

Lab RSMortgage Customer Account Service

**Spring Cloud RSMortgage Customer Account 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 0

1.5 – Let Spring Tool Suite Prepare the Project 1

1.6 – Make sure the following looks like below 2

1.7 – Spring Boot Maven Parent Section 2

1.8 – Maven Properties Section 2

1.9 – Maven Dependency Management Section 2

1.10 – Spring Boot Actuator Dependency 3

1.11 – Spring Boot Web Dependency 3

1.12 – Spring Boot JPA Dependency 3

1.13 – Spring Boot H2 Dependency 3

1.14 – Spring Boot Test Dependency 4

1.15 – Spring Boot Jackson DataBind Dependency 4

1.16 – Spring Boot Jackson HAL Browser Dependency 5

1.17 – Spring Boot Jackson JSON Test Dependency 5

1.18 – Spring Boot Jackson Swagger Dependency 6

1.19 – Spring Boot HSQL Dependency 6

1.20 – Spring Boot MySQL Dependency 7

1.21 – Spring Cloud Eureka Dependency 7

1.22 – Spring Cloud Feign Dependency 7

1.23 – Spring Cloud Config Dependency 8

1.24 – Maven Build configuration 8

1.25 – Add api.rest package 9

1.26– Add dao.jpa package 11

1.27– Add domain package 12

1.28– Add exception package 13

1.29– Add service package 14

1.30– Create Account Domain class in the domain package 15

1.31– Generate the following for the Account class 16

1.32– Create AccountType Domain class in the domain package 17

1.33– Generate the following for the AccountType class 17

1.34– Create the Address Domain class in the domain package 18

1.35– Do the following to the Address Class 19

1.36--Create the Address Domain class in the domain package 20

1.37– Do the following to the Contact Class 21

1.38--Create the ContactType Domain class in the domain package 22

1.39– Do the following to the ContactType Class 22

1.40--Create the Customer Domain class in the domain package 23

1.41– Do the following to the Customer Class 24

1.42--Create the DegreeType Domain class in the domain package 25

1.43– Do the following to the DegreeType Class 25

1.44--Create the Education Domain class in the domain package 26

1.45– Do the following to the Education Class 27

1.46--Create the Employment Domain class in the domain package 28

1.47– Do the following to the Employment Class 30

1.48--Create the Investment Domain class in the domain package 31

1.49– Do the following to the Investment Class 32

1.50--Create the InvestmentType class in the domain package 32

1.51– Do the following to the InvestmentType Class 32

1.52--Create the Liability Domain class in the domain package 33

1.53– Do the following to the Liability Class 34

1.54--Create the LiabilityType Domain class in the domain package 34

1.55– Do the following to the LiabilityType Class 34

1.56--Create the RestAPIExceptionInfo class in the domain package 35

1.57– Generate HTTP400Exception in the exception package 36

1.58– Generate HTTP404Exception in the exception package 37

1.59– Generate DAOInterface in the dao.jpa package 38

1.60– Generate Service class in the service package 39

1.61– Generate ServiceProperties class in the service package 43

1.62– Generate ServiceHealth class in the service package 44

1.63– Generate ServiceEvent class in the service package 45

1.64– Generate CustomerClient class in the rest.api package 46

1.65– Generate AbstractRestController class in the rest.api package 47

1.66– Generate CustomerControllerclass in the rest.api package 49

1.66– Generate RestControllerAspect in the rest.api package 53

1.67– Generate RsMortgageCustomerRestAPIApplication in the rest.api package 55

1.68– Create application.yml file under resources 56

1.69– Add bootstrap.yml file in the resources folder 57

1.70 –Open Git Bash in project folder 58

1.71 –Run the first instance 58

1.72 –Run the second instance 59

1.73 – Navigate to http://localhost:8761 60

1.74 – Navigate to http://localhost:8762 61

1.75 – Navigate to http://eureka-host1:8761/ 62

1.76 – Navigate to http://eureka-host2:8762/ 63

1.77– Open Git bash in the config project directory 64

1.78– Run the config service project 64

1.79– Verify Customer Account Project mysql properties 65

1.80– Open Git bash in the project directory 66

1.81– Build the project 67

1.82 –Run the Project 68

1.83 –Verify Config Property is read and used 68

1.84 – Navigate to http://localhost:8761 69

1.85—Get an existing Customer Account 70

1.86– Create a Customer Account 71

1.87– Verify the Database 73

1.88—Try to Update a Record 73

1.89—Verify the Database 74

1.90 – Try to get a single customer 75

1.91 – Try to delete a single customer 76

1.92 – Swagger UI 77

1.93 – Conclusion 80

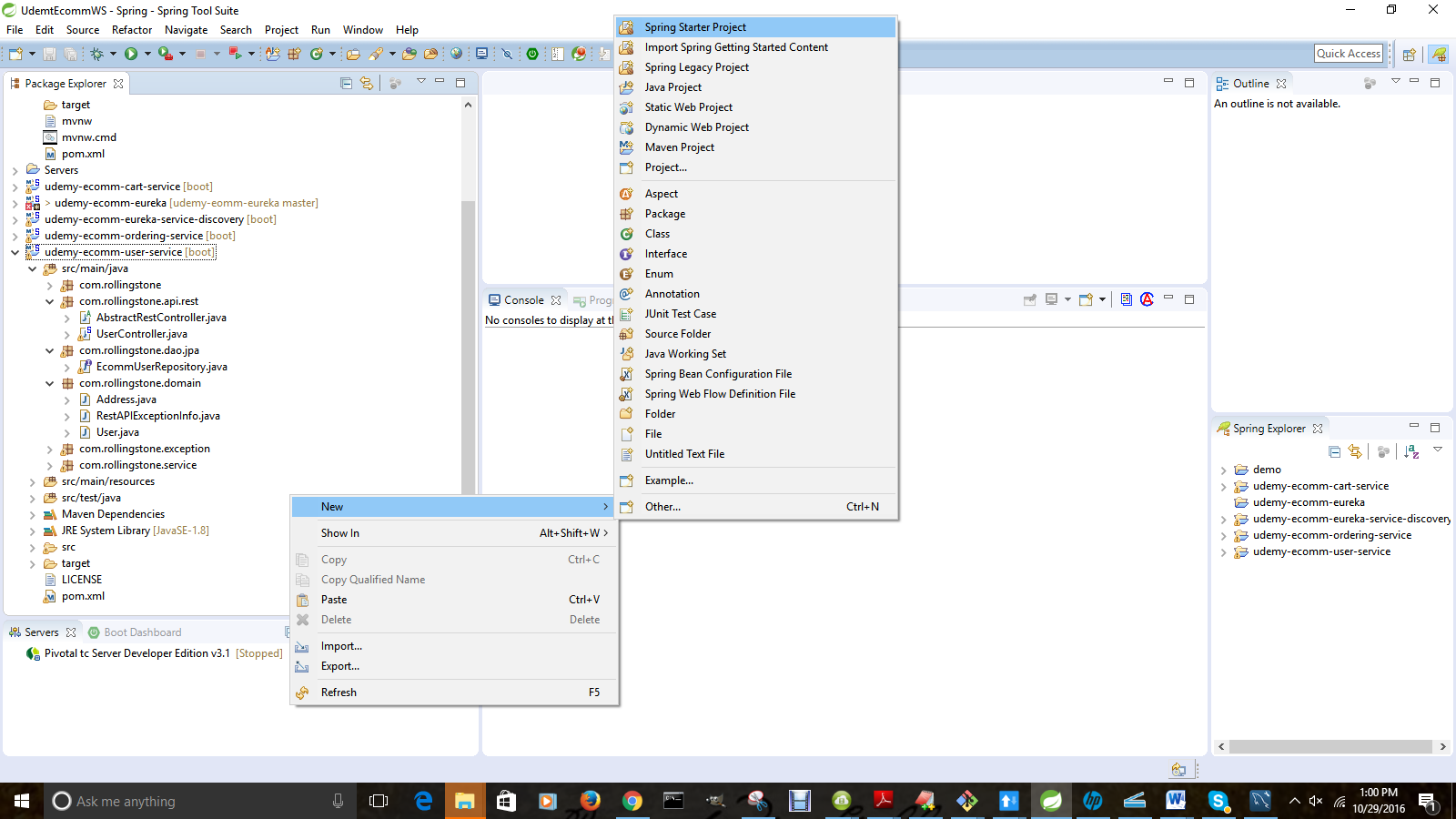
Chapter 1

*Spring Cloud Customer Account Service Project Creation*

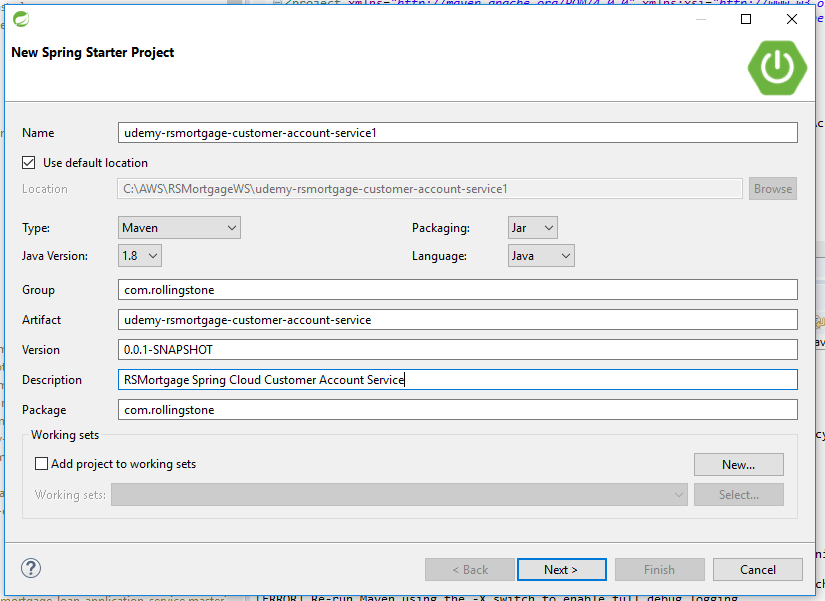
1.0 - Introduction

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

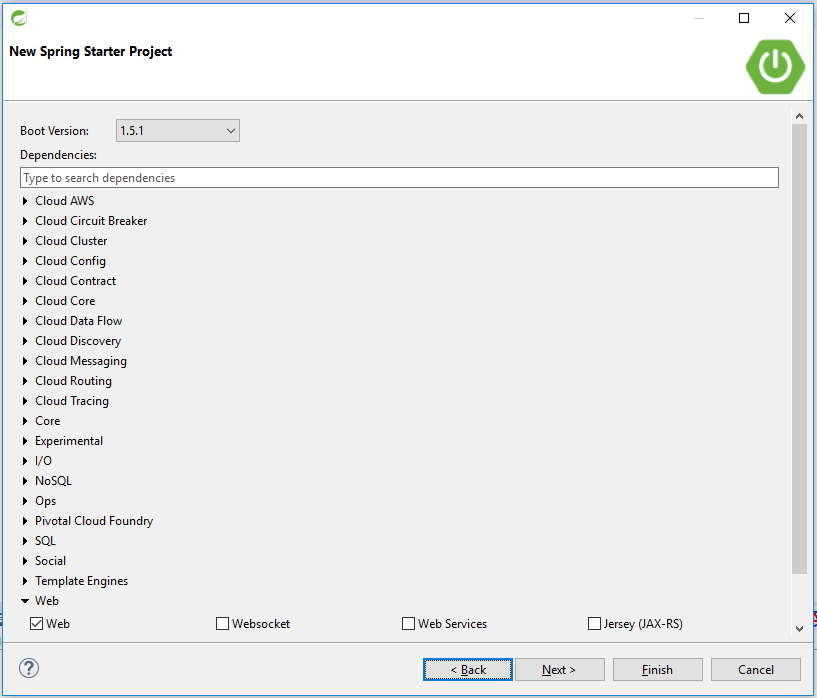
1.1 – Create a new Spring Starter Project



1.2 – Fill initial values

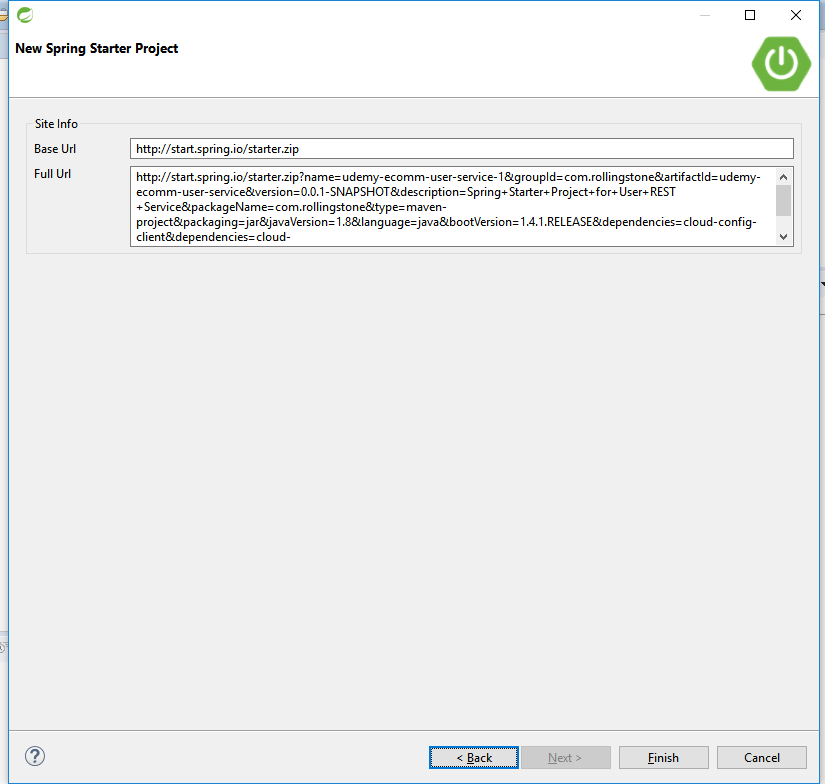


1.3 – Choose Eureka and Web as starter projects

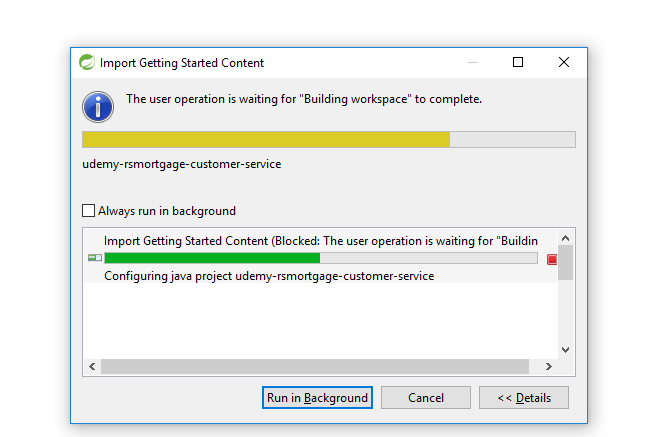


.

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-account-service</artifactId>

<version>1.0</version>

<packaging>jar</packaging>

<description>Example project demonstrating Spring Cloud based Customer Account 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.RsMortgageCustomerAccountRestAPIApplication</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 MySQL 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>

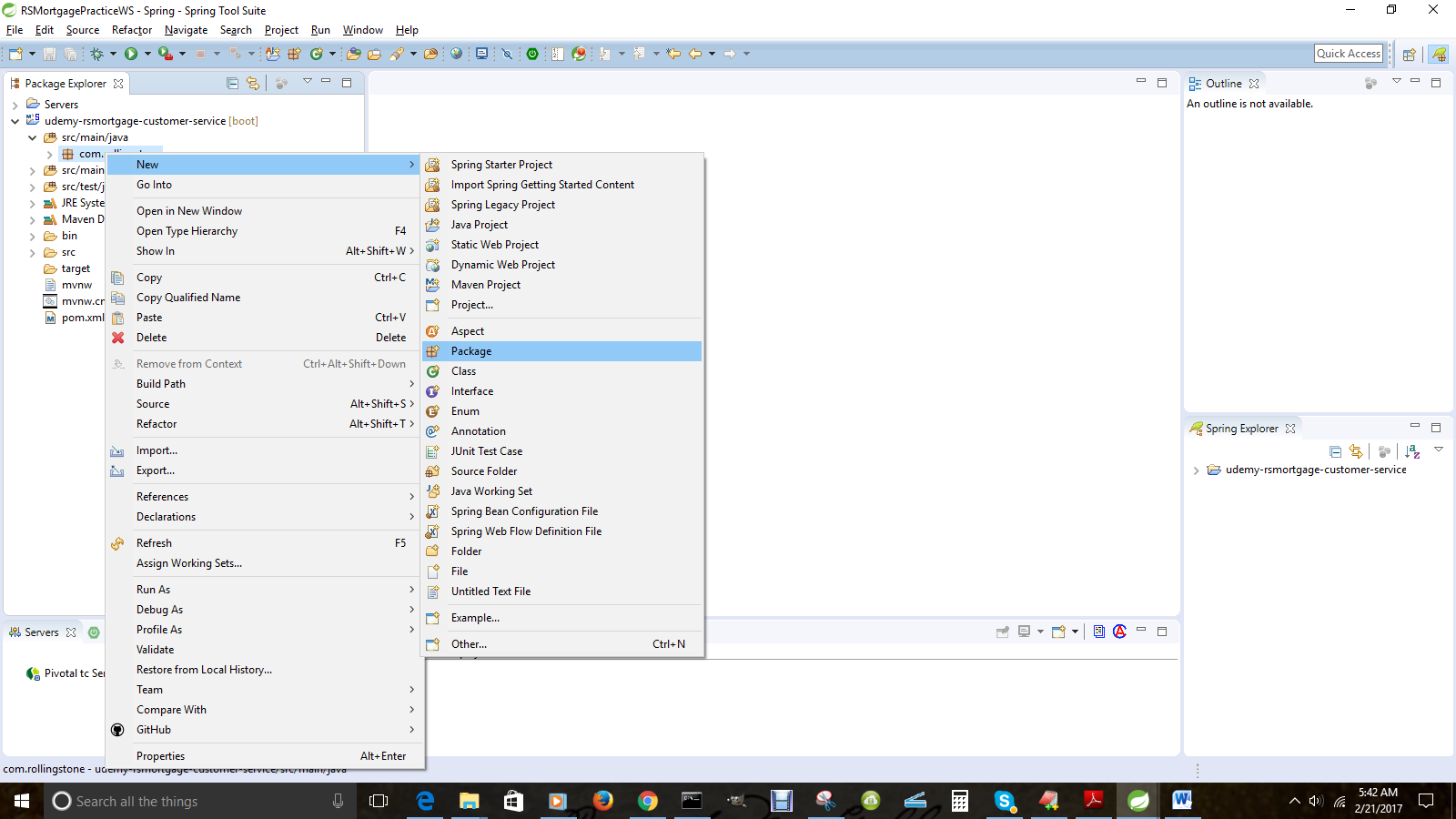
</configuration>

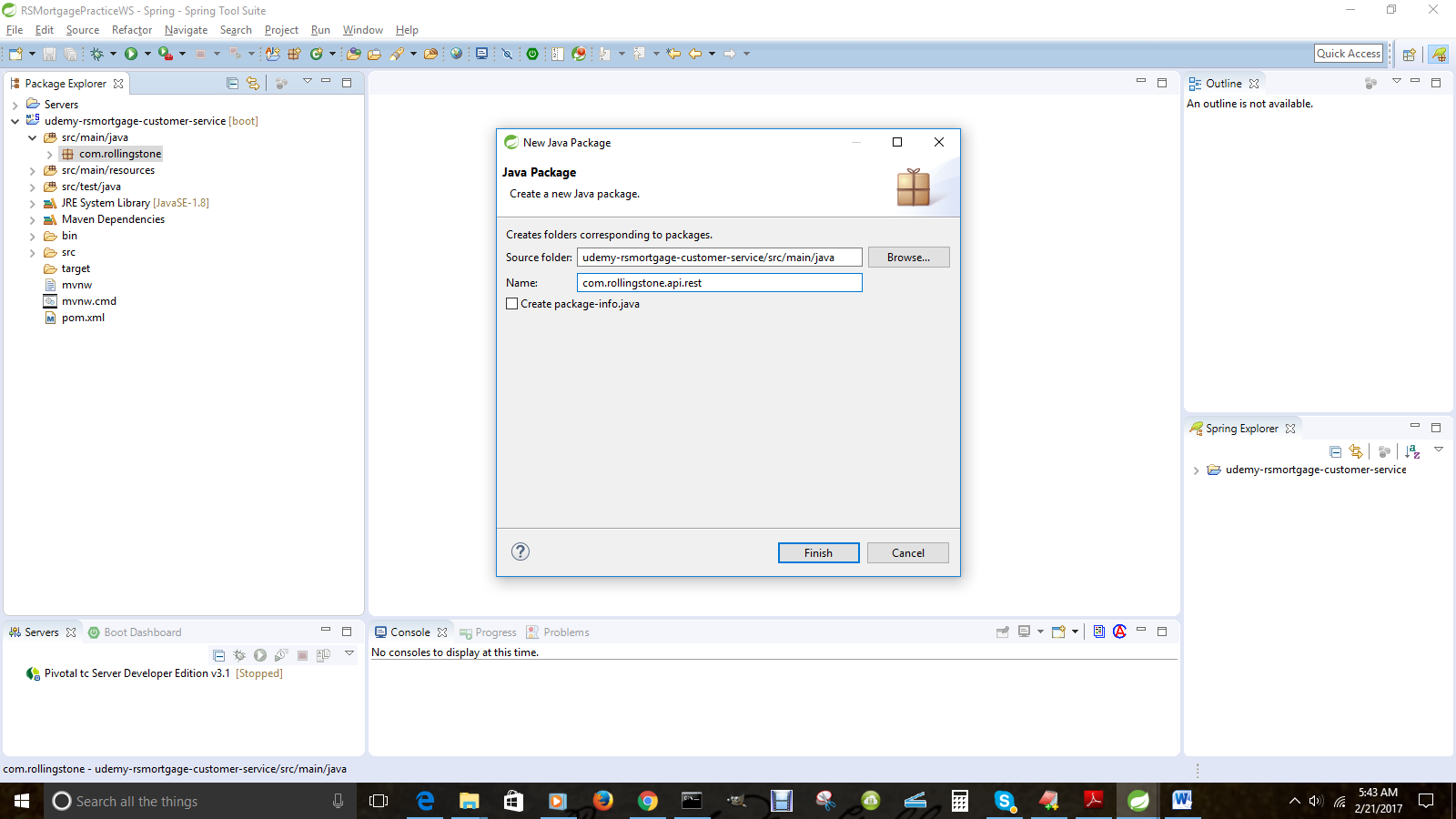
</plugin>

</plugins>

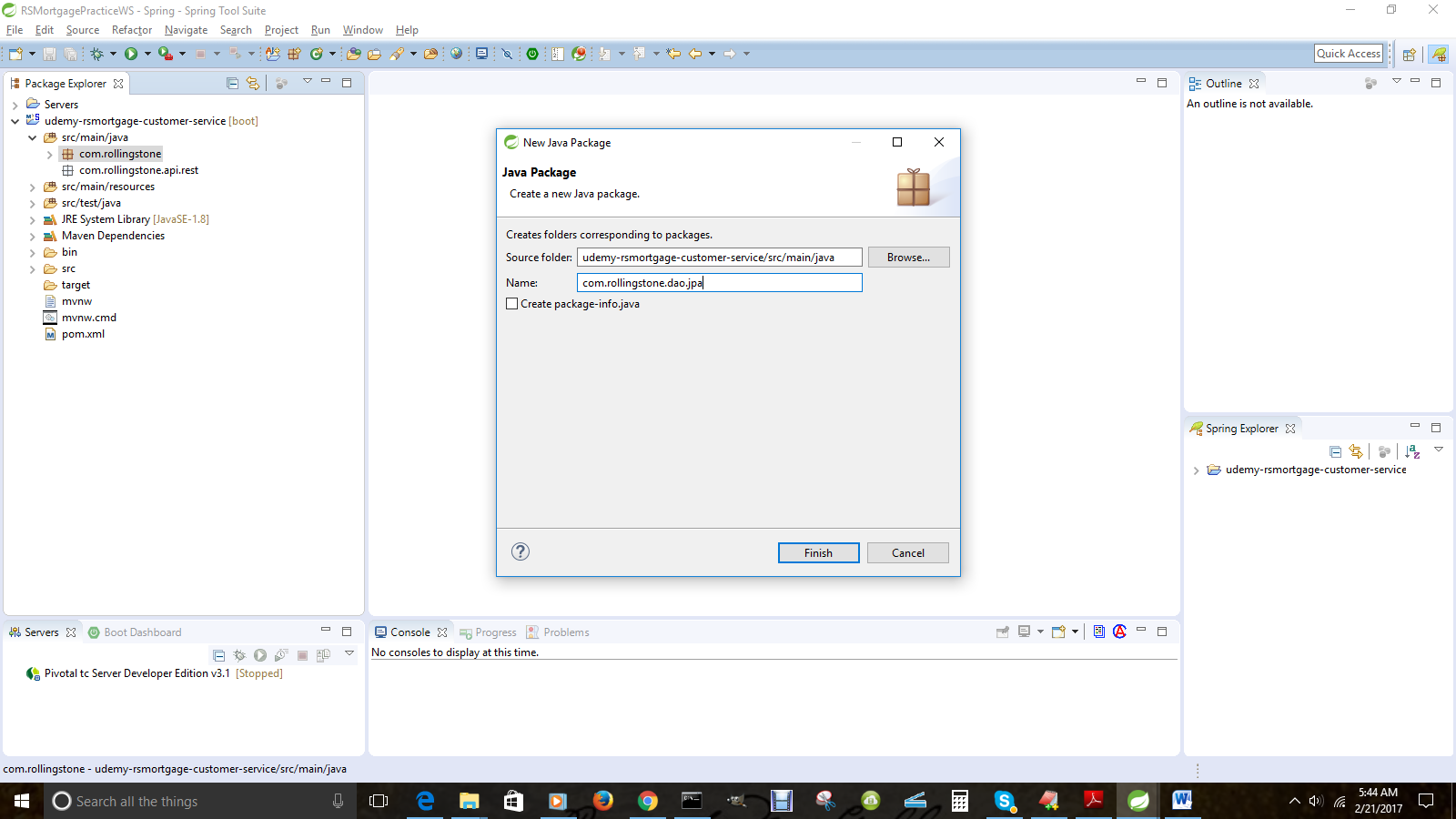
</build>

1.25 – Add api.rest package

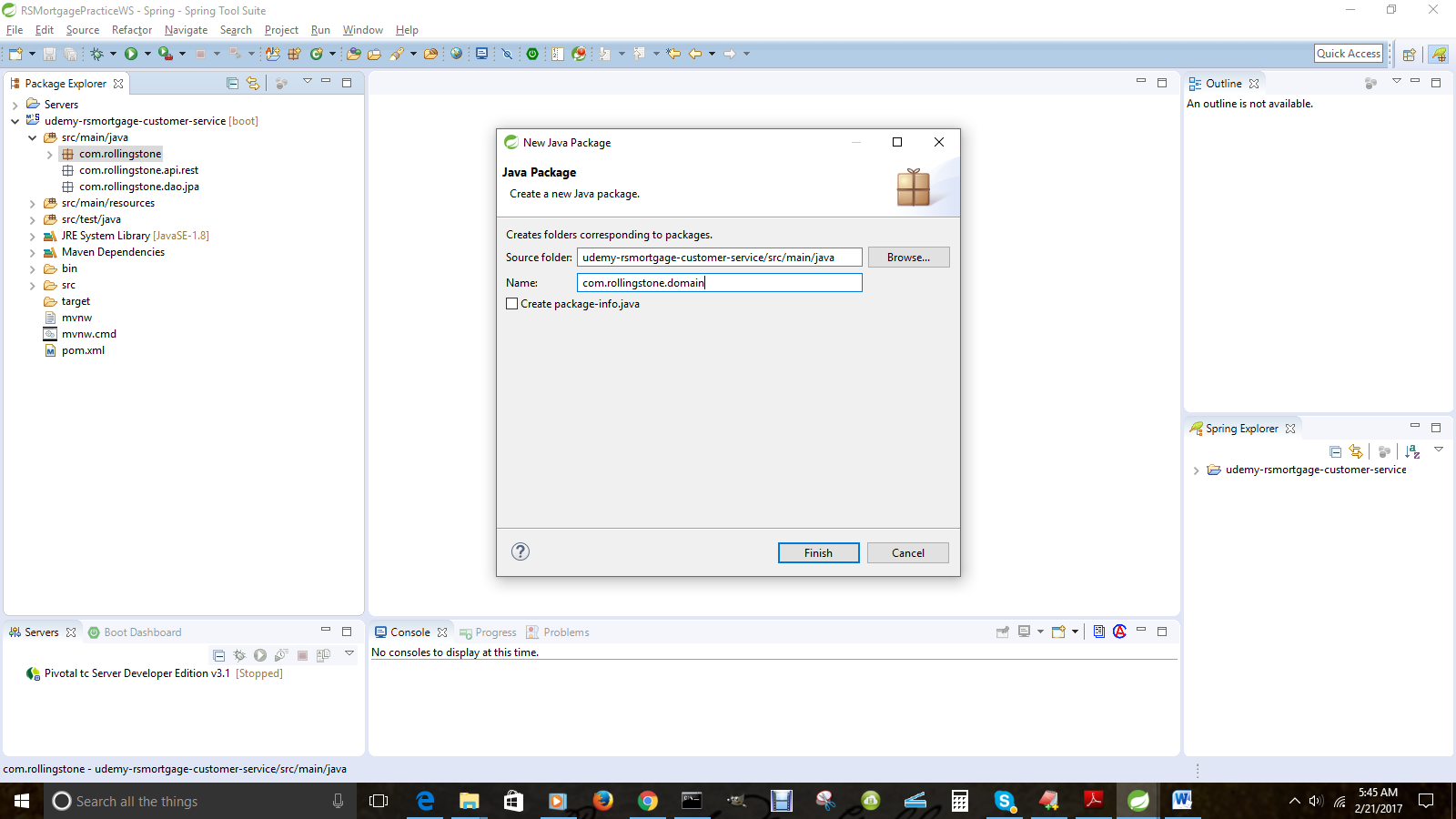




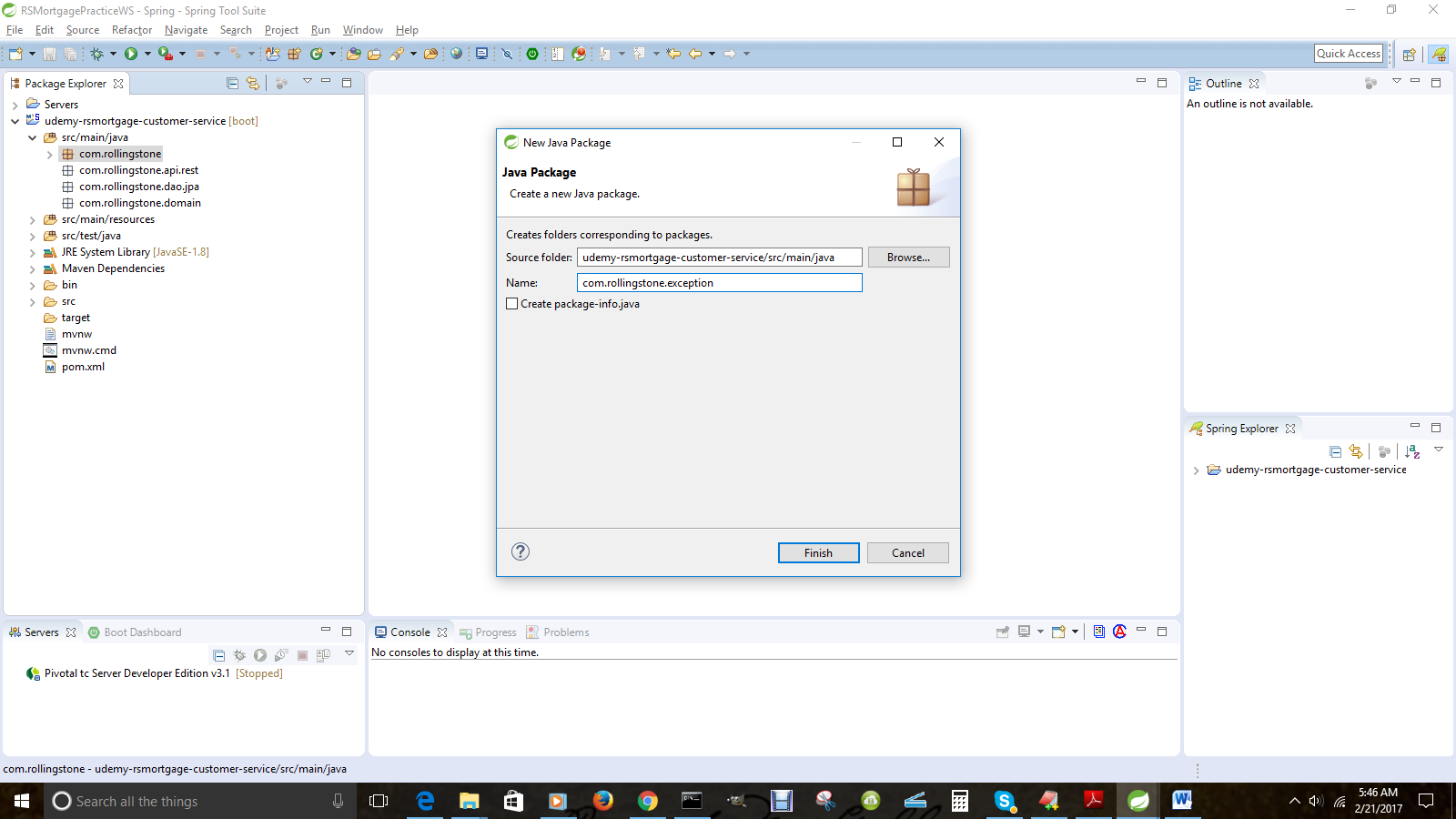
1.26– Add dao.jpa package



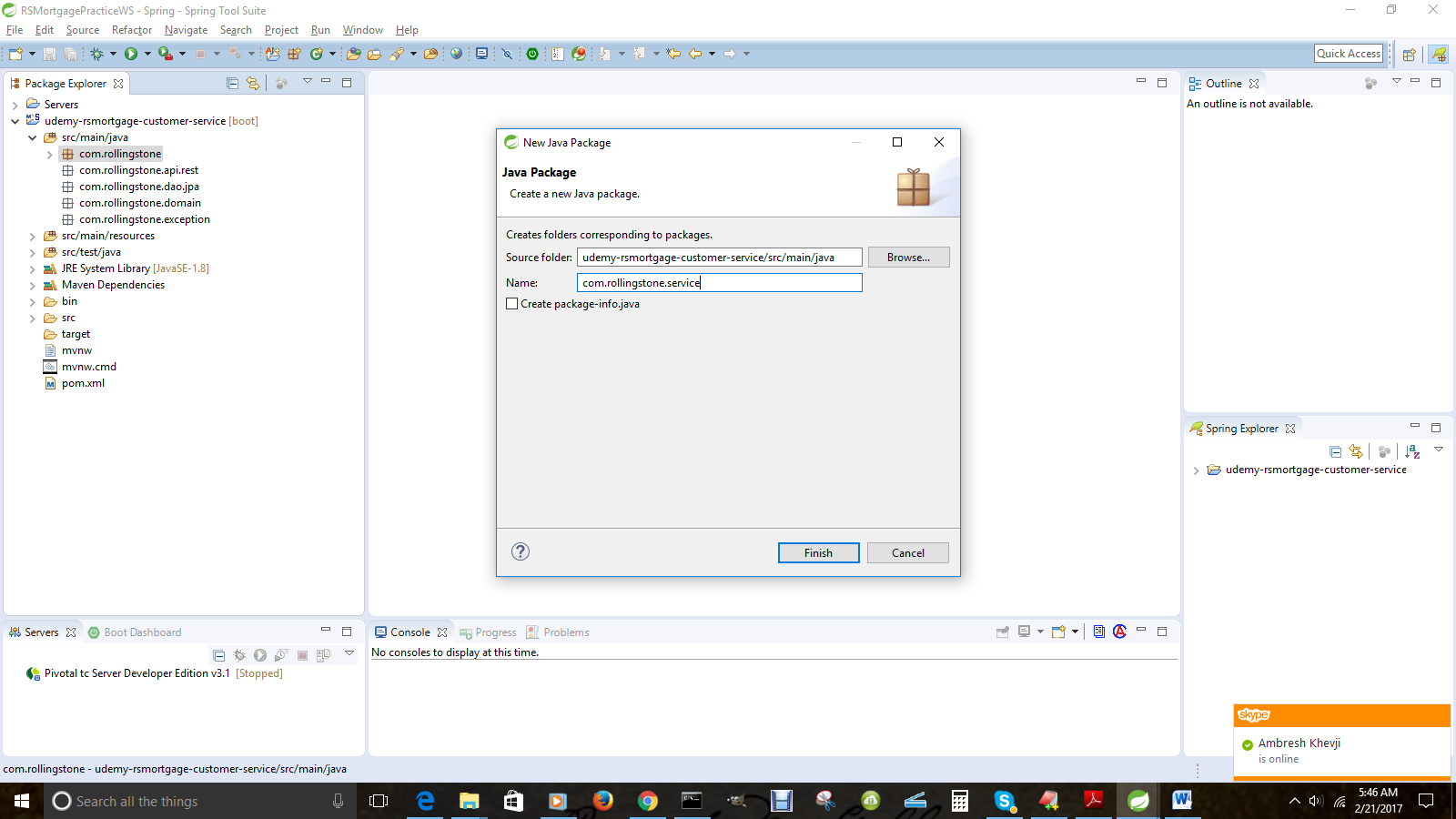
1.27– Add domain package



1.28– Add exception package



1.29– Add service package



1.30– Create Account Domain class in the domain package

package com.rollingstone.domain;

import java.util.Date;

/\*

\* A Account 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;

}

1.31– Generate the following for the Account class

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

1.32– 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;

}

1.33– 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

1.34– 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;

}

1.35– 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

1.36--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;

1.37– 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

1.38--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;

}

1.39– 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

1.40--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;

}

1.41– 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

1.42--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;

}

1.43– 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

1.44--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;

}

1.45– 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

1.46--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;

}

1.47– 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

1.48--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;

}

1.49– 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

1.50--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;

}

1.51– 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

1.52--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;

}

1.53– 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

1.54--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;

}

1.55– 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

1.56--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;

}

}

1.57– 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);

}

}

1.58– 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);

}

}

1.59– Generate DAOInterface 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.Account;

import com.rollingstone.domain.Customer;

import com.rollingstone.domain.Employment;

public interface RsMortgageCustomerAccountRepository extends PagingAndSortingRepository<Account, Long> {

List<Account> findCustomerAccountsByCustomer(Customer customer);

Page findAll(Pageable pageable);

}

1.60– 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.netflix.hystrix.contrib.javanica.annotation.HystrixCommand;

import com.rollingstone.dao.jpa.RsMortgageCustomerAccountRepository;

import com.rollingstone.domain.Account;

import com.rollingstone.domain.Customer;

/\*

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

\*/

@Service

public class RsMortgageCustomerAccountService {

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

@Autowired

private RsMortgageCustomerAccountRepository customerAccountRepository;

@Autowired

CounterService counterService;

@Autowired

GaugeService gaugeService;

@Autowired

private CustomerClient customerClient;

public RsMortgageCustomerAccountService() {

}

@HystrixCommand(fallbackMethod = "createAccountWithoutCustomerValidation")

public Account createAccount(Account account) throws Exception {

Account createdAccount = null;

if (account != null && account.getCustomer() != null){

log.info("In service account create"+ account.getCustomer().getId());

if (customerClient == null){

log.info("In customerClient null got customer");

}

else {

log.info("In customerClient not null got customer");

}

Customer customer = customerClient.getCustomer((new Long(account.getCustomer().getId())));

if (customer != null){

createdAccount = customerAccountRepository.save(account);

log.info("Valid Customer Created Account.");

}else {

log.info("Invalid Customer");

throw new Exception("Invalid Customer");

}

}

else {

throw new Exception("Invalid Customer");

}

return createdAccount;

}

public Account createAccountWithoutCustomerValidation(Account account) throws Exception {

Account createdAccount = null;

log.info("Customer Validation Failed. Creating Customer Account without validation.");

createdAccount = customerAccountRepository.save(account);

return createdAccount;

}

public Account getAccount(long id) {

return customerAccountRepository.findOne(id);

}

public void updateAccount(Account account) throws Exception {

Account createdAccount = null;

if (account != null && account.getCustomer() != null){

log.info("In service account create"+ account.getCustomer().getId());

if (customerClient == null){

log.info("In customerClient null got customer");

}

else {

log.info("In customerClient not null got customer");

}

Customer customer = customerClient.getCustomer((new Long(account.getCustomer().getId())));

if (customer != null){

createdAccount = customerAccountRepository.save(account);

}else {

log.info("Invalid Customer");

throw new Exception("Invalid Customer");

}

}

else {

throw new Exception("Invalid Customer");

}

}

public void deleteAccount(Long id) {

customerAccountRepository.delete(id);

}

public Page<Account> getAllAccountsByPage(Integer page, Integer size) {

Page pageOfAccounts = customerAccountRepository.findAll(new PageRequest(page, size));

// example of adding to the /metrics

if (size > 50) {

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

}

return pageOfAccounts;

}

public List<Account> getAllAccounts() {

Iterable<Account> pageOfAccounts = customerAccountRepository.findAll();

List<Account> customerAccounts = new ArrayList<Account>();

for (Account account : pageOfAccounts){

customerAccounts.add(account);

}

log.info("In Real Service getAllAccounts size :"+customerAccounts.size());

return customerAccounts;

}

public List<Account> getAllAccountsForCustomer(Customer customer) {

Iterable<Account> pageOfAccounts = customerAccountRepository.findCustomerAccountsByCustomer(customer);

List<Account> customerAccounts = new ArrayList<Account>();

for (Account account : pageOfAccounts){

customerAccounts.add(account);

}

log.info("In Real Service getAllAccounts size :"+customerAccounts.size());

return customerAccounts;

}

}

1.61– 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 = "CustomerAccountService";

@NotNull // you can also create configurationPropertiesValidator

private String description = "Customer Account MicroService that helps maintain the customer bank, credit card and other types of expense account";

public String getName() {

return this.name;

}

public void setName(String name) {

this.name = name;

}

public String getDescription() {

return description;

}

public void setDescription(String description) {

this.description = description;

}

}

1.62– 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 CustomerAccountServiceHealth 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() + "' "

+ "'" + this.configuration.getDescription() + "' }").status("itsok!").build();

}

}

1.63– 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 CustomerAccountServiceEvent extends ApplicationEvent {

public CustomerAccountServiceEvent(Object source) {

super(source);

}

public String toString() {

return "My CustomerAccountService Event";

}

}

1.64– Generate CustomerClient class in the rest.api package

package com.rollingstone.service;

import java.util.List;

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

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

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

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

import com.rollingstone.domain.Customer;

@FeignClient("rsmortgage-customer-service")

interface CustomerClient {

@RequestMapping(method = RequestMethod.GET, value="/rsmortgage-customerservice/v1/customer/all")

List<Customer> getCustomers();

@RequestMapping(method = RequestMethod.GET, value="/rsmortgage-customerservice/v1/customer/simple/{id}")

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

}

1.65– 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;

}

}

1.66– 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.domain.Account;

import com.rollingstone.exception.HTTP400Exception;

import com.rollingstone.service.RsMortgageCustomerAccountService;

/\*

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

\*/

@RestController

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

public class CustomerAccountController extends AbstractRestController {

@Autowired

private RsMortgageCustomerAccountService customerAccountService;

@RequestMapping(value = "",

method = RequestMethod.POST,

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

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

@ResponseStatus(HttpStatus.CREATED)

public void createCustomerAccount(@RequestBody Account account,

HttpServletRequest request, HttpServletResponse response) throws Exception {

Account createdAccount = this.customerAccountService.createAccount(account);

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

}

@RequestMapping(value = "",

method = RequestMethod.GET,

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

@ResponseStatus(HttpStatus.OK)

public

@ResponseBody

Page<Account> getAllCustomersAccountByPage(@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.customerAccountService.getAllAccountsByPage(page, size);

}

@RequestMapping(value = "/all",

method = RequestMethod.GET,

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

@ResponseStatus(HttpStatus.OK)

public

@ResponseBody

List<Account> getAllCustomerAccounts(@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.customerAccountService.getAllAccounts();

}

@RequestMapping(value = "/all/{customerId}",

method = RequestMethod.GET,

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

@ResponseStatus(HttpStatus.OK)

public

@ResponseBody

List<Account> getAllCustomerAccountsForSingleCustomer(@RequestParam(value = "page", required = true, defaultValue = DEFAULT\_PAGE\_NUM) Integer page,

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

@PathVariable("id") Long id,

HttpServletRequest request, HttpServletResponse response) {

return this.customerAccountService.getAllAccountsForCustomer(new Customer());

}

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

public Account getSimpleCustomerAccount(@PathVariable("id") Long id) {

Account account = this.customerAccountService.getAccount(id);

checkResourceFound(account);

return account;

}

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

method = RequestMethod.GET,

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

@ResponseStatus(HttpStatus.OK)

public

@ResponseBody

Account getAccount(@PathVariable("id") Long id,

HttpServletRequest request, HttpServletResponse response) throws Exception {

Account account = this.customerAccountService.getAccount(id);

checkResourceFound(account);

return account;

}

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

method = RequestMethod.PUT,

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

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

@ResponseStatus(HttpStatus.NO\_CONTENT)

public void updateCustomerAccount(@PathVariable("id") Long id, @RequestBody Account account,

HttpServletRequest request, HttpServletResponse response) throws Exception {

checkResourceFound(this.customerAccountService.getAccount(id));

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

this.customerAccountService.updateAccount(account);

}

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

method = RequestMethod.DELETE,

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

@ResponseStatus(HttpStatus.NO\_CONTENT)

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

HttpServletResponse response) {

checkResourceFound(this.customerAccountService.getAccount(id));

this.customerAccountService.deleteAccount(id);

}

}

1.66– Generate RestControllerAspect in the rest.api 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.createCustomerAccount\*(..))")

public void afterCallingCreateCustomerAccount(JoinPoint pjp) {

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

counterService.increment("com.rollingstone.api.rest.CustomerAccountController.createCustomerAccount");

}

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

public void afterCallinggetAllCustomerAccount(JoinPoint pjp) {

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

counterService.increment("com.rollingstone.api.rest.CustomerAccountController.getAllCustomerAccount");

}

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

public void afterCallinggetCustomerAccount(JoinPoint pjp) {

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

counterService.increment("com.rollingstone.api.rest.CustomerAccountController.getCustomerAccount");

}

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

public void afterCallingUpdateCustomerAccount(JoinPoint pjp) {

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

counterService.increment("com.rollingstone.api.rest.CustomerAccountController.updateCustomerAccount");

}

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

public void afterGetGreetingThrowsException(NoSuchElementException e) {

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

}

}

1.67– 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.circuitbreaker.EnableCircuitBreaker;

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

@EnableCircuitBreaker

public class RsMortgageCustomerAccountRestAPIApplication extends SpringBootServletInitializer {

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

}

}

1.68– 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: 9002

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: 9003

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: customer-account-service

version: 0.1

description: customer-account-service

info:

build:

artifact: ${project.artifactId}

name: ${project.name}

description: ${project.description}

version: ${project.version}

1.69– Add bootstrap.yml file in the resources folder

spring:

application:

name: udemy-rsmortgage-customer-account-service

profiles:

active:

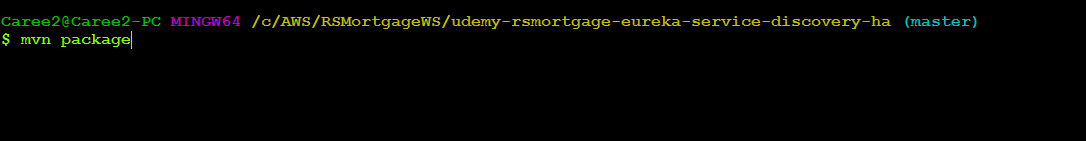
mysql

cloud:

config:

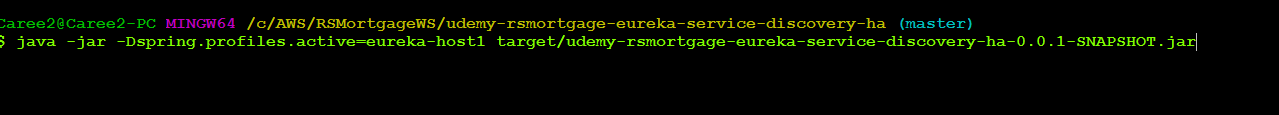
uri: http://localhost:9000

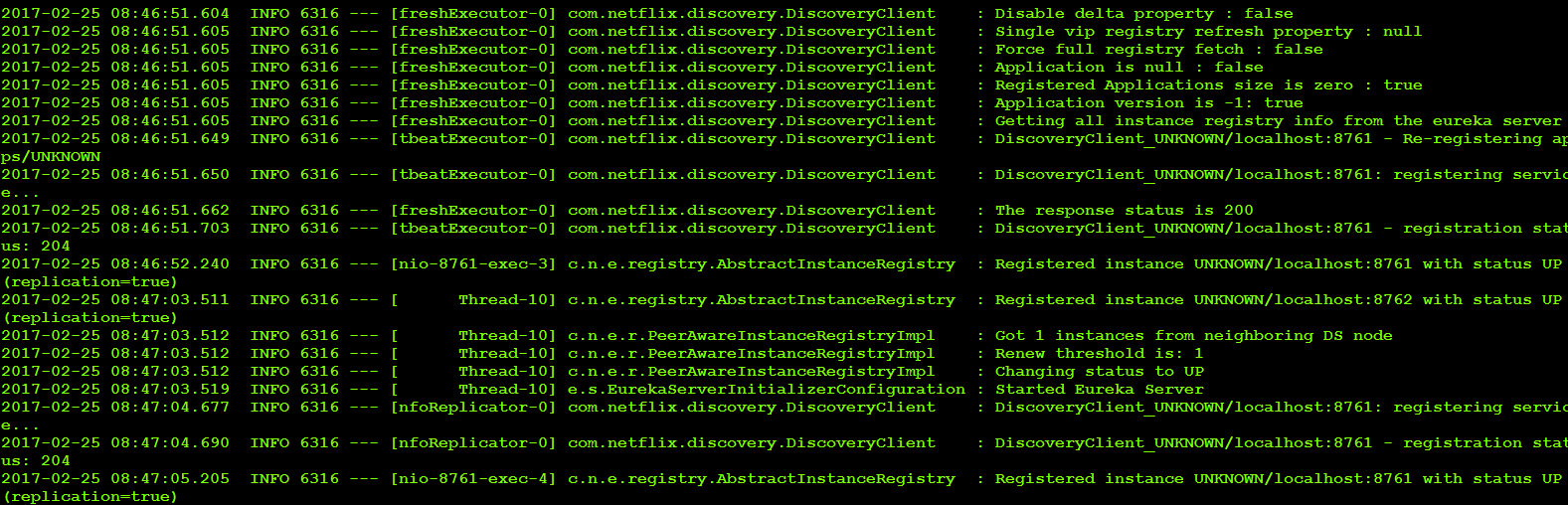
1.70 –Open Git Bash in project folder



1.71 –Run the first instance

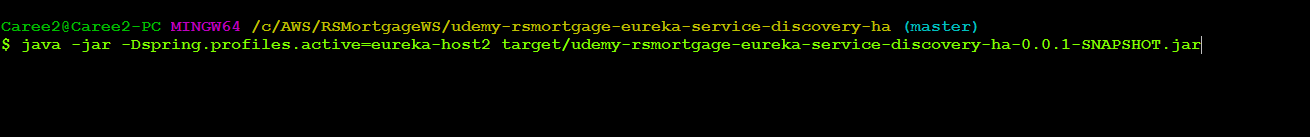
java -jar -Dspring.profiles.active=eureka-host1 target/udemy-rsmortgage-eureka-service-discovery-ha-0.0.1-SNAPSHOT.jar

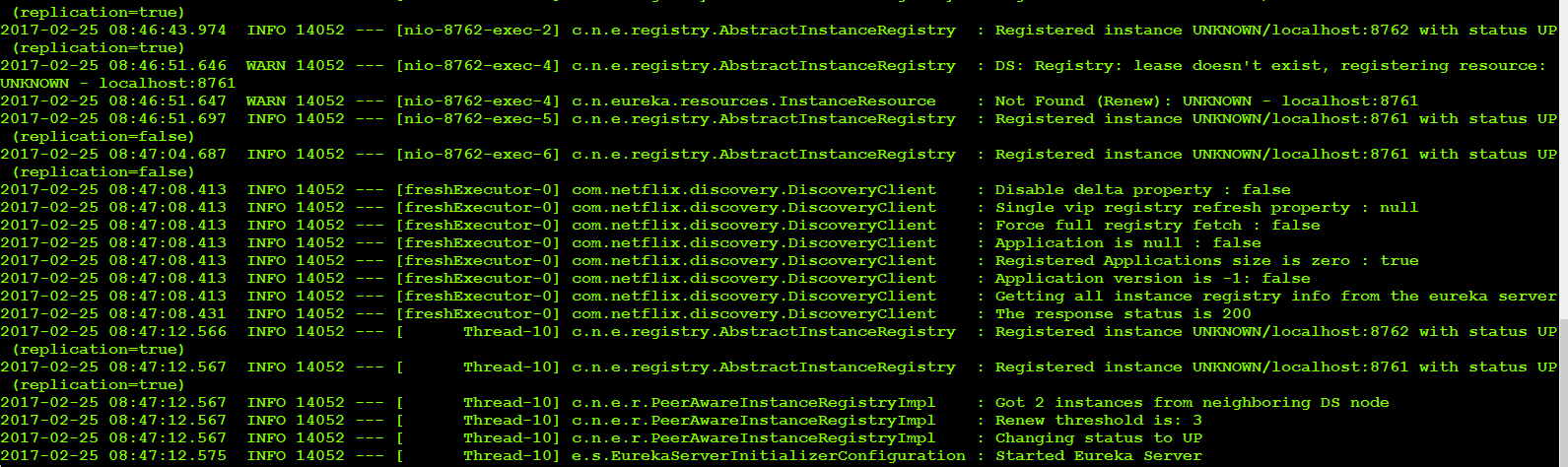




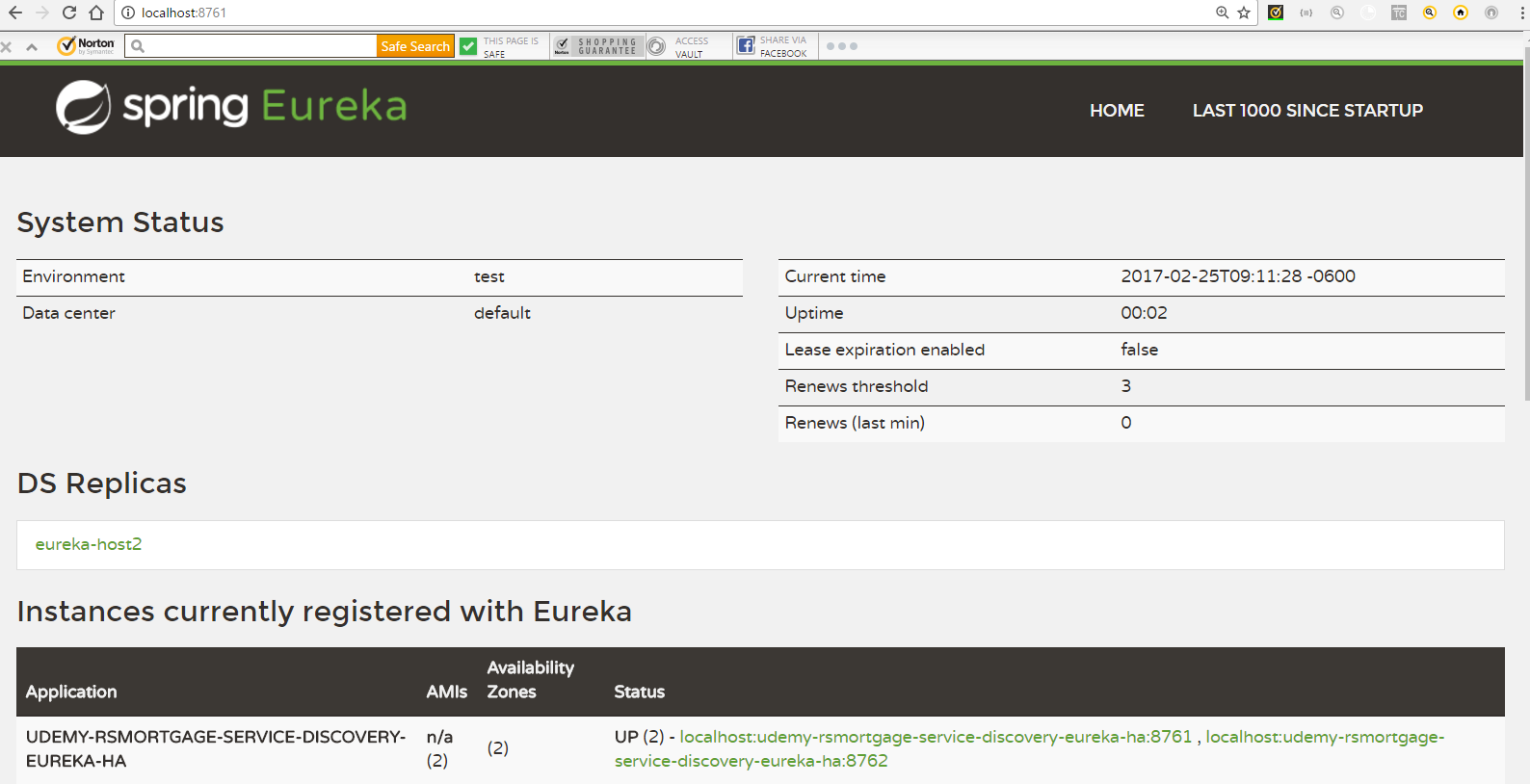
1.72 –Run the second instance

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

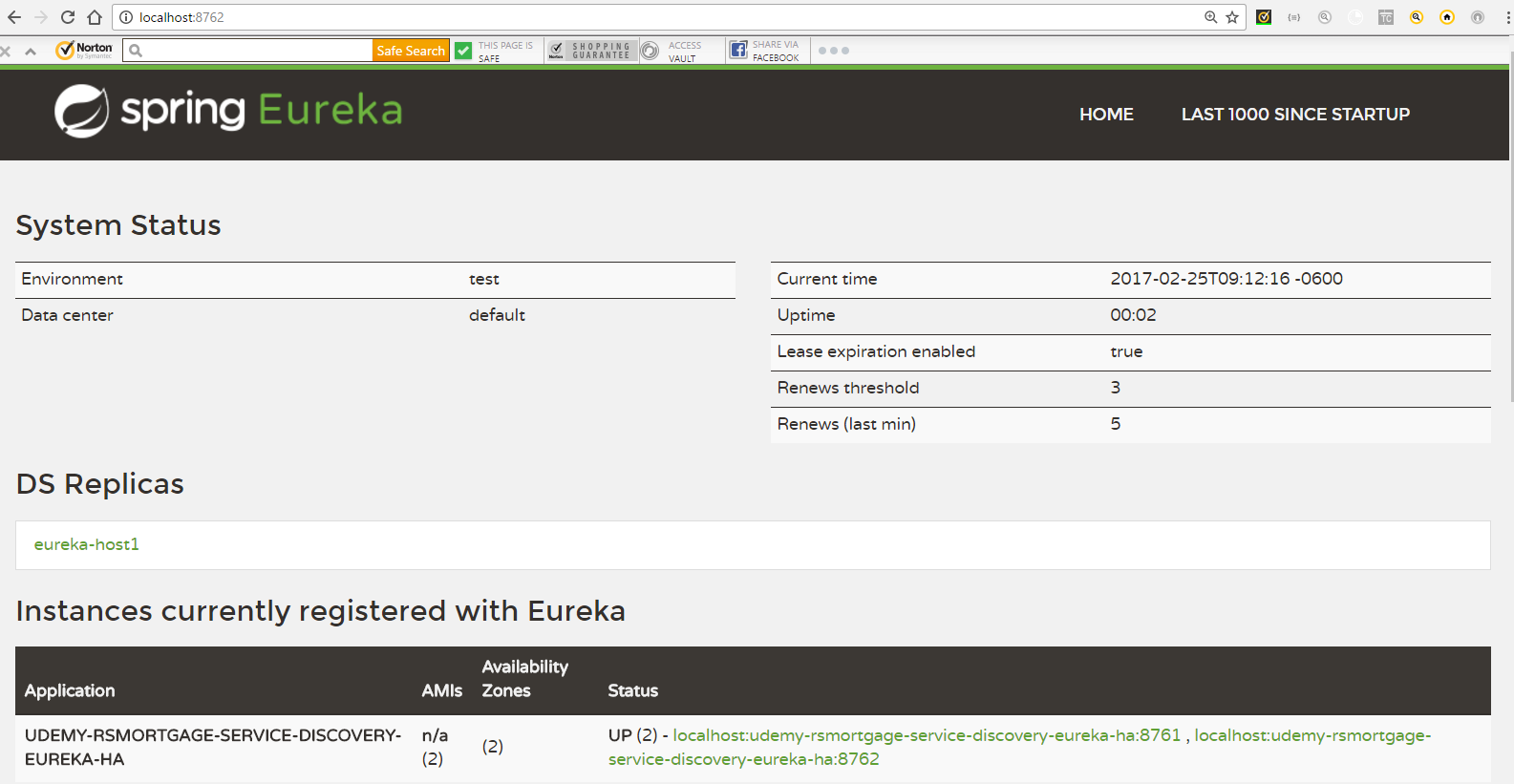




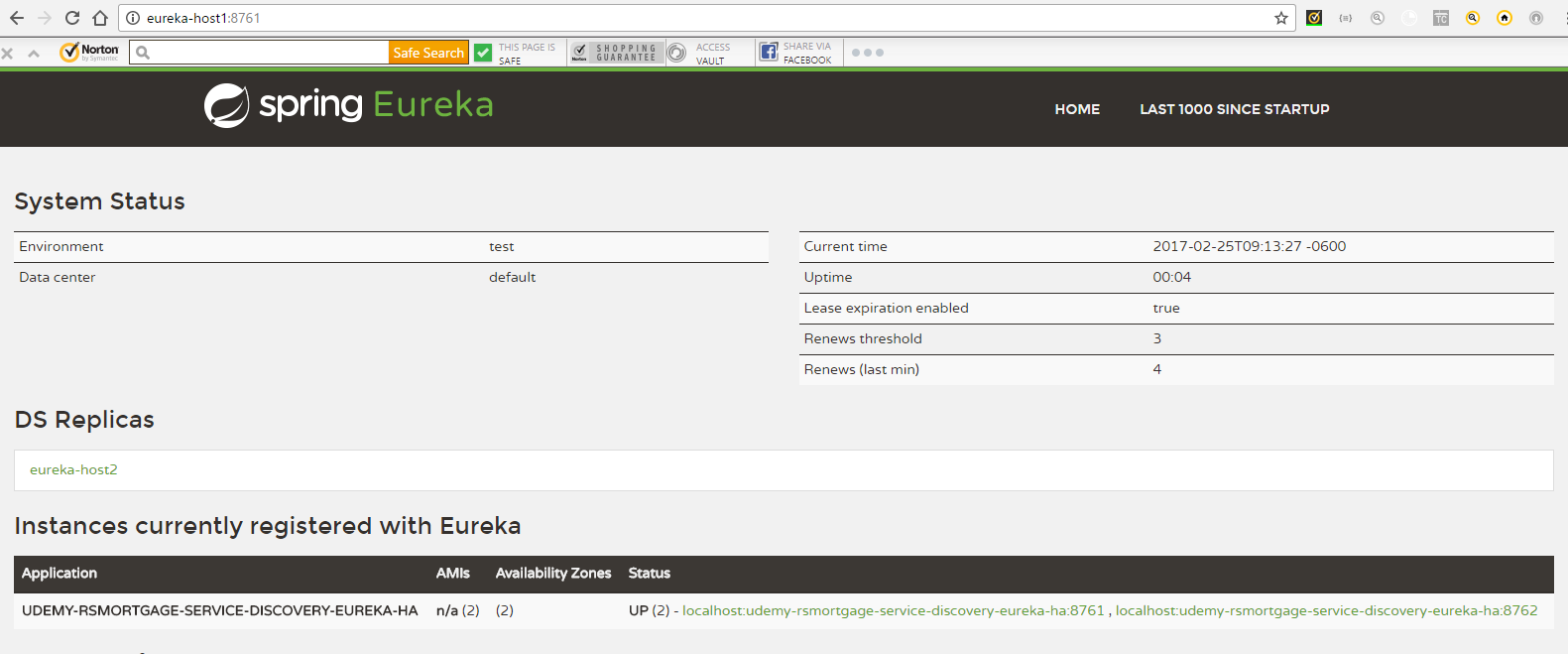
1.73 – Navigate to <http://localhost:8761>



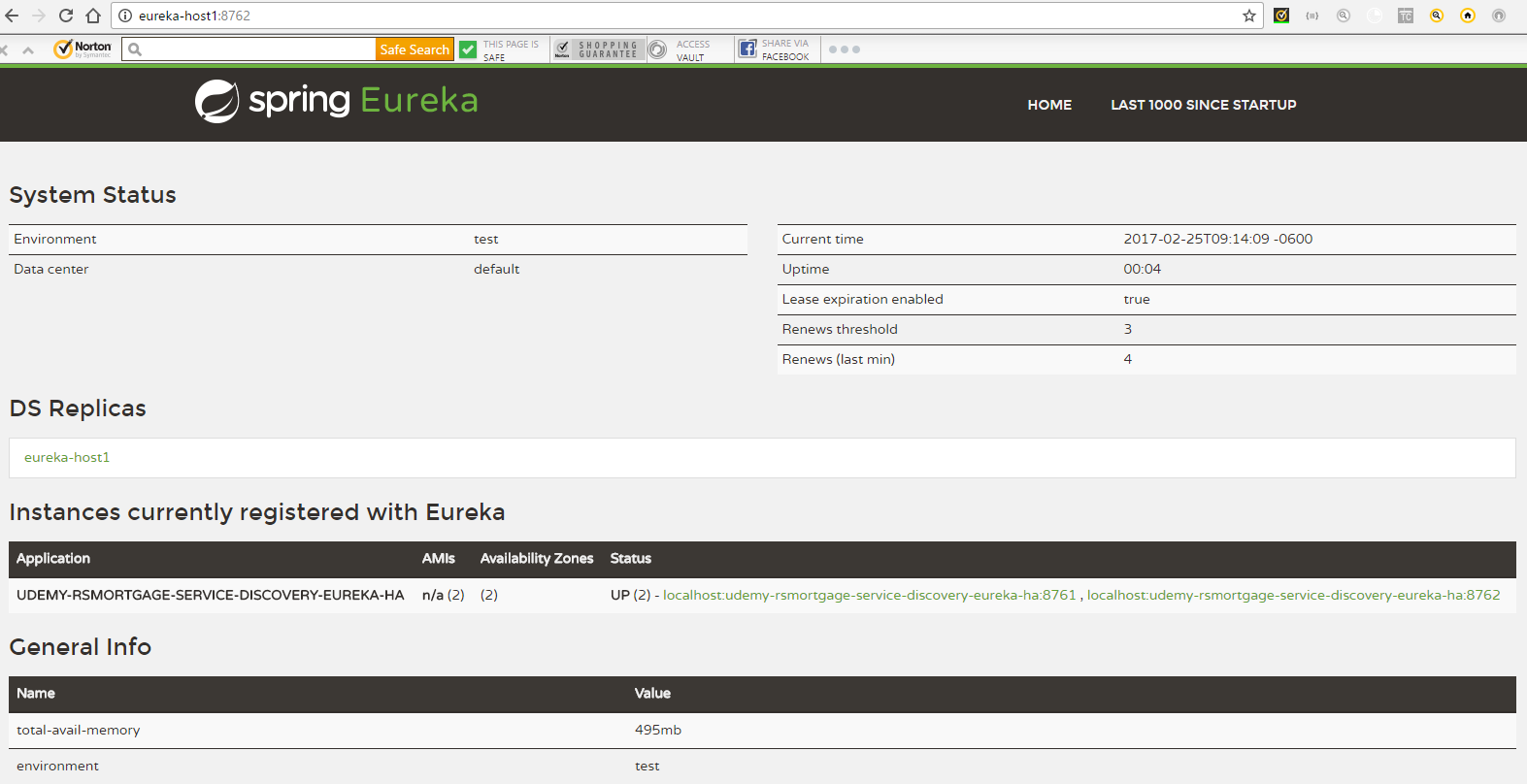
1.74 – Navigate to <http://localhost:8762>

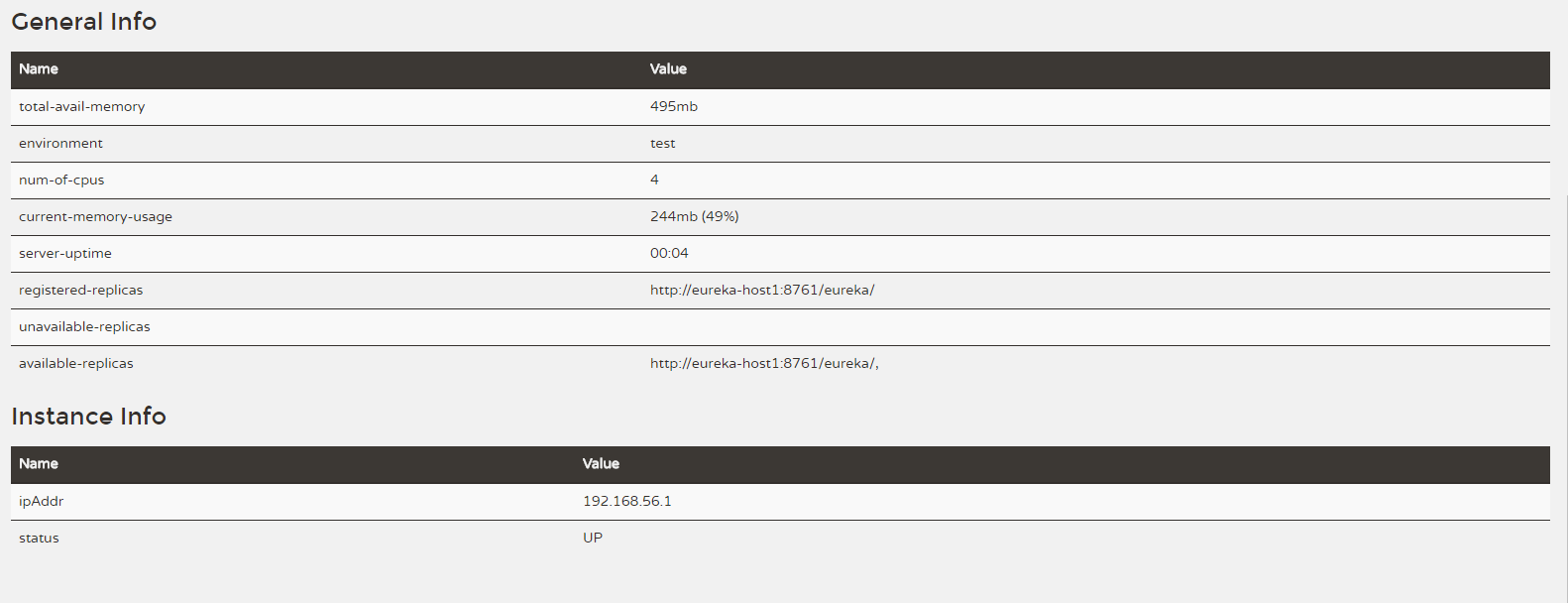


1.75 – Navigate to <http://eureka-host1:8761/>

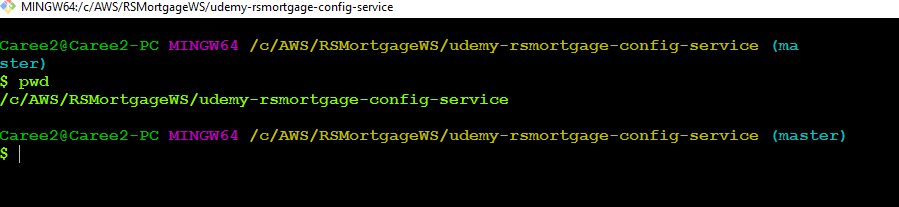


1.76 – Navigate to <http://eureka-host2:8762/>





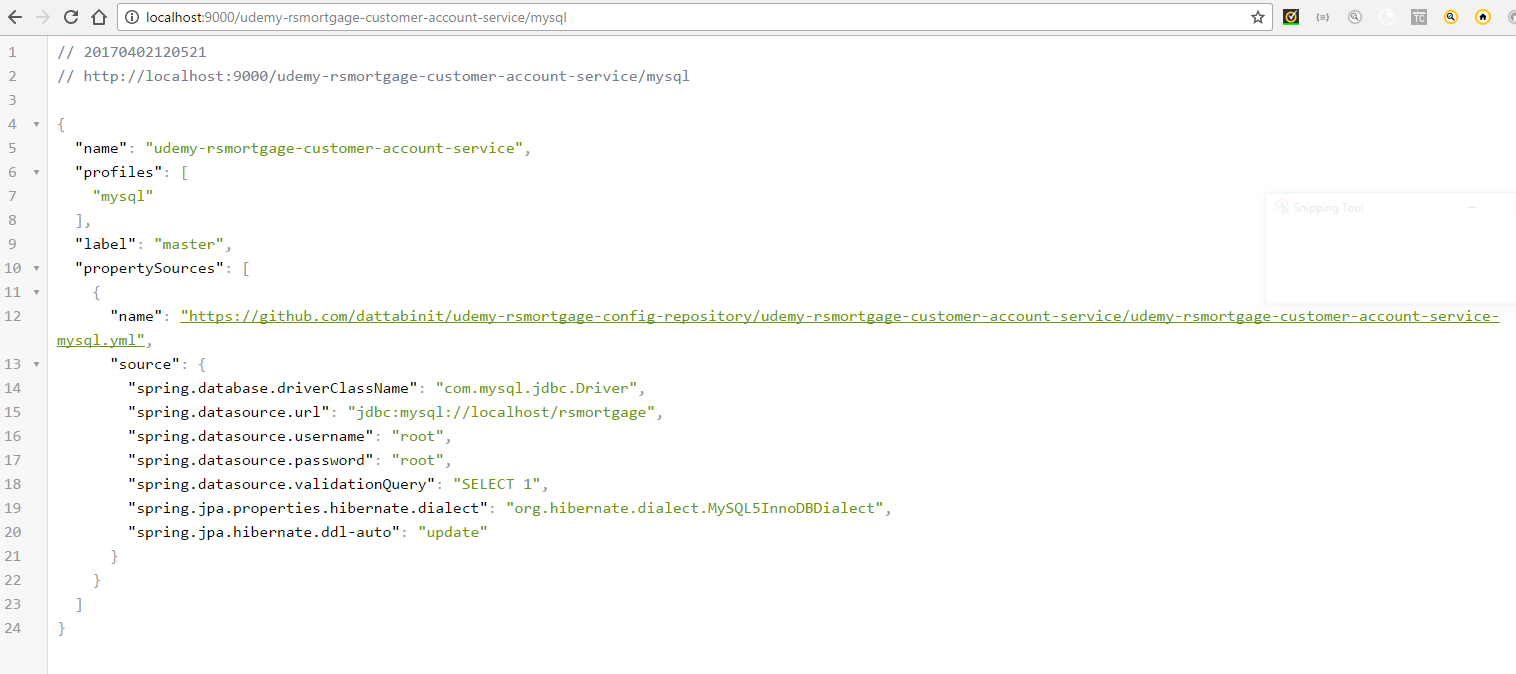
1.77– Open Git bash in the config project directory



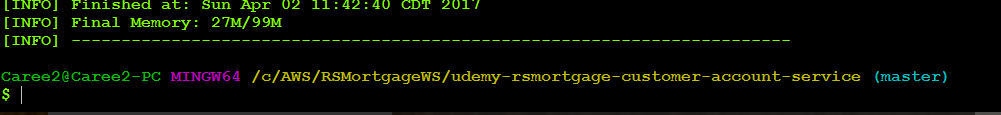
1.78– Run the config service project

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

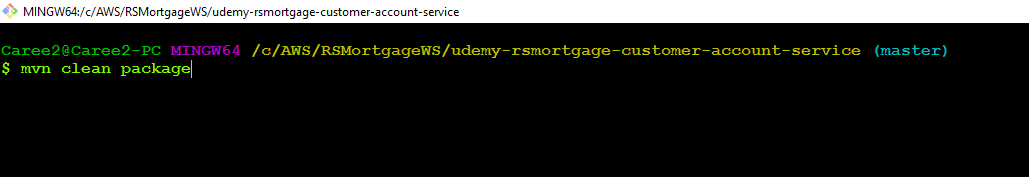
1.79– Verify Customer Account Project mysql properties

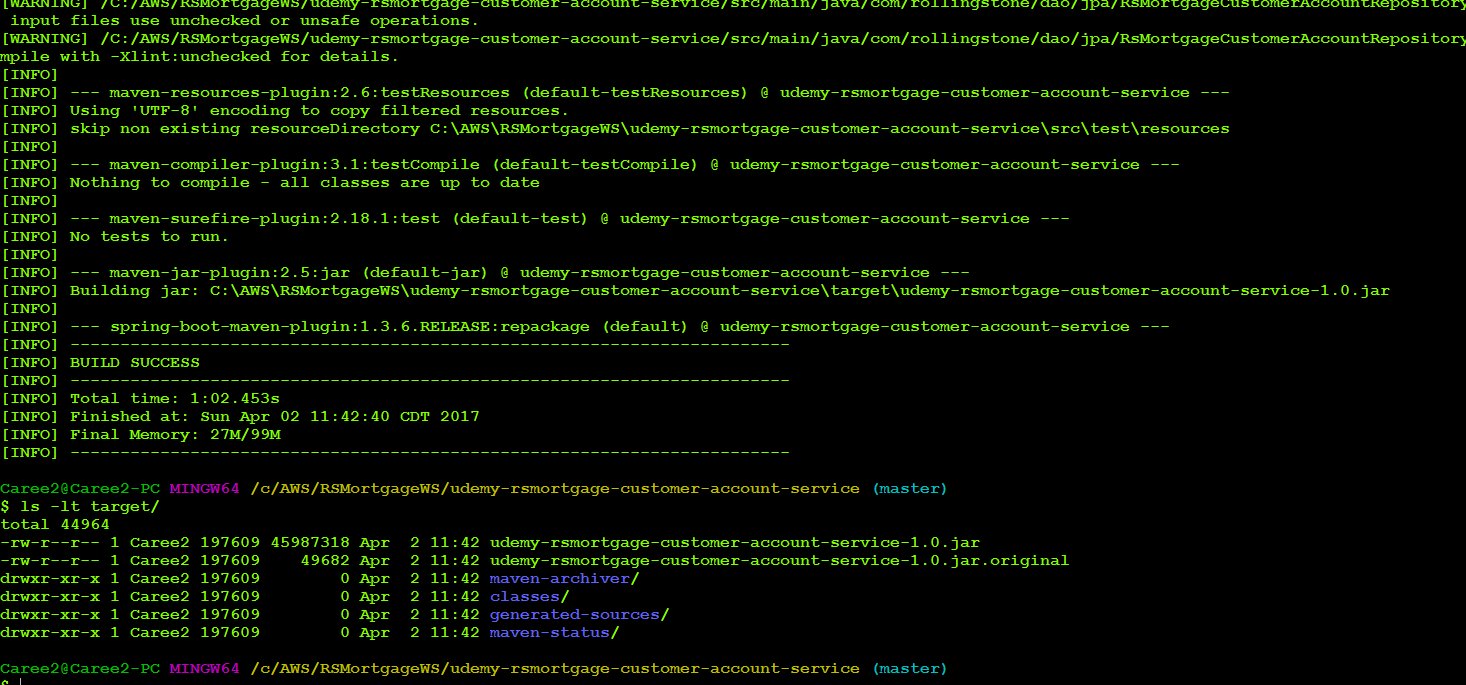


1.80– Open Git bash in the project directory



1.81– Build the project

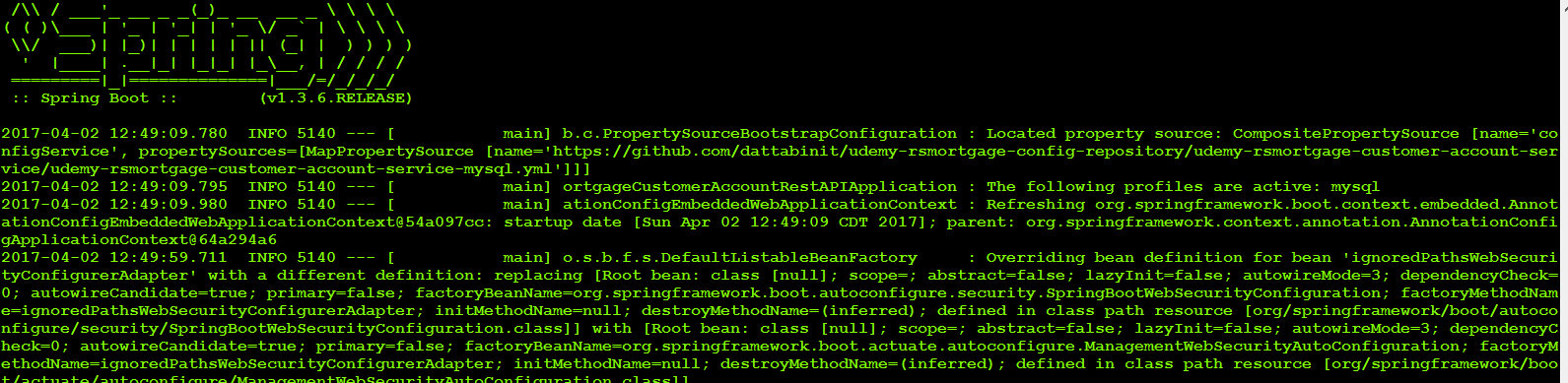




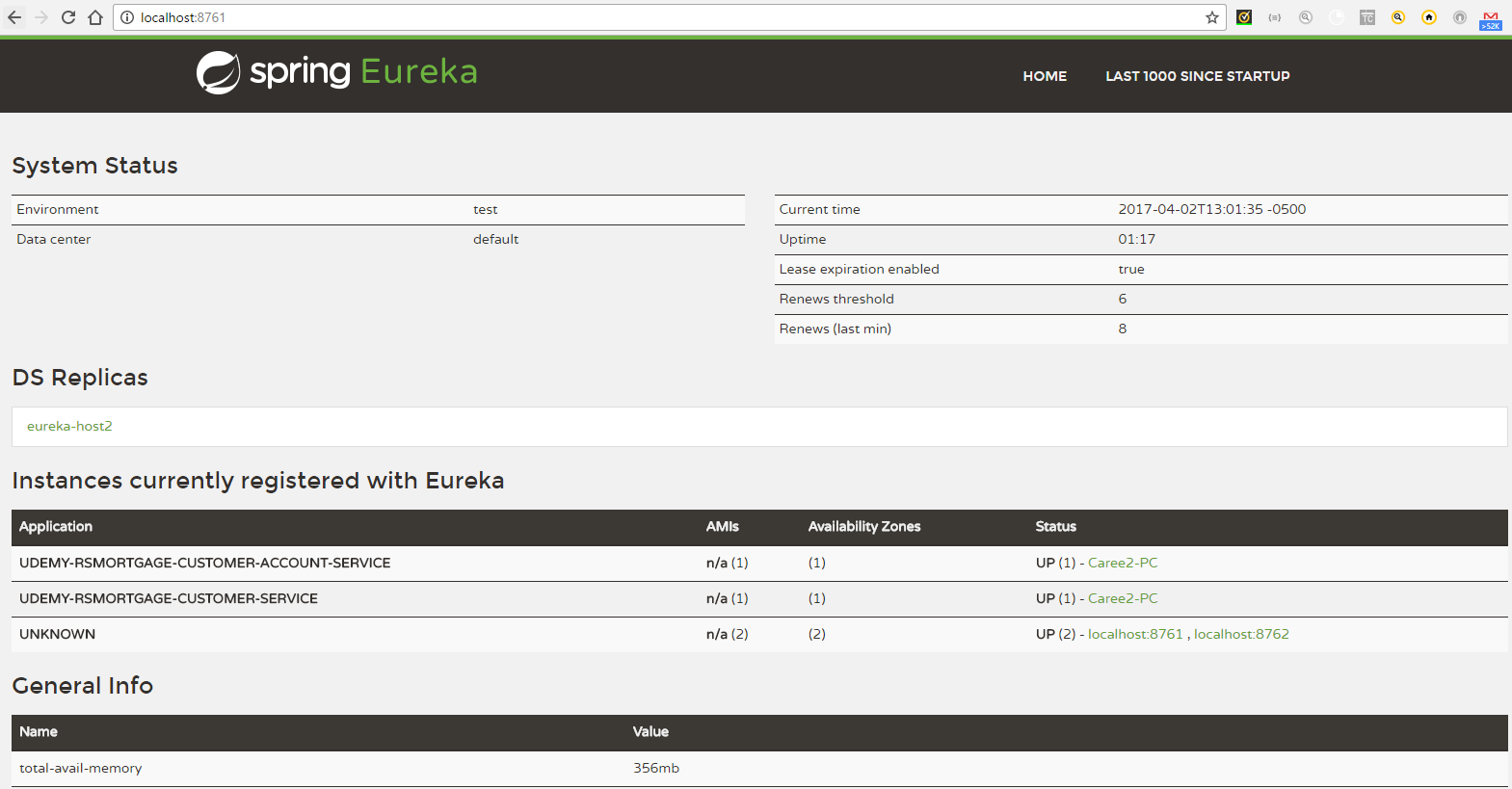
1.82 –Run the Project

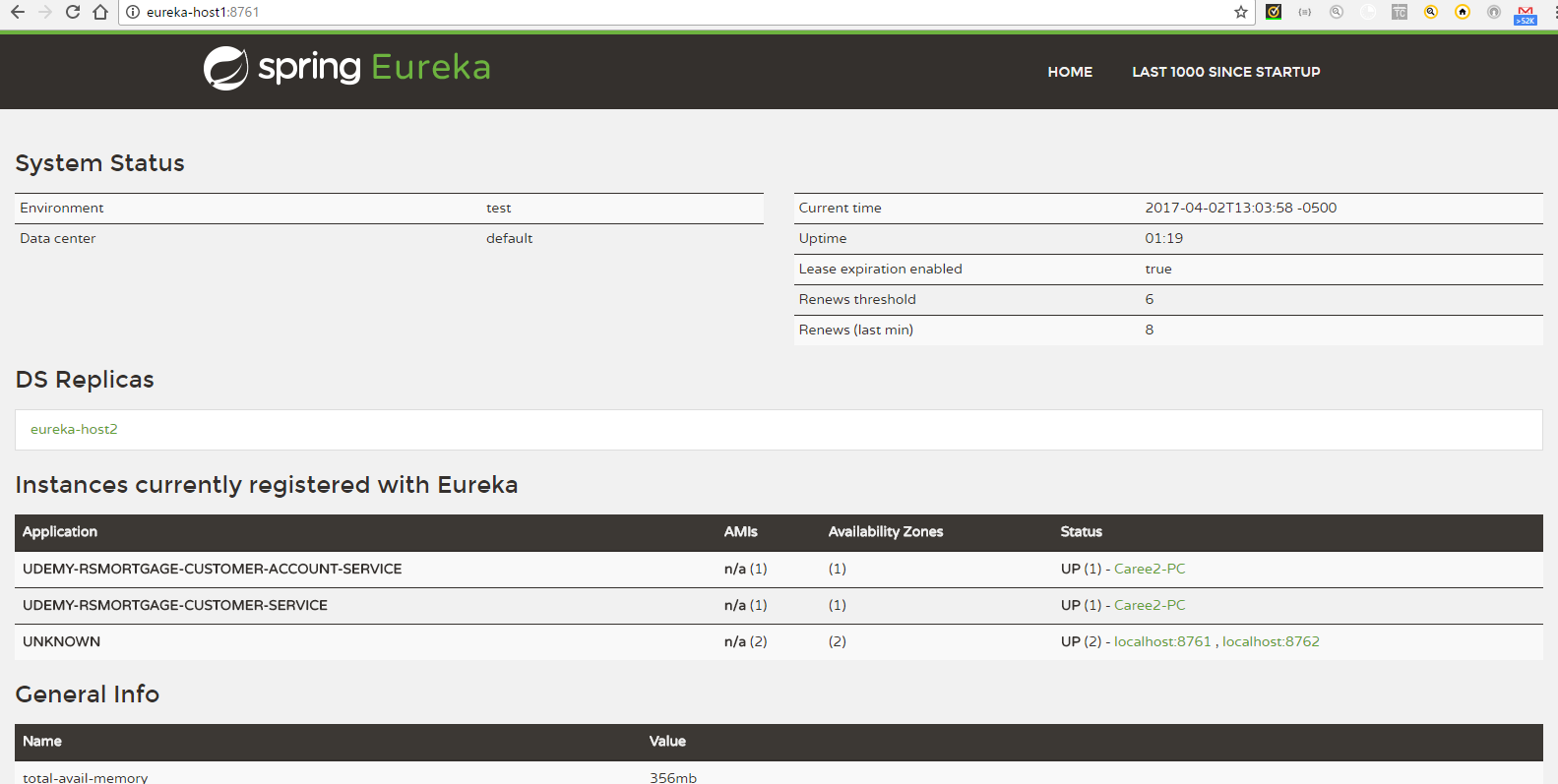
java -jar -Dspring.profiles.active=mysql target/udemy-rsmortgage-customer-account-service-1.0.jar

1.83 –Verify Config Property is read and used



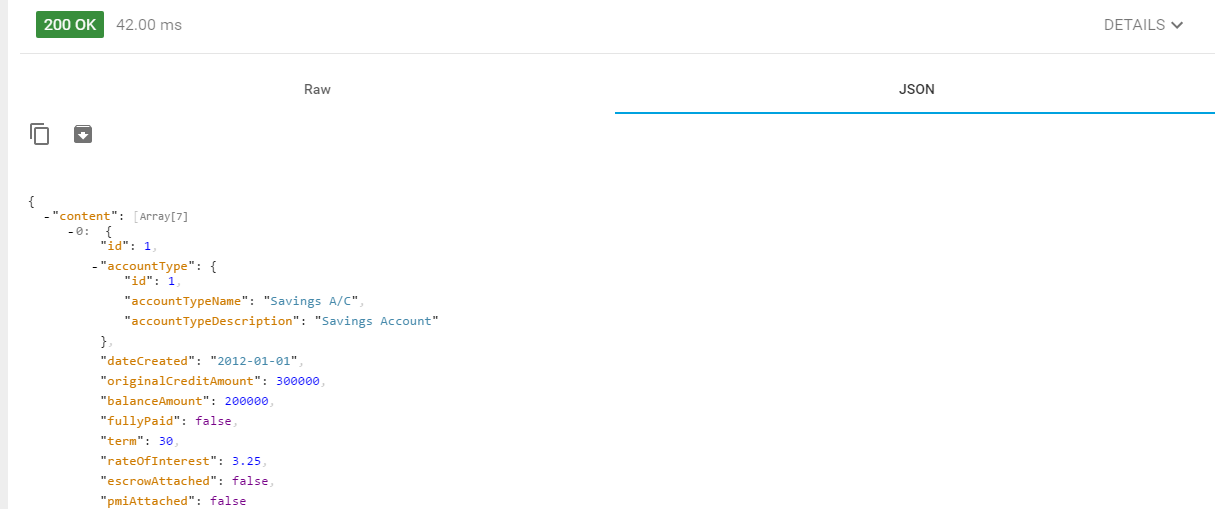
1.84 – Navigate to <http://localhost:8761>





1.85—Get an existing Customer Account

http://localhost:9003/rsmortgage-customer-account-service/v1/customer-account



1.86– Create a Customer Account

<http://localhost:9003/rsmortgage-customer-account-service/v1/customer-account>

Headers

Accept: application/json

Content-Type: application/json

Payload

{

"customer": {

"id": 1,

"firstName": "Jordaya",

"lastName": "Scott",

"dateOfBirth": "1980-04-12",

"totalLoanAmount": 287000,

"bonusPoints": 70000,

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

"socialSecurityNumber": "449-84-4944",

"rating": 7

},

"accountType": {

"id": 2,

"accountTypeName": "Savings A/C",

"accountTypeDescription": "Savings Account"

},

"dateCreated": "2017-01-01",

"originalCreditAmount": 300000,

"balanceAmount": 200000,

"fullyPaid": false,

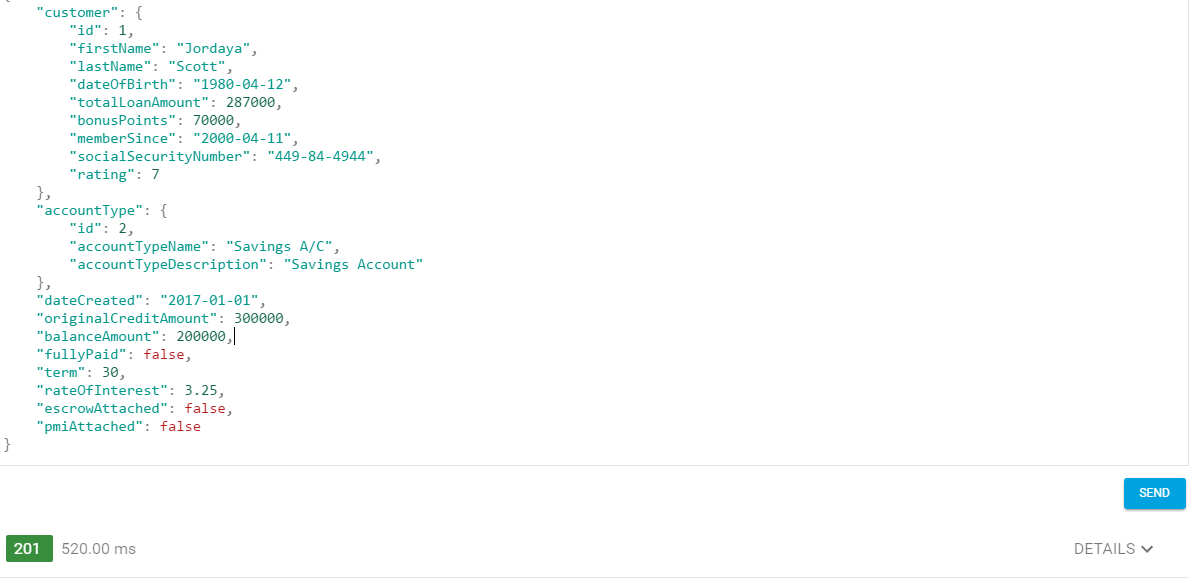
"term": 30,

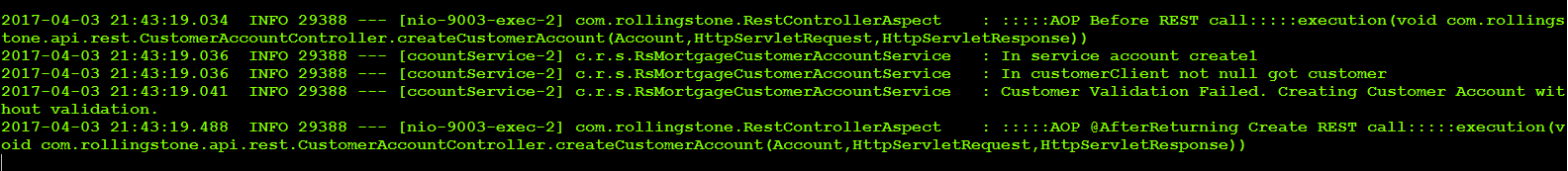
"rateOfInterest": 3.25,

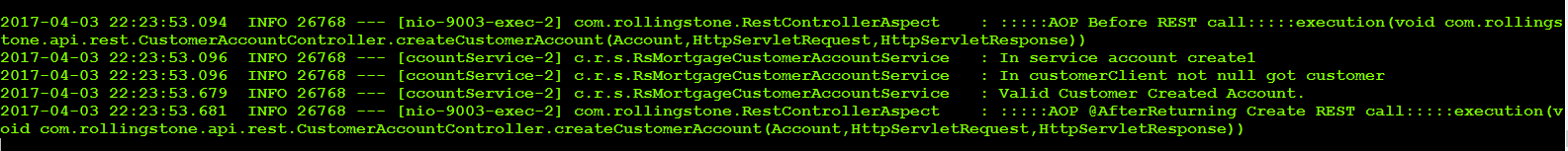
"escrowAttached": false,

"pmiAttached": false

}







1.87– Verify the Database



1.88—Try to Update a Record

<http://localhost:9003/rsmortgage-customer-account-service/v1/customer-account/21>

payload

{

"id":21,

"customer": {

"id": 1,

"firstName": "Jordaya",

"lastName": "Scott",

"dateOfBirth": "1980-04-12",

"totalLoanAmount": 287000,

"bonusPoints": 70000,

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

"socialSecurityNumber": "449-84-4944",

"rating": 7

},

"accountType": {

"id": 2,

"accountTypeName": "Savings A/C",

"accountTypeDescription": "Savings Account"

},

"dateCreated": "2017-01-01",

"originalCreditAmount": 500000,

"balanceAmount": 345670,

"fullyPaid": false,

"term": 60,

"rateOfInterest": 2.25,

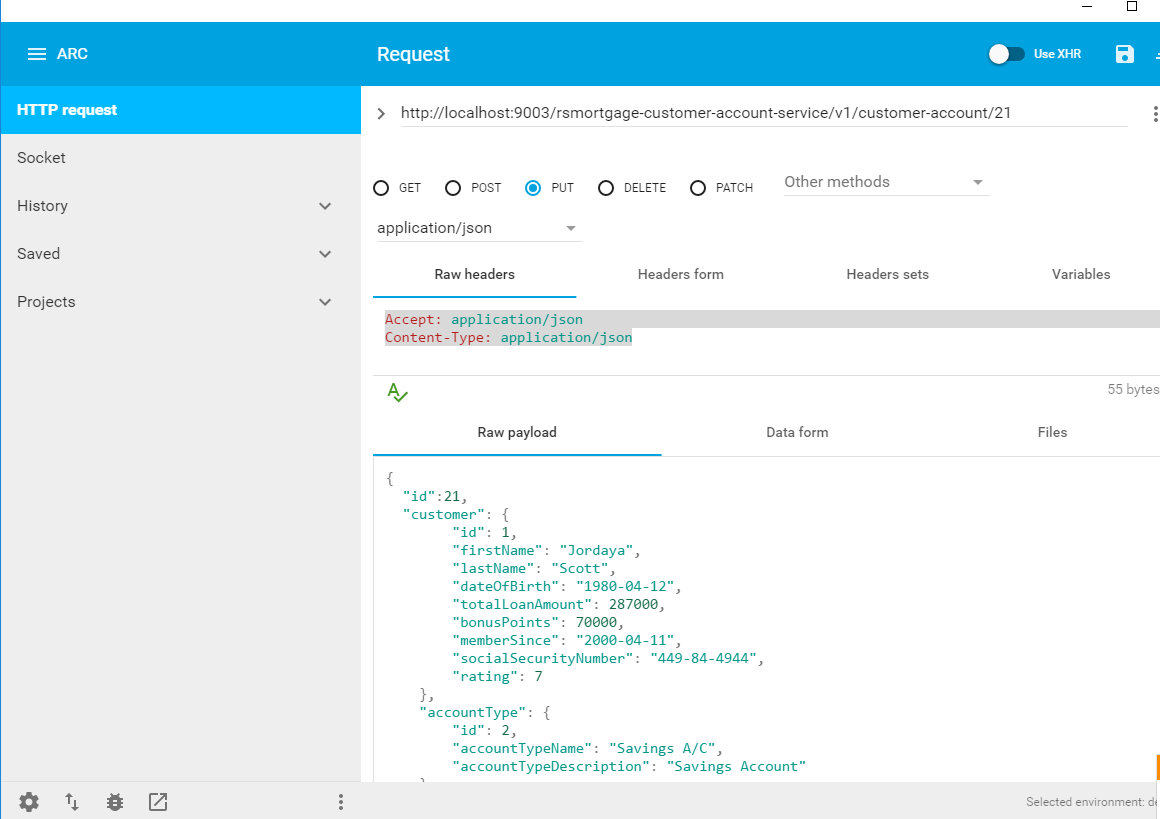
"escrowAttached": false,

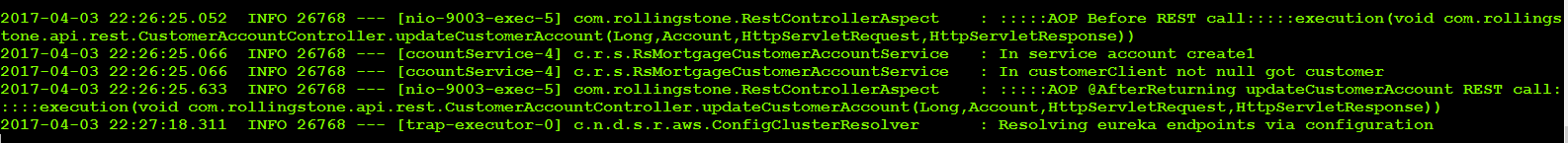
"pmiAttached": false

}

Accept: application/json

Content-Type: application/json



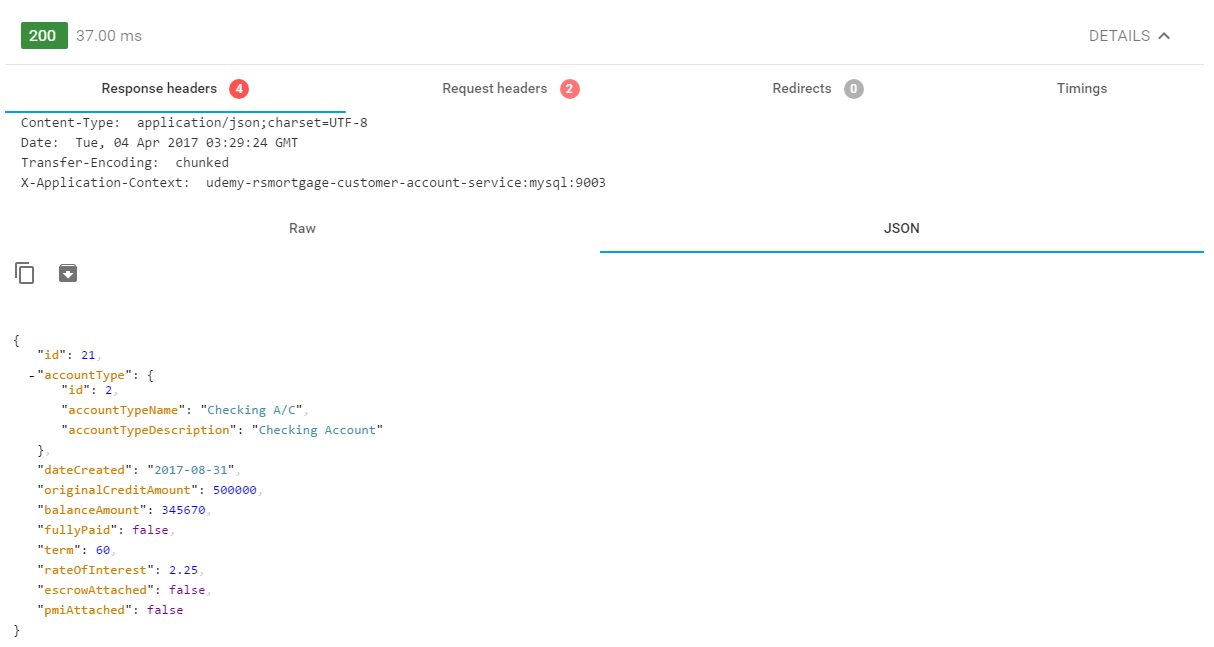


1.89—Verify the Database



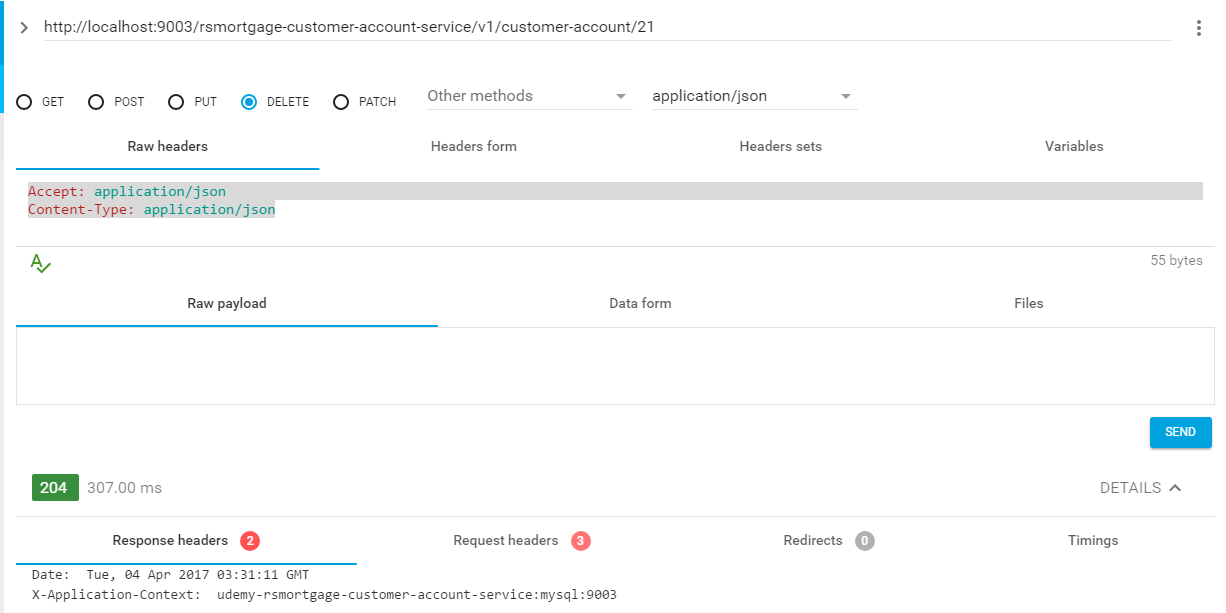
1.90 – Try to get a single customer

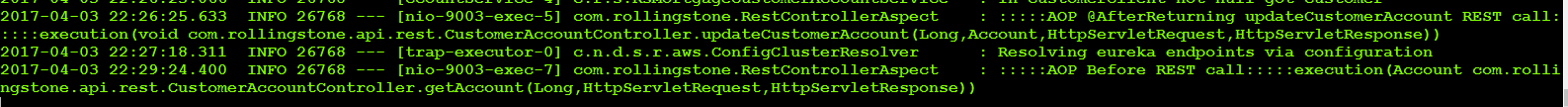
http://localhost:9003/rsmortgage-customer-account-service/v1/customer-account/21



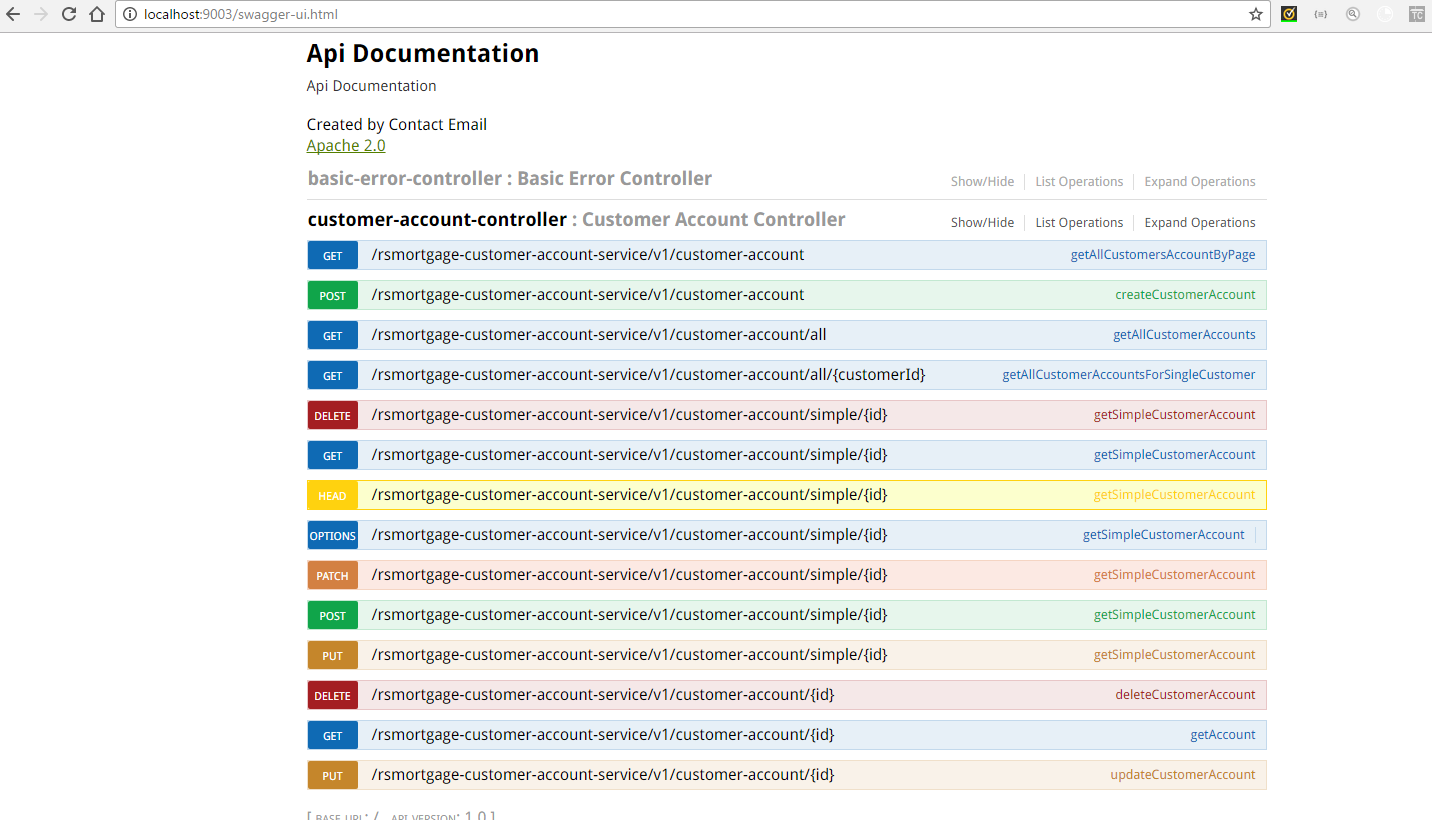
1.91 – Try to delete a single customer

http://localhost:9003/rsmortgage-customer-account-service/v1/customer-account/21

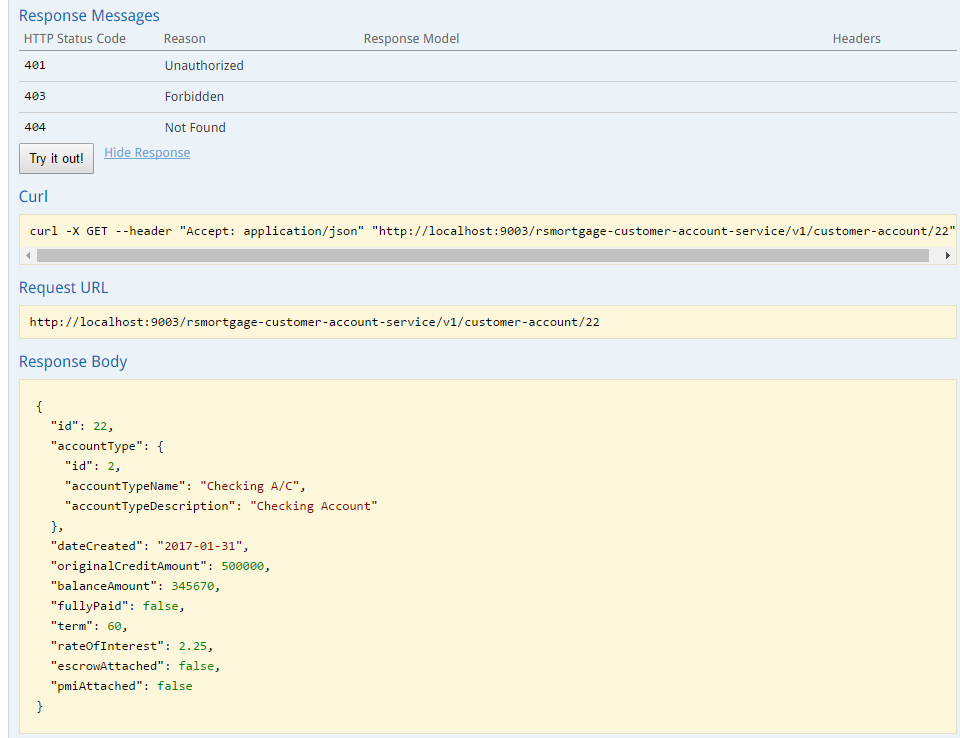




1.92 – Swagger UI







1.93 – Conclusion

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