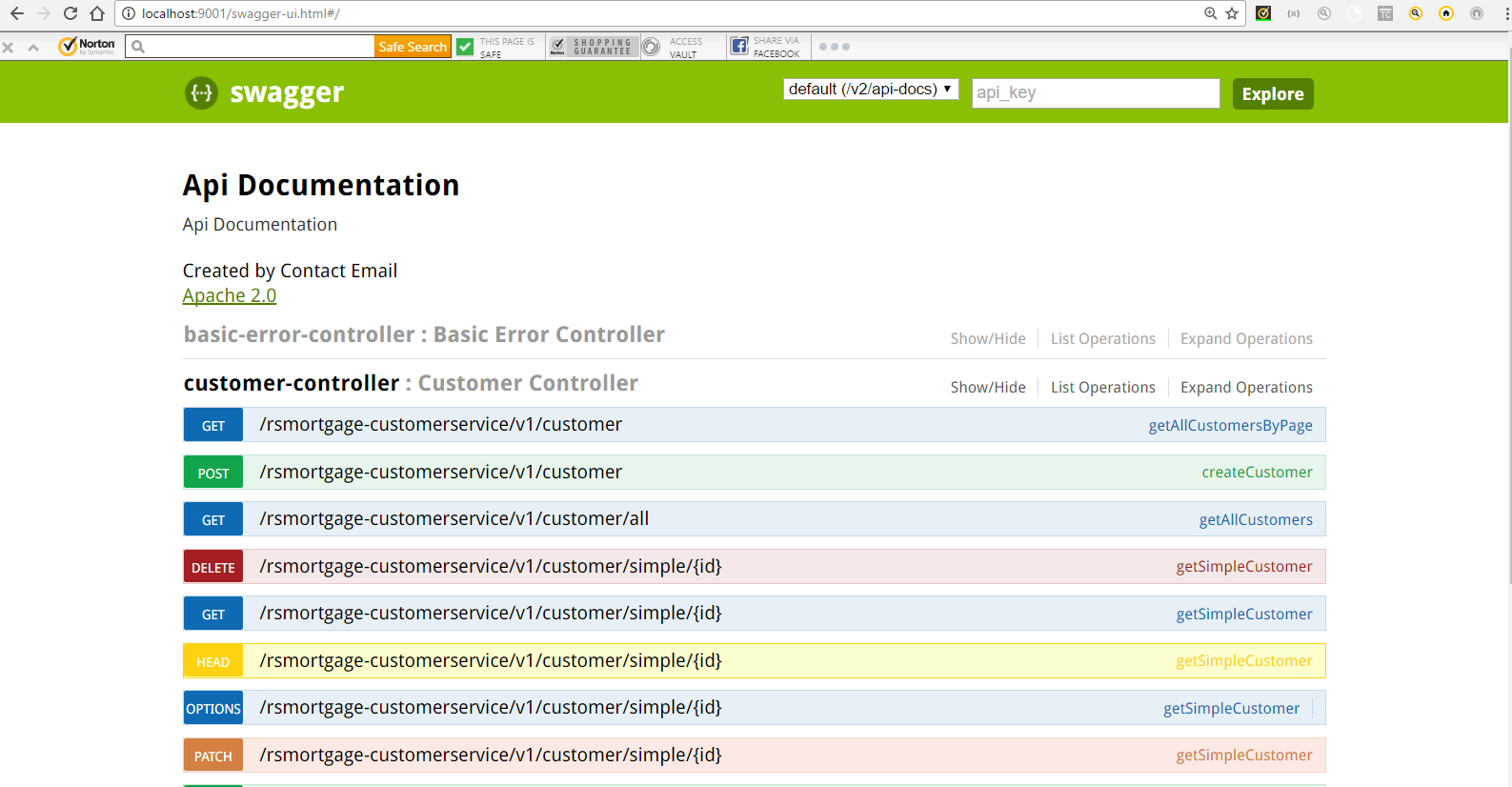
1.88 – Try to get a single customer



1.89 – Try to delete a single customer



1.90 – Swagger UI



1.91 – Conclusion

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