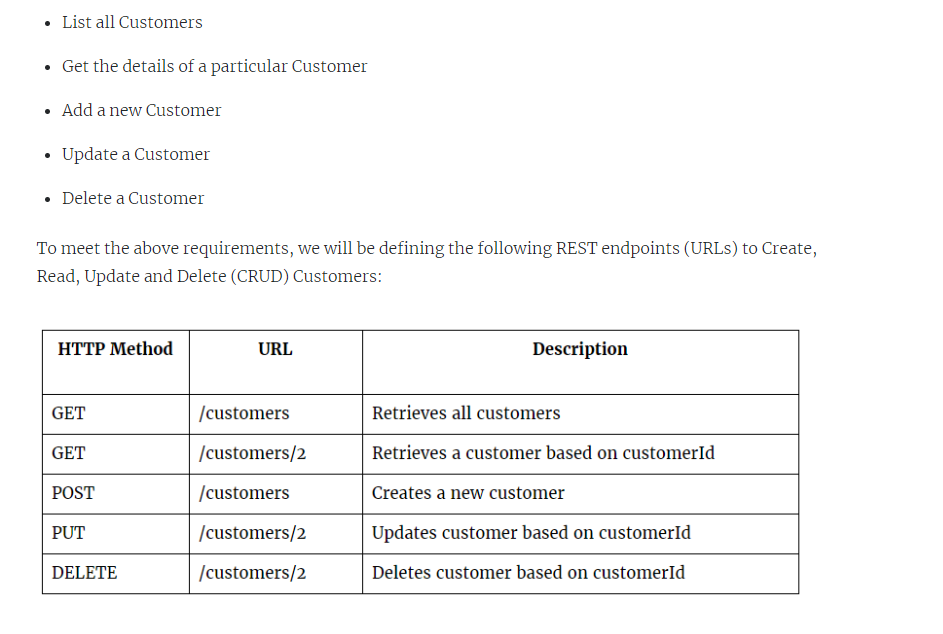
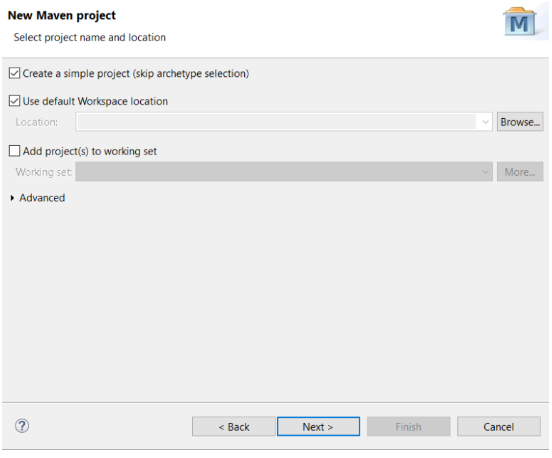
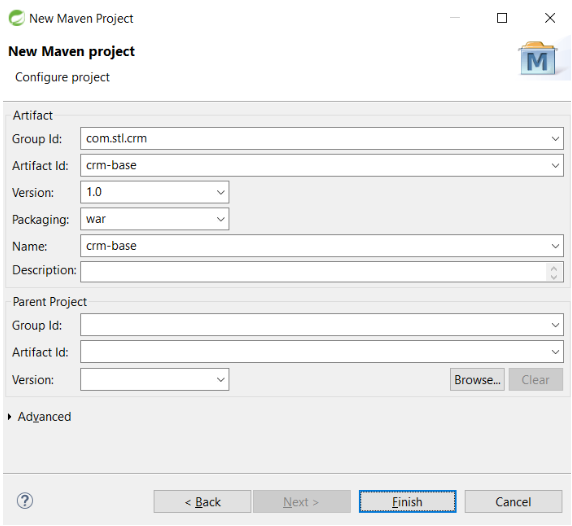
Spring REST API using Spring MVC 5





Enter the following project Artifact as seen in the following screenshot, then click Finish button.

Group Id: com.stl.crm  
Artifact Id: crm-base  
Version: 1.0  
Packaging: war  
Name: crm-base



**Pom.xml**

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.stl.crm</groupId>

<artifactId>crm-base</artifactId>

<version>1.0</version>

<packaging>war</packaging>

<name>crm-base</name>

<properties>

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

<org.springframework.version>5.2.2.RELEASE</org.springframework.version>

<jackson.version>2.10.1</jackson.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>${org.springframework.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

<version>${org.springframework.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>${org.springframework.version}</version>

</dependency>

<!-- Jackson -->

<dependency>

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

<artifactId>jackson-databind</artifactId>

<version>${jackson.version}</version>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>3.0.1</version>

<scope>provided</scope>

</dependency>

</dependencies>

<build>

<finalName>${project.artifactId}</finalName>

<plugins>

<plugin>

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

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

<configuration>

<source>1.8</source>

<target>1.8</target>

<encoding>${project.build.sourceEncoding}</encoding>

</configuration>

</plugin>

<plugin>

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

<artifactId>maven-surefire-plugin</artifactId>

</plugin>

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>build-helper-maven-plugin</artifactId>

</plugin>

<plugin>

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

<artifactId>maven-jar-plugin</artifactId>

</plugin>

<plugin>

<artifactId>maven-war-plugin</artifactId>

<version>2.4</version>

<configuration>

<failOnMissingWebXml>false</failOnMissingWebXml>

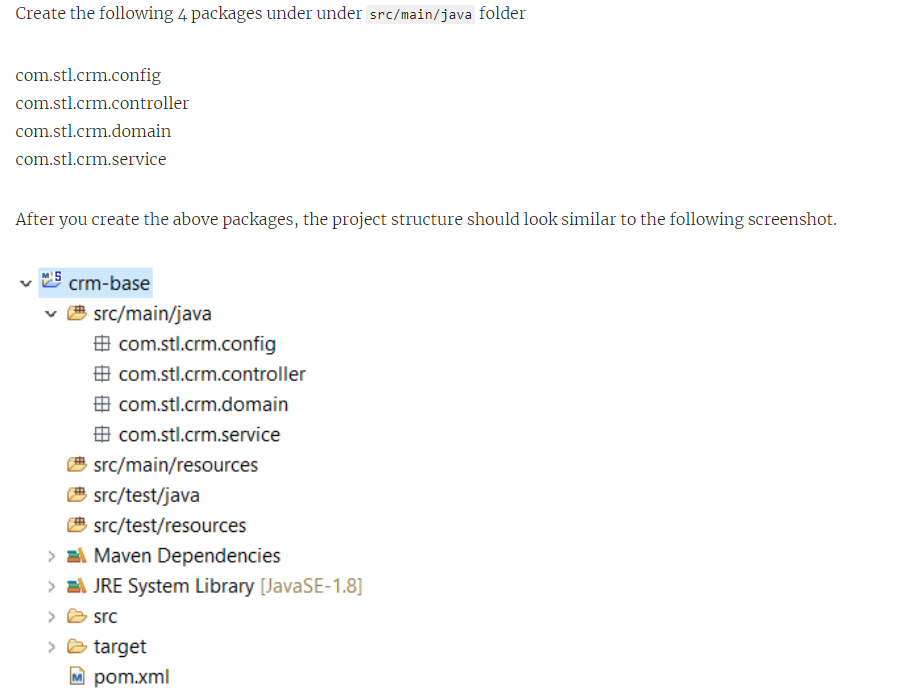
</configuration>

</plugin>

</plugins>

</build>

</project>



**Domain Layer:**

Let’s start with Domain layer. The Domain layer typically consists of a domain objects, which are modeled after database tables. In this post we are not using any database persistence, but we will see it in the next post.

Create a class called Customer under the package com.stl.crm.domain and add the following code into it. As you can see Customer class has minimum set of fields.

**package com.stl.crm.domain;**

**public class Customer {**

**private long id;**

**private String name;**

**private String address;**

**private String phone;**

**private String contact;**

**public long getId() {**

**return id;**

**}**

**public void setId(long id) {**

**this.id = id;**

**} public String getName() {**

**return name;**

**}**

**public void setName(String name) {**

**this.name = name;**

**}**

**public String getAddress() {**

**return address;**

**}**

**public void setAddress(String address) {**

**this.address = address;**

**}**

**public String getPhone() {**

**return phone;**

**}**

**public void setPhone(String phone) {**

**this.phone = phone;**

**}**

**public String getContact() {**

**return contact;**

**}**

**public void setContact(String contact) {**

**this.contact = contact;**

**}**

**}**

**Service Layer:**

Next, let’s create a Service layer. It’s a best practice to use service layer that process any business logic and call Repository layer to persist data. In the next post we will see how to persist data in the database.

Create a class called CustomerService under the package com.stl.crm.service and add the following code into it.

**package com.stl.crm.service;**

**import java.util.ArrayList;**

**import java.util.List;**

**import javax.annotation.PostConstruct;**

**import org.springframework.stereotype.Service;**

**import com.stl.crm.domain.Customer;**

***@Service***

**public class CustomerService {**

**public List<Customer> customerList = null;**

**/\*\***

**\* get all customers**

**\* @return**

**\*/**

**public List<Customer> getCustomers() {**

**return customerList;**

**}**

**/\*\***

**\* get a customer based on the customerId**

**\* @param customerId**

**\* @return**

**\*/**

**public Customer getCustomer(long customerId) {**

**for (Customer customer : customerList) {**

**if (customer.getId() == customerId) {**

**return customer;**

**}**

**}**

**return null;**

**}**

**/\*\***

**\* update existing customer**

**\* @param customerId**

**\* @param customer**

**\* @return**

**\*/**

**public Customer updateCustomer(long customerId, Customer customer) {**

**for (Customer existingCustomer : customerList) {**

**if (existingCustomer.getId() == customer.getId()) {**

**existingCustomer.setName(customer.getName());**

**existingCustomer.setAddress(customer.getAddress());**

**existingCustomer.setPhone(customer.getPhone());**

**existingCustomer.setContact(customer.getContact());**

**return existingCustomer;**

**}**

**}**

**return null;**

**}**

**/\*\***

**\* add a customer**

**\* @param customer**

**\* @return**

**\*/**

**public Customer addCustomer(Customer customer) {**

**customer.setId(customerList.size() + 1);**

**customerList.add(customer);**

**return customer;**

**}**

**/\*\***

**\* delete a customer**

**\* @param customer**

**\*/**

**public void deleteCustomer(Customer customer) {**

**customerList.remove(customer);**

**}**

**/\*\***

**\* set up a few customers on the startup**

**\***

**\*/**

***@PostConstruct***

**private void setupCustomers() {**

**customerList = new ArrayList<Customer>();**

**Customer customer = new Customer();**

**customer.setId(1);**

**customer.setName("Google Inc");**

**customer.setAddress("11600 Amphitheatre Parkway, Mountain View, CA");**

**customer.setPhone("111-222-3333");**

**customer.setContact("VP");**

**customerList.add(customer);**

**customer = new Customer();**

**customer.setId(2);**

**customer.setName("Amazon Inc");**

**customer.setAddress("410 Terry Ave. North, Seattle, WA");**

**customer.setPhone("777-222-3333");**

**customer.setContact("Sales Manager");**

**customerList.add(customer);**

**}**

**}**

**Controller Layer:**

The Controller (API) layer is responsible for receiving client requests, validating user input, interacting with a service layer, and generating a response.

Create a class called CustomerController under the package com.stl.crm.controller and add the following code into it.

**package com.stl.crm.controller;**

**import java.util.List;**

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

**import org.springframework.http.HttpStatus;**

**import org.springframework.http.ResponseEntity;**

**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.RestController;**

**import com.stl.crm.domain.Customer;**

**import com.stl.crm.service.CustomerService;**

***@RestController***

**public class CustomerController {**

***@Autowired***

**private CustomerService customerService;**

**/\*\***

**\***

**\* this method maps the following URL & http method**

**\* URL: http://hostname:port/crm-base/customers**

**\* HTTP method: GET**

**\***

**\*/**

***@RequestMapping*(value="/customers", method = RequestMethod.GET)**

**public ResponseEntity<?> getCustomers() {**

**List<Customer> customerList = customerService.getCustomers();**

**return new ResponseEntity<>(customerList, HttpStatus.OK);**

**}**

**/\*\***

**\***

**\* this method maps the following URL & http method**

**\* URL: http://hostname:port/appName/customers/{customerId}**

**\* HTTP method: GET**

**\***

**\*/**

**@RequestMapping(value="/customers/{customerId}", method = RequestMethod.GET)**

**public ResponseEntity<?> getCustomer(*@PathVariable* long customerId) {**

**Customer customer = customerService.getCustomer(customerId);**

**return new ResponseEntity<>(customer, HttpStatus.OK);**

**}**

**/\*\***

**\***

**\* this method maps the following URL & http method**

**\* URL: http://hostname:port/appName/customers**

**\* HTTP method: POST**

**\***

**\*/**

***@RequestMapping*(value="/customers", method = RequestMethod.POST)**

**public ResponseEntity<?> addCustomer(@RequestBody Customer customer) {**

**Customer newCustomer = customerService.addCustomer(customer);**

**return new ResponseEntity<>(newCustomer, HttpStatus.CREATED);**

**}**

**/\*\***

**\***

**\* this method maps the following URL & http method**

**\* URL: http://hostname:port/appName/customers/customerId**

**\* HTTP method: PUT**

**\***

**\*/**

**@RequestMapping(value = "/customers/{customerId}", method = RequestMethod.PUT)**

**public ResponseEntity<?> updateCustomer(*@PathVariable* long customerId,**

***@RequestBody* Customer customer) {**

**Customer updatedCustomer = customerService.updateCustomer(customerId, customer);**

**return new ResponseEntity<>(updatedCustomer, HttpStatus.OK);**

**}**

**/\*\***

**\***

**\* this method maps the following URL & http method**

**\* URL: http://hostname:port/appName/customers/customerId**

**\* HTTP method: DELETE**

**\***

**\*/**

***@RequestMapping*(value = "/customers/{customerId}", method = RequestMethod.DELETE)**

**public ResponseEntity<?> deleteCustomer(@PathVariable long customerId) {**

**Customer customer = customerService.getCustomer(customerId);**

**customerService.deleteCustomer(customer);**

**return new ResponseEntity<>(HttpStatus.OK);**

**}**

**/\*\***

**\***

**\* this method maps the following URL & http method**

**\* URL: http://hostname:port/appName**

**\* HTTP method: GET**

**\***

**\*/**

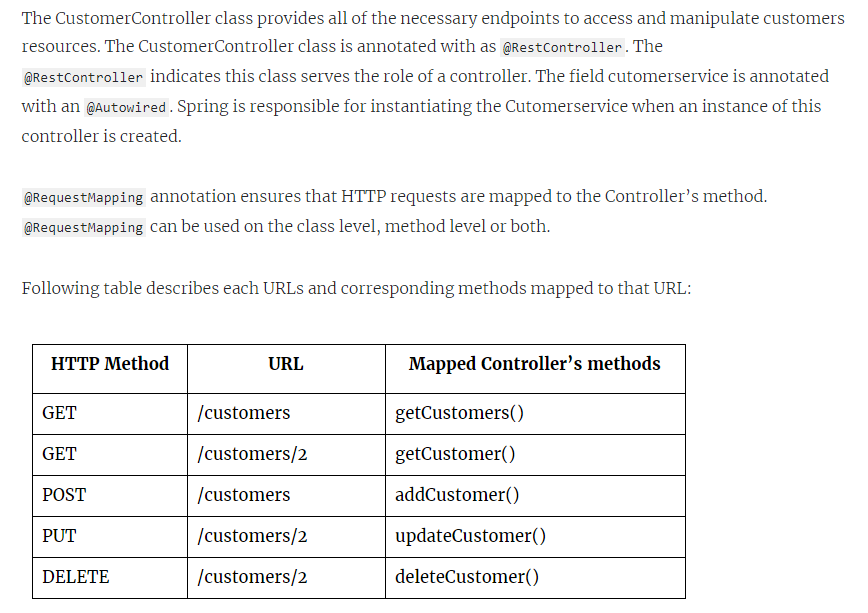
**@RequestMapping(value = "/", method = RequestMethod.GET)**

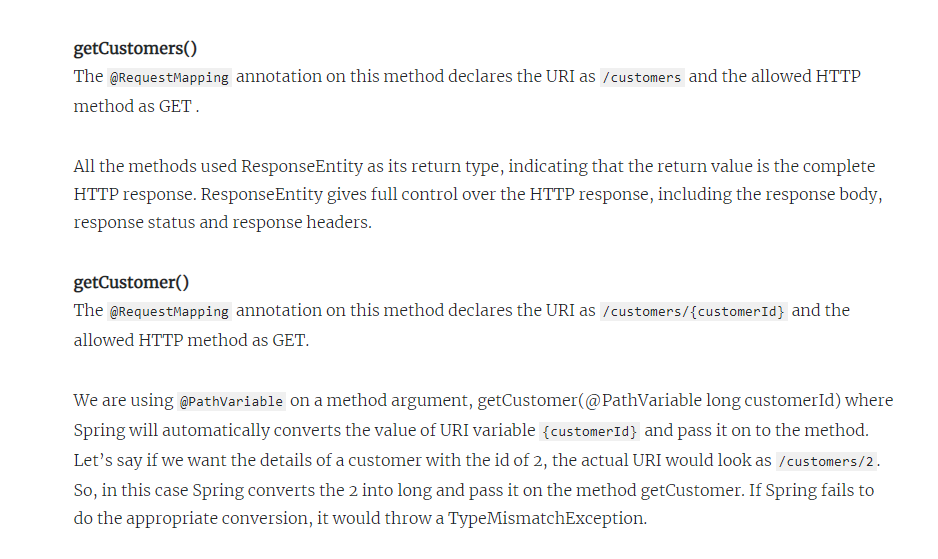
**public ResponseEntity<?> home() {**

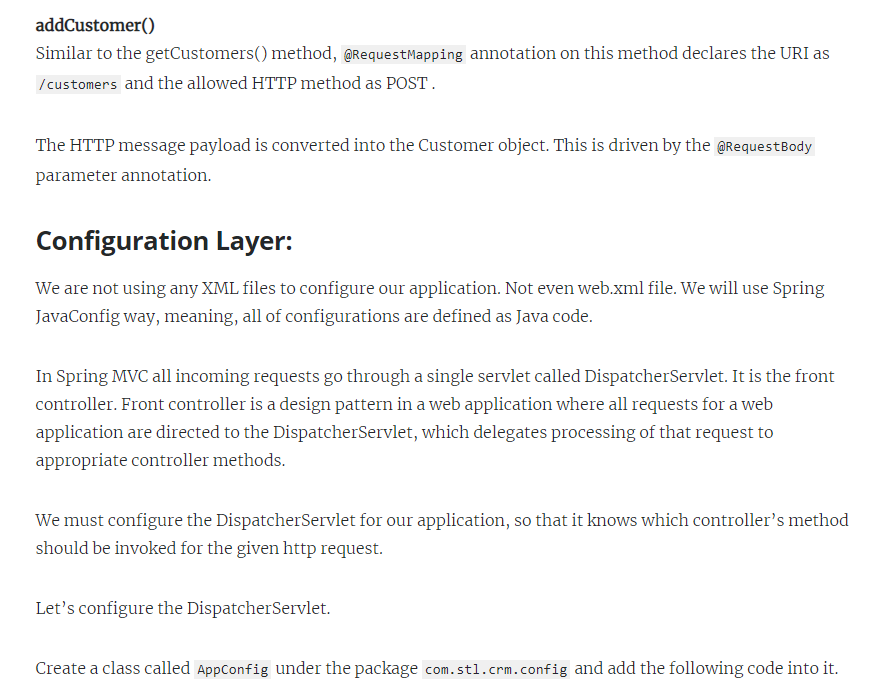
**return new ResponseEntity<>("CRM REST API - Base 1", HttpStatus.OK);**

**}**

**}**







Create a class called AppConfig under the package com.stl.crm.config and add the following code into it.

**package com.stl.crm.config;**

**import org.springframework.context.annotation.ComponentScan;**

**import org.springframework.context.annotation.Configuration;**

**import org.springframework.web.servlet.config.annotation.EnableWebMvc;**

**@Configuration**

**@EnableWebMvc**

**@ComponentScan(basePackages = {"com.stl.crm"})**

**public class AppConfig {**

**}**

Also create another class called AppInitializer under the package com.stl.crm.config and add the following code into it.

package com.stl.crm.config;

**import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;**

**public class AppInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {**

**@Override**

**protected Class<?>[] getRootConfigClasses() {**

**return null;**

**}**

**@Override**

**protected Class<?>[] getServletConfigClasses() {**

**return new Class<?>[] { AppConfig.class};**

**}**

**@Override**

**protected String[] getServletMappings() {**

**return new String[] { "/" };**

**}**

**}**

If you have done any coding on Java web application, you might be familiar with servlet configuration and web.xml. In our case, consider AppInitializer class is similar to web.xml. AbstractAnnotationConfigDispatcherServletInitializer class is used to make pure java based configuration without using a web.xml

We configure the DispatcherServlet to our application using AppInitializer class which extends AbstractAnnotationConfigDispatcherServletInitializer class, where Spring MVC provides the DispatcherServlet.

Now we need to give some hints to DispatcherServlet about the location of the controller classes. We do this in getServeltConfigClasses method by adding the main config class, AppConfig. We have to add this to get things started because this is the class that tells Spring to scan the rest of the classes with the @ComponentScan, which triggers Spring to read and register our controller classes.

As we mentioned, every incoming request first comes to the Dispatcher servlet, but how does the Dispatcher servlet know it should handle every incoming request? We configure this in getServletMappings method where we return the string array containing only the “/” character. This makes sure that every incoming request will be handled by the DispatcherServlet.

Let’s review the annotations mentioned in the AppConfig class:

@Configuration: This annotation tags the class as a source of bean definitions for the application context.

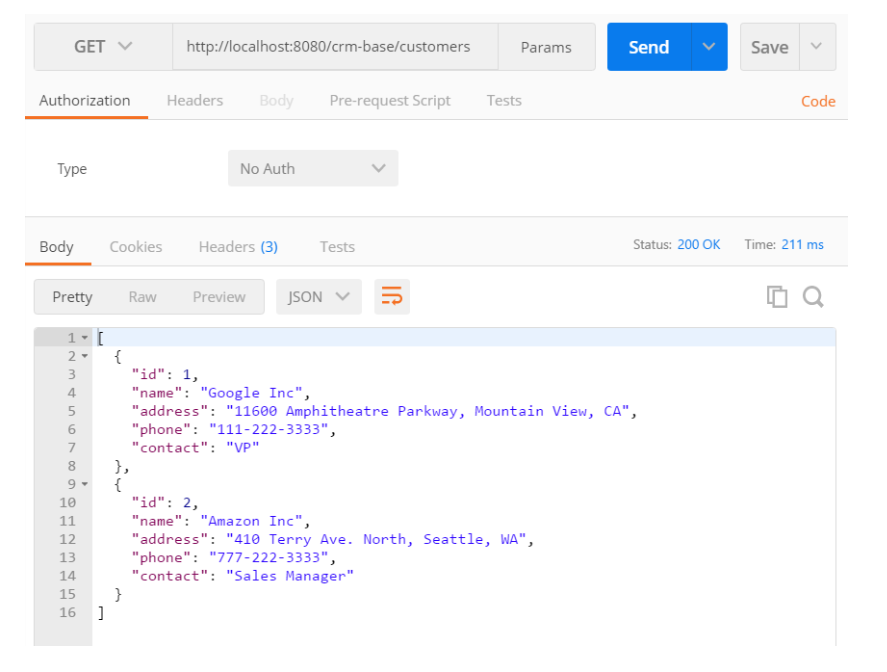
@EnableWebMvc: This annotation enables our application as Spring MVC. It enables support for @RestController annotated classes that use @RequestMapping to map incoming requests to a certain method.

@ComponentScan: This tells Spring to scan the package “com.stl.crm” for any annotated classes. Spring will scan this package recursively and registers any classes annotated by the normal Spring annotations @RestController, @Service, @Component, @Repository and so on.

**List all Customers:**

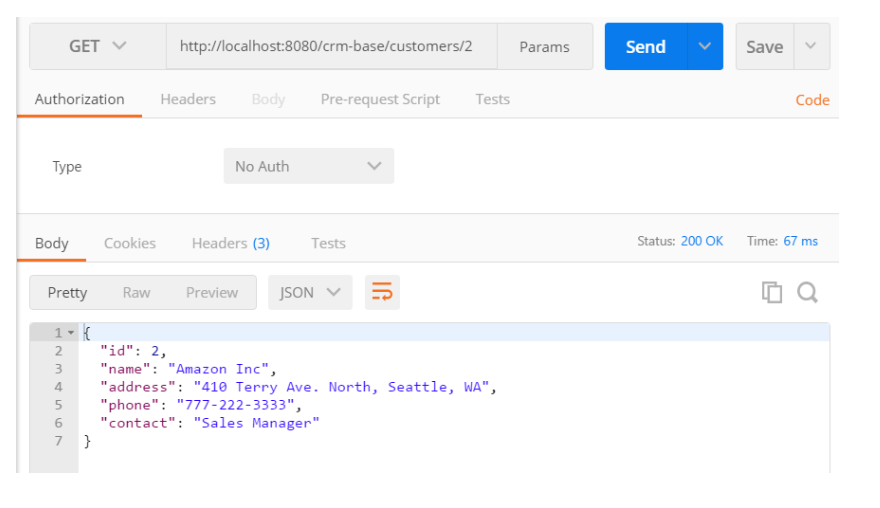
Enter the following URL in the ‘Enter request URL’ box and click on send button.  
http://localhost:8080/crm-base/customers

You would get the response with the list of 2 customers as follows:



**Get the details of a particular Customer:**

Enter the following URL and click on the send button. You will get one customer as follows:  
http://localhost:8080/crm-base/customers/2



**Add a new Customer:**

Enter the following URL and make sure to select the HTTP method as POST.  
http://localhost:8080/crm-base/customers

Select the ‘Body’ tag, click on the ‘raw’ radio button under the Body tag, click the drop down arrow on the ‘Text’ and select JSON(application/json). Enter the following JSON data as the body.

Your selection should match the following screen. Click on the Send button and verify the response.

{

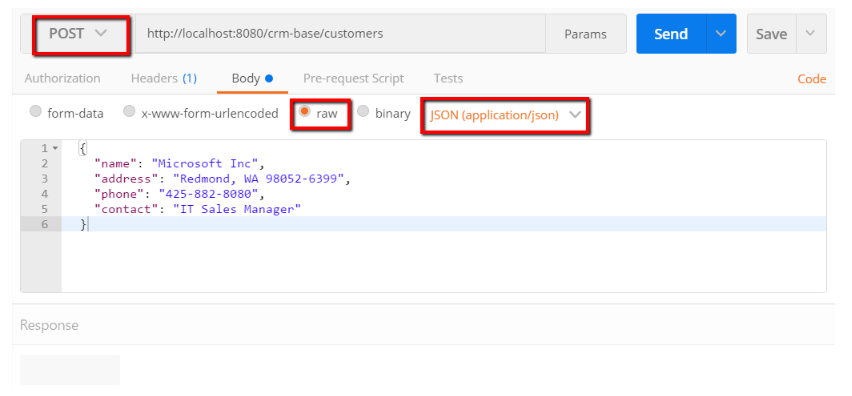
"name": "Microsoft Inc",

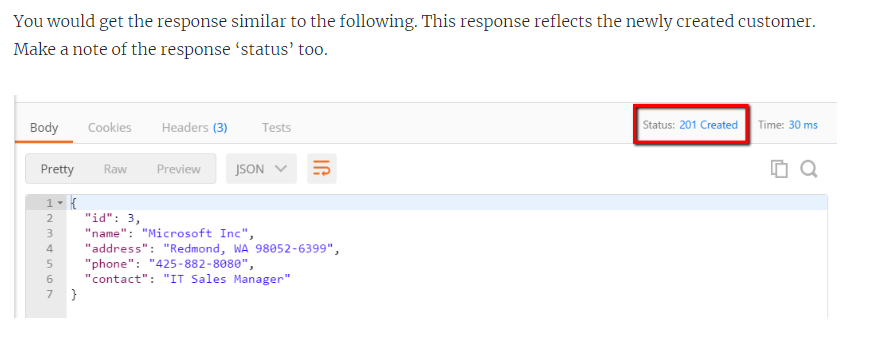
"address": "Redmond, WA 98052-6399",

"phone": "425-882-8080",

"contact": "IT Sales Manager"

}





**Update a Customer:**

Enter the following URL and make sure to select the HTTP method as PUT.  
http://localhost:8080/crm-base/customers/3

Select the ‘Body’ tag, click on the ‘raw’ radio button and select JSON(application/json). Enter the following JSON data as we are making changes to contact and phone fields.

{

"id": 3,

"name": "Microsoft Inc",

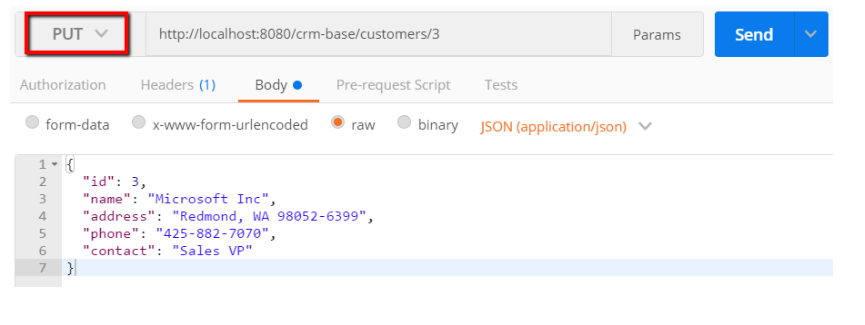
"address": "Redmond, WA 98052-6399",

"phone": "1-800-882-8080",

"contact": "Sales VP"

}

Your selection should match the following screen. Click on the Send button and verify the response. You would get the updated customer data back.



**Delete a Customer:**

Enter the following URL and make sure to select the HTTP method as DELETE.  
http://localhost:8080/crm-base/customers/3

Click on the Send button and verify the response status ‘200 OK’.

