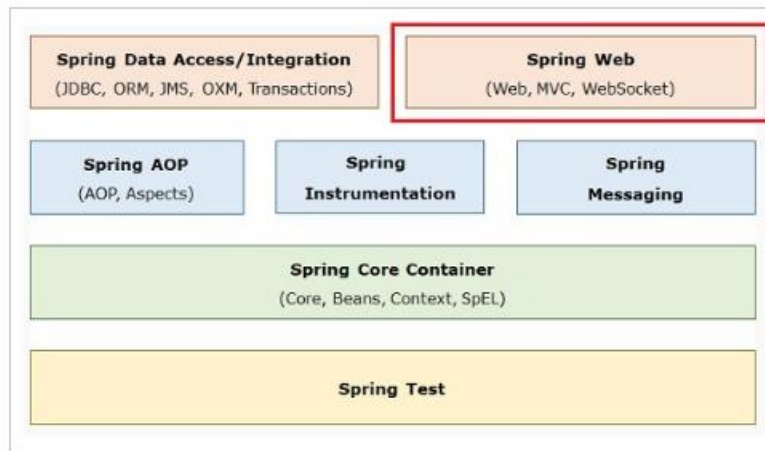


## Unit-5

### Spring REST

#### Spring REST- An Introduction

Spring's web module carries the support for REST as well.



#### Spring MVC and REST

Spring Web MVC is the source of Spring REST as well. So, let us just quickly understand the internals of Spring MVC in detail.

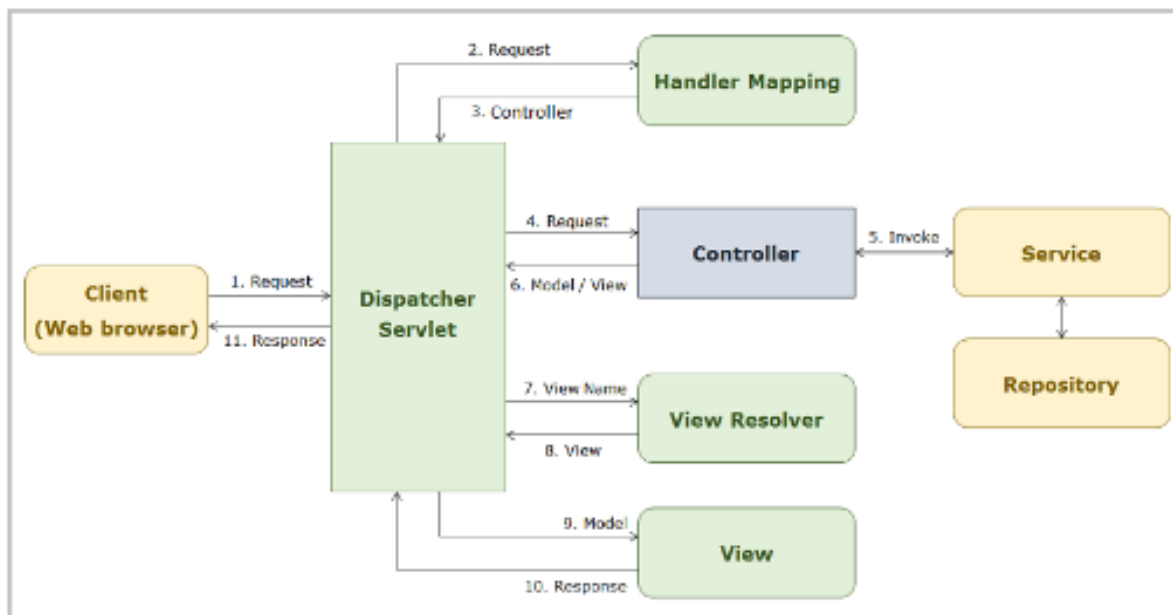
Spring MVC framework is based on MVC and the front controller design patterns.

The core element of Spring MVC is the Dispatcher Servlet and, that is the Front Controller which provides an entry point for handling all the requests and forwards the same to the appropriate components.

The Dispatcher Servlet interacts with handler mappings to determine which controller to execute upon user request. Same way, it consults view resolver to decide the view in which the response to be rendered.

#### Spring MVC Architecture

The following figure shows the working internals of Spring MVC.



The DispatcherServlet intercepts the incoming HTTP request.

1. An application can have multiple controllers. So, the DispatcherServlet consults the handler mapping to decide the controller that should work on the request.
2. The handler mapping uses request URL to choose the controller. Once done, this decision should be sent to the DispatcherServlet back.
3. After receiving appropriate controller name, DispatcherServlet sends the request to the controller.
4. The controller receives the request from the DispatcherServlet and executes the business logic that is available in the service layer.
5. Once done with the business logic, controller generates some information that needs to be sent back to the client and displayed in the web browser. This information is called as model. Now, the controller sends the model and logical name of the view to the DispatcherServlet.
6. The DispatcherServlet passes the logical View name to the ViewResolver, which determines the actual view.
7. The actual view is sent back to the DispatcherServlet, now.
8. The DispatcherServlet then passes the model to the View, which generates the response.
9. The generated response is sent back to the DispatcherServlet.
10. The DispatcherServlet returns the generated response over to the client.

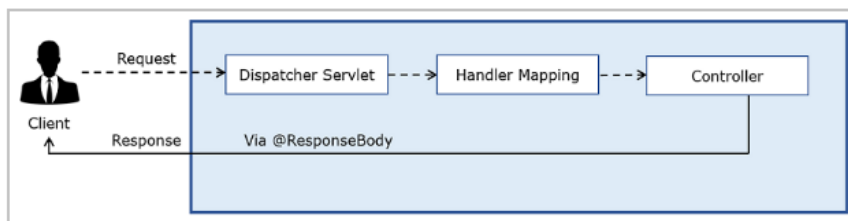
*Note: By default, DispatcherServlet supports GET, HEAD, POST, PUT, PATCH and DELETE HTTP methods only.*

## Spring REST

Spring provides support for creating RESTful web services using Spring MVC. Availing this support requires, Spring version 3.0 and above.

The REST controllers are different from MVC controllers because REST controllers' methods return results which can be mapped to a representation rather than a view.

For this, Spring 3.0 introduced `@ResponseBody` annotation. So, the methods of REST controllers that are annotated with `@ResponseBody` tells the DispatcherServlet that the result of execution need not to be mapped with a view.



**@ResponseBody** annotation automatically converts the response to a JSON string literal by applying serialization on the return value of the method.

## @RestController

In Spring 4.0, the **@RestController** annotation was introduced.

This annotation is a combination of `@Controller` and `@ResponseBody`.

This annotation when used on a REST controller class bounds all the values returned by controller methods to the response body.

Developing Spring REST as a Boot project still simplifies the developers' job as a lot of things are auto-configured.

In our course, we will learn, how to develop a **Spring REST** application using **Spring Boot**.

## Spring MVC vs Spring REST

The below table summarizes the differences between Spring MVC and Spring REST.

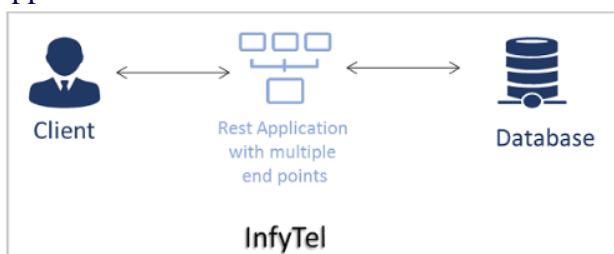
Spring MVC	Spring REST
Spring MVC response is a View/Page by default	In Spring REST data is returned directly back to the client

### Introducing Infytel Case Study

Spring Boot helps to create a REST application with less effort and time. This is because Spring Boot manages dependencies in an efficient way. Also, it abstracts the configurations that help the developers focus only on the business.

Additionally, Spring Boot provides embedded defaults (example - deploys a web application in the embedded Tomcat server) and opinions (example - suggests Hibernate implementation of JPA for ORM).

In this course, we will look at a telecom application called Infytel. As the course proceeds, we will build this app incrementally to make it a collection of REST endpoints. And, we will use Spring Boot to develop this application.

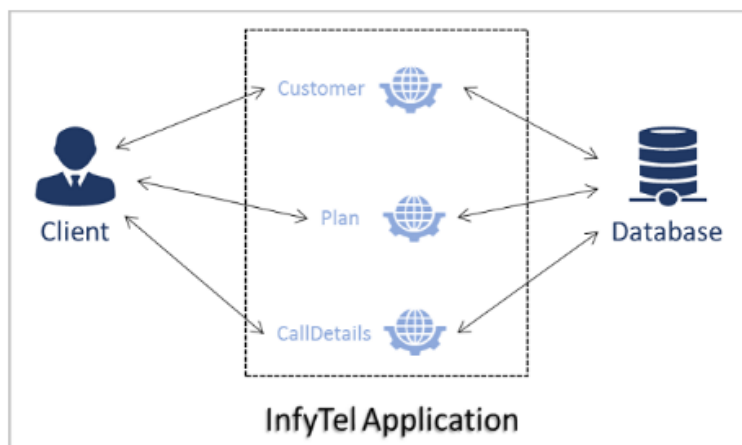


### Infytel - An Overview

The Infytel application will have the following REST resources.

Controller	Functionality
CustomerController	Add, update, delete and fetch customers
PlanController	Fetch plan details
CallDetailsController	Fetch call details

And, these resources contact the same database.



### Steps for executing Infytel

**About the course** section has the complete set of downloadable code and script for the Infytel case study.

Once downloaded, proceed with the following steps for execution.

**Step 1:** Extract the project and import the same as Maven project in STS.

**Step 2:** Make sure the database is up and ready with tables.

**Step 3:** Execute the project as Spring Boot application.

**Step 4:** Pay attention to the console to gain information on port number and the context path.

**Step 5:** Gain request mapping details of the RESTful service methods. That is, have a look at the request and method mapping annotations of the controllers to understand the

- URI format
- template parameters, if any
- request method (GET, POST, etc.,)
- MIME type of Java objects to be passed

Finally, trigger requests to the endpoints using a tool like PostMan.

## Creating a Spring REST Controller

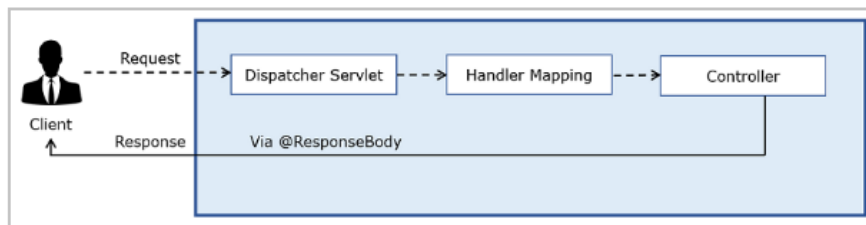
### Spring REST Internals

Spring Web MVC module is the source of Spring REST as well.

#### Working Internals of Spring REST:

Spring REST requests are delegated to the DispatcherServlet that identifies the specific controller with the help of handler mapper. Then, the identified controller processes the request and renders the response. This response, in turn, reaches the dispatcher servlet and finally gets rendered to the client.

Here, ViewResolver has no role to play.



#### Steps involved in exposing the business functionality as a RESTful web service

**What are the steps involved in exposing business functionality as a RESTful web service?**

**Step 1:** Create a REST Resource

**Step 2:** Add the service methods that are mapped against the standard HTTP methods

**Step 3:** Configure and deploy the REST application

Since we are going to develop REST applications using Spring Boot, lot of configurations needed in Step-3 can be avoided as Spring Boot takes care of the same.

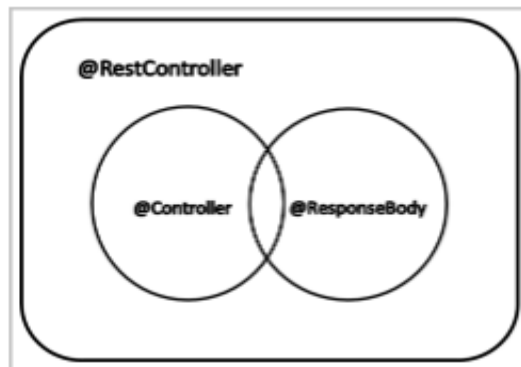
Let us look at each step in detail.

## Creating a REST Resource

Any class that needs to be exposed as a RESTful resource has to be annotated with `@RestController`

### @RestController

- This annotation is used to create REST controllers.
- It is applied on a class in order to mark it as a request handler/REST resource.
- This annotation is a combination of `@Controller` and `@ResponseBody` annotations.
- `@ResponseBody` is responsible for the automatic conversion of the response to a JSON string literal. If `@RestController` is in place, there is no need to use `@ResponseBody` annotation to denote that the Service method simply returns data, not a view.
- `@RestController` is an annotation that takes care of instantiating the bean and marking the same as REST controller.
- It belongs to the package, `org.springframework.web.bind.annotation.RestController`



### @RequestMapping

- This annotation is used for mapping web requests onto methods that are available in the resource classes. It is capable of getting applied at both class and method levels. At method level, we use this annotation mostly to specify the HTTP method.

```
1. import org.springframework.web.bind.annotation.RestController;
2. import org.springframework.web.bind.annotation.RequestMapping;
3. @RestController
4. @RequestMapping("/customers")
5. public class CustomerController
6. {
7.     @RequestMapping(method=RequestMethod.POST)
8.     public String createCustomer()
9.     {
10.         //Functionality goes here
11.     }
12. }
```

## Adding request handler methods

REST resources have handler methods with appropriate HTTP method mappings to handle the incoming HTTP requests. And, this method mapping usually happens with annotations.

For example, in the Infytel application, developed for a telecom company, we would like to create a REST resource that can deal with operations like creating, deleting, fetching and updating the customers. Here comes the summary of HTTP operations and the corresponding mappings. An important point to be noted here is, Infytel is a Spring Boot application.

URI	HTTP Method	CustomerController method	Method description	Annotation to be applied at the method level	New Annotation that can be applied instead of @RequestMapping at the method level
/customers	GET	fetchCustomer()	Will fetch all the customers of Infytel App and return the same.	@RequestMapping(method = RequestMethod.GET)	@GetMapping
/customers	POST	createCustomer()	Will create a new customer	@RequestMapping(method = RequestMethod.POST)	@PostMapping
/customers	DELETE	deleteCustomer()	Will delete an existing customer	@RequestMapping(method = RequestMethod.DELETE)	@DeleteMapping
/customers	UPDATE	updateCustomer()	Will update the details of an existing customer	@RequestMapping(method = RequestMethod.PUT)	@PutMapping

### CustomerController with handler methods

Following is the code of a REST controller that has several handler methods with appropriate method mapping annotations.

```

1. @RestController
2. @RequestMapping("/customers")
3. public class CustomerController
4. {
5.     //Fetching the customer details
6.     @GetMapping
7.     public String fetchCustomer()
8.     {
9.         //This method will fetch the customers of Infytel and return the same.
10.    return "customers fetched successfully";

```

```
11. }
12. //Adding a new customer
13. @PostMapping
14. public String createCustomer()
15. {
16. //This method will persist the details of a customer
17. return "Customer added successfully";
18. }
19. //Updating an existing customer
20. @PutMapping
21. public String updateCustomer()
22. {
23. //This method will update the details of an existing customer
24. return "customer details updated successfully";
25. }
26. //Deleting a customer
27. @DeleteMapping
28. public String deleteCustomer()
29. {
30. //This method will delete a customer
31. return "customer details deleted successfully";
32. }
33. }
```

### Configuring a REST Controller

Since we are creating a Spring Boot application with spring-boot-starter-web dependency, we need not pay much focus on the configurations.

- spring-boot-starter-web dependency in the pom.xml will provide the dependencies that are required to build a Spring MVC application including the support for REST. Also, it will ensure that our application is deployed on an embedded Tomcat server and we can replace this option with the one that we prefer based on the requirement.

```
1. <dependency>
2. <groupId>org.springframework.boot</groupId>
3. <artifactId>spring-boot-starter</artifactId>
4. </dependency>
5. <!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-web -->
6. <dependency>
7. <groupId>org.springframework.boot</groupId>
8. <artifactId>spring-boot-starter-web</artifactId>
9. </dependency>
```

- Application class is simply annotated with @SpringBootApplication

```
1. @SpringBootApplication
```

```
2. public class InfytelDemo1Application {
3.     public static void main(String[] args) {
4.         SpringApplication.run(InfytelDemo1Application.class, args);
5.     }
6. }
```

@SpringBootApplication is a convenient annotation that adds the following:

- @Configuration marks the class as a source where bean definitions can be found for the application context.
- @EnableAutoConfiguration tells Spring Boot to start adding beans based on the classpath and property settings.
- @ComponentScan tells Spring to look for other components, configurations, and, services in the packages being specified.

And, one more point to be added here is,

@EnableWebMvc is used explicitly in a regular Spring MVC application. But, Spring Boot adds it automatically when it sees spring-webmvc on the classpath. This marks the application as a web application and enables the key behaviors such as setting up the DispatcherServlet etc.

The main() method uses Spring Boot's SpringApplication.run() method to launch an application. Notice, we haven't written any web.xml file. This web application uses 100% pure Java-based configuration and we need not deal with any configurations (no XML files except pom.xml) pertaining to infrastructure.

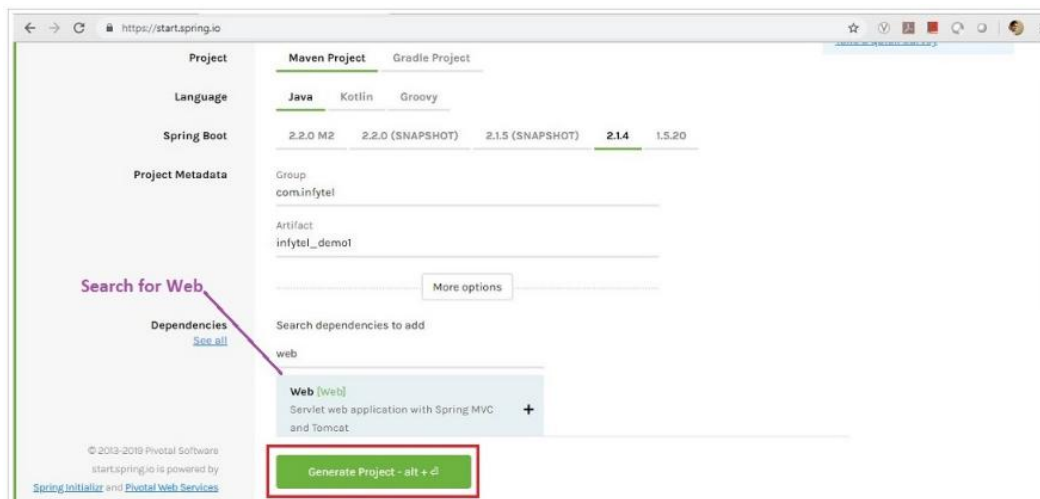
## Demo:

### Objectives:

To create a Spring REST application using Spring Boot.

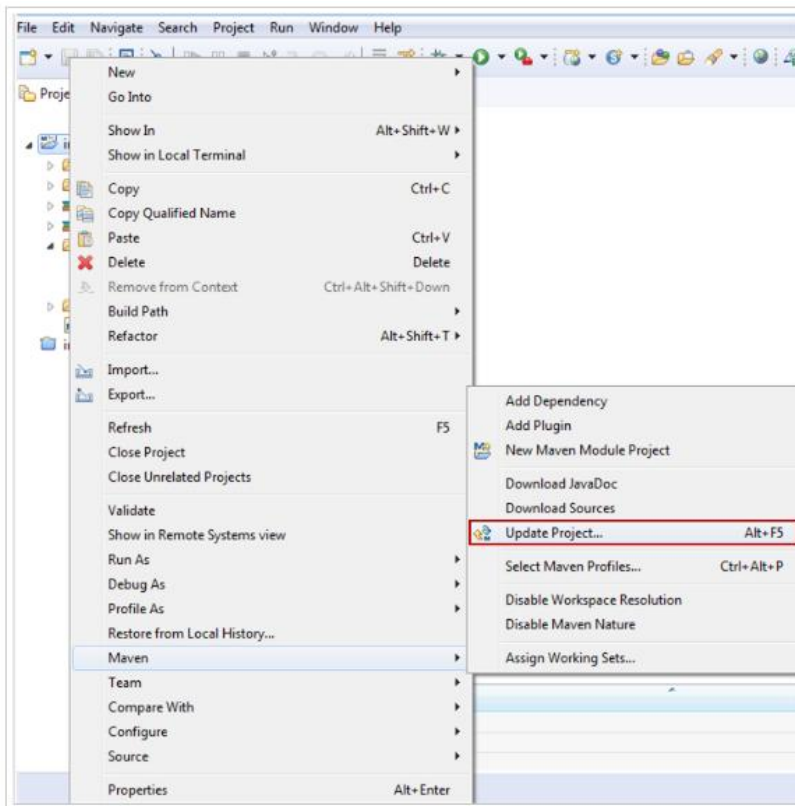
### Steps:

**Step 1:** Create a Maven project using Spring Initializer with maven web dependencies and import the same in STS.

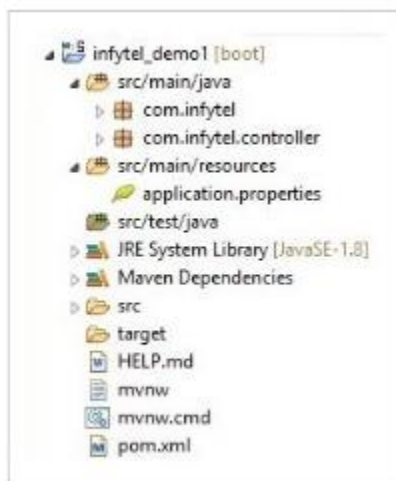


After importing the project, if the dependencies are not getting downloaded automatically, update Maven project as shown below.





**Step 2:** Modify the imported project according to the following project structure:



**Step 3:** Look at the class, **InfytelDemo1Application** in **com.infytel** package, this will be auto-created when the project is generated. We will modify this only when we need to include custom configurations.

1. package com.infytel;
2. import org.springframework.boot.SpringApplication;
3. import org.springframework.boot.autoconfigure.SpringBootApplication;
4. @SpringBootApplication
5. public class InfytelDemo1Application {
6. public static void main(String[] args) {
7. SpringApplication.run(InfytelDemo1Application.class, args);
8. }

9. }

**Step 4:** Create a class **CustomerController** in com.infytel.controller package:

```
1. package com.infytel.controller;
2. import org.springframework.web.bind.annotation.DeleteMapping;
3. import org.springframework.web.bind.annotation.GetMapping;
4. import org.springframework.web.bind.annotation.PostMapping;
5. import org.springframework.web.bind.annotation.PutMapping;
6. import org.springframework.web.bind.annotation.RequestMapping;
7. import org.springframework.web.bind.annotation.RestController;
8. @RestController
9. @RequestMapping("/customers")
10. public class CustomerController
11. {
12. //Fetching the customer details
13. @GetMapping
14. public String fetchCustomer()
15. {
16. //This method will fetch the customers of Infytel and return the same.
17. return "customers fetched successfully";
18. }
19. //Adding a new customer
20. @PostMapping
21. public String createCustomer()
22. {
23. //This method will persist the details of a customer
24. return "Customer added successfully";
25. }
26. //Updating an existing customer
27. @PutMapping
28. public String updateCustomer()
29. {
30. //This method will update the details of an existing customer
31. return "customer details updated successfully";
32. }
33. //Deleting a customer
34. @DeleteMapping
35. public String deleteCustomer()
36. {
37. //This method will delete a customer
38. return "customer details deleted successfully";
39. }
```

```
40. }
```

**Step 5:** Add the following content to **application.properties** file:

1. server.port = 8080
2. server.servlet.context-path=/infytel-1

**Step 6:** Make sure that the project's pom.xml looks similar to the one that is shown below.

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <project xmlns="http://maven.apache.org/POM/4.0.0"
3.   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
   4.0.0.xsd">
5.   <modelVersion>4.0.0</modelVersion>
6.   <parent>
7.     <groupId>org.springframework.boot</groupId>
8.     <artifactId>spring-boot-starter-parent</artifactId>
9.     <version>2.1.4.RELEASE</version>
10.    <relativePath /> <!-- lookup parent from repository -->
11.  </parent>
12.  <groupId>com.infytel</groupId>
13.  <artifactId>infytel_demo1</artifactId>
14.  <version>0.0.1-SNAPSHOT</version>
15.  <name>infytel_demo1</name>
16.  <description>Spring REST using Spring Boot - Basic Annotations</description>
17.  <properties>
18.    <java.version>1.8</java.version>
19.  </properties>
20.  <dependencies>
21.    <!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-web -->
22.    <dependency>
23.      <groupId>org.springframework.boot</groupId>
24.      <artifactId>spring-boot-starter-web</artifactId>
25.    </dependency>
26.    <dependency>
27.      <groupId>org.springframework.boot</groupId>
28.      <artifactId>spring-boot-starter-test</artifactId>
29.      <scope>test</scope>
30.    </dependency>
31.  </dependencies>
32.  <build>
33.    <plugins>
34.      <plugin>
35.        <groupId>org.springframework.boot</groupId>
```

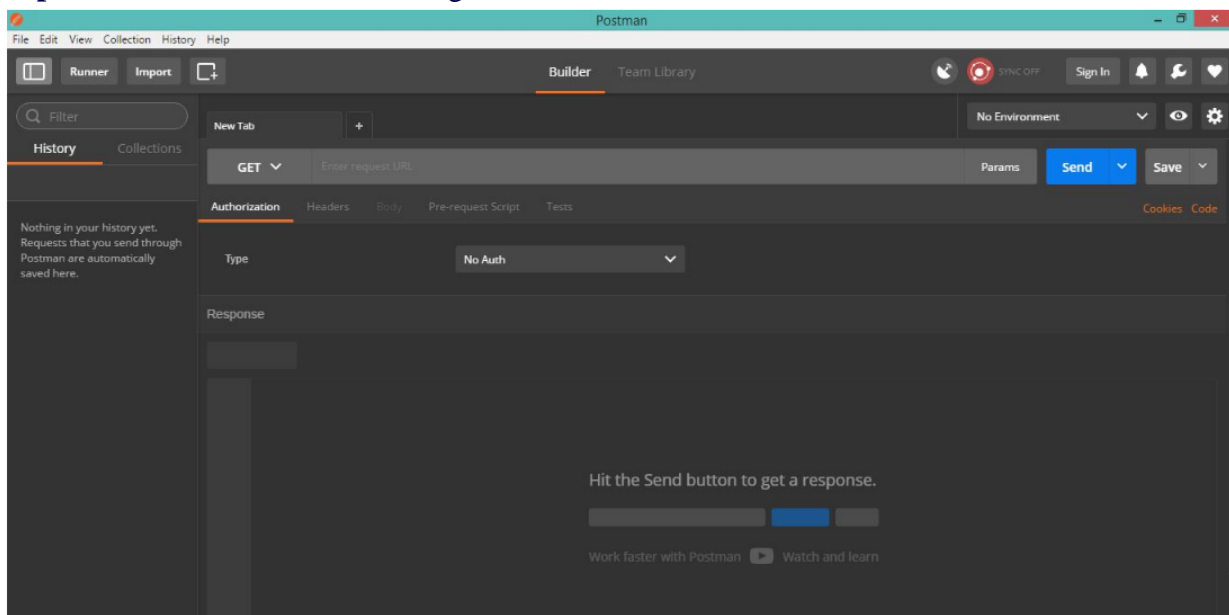
```
36. <artifactId>spring-boot-maven-plugin</artifactId>
37. </plugin>
38. </plugins>
39. </build>
40. </project>
```

**Step 7:** Deploy the service on the server by executing the class containing the main method. So, we have successfully created and deployed RESTful web service. Now let us see how the same can be tested using Postman client.

## To create a Spring REST application using Spring Boot.

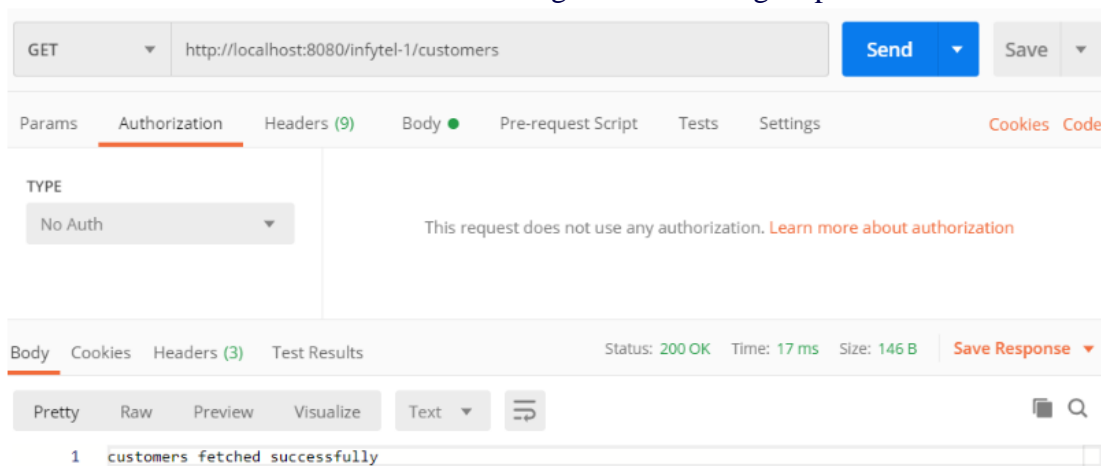
### Testing Web Service using Postman

**Step 1:** Launch Postman. You will get a screen as shown below:



### Step 2: Testing GET request

From the drop-down, select GET and enter **http://localhost:8080/infytel-1/customers** into the URL field to test the service. Click on Send. You will get the following response:



**Step 3: Testing POST request**

From the drop-down, select POST and enter **http://localhost:8080/infytel-1/customers** into the URL field to test the service. Click on Send. You will get the following response:

The screenshot shows the REST Client interface for a POST request. The URL is `http://localhost:8080/infytel-1/customers`. The request is sent, and the response is displayed in the 'Body' tab. The response status is 200 OK, with a time of 38 ms and a size of 143 B. The response body is `1 Customer added successfully`.

**Step 4: Testing PUT request**

From the drop-down, select PUT and enter **http://localhost:8080/infytel-1/customers** into the URL field to test the service. Click on Send. You will get the following response:

The screenshot shows the REST Client interface for a PUT request. The URL is `http://localhost:8080/infytel-1/customers`. The request is sent, and the response is displayed in the 'Body' tab. The response status is 200 OK, with a time of 19 ms and a size of 153 B. The response body is `1 customer details updated successfully`.

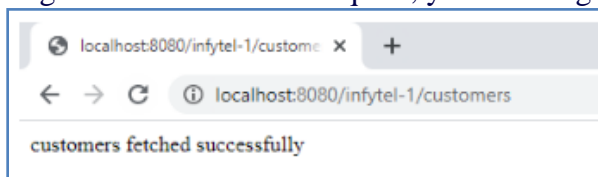
**Step 5: Testing DELETE request**

From the drop-down, select DELETE and enter **http://localhost:8080/infytel-1/customers** into the URL field to test the service. Click on Send. You will get the following response:

The screenshot shows the REST Client interface for a DELETE request. The URL is `http://localhost:8080/infytel-1/customers`. The request is sent, and the response is displayed in the 'Body' tab. The response status is 200 OK, with a time of 384 ms and a size of 153 B. The response body is `1 Customer details deleted successfully`.

**Note:** For GET operation, you can test on a browser as well.

Enter our application URL: **http://localhost:8080/infytel-1/customers** into the address bar of your browser to get the result of GET request, you should get the following response.



### @RequestBody

**@RequestBody** is the annotation that helps map our HTTP request body to a Java DTO. And, this annotation has to be applied on the local parameter (Java DTO) of the request method.

Whenever Spring encounters **@RequestBody**, it takes the help of the registered **HttpMessageConverters** which will help convert the HTTP request body to Java Object depending on the MIME type of the Request body.

Example: In the below code snippet, the incoming HTTP request body is deserialized to **CustomerDTO**. If the MIME type of the incoming data is not mentioned, it will be considered as JSON by default and Spring will use the JSON message converter to deserialize the incoming data.

```
1. @PostMapping
2. public String createCustomer( @RequestBody CustomerDTO customerDTO)
3. {
4. // logic goes here
5. }
```

We can specify the expected Mime type using the **consumes** attribute of the HTTP method matching annotation

**Example:** The above code is equivalent to the below code that promotes the application of the attribute, **consumes**.

```
1. @PostMapping(consumes="application/json")
2. public ResponseEntity<String> createCustomer( @RequestBody CustomerDTO customerDTO)
3. {
4. // logic goes here
5. }
```

**consumes** attribute can be supplied with a single value or an array of media types as shown below

```
1. consumes = "text/plain"
2. or
3. consumes = {"text/plain", "application/json"}
4. Some more valid values:
5. consumes = "application/json"
6. consumes = {"application/xml", "application/json"}
```

```
7. consumes = {"text/plain", "application/*"}
```

## ResponseEntity

How to send a Java Object in the response?

### Scenario-1

Java objects can be returned by the handler method just like how the normal Java methods can return an object.

**Example:** In the below example, the `fetchCustomer()` returns a list of Customer Objects. This list will be converted by Spring's message converter to JSON data.

```
1. @GetMapping
2. public List<CustomerDTO> fetchCustomer()
3. {
4.     //business logic goes here
5.     return customerService.fetchCustomer();
6. }
```

### Scenario-2

We can specify the MIME type, to which the data to be serialized, using the **produces** attribute of HTTP method matching annotations

```
1. @GetMapping(produces="application/json")
2. public List<CustomerDTO> fetchCustomer()
3. {
4.     //This method will return the customers of Infytel
5.     return customerService.fetchCustomer();
6. }
```

Just like **consumes**, the attribute, **produces** can also take a single value or an array of MIME types

```
1. Valid values for produces attribute:
2. produces = "text/plain"
3. produces = {"text/plain", "application/json"}
4. produces = {"application/xml", "application/json"}
```

## Setting response using ResponseEntity

While sending a response, we may like to set the HTTP status code and headers .To help achieving this, we can use `ResponseEntity` class.

**ResponseEntity<T>** Will help us add a `HttpStatus` status code and headers to our response.

**Example:** In the below code snippet, `createCustomer()` method is returning a String value and setting the status code as 200.

```
1. @PostMapping(consumes="application/json")
2. public ResponseEntity<String> createCustomer(@RequestBody CustomerDTO customerDTO)
3. {
```

```

4. //This method will create a customer
5. String response = customerService.createCustomer(customerDTO);
6. return ResponseEntity.ok(response);
7. }

```

As shown in the code snippet, we can use the static functions of `ResponseEntity` class that are available for standard Http codes (`ok()` method is used, here). One more way of setting response entity is as follows.

```

1. ResponseEntity(T body, MultiValueMap<String,String> headers, HttpStatus status)

```

#### Example:

```

1. @PostMapping(consumes="application/json")
2. public ResponseEntity<String> createCustomer(@RequestBody CustomerDTO customerDTO)
3. {
4.     HttpHeaders responseHeaders = new HttpHeaders();
5.     responseHeaders.set("MyResponseHeaders", "Value1");
6.     String response = customerService.createCustomer(customerDTO);
7.     return new ResponseEntity<String>(response, responseHeaders, HttpStatus.CREATED);
8. }

```

### ResponseEntity - Constructors and Methods

Below is the list of constructors available to create `ResponseEntity`

Constructor	Description
<code>ResponseEntity(HttpStatus status)</code>	Creates a <code>ResponseEntity</code> with only status code and no body
<code>ResponseEntity(MultiValueMap&lt;String,String&gt; headers, HttpStatus status)</code>	Creates a <code>ResponseEntity</code> object with headers and statuscode but, no body
<code>ResponseEntity(T body, HttpStatus status)</code>	Creates a <code>ResponseEntity</code> with a body of type T and HTTP status
<code>ResponseEntity(T body, MultiValueMap&lt;String,String&gt; headers, HttpStatus status)</code>	Creates a <code>ResponseEntity</code> with a body of type T, header and HTTP status

**Note:** There are some `ResponseEntity` methods available as well.

ResponseEntity method	Description
<code>ok(T body)</code>	Method that creates a <code>ResponseEntity</code> with status, ok and body, T
<code>ResponseBuilder badRequest()</code>	<p>Returns a <code>ResponseBuilder</code> with the status, <code>BAD_REQUEST</code>. In case, body has to be added, then <code>body()</code> method should be invoked on the <code>ResponseBuilder</code> being received and finally <code>build()</code> should get invoked in order to build <code>ResponseEntity</code>.</p> <p><b>Usage Example:</b></p> <pre>ResponseEntity.badRequest().body(message).build();</pre>
<code>ResponseBuilder</code>	Returns a <code>ResponseBuilder</code> with the status, <code>NOT_FOUND</code> .



ResponseEntity method	Description
notFound()	ResponseEntity.notFound().build();

Let us have a look at the demo that covers these concepts.

## Demo 2 - Consuming and producing Java objects

### Objectives:

To create a Spring REST application using Spring Boot, where the handler methods of a REST controller, consumes and produces Java objects. We will learn,

1. The usage of **produces** and **consumes** attributes of HTTP method handling annotations.
2. The usage of **ResponseEntity**

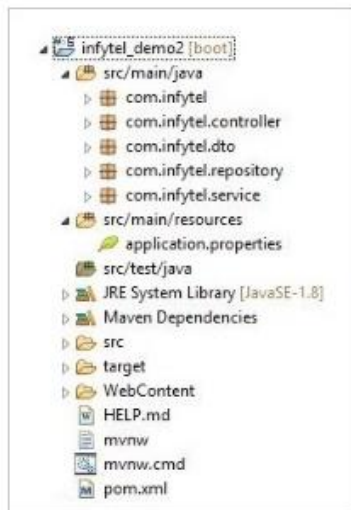
**Scenario:** An online telecom app called **Infytel** is exposing its customer management profile as a RESTful service. The Customer resource titled CustomerController allows us to create, fetch, delete and update customer details. This demo works with the following HTTP operations.

MethodName	URI	HTTP Method	Remarks
createCustomer()	/customers	POST	Uses <b>consumes</b> attribute, <b>@RequestBody</b>
fetchCustomer()	/customers	GET	Uses <b>produces</b> attribute

### Steps:

**Step 1:** Create a Maven project using Spring Initializer with web dependency and import the same in STS.

**Step 2:** Modify the imported project according to the following project structure:



**Step 3:** Look at the class InfytelDemo2Application in com.infytel package that gets created automatically when the project is generated. We will modify this only when we need to go with custom configuration.

1. package com.infytel;
2. import org.springframework.boot.SpringApplication;

```
3. import org.springframework.boot.autoconfigure.SpringBootApplication;
4. @SpringBootApplication
5. public class InfytelDemo2Application {
6.     public static void main(String[] args) {
7.         SpringApplication.run(InfytelDemo2Application.class, args);
8.     }
9. }
```

**Step 4:** Create the class **CustomerController** under **com.infytel.controller** package:

```
1. package com.infytel.controller;
2. import java.util.List;
3. import org.springframework.beans.factory.annotation.Autowired;
4. import org.springframework.http.ResponseEntity;
5. import org.springframework.web.bind.annotation.DeleteMapping;
6. import org.springframework.web.bind.annotation.GetMapping;
7. import org.springframework.web.bind.annotation.PostMapping;
8. import org.springframework.web.bind.annotation.PutMapping;
9. import org.springframework.web.bind.annotation.RequestBody;
10. import org.springframework.web.bind.annotation.RequestMapping;
11. import org.springframework.web.bind.annotation.RestController;
12. import com.infytel.dto.CustomerDTO;
13. import com.infytel.service.CustomerService;
14. @RestController
15. @RequestMapping("/customers")
16. public class CustomerController
17. {
18.     //CustomerController needs to contact CustomerService, hence dependency injecting the
        customerService reference
19.     @Autowired
20.     private CustomerService customerService;
21.     //Fetching customer details
22.     @GetMapping(produces="application/json")
23.     public List<CustomerDTO> fetchCustomer()
24.     {
25.         //This method will return the customers of Infytel
26.         return customerService.fetchCustomer();
27.     }
28.     //Adding a customer
29.     @PostMapping(consumes="application/json")
30.     public ResponseEntity<String> createCustomer(@RequestBody CustomerDTO customerDTO)
31.     {
32.         //This method will create a customer
33.         String response = customerService.createCustomer(customerDTO);
```

```
34. return ResponseEntity.ok(response);
35. }
36. //Updating an existing customer
37. @PutMapping
38. public String updateCustomer()
39. {
40. //This method will update an existing customer
41. return "customer details updated successfully";
42. }
43. //Deleting a customer
44. @DeleteMapping
45. public String deleteCustomer()
46. {
47. //This method will delete a customer
48. return "customer details deleted successfully";
49. }
50. }
```

**Step 5:** Create the DTO classes **CustomerDTO**, **FriendFamilyDTO** and **PlanDTO** under **com.infytel.dto** package:

```
1. package com.infytel.dto;
2. import java.util.List;
3. public class CustomerDTO {
4. long phoneNo;
5. String name;
6. String email;
7. int age;
8. char gender;
9. List<FriendFamilyDTO> friendAndFamily;
10. String password;
11. String address;
12. PlanDTO currentPlan;
13. public String getEmail() {
14. return email;
15. }
16. public void setEmail(String email) {
17. this.email = email;
18. }
19. public PlanDTO getCurrentPlan() {
20. return currentPlan;
21. }
22. public void setCurrentPlan(PlanDTO currentPlan) {
23. this.currentPlan = currentPlan;
```

```
24. }
25. public String getPassword() {
26. return password;
27. }
28. public void setPassword(String password) {
29. this.password = password;
30. }
31. public String getAddress() {
32. return address;
33. }
34. public void setAddress(String address) {
35. this.address = address;
36. }
37. public List<FriendFamilyDTO> getFriendAndFamily() {
38. return friendAndFamily;
39. }
40. public void setFriendAndFamily(List<FriendFamilyDTO> friendAndFamily) {
41. this.friendAndFamily = friendAndFamily;
42. }
43. public long getPhoneNo() {
44. return phoneNo;
45. }
46. public void setPhoneNo(long phoneNo) {
47. this.phoneNo = phoneNo;
48. }
49. public String getName() {
50. return name;
51. }
52. public void setName(String name) {
53. this.name = name;
54. }
55. public int getAge() {
56. return age;
57. }
58. public void setAge(int age) {
59. this.age = age;
60. }
61. public char getGender() {
62. return gender;
63. }
64. public void setGender(char gender) {
65. this.gender = gender;
```

```
66. }
67. @Override
68. public String toString() {
69. return "CustomerDTO [phoneNo=" + phoneNo + ", name=" + name + ", age=" + age + ", gender="
    + gender + ", friendAndFamily=" + friendAndFamily + ", password=" + password + ", address=" +
    address + "]";
70. }
71. }
72.
1. package com.infytel.dto;
2. public class FriendFamilyDTO {
3. long phoneNo;
4. long friendAndFamily;
5. public long getPhoneNo() {
6. return phoneNo;
7. }
8. public void setPhoneNo(long phoneNo) {
9. this.phoneNo = phoneNo;
10. }
11. public long getFriendAndFamily() {
12. return friendAndFamily;
13. }
14. public void setFriendAndFamily(long friendAndFamily) {
15. this.friendAndFamily = friendAndFamily;
16. }
17. public FriendFamilyDTO(long phoneNo, long friendAndFamily) {
18. this();
19. this.phoneNo = phoneNo;
20. this.friendAndFamily = friendAndFamily;
21. }
22. public FriendFamilyDTO() {
23. super();
24. }
25. @Override
26. public String toString() {
27. return "FriendFamilyDTO [phoneNo=" + phoneNo + ", friendAndFamily=" + friendAndFamily +
    "]";
28. }
29. }
30.
1. package com.infytel.dto;
2. import javax.xml.bind.annotation.XmlRootElement;
```

```
3. @XmlElement //Maps a class or an enum type to an XML element.
4. public class PlanDTO {
5.     Integer planId;
6.     String planName;
7.     Integer nationalRate;
8.     Integer localRate;
9.     public Integer getPlanId() {
10.    return planId;
11. }
12. public void setPlanId(Integer planId) {
13.    this.planId = planId;
14. }
15. public String getPlanName() {
16.    return planName;
17. }
18. public void setPlanName(String planName) {
19.    this.planName = planName;
20. }
21. public Integer getNationalRate() {
22.    return nationalRate;
23. }
24. public void setNationalRate(Integer nationalRate) {
25.    this.nationalRate = nationalRate;
26. }
27. public Integer getLocalRate() {
28.    return localRate;
29. }
30. public void setLocalRate(Integer localRate) {
31.    this.localRate = localRate;
32. }
33. public PlanDTO() {
34.    super();
35. }
36. @Override
37. public String toString() {
38.    return "PlanDTO [planId=" + planId + ", planName=" + planName + ", nationalRate=" +
        nationalRate + ", localRate=" + localRate + "];"
39. }
40. }
```

**Step 6:** Create the class **CustomerRepository** under com.infytel.repository package

```
1. package com.infytel.repository;
2. import java.util.ArrayList;
```

```
3. import java.util.List;
4. import javax.annotation.PostConstruct;
5. import org.springframework.stereotype.Repository;
6. import com.infytel.dto.CustomerDTO;
7. import com.infytel.dto.FriendFamilyDTO;
8. import com.infytel.dto.PlanDTO;
9. @Repository
10. public class CustomerRepository {
11. List<CustomerDTO> customers = null;
12. //Equivalent/similar to constructor. Here, populates the DTOs in a hard-coded way
13. @PostConstruct
14. public void initializer()
15. {
16. CustomerDTO customerDTO = new CustomerDTO();
17. PlanDTO planDTO = new PlanDTO();
18. planDTO.setPlanId(1);
19. planDTO.setPlanName("Simple");
20. planDTO.setLocalRate(3);
21. planDTO.setNationalRate(5);
22. customerDTO.setAddress("Chennai");
23. customerDTO.setAge(18);
24. customerDTO.setCurrentPlan(planDTO);
25. customerDTO.setGender('m');
26. customerDTO.setName("Jack");
27. customerDTO.setEmail("Jack@infy.com");
28. customerDTO.setPassword("ABC@123");
29. customerDTO.setPhoneNo(99512122221);
30. List<FriendFamilyDTO> friendAndFamily = new ArrayList<>();
31. friendAndFamily.add(new FriendFamilyDTO(customerDTO.getPhoneNo(),800000145));
32. friendAndFamily.add(new FriendFamilyDTO(customerDTO.getPhoneNo(),700000145));
33. customerDTO.setFriendAndFamily(friendAndFamily);
34. customers = new ArrayList<>();
35. customers.add(customerDTO);
36. }
37. //adds the received customer object to customers list
38. public void createCustomer(CustomerDTO customerDTO)
39. {
40. customers.add(customerDTO);
41. }
42. //returns a list of customers
43. public List<CustomerDTO> fetchCustomer()
44. {
```

```
45. return customers;
46. }
47. }
```

**Step 7:** Create the class **CustomerService** under **com.infytel.service** package:

```
1. package com.infytel.service;
2. import java.util.List;
3. import org.springframework.beans.factory.annotation.Autowired;
4. import org.springframework.stereotype.Service;
5. import com.infytel.dto.CustomerDTO;
6. import com.infytel.repository.CustomerRepository;
7. @Service
8. public class CustomerService
9. {
10. /* CustomerService needs to contact CustomerRepository,
11. hence injecting the customerRepository dependency
12. */
13. @Autowired
14. private CustomerRepository customerRepository;
15. //makes a call to repository method for adding the customer
16. public String createCustomer(CustomerDTO customerDTO)
17. {
18. customerRepository.createCustomer(customerDTO);
19. return "Customer with "+customerDTO.getPhoneNo()+" added successfully";
20. }
21. //makes a call to repository method for returning a list of customers
22. public List<CustomerDTO> fetchCustomer()
23. {
24. return customerRepository.fetchCustomer();
25. }
26. }
```

**Step 8:** Add the following content to **application.properties** file:

```
1. server.port = 8080
2. server.servlet.context-path=/infytel-2
```

**Step 9:** Make sure that the project's **pom.xml** looks similar to the one that is shown below.

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <project xmlns="http://maven.apache.org/POM/4.0.0"
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
5. <modelVersion>4.0.0</modelVersion>
```



```
6. <parent>
7. <groupId>org.springframework.boot</groupId>
8. <artifactId>spring-boot-starter-parent</artifactId>
9. <version>2.1.4.RELEASE</version>
10. <relativePath /> <!-- lookup parent from repository -->
11. </parent>
12. <groupId>com.infosys</groupId>
13. <artifactId>infytel_demo2</artifactId>
14. <version>0.0.1-SNAPSHOT</version>
15. <name>infytel_demo2</name>
16. <description>Understanding the differences between Spring MVC and Spring
    REST</description>
17. <properties>
18. <java.version>1.8</java.version>
19. </properties>
20. <dependencies>
21. <dependency>
22. <groupId>org.springframework.boot</groupId>
23. <artifactId>spring-boot-starter-web</artifactId>
24. </dependency>
25. <dependency>
26. <groupId>org.springframework.boot</groupId>
27. <artifactId>spring-boot-starter-test</artifactId>
28. <scope>test</scope>
29. </dependency>
30. </dependencies>
31. <build>
32. <plugins>
33. <plugin>
34. <groupId>org.springframework.boot</groupId>
35. <artifactId>spring-boot-maven-plugin</artifactId>
36. </plugin>
37. </plugins>
38. </build>
39. </project>
```

**Step 10:** Deploy the application by executing the class that contains the main method.

So, we have successfully created and deployed the REST endpoints. Now, let us see how to test the same using Postman client.

### Output Screenshots

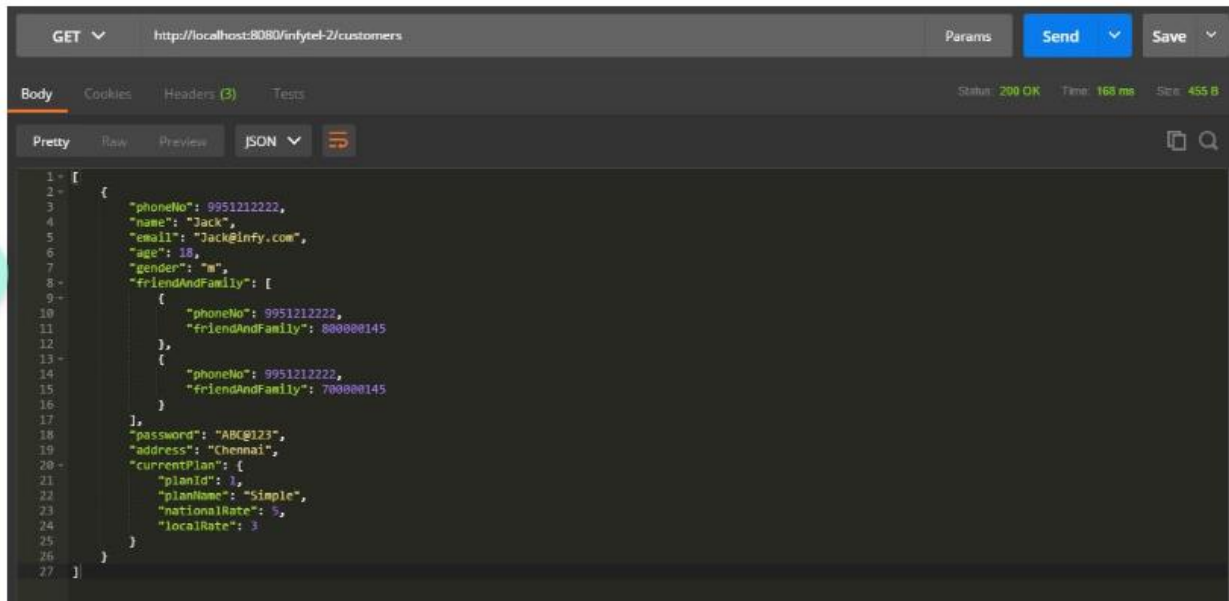
#### Testing Web Service using Postman

**Step 1:** Launch Postman.

**Step 2:** Test GET request

From the drop-down select **GET** and enter **http://localhost:8080/infytel-2/customers** into the URL field to test the service. Click on Send. You will get the following response:

This is the pre-populated data that is appearing.



**Step 3:** Test POST request

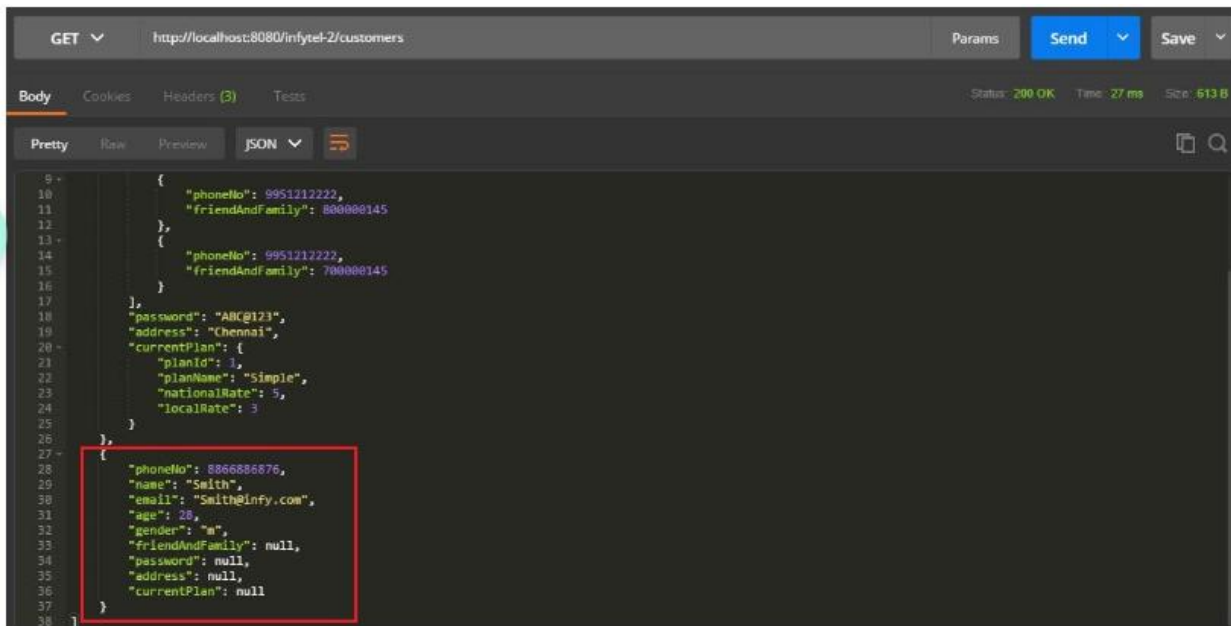
From the drop-down select POST and enter **http://localhost:8080/infytel-2/customers** into the URL field to test the service. And click on body to enter following JSON data. The below data is used to create a new customer in infytel.

The JSON data has to match the DTO object to which it is converted.

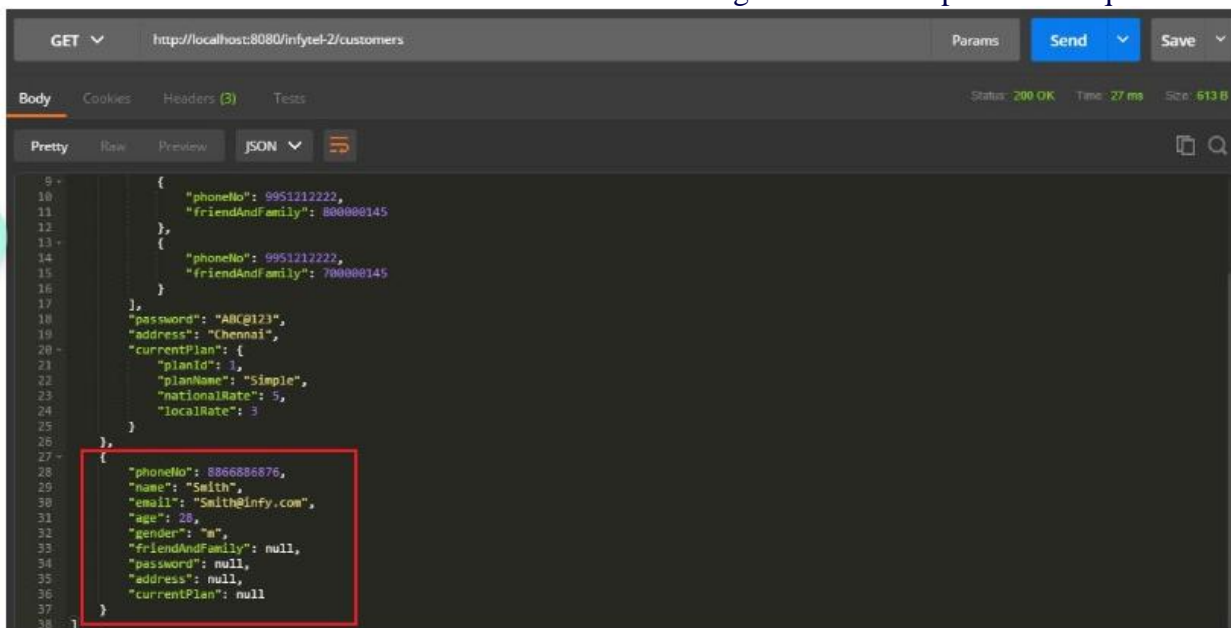
1. {"phoneNo":8866886876,"name":"Smith","email":"Smith@infy.com","age":28,
2. "gender":"m"}

**Note:** Observe the CustomerDTO's variable names. It is important that the variable name in CustomerDTO and JSON key has to match exactly, for the correct mapping to happen.

Click on Send. The following response will be generated.



Now, from the drop-down, select GET and enter the URL **http://localhost:8080/infytel-2/customers** into the URL field to test the service. Click on Send. Following will be the output of the request.



**Note:** For GET operation, we can test on a browser as well.

Enter the URL: **http://localhost:8080/infytel-2/customers** into the address bar of the browser to get the result of GET request.



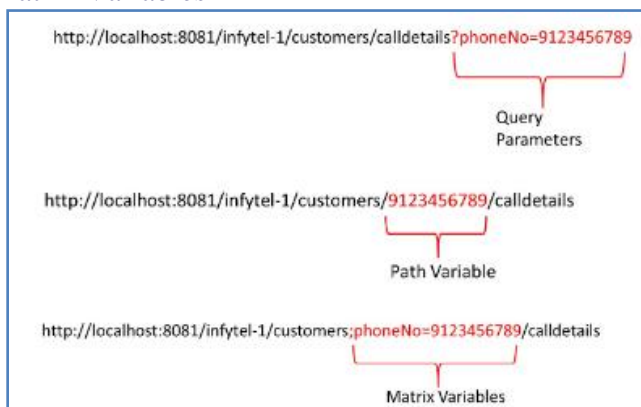
### URI Parameter Injection Annotations

It is not always that the client prefers sending the data as request body. The client can even choose to send the data as part of the request URI. For example, if the client feels that the data is not sensitive and

doesn't need a separate channel to get transferred. So, how to receive this kind of data that appears in the URI.

Before we look into the ways to extract the URI data, we will have a look at the formal categorization of the data that appear in the request URI.

1. Query Parameter
2. Path Variables
3. Matrix Variables



Let us understand these URI parameters in detail

### Query Parameter:

- Query parameters or request parameters usually travel with the URI and are delimited by question mark.
- The query parameters in turn are delimited by ampersand from one another.
- The annotation `@RequestParam` helps map query/request parameters to the method arguments.
- **@RequestParam** annotation expects the name that it holds to be similar to the one that is present in the request URI. This makes the code tightly coupled with the request URI.

**Example:** Assume, there is a RestController called `CallDetailsController` that has a service method to return the call details of a specific phone number on a particular date. The service method here, takes the `phoneNo` of the customer and the date as query /request parameters.

### Example URI:

1. URI: `http://<<hostname>>:<<port>>/<<contextpath>>/calldetails?calledBy=9123456789&calledOn=12/5/2019`
1. `@RestController`
2. `@RequestMapping("/calldetails")`
3. `public class CallDetailsController`
4. `{`
5. `//Fetching call details based on the request parameters being passed along with the URI`
6. `@GetMapping(produces = "application/json")`
7. `public List<CallDetailsDTO> callDetails(`
8. `@RequestParam("calledBy") long calledBy, @RequestParam("calledOn") String calledOn)`
9. `{`
10. `//code goes here`

```
11. }  
12. }
```

### Path Variables

- Path variables are usually available at the end of the request URIs delimited by slash (/).
- @PathVariable annotation is applied on the argument of the controller method whose value needs to be extracted out of the request URI.
- A request URI can have any number of path variables.
- Multiple path variables require the usage of multiple @PathVariable annotations.
- @PathVariable can be used with any type of request method. For example, GET, POST, DELETE, etc.,

We have to make sure that the name of the local parameter (int id) and the placeholder ({id}) are same. Name of the PathVariable annotation's argument (@PathVariable("id")) and the placeholder ({id}) should be equal, otherwise.

```
1. @GetMapping("/{id}")  
2. public String controllerMethod(@PathVariable int id){ }
```

OR

```
1. @GetMapping("/{id}")  
2. public String controllerMethod(@PathVariable("id") int empId){ }
```

**Example:** Assume, there is a REST controller called CustomerController that has service methods to update and delete the customers. These operations are simply based out of the phone number of the customer that is passed as part of the request URI.

Below are the sample URIs that contain data as part of them.

```
1. URI: PUT:http://<<hostname>>:<<port>>/<<contextpath>>/customers/9123456789  
2. DELETE:http://<<hostname>>:<<port>>/<<contextpath>>/customers/9123456789
```

The handler methods with the provision to extract the URI parameters are presented below.

```
1. @RestController  
2. @RequestMapping("/customers")  
3. public class CustomerController  
4. {  
5. //Updating an existing customer  
6. @PutMapping(value =("/{phoneNumber}", consumes = "application/json")  
7. public String updateCustomer(  
8. @PathVariable("phoneNumber") long phoneNumber,  
9. @RequestBody CustomerDTO customerDTO) {  
10. //code goes here  
11. }  
12. // Deleting a customer  
13. @DeleteMapping(value="/{phoneNumber}",produces="text/html")  
14. public String deleteCustomer(
```

```

15. @PathVariable("phoneNumber") long phoneNumber)
16. throws NoSuchCustomerException {
17. //code goes here
18. }
19. }

```

### Matrix variables:

- Matrix variables are a block/segment of values that travel along with the URI. For example, **/localRate=1,2,3/**
- These variables may appear in the middle of the path unlike query parameters which appear only towards the end of the URI.
- Matrix variables follow **name=value** format and use semicolon to get delimited from one other matrix variable.
- A matrix variable can carry any number of values, delimited by commas.
- @MatrixVariable is used to extract the matrix variables.

**Example:** Assume, there is a REST controller called PlanController that has a service to return the details of Plans based on the search criteria, localRates.

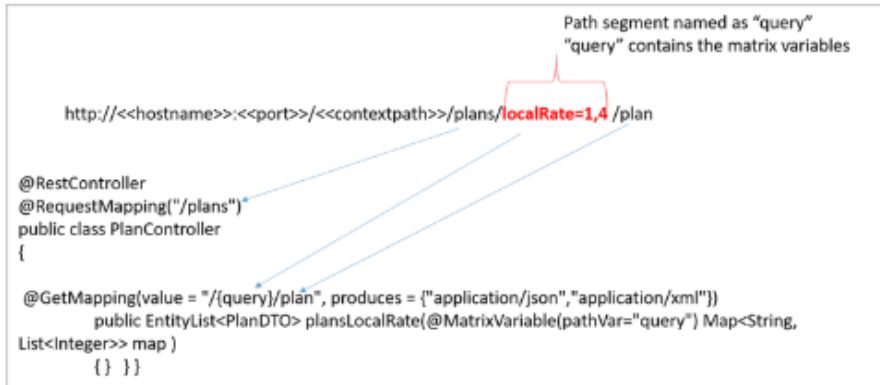
Below is the URI path. Observe that a variable localRate with two values separated by "," appear in the middle of the path.

```

1. URI:http://<<hostname>>:<<port>>/<<contextpath>>/plans/localRate=1,4 /plan

```

See how the matrix variable, **localRate** gets extracted in the controller method.



### Code:

```

1. @RestController
2. @RequestMapping("/plans")
3. public class PlanController
4. {
5. // {query} here is a place holder for the matrix variables that travel in the URI,
6. // it is not mandatory that the client URI should hold a string literal called query
7. @GetMapping(value =("/{query})/plan", produces = {"application/json","application/xml"})
8. public EntityList<PlanDTO> plansLocalRate(
9. @MatrixVariable(pathVar="query") Map<String, List<Integer>> map ) {
10. //code goes here
11. }

```

```
12. }
```

**@MatrixVariable(pathVar="query") Map<String, List<Integer>> map**: The code snippet mentions that all the matrix variables that appear in the path segment of name **query** should be stored in a Map instance called **map**. Here, the map's key is nothing but the name of the matrix variable and that is nothing but **localRate**. And, the **value** of the map is a collection of **localRates (1,4)** of type Integer.

**Note:** If the matrix variable appears towards the end of the URI as in the below example,

```
1. URI:http://localhost:8081/infytel-1/customers/calldetails/phoneNo=9123456789
```

we can use the following approach to read the matrix variable.

```
1. @GetMapping("/customers/{query}")
2. public ResponseEntity<CallDetails> getCallDetails(@MatrixVariable String phoneNo)
3. {
4. //code goes here
5. }
```

Let us put all these together and look at an example.

### Demo 3- Reading path variables using @PathVariable

#### Objectives:

To create a Spring REST application using Spring Boot where some of the REST methods take path variables as arguments. In this demo, we will learn the usage of @PathVariable.

**Scenario:** An online telecom app called Infytel is exposing its “customer management profile” as RESTful service. The Customer resource allows us to create, fetch, delete and update customer details. This demo concentrates on the following HTTP operations.

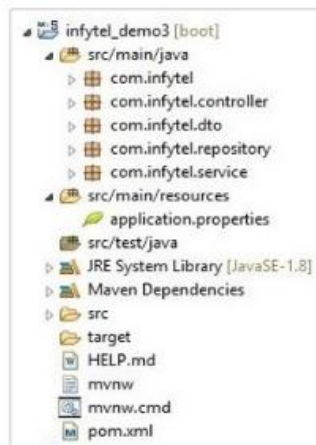
MethodName	URI	HTTP Method	Remarks
updateCustomer	/customers/{phoneNumber}	PUT	To update Customer details of a customer identified via his phoneNumber passed as path variable
deleteCustomer	/customers/{phoneNumber}	DELETE	To delete an existing Customer of Infytel identified via his phoneNumber passed as path variable

#### Steps:

**Step 1:** Create a Maven project using Spring Initializer with web dependency and import the same in STS.

**Step 2:** Modify the imported project according to the following project structure:





**Step 3:** Look at the class **InfytelDemo3Application** in **com.infytel** package that gets created automatically when the project is generated. We will modify this only when we need to go with custom configuration.

```
1. package com.infytel;
2. import org.springframework.boot.SpringApplication;
3. import org.springframework.boot.autoconfigure.SpringBootApplication;
4. @SpringBootApplication
5. public class InfytelDemo3Application {
6.     public static void main(String[] args) {
7.         SpringApplication.run(InfytelDemo3Application.class, args);
8.     }
9. }
```

**Step 4:** Create the class **CustomerController** under **com.infytel.controller** package.

Notice the usage of **@PathVariable** and **/ {phoneNumber}** in **updateCustomer()** and **deleteCustomer()** methods.

```
1. package com.infytel.controller;
2. import java.util.List;
3. import org.springframework.beans.factory.annotation.Autowired;
4. import org.springframework.http.ResponseEntity;
5. import org.springframework.web.bind.annotation.DeleteMapping;
6. import org.springframework.web.bind.annotation.GetMapping;
7. import org.springframework.web.bind.annotation.PathVariable;
8. import org.springframework.web.bind.annotation.PostMapping;
9. import org.springframework.web.bind.annotation.PutMapping;
10. import org.springframework.web.bind.annotation.RequestBody;
11. import org.springframework.web.bind.annotation.RequestMapping;
12. import org.springframework.web.bind.annotation.RestController;
13. import com.infytel.dto.CustomerDTO;
14. import com.infytel.service.CustomerService;
15. @RestController
```



```
16. @RequestMapping("/customers")
17. public class CustomerController
18. {
19. //CustomerController needs to contact CustomerService, hence dependency injecting the
    customerService reference
20. @Autowired
21. private CustomerService customerService;
22. //Fetching customer details
23. @GetMapping(produces="application/json")
24. public List<CustomerDTO> fetchCustomer()
25. {
26. return customerService.fetchCustomer();
27. }
28. //Adding a customer
29. @PostMapping(consumes="application/json")
30. public ResponseEntity<String> createCustomer( @RequestBody CustomerDTO customerDTO)
31. {
32. String response = customerService.createCustomer(customerDTO);
33. return ResponseEntity.ok(response);
34. }
35. //Updating an existing customer
36. @PutMapping(value =("/{phoneNumber}", consumes = "application/json")
37. public String updateCustomer(@PathVariable("phoneNumber") long phoneNumber,
    @RequestBody CustomerDTO customerDTO)
38. {
39. return customerService.updateCustomer(phoneNumber, customerDTO);
40. }
41. //Deleting a customer
42. @DeleteMapping("/{phoneNumber}")
43. public String deleteCustomer(@PathVariable("phoneNumber") long phoneNumber)
44. {
45. return customerService.deleteCustomer(phoneNumber);
46. }
47. }
```

**Step 5:** Create the classes **CustomerDTO**, **FriendFamilyDTO** and **PlanDTO** under **com.infytel.dto** package:

```
1. package com.infytel.dto;
2. import java.util.List;
3. public class CustomerDTO
4. {
5. long phoneNo;
```

```
6. String name;
7. String email;
8. int age;
9. char gender;
10. List<FriendFamilyDTO> friendAndFamily;
11. String password;
12. String address;
13. PlanDTO currentPlan;
14. public String getEmail() {
15. return email;
16. }
17. public void setEmail(String email) {
18. this.email = email;
19. }
20. public PlanDTO getCurrentPlan() {
21. return currentPlan;
22. }
23. public void setCurrentPlan(PlanDTO currentPlan) {
24. this.currentPlan = currentPlan;
25. }
26. public String getPassword() {
27. return password;
28. }
29. public void setPassword(String password) {
30. this.password = password;
31. }
32. public String getAddress() {
33. return address;
34. }
35. public void setAddress(String address) {
36. this.address = address;
37. }
38. public List<FriendFamilyDTO> getFriendAndFamily() {
39. return friendAndFamily;
40. }
41. public void setFriendAndFamily(List<FriendFamilyDTO> friendAndFamily) {
42. this.friendAndFamily = friendAndFamily;
43. }
44. public long getPhoneNo() {
45. return phoneNo;
46. }
47. public void setPhoneNo(long phoneNo) {
```

```
48. this.phoneNo = phoneNo;
49. }
50. public String getName() {
51. return name;
52. }
53. public void setName(String name) {
54. this.name = name;
55. }
56. public int getAge() {
57. return age;
58. }
59. public void setAge(int age) {
60. this.age = age;
61. }
62. public char getGender() {
63. return gender;
64. }
65. public void setGender(char gender) {
66. this.gender = gender;
67. }
68. @Override
69. public String toString() {
70. return "CustomerDTO [phoneNo=" + phoneNo + ", name=" + name + ", age=" + age + ", gender="
    + gender + ", friendAndFamily=" + friendAndFamily + ", password=" + password + ", address=" +
    address + "]";
71. }
72. }
73.
1. package com.infytel.dto;
2. public class FriendFamilyDTO {
3. long phoneNo;
4. long friendAndFamily;
5. public long getPhoneNo() {
6. return phoneNo;
7. }
8. public void setPhoneNo(long phoneNo) {
9. this.phoneNo = phoneNo;
10. }
11. public long getFriendAndFamily() {
12. return friendAndFamily;
13. }
14. public void setFriendAndFamily(long friendAndFamily) {
```

```
15. this.friendAndFamily = friendAndFamily;
16. }
17. public FriendFamilyDTO(long phoneNo, long friendAndFamily) {
18. this();
19. this.phoneNo = phoneNo;
20. this.friendAndFamily = friendAndFamily;
21. }
22. public FriendFamilyDTO() {
23. super();
24. }
25. @Override
26. public String toString() {
27. return "FriendFamilyDTO [phoneNo=" + phoneNo + ", friendAndFamily=" + friendAndFamily +
    "]" ;
28. }
29. }
30.
```

```
1. package com.infytel.dto;
2. import javax.xml.bind.annotation.XmlRootElement;
3. @XmlRootElement
4. public class PlanDTO {
5. Integer planId;
6. String planName;
7. Integer nationalRate;
8. Integer localRate;
9. public Integer getPlanId() {
10. return planId;
11. }
12. public void setPlanId(Integer planId) {
13. this.planId = planId;
14. }
15. public String getPlanName() {
16. return planName;
17. }
18. public void setPlanName(String planName) {
19. this.planName = planName;
20. }
21. public Integer getNationalRate() {
22. return nationalRate;
23. }
24. public void setNationalRate(Integer nationalRate) {
25. this.nationalRate = nationalRate;
```

```
26. }
27. public Integer getLocalRate() {
28. return localRate;
29. }
30. public void setLocalRate(Integer localRate) {
31. this.localRate = localRate;
32. }
33. public PlanDTO() {
34. super();
35. }
36. @Override
37. public String toString() {
38. return "PlanDTO [planId=" + planId + ", planName=" + planName + ", nationalRate=" +
    nationalRate + ", localRate=" + localRate + "]";
39. }
40. }
```

**Step 6:** Create the class called **CustomerRepository** under com.infytel.repository package:

```
1. package com.infytel.repository;
2. import java.util.ArrayList;
3. import java.util.List;
4. import javax.annotation.PostConstruct;
5. import org.springframework.stereotype.Repository;
6. import com.infytel.dto.CustomerDTO;
7. import com.infytel.dto.FriendFamilyDTO;
8. import com.infytel.dto.PlanDTO;
9. @Repository
10. public class CustomerRepository {
11. List<CustomerDTO> customers = null;
12. //Similar to the constructor. Populates the DTOs in a hard-coded way
13. @PostConstruct
14. public void initializer()
15. {
16. CustomerDTO customerDTO = new CustomerDTO();
17. PlanDTO planDTO = new PlanDTO();
18. planDTO.setPlanId(1);
19. planDTO.setPlanName("Simple");
20. planDTO.setLocalRate(3);
21. planDTO.setNationalRate(5);
22. customerDTO.setAddress("Chennai");
23. customerDTO.setAge(18);
24. customerDTO.setCurrentPlan(planDTO);
}
```

```
25. customerDTO.setGender('m');
26. customerDTO.setName("Jack");
27. customerDTO.setEmail("Jack@infy.com");
28. customerDTO.setPassword("ABC@123");
29. customerDTO.setPhoneNo(99512122221);
30. List<FriendFamilyDTO> friendAndFamily = new ArrayList<>();
31. friendAndFamily.add(new FriendFamilyDTO(customerDTO.getPhoneNo(),800000145));
32. friendAndFamily.add(new FriendFamilyDTO(customerDTO.getPhoneNo(),700000145));
33. customerDTO.setFriendAndFamily(friendAndFamily);
34. customers = new ArrayList<>();
35. customers.add(customerDTO);
36. }
37. //adds the received customer object to customers list
38. public void createCustomer(CustomerDTO customerDTO)
39. {
40. customers.add(customerDTO);
41. }
42. //returns a list of customers
43. public List<CustomerDTO> fetchCustomer()
44. {
45. return customers;
46. }
47. //deletes the passed customer from the list
48. public void deleteCustomer(CustomerDTO customer)
49. {
50. customers.remove(customer);
51. }
52. }
```

**Step 7:** Create the class **CustomerService** under com.infytel.service package:

```
1. package com.infytel.service;
2. import java.util.List;
3. import org.springframework.beans.factory.annotation.Autowired;
4. import org.springframework.stereotype.Service;
5. import com.infytel.dto.CustomerDTO;
6. import com.infytel.repository.CustomerRepository;
7. @Service
8. public class CustomerService
9. {
10. /* CustomerService needs to contact CustomerRepository,
11. hence injecting the customerRepository dependency
12. */
```

```
13. @Autowired
14. private CustomerRepository customerRepository;
15. //makes a call to repository method for adding the customer
16. public String createCustomer(CustomerDTO customerDTO)
17. {
18. customerRepository.createCustomer(customerDTO);
19. return "Customer with "+customerDTO.getPhoneNo()+" added successfully";
20. }
21. //makes a call to repository method for returning a list of customers
22. public List<CustomerDTO> fetchCustomer()
23. {
24. return customerRepository.fetchCustomer();
25. }
26. /* makes a call to repository method for fetching the customers list and
27. updates the customer's details
28. */
29. public String updateCustomer(long phoneNumber, CustomerDTO customerDTO)
30. {
31. String response = "Customer of: "+phoneNumber+" does not exist";
32. for(CustomerDTO customer : customerRepository.fetchCustomer())
33. {
34. if(customer.getPhoneNo() == phoneNumber)
35. {
36. if(customerDTO.getName()!=null)
37. customer.setName(customerDTO.getName());
38. if(customerDTO.getAddress()!=null)
39. customer.setAddress(customerDTO.getAddress());
40. if(customerDTO.getPassword()!=null)
41. customer.setPassword(customerDTO.getPassword());
42. response = "Customer of phoneNumber "+customer.getPhoneNo()+" got updated successfully";
43. break;
44. }
45. }
46. return response;
47. }
48. /* makes a call to repository method for fetching the customers list and
49. then calls the repository's deleteCustomer() method with the customer to be deleted
50. */
51. public String deleteCustomer(long phoneNumber)
52. {
53. String response = "Customer of: "+phoneNumber+" does not exist";
54. for (CustomerDTO customer : customerRepository.fetchCustomer()) {
```

```
55. if (customer.getPhoneNo() == phoneNumber) {  
56. customerRepository.deleteCustomer(customer);  
57. response = customer.getName() + " of phoneNo " + customer.getPhoneNo()  
58. + " got deleted successfully";  
59. break;  
60. }  
61. }  
62. return response;  
63. }  
64. }
```

**Step 8:** Add the following content to **application.properties** file:

1. server.port = 8080
2. server.servlet.context-path=/infytel-3

**Step 9:** Make sure that the project's pom.xml looks similar to the one shown below.

```
1. <?xml version="1.0" encoding="UTF-8"?>  
2. <project xmlns="http://maven.apache.org/POM/4.0.0"  
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
4. xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-  
4.0.0.xsd">  
5. <modelVersion>4.0.0</modelVersion>  
6. <parent>  
7. <groupId>org.springframework.boot</groupId>  
8. <artifactId>spring-boot-starter-parent</artifactId>  
9. <version>2.1.4.RELEASE</version>  
10. <relativePath /> <!-- lookup parent from repository -->  
11. </parent>  
12. <groupId>infytel</groupId>  
13. <artifactId>infytel_demo3</artifactId>  
14. <version>0.0.1-SNAPSHOT</version>  
15. <name>infytel_demo3</name>  
16. <description>Demo project for Spring Boot</description>  
17. <properties>  
18. <java.version>1.8</java.version>  
19. </properties>  
20. <dependencies>  
21. <dependency>  
22. <groupId>org.springframework.boot</groupId>  
23. <artifactId>spring-boot-starter-web</artifactId>  
24. </dependency>  
25. <dependency>
```



```
26. <groupId>org.springframework.boot</groupId>
27. <artifactId>spring-boot-starter-test</artifactId>
28. <scope>test</scope>
29. </dependency>
30. </dependencies>
31. <build>
32. <plugins>
33. <plugin>
34. <groupId>org.springframework.boot</groupId>
35. <artifactId>spring-boot-maven-plugin</artifactId>
36. </plugin>
37. </plugins>
38. </build>
39. </project>
```

**Step 10:** Deploy the application on the server by executing the class containing the main method.

So, we have successfully created and deployed REST endpoints. Now, let us see how to test the same using Postman client.

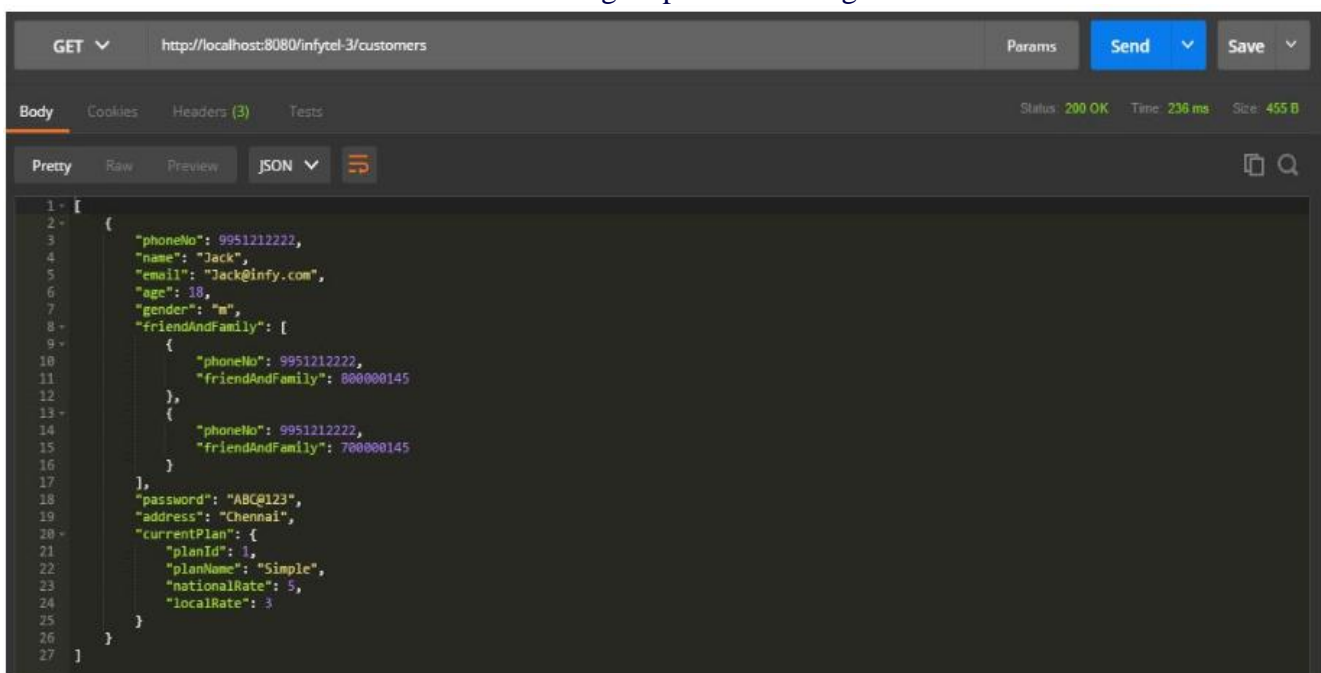
## Output

### Testing the REST endpoints using Postman

**Step 1:** Launch Postman.

**Step 2:** Test GET request

From the dropdown, select GET and enter **http://localhost:8080/infytel-3/customers** into the URL field to test the service. Click on Send. The following response will be generated



**Step 3:** Test PUT request

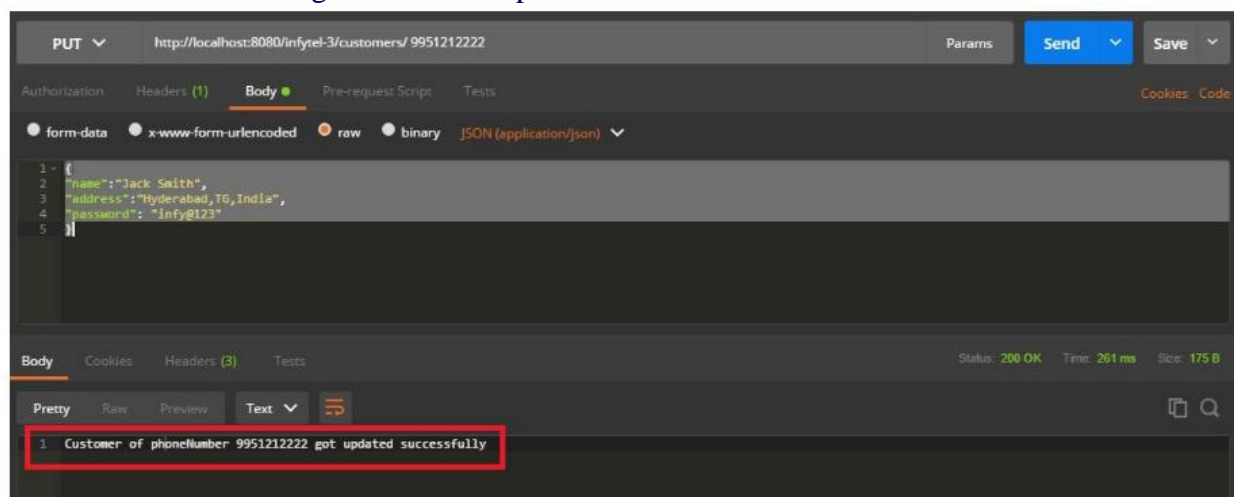
From the drop-down, select PUT and enter **http://localhost:8080/infytel-3/customers/ 9951212222** into the URL field to test the service. And, click on the body to enter the following JSON data. The below data is used to update the details of an existing customer (customer of phone number, 9951212222)

The JSON data has to match the DTO object to which it is converted.

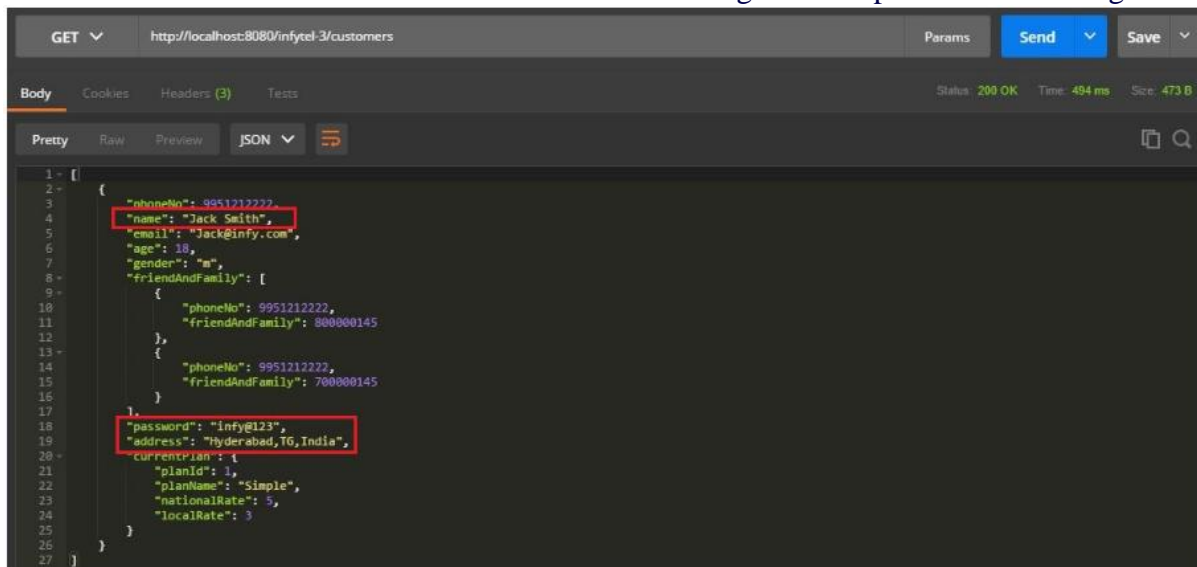
1. {
2. "name": "Jack Smith",
3. "address": "Hyderabad,TG,India",
4. "password": "infy@123"
5. }

**Note:** Observe the CustomerDTO's variable names. It is important that the variable name in CustomerDTO and JSON key has to match exactly, for the correct mapping to happen.

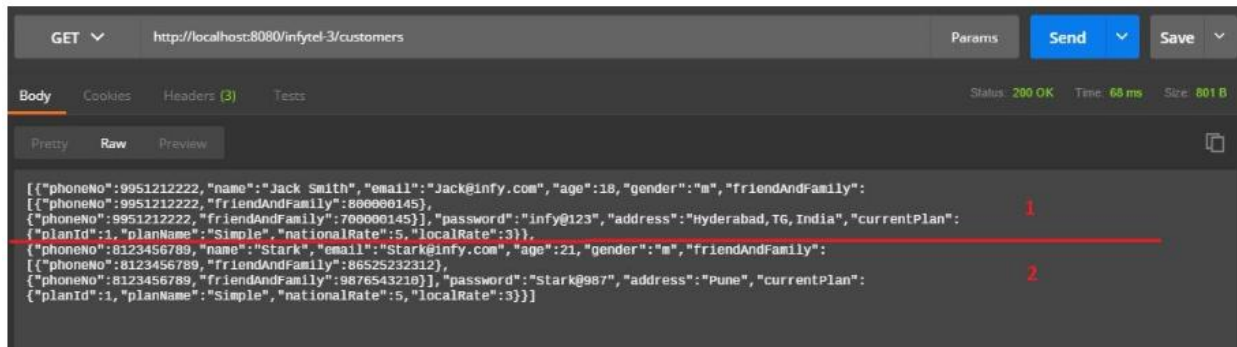
Click on Send. Following will be the response.



Now, from the drop-down, select GET and enter the URL `http://localhost:8080/infytel-3/customers` into the URL field to test the service. Click on Send. Following is the response that will be generated

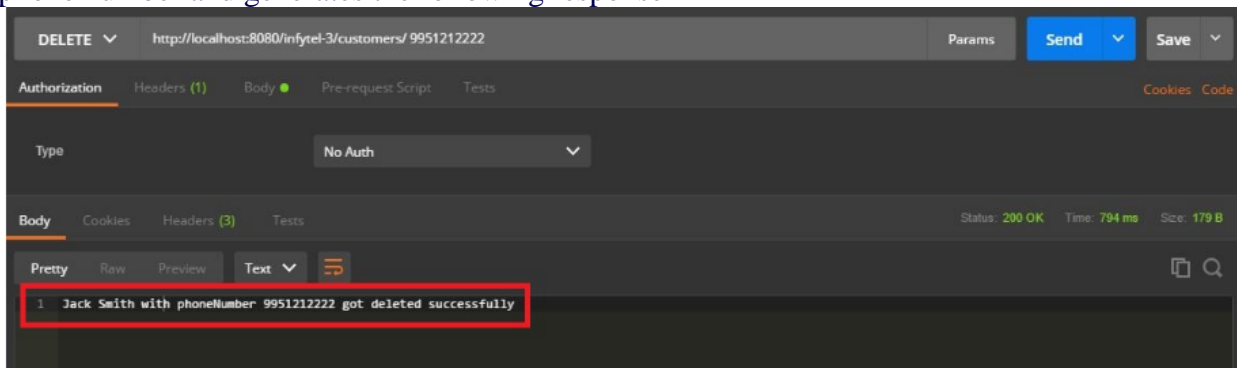


Now, let us assume, we have two Customer Objects in Infytel repository.



#### Step 4: Test DELETE request

From the drop down, select DELETE and enter `http://localhost:8080/infytel-3/customers/9951212222` into the URL field to test the service. Click on Send. It deletes the customer with the given phone number and generates the following response



Now, from the drop down, select GET and enter the URL `http://localhost:8080/infytel-3/customers` into the URL field to test the service. Click on Send. Following will be the response



Notice that the details of the customer of phone number, 9951212222 is not appearing while fetching the data after the delete operation.

#### Exercise - Usage of @PathVariable

CookPick online grocery application has a requirement where the users should be able to search for a product based on its name. As a result of the search, a list of product details from different vendors should be rendered.

Write a RESTful endpoint that could serve this purpose.

RESTful URL	HTTP Action	Business Operation
/product/{productName}	GET	getProducts()

**Method description as follows:**

```
public List<ProductDTO> getProducts(String productName)
```

This method,

- Takes product name
- Renders the list of products with attributes as follows,
  - productCode : long
  - productName : String
  - productVendor : String
  - productPrice : double
  - productInStock: int

**Note:**

1. You can implement this requirement in the project that you used for previous exercise.
2. List of products needs to be rendered in JSON format.
3. You can simply hardcode the product details. It is optional to use a database to retrieve the product details.

\*Time given for this exercise indicates the time required to create the REST endpoint and is excluding the time required to write the logic for retrieving the product details from database.

**@RequestParam Video****Demo 4 - Reading Request Parameters using @RequestParam**

**Objectives:** To learn how to read query strings from the URI.

To create a Spring REST application using Spring Boot where the REST endpoint works with request parameters. Here, we will learn the

- Usage of @RequestParam

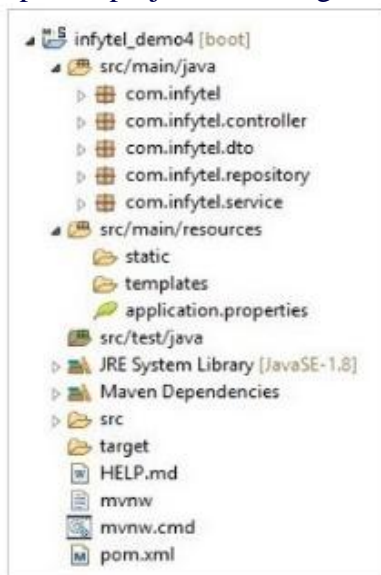
**Scenario:** An online telecom app called Infytel is exposing its customer management profile as a RESTful service. The application has a customer resource titled CustomerController allows us to create, fetch, delete and update customer details. This application has two more controllers, CallDetailsController and PlanController to deal with call and plan details respectively. This demo has the following HTTP operations.

Controller Class	Method Name	URI	HTTP Method	Remarks
CallDetailsController	fetchCallDetails()	/calldetails?calledBy=&calledOn=	GET	This fetches the calls made by a customer on a certain date. Uses @RequestParam to read query string values passed
CustomerController	createCustomer()	/customers	POST	To create a new customer
CustomerController	fetchCustomer()	/customers	GET	To fetch all the

Controller Class	Method Name	URI	HTTP Method	Remarks
				existing customers
CustomerController	updateCustomer()	/customers/{phoneNumber}	PUT	To update a customer details
CustomerController	deleteCustomer	/customers/{phoneNumber}	DELETE	To delete an existing customer

**Step 1:** Create a Maven project using Spring Initializer with web dependency and import the same in STS.

**Step 2:** Modify the imported project according to the following project structure:



**Step 3:** Look at the class InfytelDemo4Application in com.infytel package that gets created automatically when the project is generated. We will modify this only when we need to go with custom configuration

```

1. package com.infytel;
2. import org.springframework.boot.SpringApplication;
3. import org.springframework.boot.autoconfigure.SpringBootApplication;
4. @SpringBootApplication
5. public class InfytelDemo4Application {
6.     public static void main(String[] args) {
7.         SpringApplication.run(InfytelDemo4Application.class, args);
8.     }
9. }

```

**Step 4:** Create the controllers, **CallDetailsController**, **CustomerController** under com.infytel.controller package:

```

1. package com.infytel.controller;
2. import java.time.LocalDate;
3. import java.time.format.DateTimeFormatter;
4. import java.util.List;

```

```
5. import org.springframework.beans.factory.annotation.Autowired;
6. import org.springframework.web.bind.annotation.GetMapping;
7. import org.springframework.web.bind.annotation.RequestMapping;
8. import org.springframework.web.bind.annotation.RequestParam;
9. import org.springframework.web.bind.annotation.RestController;
10. import com.infytel.dto.CallDetailsDTO;
11. import com.infytel.service.CallDetailsService;
12. @RestController
13. @RequestMapping("/calldetails")
14. public class CallDetailsController
15. {
16. @Autowired
17. private CallDetailsService callDetailsService;
18. //Fetching call details based on the request parameters being passed along with the URI
19. //Make sure giving the current date (calledOn) on which the demo gets executed
20. //CallDetailsRepository has code to populate calledOn with the current date
21. @GetMapping(produces = "application/json")
22. public List<CallDetailsDTO> fetchCallDetails(@RequestParam("calledBy") long calledBy,
    @RequestParam("calledOn") String calledOn)
23. {
24. return callDetailsService.fetchCallDetails(calledBy, LocalDate.parse(calledOn,
    DateTimeFormatter.ofPattern("MM-dd-yyyy")));
25. }
26. }
27.
```

```
1. package com.infytel.controller;
2. import java.util.List;
3. import org.springframework.beans.factory.annotation.Autowired;
4. import org.springframework.http.ResponseEntity;
5. import org.springframework.web.bind.annotation.DeleteMapping;
6. import org.springframework.web.bind.annotation.GetMapping;
7. import org.springframework.web.bind.annotation.PathVariable;
8. import org.springframework.web.bind.annotation.PostMapping;
9. import org.springframework.web.bind.annotation.PutMapping;
10. import org.springframework.web.bind.annotation.RequestBody;
11. import org.springframework.web.bind.annotation.RequestMapping;
12. import org.springframework.web.bind.annotation.RestController;
13. import com.infytel.dto.CustomerDTO;
14. import com.infytel.dto.PlanDTO;
15. import com.infytel.service.CustomerService;
16. @RestController
17. @RequestMapping("/customers")
```

```
18. public class CustomerController
19. {
20. List<CustomerDTO> customers = null;
21. List<Long> friendFamily = null;
22. PlanDTO plan = null;
23. @Autowired
24. private CustomerService customerService;
25. //Fetching customer details
26. @GetMapping(produces="application/xml")
27. public List<CustomerDTO> fetchCustomer()
28. {
29. return customerService.fetchCustomer();
30. }
31. //Adding a customer
32. @PostMapping(consumes="application/json")
33. public ResponseEntity<String> createCustomer( @RequestBody CustomerDTO customerDTO)
34. {
35. String response = "";
36. response = customerService.createCustomer(customerDTO);
37. return ResponseEntity.ok(response);
38. }
39. //Updating an existing customer
40. @PutMapping(value =("/{phoneNumber}", consumes = "application/json")
41. public String updateCustomer(@PathVariable("phoneNumber") long phoneNumber,
    @RequestBody CustomerDTO customerDTO)
42. {
43. return customerService.updateCustomer(phoneNumber, customerDTO);
44. }
45. //Deleting a customer
46. @DeleteMapping("/{phoneNumber}")
47. public String deleteCustomer(@PathVariable("phoneNumber") long phoneNumber)
48. {
49. return customerService.deleteCustomer(phoneNumber);
50. }
51. }
```

**Step 5:** Create classes **CallDetailsDTO**, **CustomerDTO**, **ErrorMessage**, **FriendFamilyDTO**, **PlanDTO** under **com.infytel.dto** package:

```
1. package com.infytel.dto;
2. import java.time.LocalDate;
3. public class CallDetailsDTO {
4. long calledBy;
5. long calledTo;
```

```
6. LocalDate calledOn;
7. int duration;
8. public long getCalledBy() {
9.     return calledBy;
10. }
11. public void setCalledBy(long calledBy) {
12.     this.calledBy = calledBy;
13. }
14. public long getCalledTo() {
15.     return calledTo;
16. }
17. public void setCalledTo(long calledTo) {
18.     this.calledTo = calledTo;
19. }
20. public LocalDate getCalledOn() {
21.     return calledOn;
22. }
23. public void setCalledOn(LocalDate calledOn) {
24.     this.calledOn = calledOn;
25. }
26. public int getDuration() {
27.     return duration;
28. }
29. public void setDuration(int duration) {
30.     this.duration = duration;
31. }
32. @Override
33. public String toString() {
34.     return "CallDetailsDTO [calledBy=" + calledBy + ", calledTo=" + calledTo + ", calledOn=" +
        calledOn + ", duration=" + duration + " ]";
35. }
36. }
```

```
1. package com.infytel.dto;
2. import java.util.List;
3. public class CustomerDTO {
4.     long phoneNo;
5.     String name;
6.     String email;
7.     public String getEmail() {
8.         return email;
9.     }
```



```
10. public void setEmail(String email) {
11. this.email = email;
12. }
13. int age;
14. char gender;
15. List<FriendFamilyDTO> friendAndFamily;
16. String password;
17. String address;
18. PlanDTO currentPlan;
19. public PlanDTO getCurrentPlan() {
20. return currentPlan;
21. }
22. public void setCurrentPlan(PlanDTO currentPlan) {
23. this.currentPlan = currentPlan;
24. }
25. public String getPassword() {
26. return password;
27. }
28. public void setPassword(String password) {
29. this.password = password;
30. }
31. public String getAddress() {
32. return address;
33. }
34. public void setAddress(String address) {
35. this.address = address;
36. }
37. public List<FriendFamilyDTO> getFriendAndFamily() {
38. return friendAndFamily;
39. }
40. public void setFriendAndFamily(List<FriendFamilyDTO> friendAndFamily) {
41. this.friendAndFamily = friendAndFamily;
42. }
43. public long getPhoneNo() {
44. return phoneNo;
45. }
46. public void setPhoneNo(long phoneNo) {
47. this.phoneNo = phoneNo;
48. }
49. public String getName() {
50. return name;
51. }
```

```
52. public void setName(String name) {
53. this.name = name;
54. }
55. public int getAge() {
56. return age;
57. }
58. public void setAge(int age) {
59. this.age = age;
60. }
61. public char getGender() {
62. return gender;
63. }
64. public void setGender(char gender) {
65. this.gender = gender;
66. }
67. @Override
68. public String toString() {
69. return "CustomerDTO [phoneNo=" + phoneNo + ", name=" + name + ", age=" + age + ", gender="
    + gender + ", friendAndFamily=" + friendAndFamily + ", password=" + password + ", address=" +
    address + "]\n";
70. }
71. }
```

```
1. package com.infytel.dto;
2. public class ErrorMessage {
3. private int errorCode;
4. private String message;
5. public int getErrorCode() {
6. return errorCode;
7. }
8. public void setErrorCode(int errorCode) {
9. this.errorCode = errorCode;
10. }
11. public String getMessage() {
12. return message;
13. }
14. public void setMessage(String message) {
15. this.message = message;
16. }
17. }
```

```
1. package com.infytel.dto;
2. public class FriendFamilyDTO {
3.     long phoneNo;
4.     long friendAndFamily;
5.     public long getPhoneNo() {
6.         return phoneNo;
7.     }
8.     public void setPhoneNo(long phoneNo) {
9.         this.phoneNo = phoneNo;
10.    }
11.    public long getFriendAndFamily() {
12.        return friendAndFamily;
13.    }
14.    public void setFriendAndFamily(long friendAndFamily) {
15.        this.friendAndFamily = friendAndFamily;
16.    }
17.    public FriendFamilyDTO(long phoneNo, long friendAndFamily) {
18.        this();
19.        this.phoneNo = phoneNo;
20.        this.friendAndFamily = friendAndFamily;
21.    }
22.    public FriendFamilyDTO() {
23.        super();
24.    }
25.    @Override
26.    public String toString() {
27.        return "FriendFamilyDTO [phoneNo=" + phoneNo + ", friendAndFamily=" + friendAndFamily +
28.            "]" ;
29.    }
```

```
1. package com.infytel.dto;
2. import javax.xml.bind.annotation.XmlRootElement;
3. @XmlRootElement
4. public class PlanDTO {
5.     Integer planId;
6.     String planName;
7.     Integer nationalRate;
8.     Integer localRate;
9.     public Integer getPlanId() {
10.        return planId;
11.    }
```

```
12. public void setPlanId(Integer planId) {
13. this.planId = planId;
14. }
15. public String getPlanName() {
16. return planName;
17. }
18. public void setPlanName(String planName) {
19. this.planName = planName;
20. }
21. public Integer getNationalRate() {
22. return nationalRate;
23. }
24. public void setNationalRate(Integer nationalRate) {
25. this.nationalRate = nationalRate;
26. }
27. public Integer getLocalRate() {
28. return localRate;
29. }
30. public void setLocalRate(Integer localRate) {
31. this.localRate = localRate;
32. }
33. public PlanDTO() {
34. super();
35. }
36. @Override
37. public String toString() {
38. return "PlanDTO [planId=" + planId + ", planName=" + planName + ", nationalRate=" +
    nationalRate + ", localRate=" + localRate + "]";
39. }
40. }
```

**Step** **6: Create classes `CallDetailsRepository` and `CustomerRepository` under `com.infytel.repository` package:**

```
1. package com.infytel.repository;
2. import java.time.LocalDate;
3. import java.util.ArrayList;
4. import java.util.List;
5. import javax.annotation.PostConstruct;
6. import org.springframework.stereotype.Component;
7. import com.infytel.dto.CallDetailsDTO;
8. @Component
9. public class CallDetailsRepository {
10. List<CallDetailsDTO> callDetails = null;
```

```
11. CallDetailsDTO callDetailsDTO = null;
12. CallDetailsDTO callDetailsDTO1 = null;
13. LocalDate calledOn = null;
14. // Populating CallDetailsDTO
15. @PostConstruct
16. public void populatecalledOn() {
17. callDetails = new ArrayList<>();
18. callDetailsDTO = new CallDetailsDTO();
19. callDetailsDTO1 = new CallDetailsDTO();
20. calledOn = LocalDate.now();
21. callDetailsDTO.setCalledBy(88701064651);
22. callDetailsDTO.setCalledTo(99305084951);
23. callDetailsDTO.setCalledOn(calledOn);
24. callDetailsDTO.setDuration(3);
25. callDetailsDTO1.setCalledBy(88701064651);
26. callDetailsDTO1.setCalledTo(99305084951);
27. callDetailsDTO1.setCalledOn(calledOn);
28. callDetailsDTO1.setDuration(5);
29. callDetails.add(callDetailsDTO);
30. callDetails.add(callDetailsDTO1);
31. }
32. // To fetch call details based on calledBy and calledOn attributes of
33. // CallDetailsDTO
34. public List<CallDetailsDTO> fetchCallDetails(long calledBy, LocalDate calledOn) {
35. List<CallDetailsDTO> callDetailsResultSet = new ArrayList<>();
36. for (CallDetailsDTO callDetail : callDetails) {
37. if (callDetail.getCalledBy() == calledBy && callDetail.getCalledOn().equals(calledOn))
38. callDetailsResultSet.add(callDetail);
39. }
40. return callDetailsResultSet;
41. }
42. }
```

```
1. package com.infytel.repository;
2. import java.util.ArrayList;
3. import java.util.List;
4. import javax.annotation.PostConstruct;
5. import org.springframework.stereotype.Component;
6. import com.infytel.dto.CustomerDTO;
7. import com.infytel.dto.FriendFamilyDTO;
8. import com.infytel.dto.PlanDTO;
9. @Component
```

```
10. public class CustomerRepository {
11. List<CustomerDTO> customers = null;
12. // Populating CustomerDTO - hard-coded
13. @PostConstruct
14. public void initializer() {
15. CustomerDTO customerDTO = new CustomerDTO();
16. PlanDTO planDTO = new PlanDTO();
17. planDTO.setPlanId(1);
18. planDTO.setPlanName("Simple");
19. planDTO.setLocalRate(3);
20. planDTO.setNationalRate(5);
21. customerDTO.setAddress("Chennai");
22. customerDTO.setAge(18);
23. customerDTO.setCurrentPlan(planDTO);
24. customerDTO.setGender('m');
25. customerDTO.setName("Jack");
26. customerDTO.setEmail("Jack@infy.com");
27. customerDTO.setPassword("ABC@123");
28. customerDTO.setPhoneNo(9951212222l);
29. List<FriendFamilyDTO> friendAndFamily = new ArrayList<>();
30. friendAndFamily.add(new FriendFamilyDTO(customerDTO.getPhoneNo(), 800000145));
31. friendAndFamily.add(new FriendFamilyDTO(customerDTO.getPhoneNo(), 700000145));
32. customerDTO.setFriendAndFamily(friendAndFamily);
33. customers = new ArrayList<>();
34. customers.add(customerDTO);
35. }
36. // creates customer
37. public String createCustomer(CustomerDTO customerDTO) {
38. customers.add(customerDTO);
39. return "Customer with" + customerDTO.getPhoneNo() + "added successfully";
40. }
41. // fetches customers
42. public List<CustomerDTO> fetchCustomer() {
43. return customers;
44. }
45. // deletes customers
46. public String deleteCustomer(long phoneNumber) {
47. String response = "Customer of:" + phoneNumber + "\t does not exist";
48. for (CustomerDTO customer : customers) {
49. if (customer.getPhoneNo() == phoneNumber) {
50. customers.remove(customer);
51. response = customer.getName() + "of phoneNumber" + customer.getPhoneNo()
```

```
52. + "\t got deleted successfully";
53. break;
54. }
55. }
56. return response;
57. }
58. // updates customers
59. public String updateCustomer(long phoneNumber, CustomerDTO customerDTO) {
60. String response = "Customer of:" + phoneNumber + "\t does not exist";
61. for (CustomerDTO customer : customers) {
62. if (customer.getPhoneNo() == phoneNumber) {
63. if (customerDTO.getName() != null)
64. customer.setName(customerDTO.getName());
65. if (customerDTO.getAddress() != null)
66. customer.setAddress(customerDTO.getAddress());
67. if (customerDTO.getPassword() != null)
68. customer.setPassword(customerDTO.getPassword());
69. customers.set(customers.indexOf(customer), customer);
70. response = "Customer of phoneNumber" + customer.getPhoneNo() + "\t got updated successfully";
71. break;
72. }
73. }
74. return response;
75. }
76. }
```

**Step 7:** Create classes **CallDetailsService** and **CustomerService** under com.infytel.service package:

```
1. package com.infytel.service;
2. import java.time.LocalDate;
3. import java.util.List;
4. import org.springframework.beans.factory.annotation.Autowired;
5. import com.infytel.dto.CallDetailsDTO;
6. import com.infytel.repository.CallDetailsRepository;
7. public class CallDetailsService {
8.     @Autowired
9.     private CallDetailsRepository callDetailsRepository;
10. // contacts repository to fetch the call details
11. public List<CallDetailsDTO> fetchCallDetails(long calledBy, LocalDate calledOn) {
12. return callDetailsRepository.fetchCallDetails(calledBy, calledOn);
13. }
14. }
```

```
1. package com.infytel.service;
2. import java.util.List;
3. import org.springframework.beans.factory.annotation.Autowired;
4. import com.infytel.dto.CustomerDTO;
5. import com.infytel.repository.CustomerRepository;
6. public class CustomerService {
7.     @Autowired
8.     private CustomerRepository customerRepository;
9.     // calls repository layer method to create customer
10.    public String createCustomer(CustomerDTO customerDTO) {
11.        return customerRepository.createCustomer(customerDTO);
12.    }
13.    // calls repository layer method to fetch customers
14.    public List<CustomerDTO> fetchCustomer() {
15.        return customerRepository.fetchCustomer();
16.    }
17.    // calls repository layer method to delete customer
18.    public String deleteCustomer(long phoneNumber) {
19.        return customerRepository.deleteCustomer(phoneNumber);
20.    }
21.    // calls repository layer method to update customer
22.    public String updateCustomer(long phoneNumber, CustomerDTO customerDTO) {
23.        return customerRepository.updateCustomer(phoneNumber, customerDTO);
24.    }
25. }
```

**Step 8:** Add the following content to application.properties file:

```
1. server.port = 8080
2. server.servlet.context-path=/infytel-4
```

**Step 9:** Make sure that the project's pom.xml looks similar to the one that is shown below.

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <project xmlns="http://maven.apache.org/POM/4.0.0"
3.    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
5.        4.0.0.xsd">
6.    <modelVersion>4.0.0</modelVersion>
7.    <parent>
8.        <groupId>org.springframework.boot</groupId>
9.        <artifactId>spring-boot-starter-parent</artifactId>
10.       <version>2.1.4.RELEASE</version>
11.       <relativePath /> <!-- lookup parent from repository -->
12.    </parent>
```



```
12. <groupId>infytel</groupId>
13. <artifactId>infytel_demo4</artifactId>
14. <version>0.0.1-SNAPSHOT</version>
15. <name>infytel_demo4</name>
16. <description>Demo project for Spring Boot</description>
17. <properties>
18. <java.version>1.8</java.version>
19. </properties>
20. <dependencies>
21. <dependency>
22. <groupId>org.springframework.boot</groupId>
23. <artifactId>spring-boot-starter-web</artifactId>
24. </dependency>
25. <dependency>
26. <groupId>org.springframework.boot</groupId>
27. <artifactId>spring-boot-starter-test</artifactId>
28. <scope>test</scope>
29. </dependency>
30. </dependencies>
31. <build>
32. <plugins>
33. <plugin>
34. <groupId>org.springframework.boot</groupId>
35. <artifactId>spring-boot-maven-plugin</artifactId>
36. </plugin>
37. </plugins>
38. </build>
39. </project>
```

**Step 10:** Deploy the application on the server by executing the class containing the main method.

So, we have successfully created and deployed the REST endpoints. Now, let us see how can we test the same using Postman client.

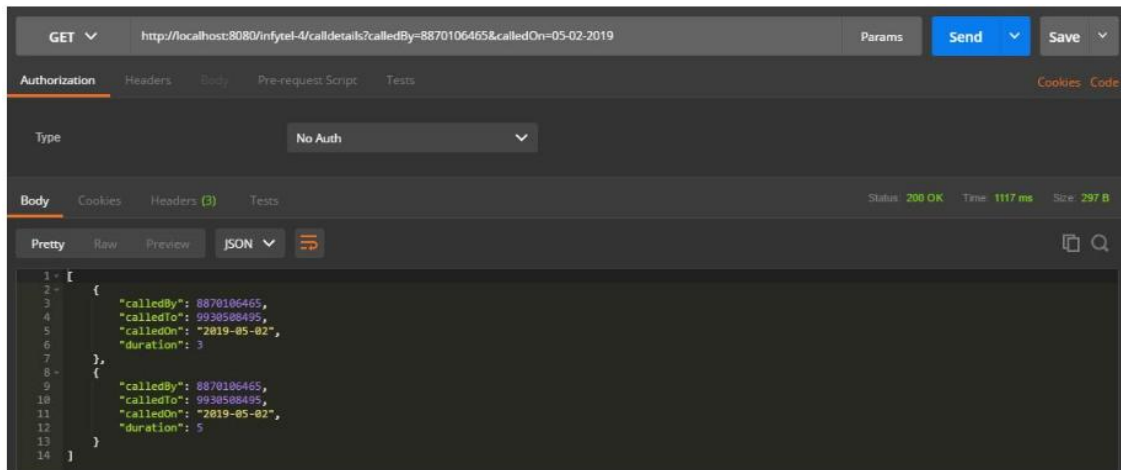
### Output Screenshots

Testing Web Service using Postman

**Step 1:** Launch Postman.

**Step 2:** Test this URL - **http://localhost:8080/infytel-**

**4/calldetails?calledBy=8870106465&calledOn=05-02-2019** using HTTP GET, where the URL contains the details of the customer(Phone number) and the date(current date in format mm-dd-yyyy) for which the call details should be fetched. Since the code works with a hard-coded collection, the date functionality has been coded to take only the current date.



Notice how, the query string has been passed and the same has been fetched in the RestController.

### Exercise - Usage of @RequestParam

CookPick online grocery application has a requirement where the users should be able to search for a product based on its name and vendor. As a result of the search, a list of product details matching with the given name and vendor should be rendered.

RESTful URL	HTTP Action	Business Operation
/product?productName=<product_name>&productVendor=<vendor_name>	GET	getProducts ()

Method description as follows:

**public List<ProductDTO> getProducts(String productName, String productVendor)**

This method,

- Takes product name and vendor as input
- Renders a list of products that match the condition

#### Note:

1. You can implement this requirement in the project that you used for previous exercise.
2. List of products needs to be rendered in JSON format.
3. You can simply hardcode the product details. It is optional to use a database to retrieve the product details.

\*Time given for this exercise indicates the time required to create the REST endpoint and is excluding the time required to write the logic for retrieving the product details from database.

### @Matrix variable video

### Demo 5 - Working with matrix variables using @MatrixVariable

**Objectives:** To create a Spring REST application that has code to work with matrix variables that are sent as part of URI. Here, we will learn

- How to configure a Spring Boot web application to provide the support for matrix variables
- The usage of @MatrixVariable

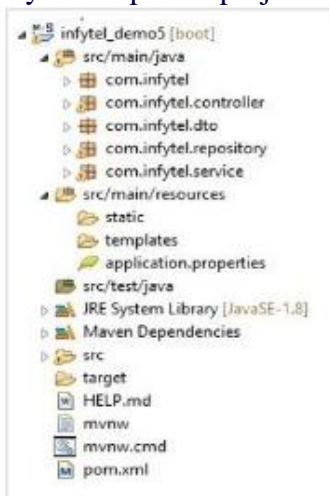
**Scenario:** An online telecom app called Infytel exposes its functionalities as RESTful resources. And, one among such resources is PlanController that deals with the plans that are available with Infytel.

Controller Class	Method Name	URI	HTTP Method	Remarks
PlanController	fetchPlans()	/plans	GET	fetches all the plans
PlanController	plansLocalRate()	/plans/{query}/plan	GET	fetches only those plans that matches the localrates provided via matrix parameter

### Steps:

**Step 1:** Create a Maven project using Spring Initializer with Web dependency and import the same in STS.

**Step 2:** Modify the imported project according to the following project structure:



**Step 3:** Look at the class InfytelDemo5Application in com.infytel package that gets created automatically when the project is generated. We will modify this only when we need to go with custom configuration. To enable matrix variables, configurePathMatch() method of WebMvcConfigurer needs to be overridden. Matrix variables are disabled by default and the following configuration

**urlPathHelper.setRemoveSemicolonContent(false);**

should be present in the overridden method to enable the same.

1. package com.infytel;
2. import org.springframework.boot.SpringApplication;
3. import org.springframework.boot.autoconfigure.SpringBootApplication;
4. import org.springframework.web.servlet.config.annotation.PathMatchConfigurer;
5. import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
6. import org.springframework.web.util.UrlPathHelper;
7. @SpringBootApplication
8. public class InfytelDemo5Application implements WebMvcConfigurer {
9. public static void main(String[] args) {
10. SpringApplication.run(InfytelDemo5Application.class, args);

```
11. }
12. // To support matrix parameters
13. @Override
14. public void configurePathMatch(PathMatchConfigurer configurer) {
15.     UrlPathHelper urlPathHelper = new UrlPathHelper();
16.     urlPathHelper.setRemoveSemicolonContent(false);
17.     configurer.setUrlPathHelper(urlPathHelper);
18. }
19. }
```

**Step 4:** Create classes **CallDetailsController**, **CustomerController** and **PlanController** under **com.infytel.controller** package:

```
1. package com.infytel.controller;
2. import java.time.LocalDate;
3. import java.time.format.DateTimeFormatter;
4. import java.util.List;
5. import org.springframework.beans.factory.annotation.Autowired;
6. import org.springframework.web.bind.annotation.GetMapping;
7. import org.springframework.web.bind.annotation.RequestMapping;
8. import org.springframework.web.bind.annotation.RequestParam;
9. import org.springframework.web.bind.annotation.RestController;
10. import com.infytel.dto.CallDetailsDTO;
11. import com.infytel.service.CallDetailsService;
12. @RestController
13. @RequestMapping("/calldetails")
14. public class CallDetailsController {
15.     @Autowired
16.     private CallDetailsService callDetailsService;
17.     // Fetching call details based on the request parameters being passed along with
18.     // the URI
19.     @GetMapping(produces = "application/json")
20.     public List<CallDetailsDTO> fetchCallDetails(@RequestParam("calledBy") long calledBy,
21.         @RequestParam("calledOn") String calledOn) {
22.         return callDetailsService.fetchCallDetails(calledBy,
23.             LocalDate.parse(calledOn, DateTimeFormatter.ofPattern("MM-dd-yyyy")));
24.     }
25. }
```

```
1. package com.infytel.controller;
2. import java.util.List;
3. import org.springframework.beans.factory.annotation.Autowired;
4. import org.springframework.http.ResponseEntity;
```

```
5. import org.springframework.web.bind.annotation.DeleteMapping;
6. import org.springframework.web.bind.annotation.GetMapping;
7. import org.springframework.web.bind.annotation.PathVariable;
8. import org.springframework.web.bind.annotation.PostMapping;
9. import org.springframework.web.bind.annotation.PutMapping;
10. import org.springframework.web.bind.annotation.RequestBody;
11. import org.springframework.web.bind.annotation.RequestMapping;
12. import org.springframework.web.bind.annotation.RestController;
13. import com.infytel.dto.CustomerDTO;
14. import com.infytel.dto.PlanDTO;
15. import com.infytel.service.CustomerService;
16. @RestController
17. @RequestMapping("/customers")
18. public class CustomerController {
19.     List<CustomerDTO> customers = null;
20.     List<Long> friendFamily = null;
21.     PlanDTO plan = null;
22.     @Autowired
23.     private CustomerService customerService;
24.     // fetch customer details
25.     @GetMapping(produces = "application/json")
26.     public List<CustomerDTO> fetchCustomer() {
27.         return customerService.fetchCustomer();
28.     }
29.     // add customer
30.     @PostMapping(consumes = "application/json")
31.     public ResponseEntity<String> createCustomer(@RequestBody CustomerDTO customerDTO) {
32.         String response = "";
33.         response = customerService.createCustomer(customerDTO);
34.         return ResponseEntity.ok(response);
35.     }
36.     // update an existing customer
37.     @PutMapping(value = "/{phoneNumber}", consumes = "application/json")
38.     public String updateCustomer(@PathVariable("phoneNumber") long phoneNumber,
39.         @RequestBody CustomerDTO customerDTO) {
39.         return customerService.updateCustomer(phoneNumber, customerDTO);
40.     }
41.     // delete customer
42.     @DeleteMapping("/{phoneNumber}")
43.     public String deleteCustomer(@PathVariable("phoneNumber") long phoneNumber) {
44.         return customerService.deleteCustomer(phoneNumber);
45.     }
```

46. }

```
1. package com.infytel.controller;
2. import java.util.ArrayList;
3. import java.util.List;
4. import java.util.Map;
5. import java.util.Set;
6. import org.springframework.beans.factory.annotation.Autowired;
7. import org.springframework.web.bind.annotation.GetMapping;
8. import org.springframework.web.bind.annotation.MatrixVariable;
9. import org.springframework.web.bind.annotation.RequestMapping;
10. import org.springframework.web.bind.annotation.RestController;
11. import com.infytel.dto.EntityList;
12. import com.infytel.dto.PlanDTO;
13. import com.infytel.service.PlanService;
14. @RestController
15. @RequestMapping("/plans")
16. public class PlanController {
17.     private EntityList<PlanDTO> plans;
18.     @Autowired
19.     private PlanService planService;
20.     // Get all available plans
21.     @GetMapping(produces = { "application/xml" })
22.     public EntityList<PlanDTO> fetchPlans() {
23.         plans = new EntityList<>(planService.fetchPlans());
24.         return plans;
25.     }
26.     @GetMapping(value =("/{query}/plan", produces = { "application/xml", "application/json" })
27.     public EntityList<PlanDTO> plansLocalRate(@MatrixVariable(pathVar = "query") Map<String,
        List<Integer>> map) {
28.         Set<String> keysLocalRates = map.keySet();
29.         List localRates = new ArrayList();
30.         for (String key : keysLocalRates) {
31.             for (int i = 0; i < map.get(key).size(); i++) {
32.                 localRates.add(map.get(key).get(i));
33.             }
34.         }
35.         plans = new EntityList<>(planService.plansLocalRate(localRates));
36.         return plans;
37.     }
38. }
```

**Step 5:** Create classes **CallDetailsDTO**, **CustomerDTO**, **EntityList**, **FriendFamilyDTO** and **PlanDTO** under com.infytel.dto package:

```
1. package com.infytel.dto;
2. import java.time.LocalDate;
3. public class CallDetailsDTO {
4.     long calledBy;
5.     long calledTo;
6.     LocalDate calledOn;
7.     int duration;
8.     public long getCalledBy() {
9.         return calledBy;
10.    }
11.    public void setCalledBy(long calledBy) {
12.        this.calledBy = calledBy;
13.    }
14.    public long getCalledTo() {
15.        return calledTo;
16.    }
17.    public void setCalledTo(long calledTo) {
18.        this.calledTo = calledTo;
19.    }
20.    public LocalDate getCalledOn() {
21.        return calledOn;
22.    }
23.    public void setCalledOn(LocalDate calledOn) {
24.        this.calledOn = calledOn;
25.    }
26.    public int getDuration() {
27.        return duration;
28.    }
29.    public void setDuration(int duration) {
30.        this.duration = duration;
31.    }
32.    @Override
33.    public String toString() {
34.        return "CallDetailsDTO [calledBy=" + calledBy + ", calledTo=" + calledTo + ", calledOn=" +
35.            calledOn + ", duration=" + duration + " ]";
36.    }
```

```
1. package com.infytel.dto;
2. import java.util.List;
```

```
3. public class CustomerDTO {
4.     long phoneNo;
5.     String name;
6.     String email;
7.     public String getEmail() {
8.         return email;
9.     }
10.    public void setEmail(String email) {
11.        this.email = email;
12.    }
13.    int age;
14.    char gender;
15.    List<FriendFamilyDTO> friendAndFamily;
16.    String password;
17.    String address;
18.    PlanDTO currentPlan;
19.    public PlanDTO getCurrentPlan() {
20.        return currentPlan;
21.    }
22.    public void setCurrentPlan(PlanDTO currentPlan) {
23.        this.currentPlan = currentPlan;
24.    }
25.    public String getPassword() {
26.        return password;
27.    }
28.    public void setPassword(String password) {
29.        this.password = password;
30.    }
31.    public String getAddress() {
32.        return address;
33.    }
34.    public void setAddress(String address) {
35.        this.address = address;
36.    }
37.    public List<FriendFamilyDTO> getFriendAndFamily() {
38.        return friendAndFamily;
39.    }
40.    public void setFriendAndFamily(List<FriendFamilyDTO> friendAndFamily) {
41.        this.friendAndFamily = friendAndFamily;
42.    }
43.    public long getPhoneNo() {
44.        return phoneNo;
```



```
45. }
46. public void setPhoneNo(long phoneNo) {
47. this.phoneNo = phoneNo;
48. }
49. public String getName() {
50. return name;
51. }
52. public void setName(String name) {
53. this.name = name;
54. }
55. public int getAge() {
56. return age;
57. }
58. public void setAge(int age) {
59. this.age = age;
60. }
61. public char getGender() {
62. return gender;
63. }
64. public void setGender(char gender) {
65. this.gender = gender;
66. }
67. @Override
68. public String toString() {
69. return "CustomerDTO [phoneNo=" + phoneNo + ", name=" + name + ", age=" + age + ", gender="
    + gender + ", friendAndFamily=" + friendAndFamily + ", password=" + password + ", address=" +
    address + "]\n";
70. }
71. }
```

```
1. package com.infytel.dto;
2. import java.util.ArrayList;
3. import java.util.List;
4. import javax.xml.bind.annotation.XmlAnyElement;
5. import javax.xml.bind.annotation.XmlRootElement;
6. import javax.xml.bind.annotation.XmlSeeAlso;
7. @XmlRootElement
8. @XmlSeeAlso({ PlanDTO.class })
9. public class EntityList<T> {
10. private List<T> listOfEntityObjects;
11. public EntityList() {
12. listOfEntityObjects = new ArrayList<>();
```

```
13. }
14. public EntityList(List<T> listOfEntityObjects) {
15. this.listOfEntityObjects = listOfEntityObjects;
16. }
17. @XmlElement
18. public List<T> getItems() {
19. return listOfEntityObjects;
20. }
21. }
```

```
1. package com.infytel.dto;
2. public class FriendFamilyDTO {
3. long phoneNo;
4. long friendAndFamily;
5. public long getPhoneNo() {
6. return phoneNo;
7. }
8. public void setPhoneNo(long phoneNo) {
9. this.phoneNo = phoneNo;
10. }
11. public long getFriendAndFamily() {
12. return friendAndFamily;
13. }
14. public void setFriendAndFamily(long friendAndFamily) {
15. this.friendAndFamily = friendAndFamily;
16. }
17. public FriendFamilyDTO(long phoneNo, long friendAndFamily) {
18. this();
19. this.phoneNo = phoneNo;
20. this.friendAndFamily = friendAndFamily;
21. }
22. public FriendFamilyDTO() {
23. super();
24. }
25. @Override
26. public String toString() {
27. return "FriendFamilyDTO [phoneNo=" + phoneNo + ", friendAndFamily=" + friendAndFamily +
    "]" ;
28. }
29. }
```

```
1. package com.infytel.dto;
2. import javax.xml.bind.annotation.XmlRootElement;
3. @XmlRootElement
4. public class PlanDTO {
5.     Integer planId;
6.     String planName;
7.     Integer nationalRate;
8.     Integer localRate;
9.     public Integer getPlanId() {
10.         return planId;
11.     }
12.     public void setPlanId(Integer planId) {
13.         this.planId = planId;
14.     }
15.     public String getPlanName() {
16.         return planName;
17.     }
18.     public void setPlanName(String planName) {
19.         this.planName = planName;
20.     }
21.     public Integer getNationalRate() {
22.         return nationalRate;
23.     }
24.     public void setNationalRate(Integer nationalRate) {
25.         this.nationalRate = nationalRate;
26.     }
27.     public Integer getLocalRate() {
28.         return localRate;
29.     }
30.     public void setLocalRate(Integer localRate) {
31.         this.localRate = localRate;
32.     }
33.     public PlanDTO() {
34.         super();
35.     }
36.     @Override
37.     public String toString() {
38.         return "PlanDTO [planId=" + planId + ", planName=" + planName + ", nationalRate=" +
            nationalRate + ", localRate=" + localRate + "];";
39.     }
40. }
```

**Step 6:** Create classes **CallDetailsRepository**, **CustomerRepository** and **PlanRepository** under **com.infytel.repository** package:

```
1. package com.infytel.repository;
2. import java.time.LocalDate;
3. import java.util.ArrayList;
4. import java.util.List;
5. import javax.annotation.PostConstruct;
6. import org.springframework.stereotype.Repository;
7. import com.infytel.dto.CallDetailsDTO;
8. @Repository
9. public class CallDetailsRepository {
10. List<CallDetailsDTO> callDetails = null;
11. CallDetailsDTO callDetailsDTO = null;
12. CallDetailsDTO callDetailsDTO1 = null;
13. LocalDate calledOn = null;
14. // Populates CallDetails in hard-coded way
15. @PostConstruct
16. public void populatecalledOn() {
17. callDetails = new ArrayList<>();
18. callDetailsDTO = new CallDetailsDTO();
19. callDetailsDTO1 = new CallDetailsDTO();
20. calledOn = LocalDate.now();
21. callDetailsDTO.setCalledBy(88701064651);
22. callDetailsDTO.setCalledTo(99305084951);
23. callDetailsDTO.setCalledOn(calledOn);
24. callDetailsDTO.setDuration(3);
25. callDetailsDTO1.setCalledBy(88701064651);
26. callDetailsDTO1.setCalledTo(99305084951);
27. callDetailsDTO1.setCalledOn(calledOn);
28. callDetailsDTO1.setDuration(5);
29. callDetails.add(callDetailsDTO);
30. callDetails.add(callDetailsDTO1);
31. }
32. // fetches call details by calledBy and calledOn
33. public List<CallDetailsDTO> fetchCallDetails(long calledBy, LocalDate calledOn) {
34. List<CallDetailsDTO> callDetailsResultSet = new ArrayList<>();
35. for (CallDetailsDTO callDetail : callDetails) {
36. if (callDetail.getCalledBy() == calledBy && callDetail.getCalledOn().equals(calledOn))
37. callDetailsResultSet.add(callDetail);
38. }
39. return callDetailsResultSet;
40. }
```

```
41. }
```

```
1. package com.infytel.repository;
2. import java.util.ArrayList;
3. import java.util.List;
4. import javax.annotation.PostConstruct;
5. import org.springframework.stereotype.Repository;
6. import com.infytel.dto.CustomerDTO;
7. import com.infytel.dto.FriendFamilyDTO;
8. import com.infytel.dto.PlanDTO;
9. @Repository
10. public class CustomerRepository {
11.     List<CustomerDTO> customers = null;
12.     // populates customer in hard-coded way
13.     @PostConstruct
14.     public void initializer() {
15.         CustomerDTO customerDTO = new CustomerDTO();
16.         PlanDTO planDTO = new PlanDTO();
17.         planDTO.setPlanId(1);
18.         planDTO.setPlanName("Simple");
19.         planDTO.setLocalRate(3);
20.         planDTO.setNationalRate(5);
21.         customerDTO.setAddress("Chennai");
22.         customerDTO.setAge(18);
23.         customerDTO.setCurrentPlan(planDTO);
24.         customerDTO.setGender('m');
25.         customerDTO.setName("Jack");
26.         customerDTO.setEmail("Jack@infy.com");
27.         customerDTO.setPassword("ABC@123");
28.         customerDTO.setPhoneNo(99512122221);
29.         List<FriendFamilyDTO> friendAndFamily = new ArrayList<>();
30.         friendAndFamily.add(new FriendFamilyDTO(customerDTO.getPhoneNo(), 800000145));
31.         friendAndFamily.add(new FriendFamilyDTO(customerDTO.getPhoneNo(), 700000145));
32.         customerDTO.setFriendAndFamily(friendAndFamily);
33.         customers = new ArrayList<>();
34.         customers.add(customerDTO);
35.     }
36.     // creates customer
37.     public String createCustomer(CustomerDTO customerDTO) {
38.         customers.add(customerDTO);
39.         return "Customer with" + customerDTO.getPhoneNo() + "added successfully";
40.     }
```

```
41. // fetches customer
42. public List<CustomerDTO> fetchCustomer() {
43. return customers;
44. }
45. // deletes customer
46. public String deleteCustomer(long phoneNumber) {
47. String response = "Customer of:" + phoneNumber + "\t does not exist";
48. for (CustomerDTO customer : customers) {
49. if (customer.getPhoneNo() == phoneNumber) {
50. customers.remove(customer);
51. response = customer.getName() + "of phoneNumber" + customer.getPhoneNo()
52. + "\t got deleted successfully";
53. break;
54. }
55. }
56. return response;
57. }
58. // updates customer
59. public String updateCustomer(long phoneNumber, CustomerDTO customerDTO) {
60. String response = "Customer of:" + phoneNumber + "\t does not exist";
61. for (CustomerDTO customer : customers) {
62. if (customer.getPhoneNo() == phoneNumber) {
63. if (customerDTO.getName() != null)
64. customer.setName(customerDTO.getName());
65. if (customerDTO.getAddress() != null)
66. customer.setAddress(customerDTO.getAddress());
67. if (customerDTO.getPassword() != null)
68. customer.setPassword(customerDTO.getPassword());
69. customers.set(customers.indexOf(customer), customer);
70. response = "Customer of phoneNumber" + customer.getPhoneNo() + "\t got updated successfully";
71. break;
72. }
73. }
74. return response;
75. }
76. }
```

```
1. package com.infytel.repository;
2. import java.util.ArrayList;
3. import java.util.Iterator;
4. import java.util.List;
5. import javax.annotation.PostConstruct;
```

```
6. import org.springframework.stereotype.Repository;
7. import com.infytel.dto.PlanDTO;
8. @Repository
9. public class PlanRepository {
10. private List<PlanDTO> plans;
11. // Populating a list of plans in hard-coded way
12. @PostConstruct
13. public void populatePlans() {
14. plans = new ArrayList<>();
15. PlanDTO plan1 = new PlanDTO();
16. plan1.setPlanId(1);
17. plan1.setPlanName("Simple");
18. plan1.setLocalRate(3);
19. plan1.setNationalRate(5);
20. plans.add(plan1);
21. PlanDTO plan2 = new PlanDTO();
22. plan2.setPlanId(2);
23. plan2.setPlanName("Medium");
24. plan2.setLocalRate(5);
25. plan2.setNationalRate(8);
26. plans.add(plan2);
27. }
28. // fetching plans
29. public List<PlanDTO> fetchPlans() {
30. return plans;
31. }
32. // fetching plans based on localRates
33. public List<PlanDTO> plansLocalRate(List localRates) {
34. List<PlanDTO> plansResponse = new ArrayList<>();
35. Iterator it = localRates.iterator();
36. while (it.hasNext()) {
37. int rate = Integer.parseInt((String) it.next());
38. for (PlanDTO plan : plans) {
39. if (rate == plan.getLocalRate())
40. plansResponse.add(plan);
41. }
42. }
43. return plansResponse;
44. }
45. }
```

**Step 7:** Create classes **CallDetailsService**, **CustomerService** and **PlanService** under **com.infytel.service** package:

```
1. package com.infytel.service;
2. import java.time.LocalDate;
3. import java.util.List;
4. import org.springframework.beans.factory.annotation.Autowired;
5. import org.springframework.stereotype.Service;
6. import com.infytel.dto.CallDetailsDTO;
7. import com.infytel.repository.CallDetailsRepository;
8. @Service
9. public class CallDetailsService {
10. @Autowired
11. private CallDetailsRepository callDetailsRepository;
12. // contacts repository to fetch the call details
13. public List<CallDetailsDTO> fetchCallDetails(long calledBy, LocalDate calledOn) {
14. return callDetailsRepository.fetchCallDetails(calledBy, calledOn);
15. }
16. }
```

```
1. package com.infytel.service;
2. import java.util.List;
3. import org.springframework.beans.factory.annotation.Autowired;
4. import org.springframework.stereotype.Service;
5. import com.infytel.dto.CustomerDTO;
6. import com.infytel.repository.CustomerRepository;
7. @Service
8. public class CustomerService {
9. @Autowired
10. private CustomerRepository customerRepository;
11. // Contacts repository layer to add customer
12. public String createCustomer(CustomerDTO customerDTO) {
13. return customerRepository.createCustomer(customerDTO);
14. }
15. // Contacts repository layer to fetch customer
16. public List<CustomerDTO> fetchCustomer() {
17. return customerRepository.fetchCustomer();
18. }
19. // Contacts repository layer to delete customer
20. public String deleteCustomer(long phoneNumber) {
21. return customerRepository.deleteCustomer(phoneNumber);
22. }
23. // Contacts repository layer to update customer
24. public String updateCustomer(long phoneNumber, CustomerDTO customerDTO) {
25. return customerRepository.updateCustomer(phoneNumber, customerDTO);
26. }
```



```
26. }  
27. }
```

```
1. package com.infytel.service;  
2. import java.util.List;  
3. import org.springframework.beans.factory.annotation.Autowired;  
4. import org.springframework.stereotype.Service;  
5. import com.infytel.dto.PlanDTO;  
6. import com.infytel.repository.PlanRepository;  
7. @Service  
8. public class PlanService {  
9.     @Autowired  
10.    private PlanRepository planRepository;  
11.    // contacts repository to fetch plans  
12.    public List<PlanDTO> fetchPlans() {  
13.        return planRepository.fetchPlans();  
14.    }  
15.    // contacts repository to fetch plans by localRates  
16.    public List<PlanDTO> plansLocalRate(List localRates) {  
17.        return planRepository.plansLocalRate(localRates);  
18.    }  
19. }
```

**Step 8:** Add the following content to **application.properties** file:

```
1. server.port = 8080  
2. server.servlet.context-path=/infytel-5
```

**Step 9:** Make sure that the project's pom.xml looks similar to the one that is shown below.

```
1. <?xml version="1.0" encoding="UTF-8"?>  
2. <project xmlns="http://maven.apache.org/POM/4.0.0"  
3.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
4.     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-  
5.         4.0.0.xsd">  
6.     <modelVersion>4.0.0</modelVersion>  
7.     <parent>  
8.         <groupId>org.springframework.boot</groupId>  
9.         <artifactId>spring-boot-starter-parent</artifactId>  
10.        <version>2.1.4.RELEASE</version>  
11.        <relativePath /> <!-- lookup parent from repository -->  
12.    </parent>
```

```
12. <groupId>infytel</groupId>
13. <artifactId>infytel_demo5</artifactId>
14. <version>0.0.1-SNAPSHOT</version>
15. <name>infytel_demo5</name>
16. <description>Demo project for Spring Boot</description>
17. <properties>
18. <java.version>1.8</java.version>
19. </properties>
20. <dependencies>
21. <dependency>
22. <groupId>org.springframework.boot</groupId>
23. <artifactId>spring-boot-starter-web</artifactId>
24. </dependency>
25. <dependency>
26. <groupId>org.springframework.boot</groupId>
27. <artifactId>spring-boot-starter-test</artifactId>
28. <scope>test</scope>
29. </dependency>
30. </dependencies>
31. <build>
32. <plugins>
33. <plugin>
34. <groupId>org.springframework.boot</groupId>
35. <artifactId>spring-boot-maven-plugin</artifactId>
36. </plugin>
37. </plugins>
38. </build>
39. </project>
```

**Step 10:** Deploy the application on the server by executing the class containing the main method.

So, we have successfully created and deployed the REST endpoints. Now, let us see how can we test the same using Postman client.

## Output

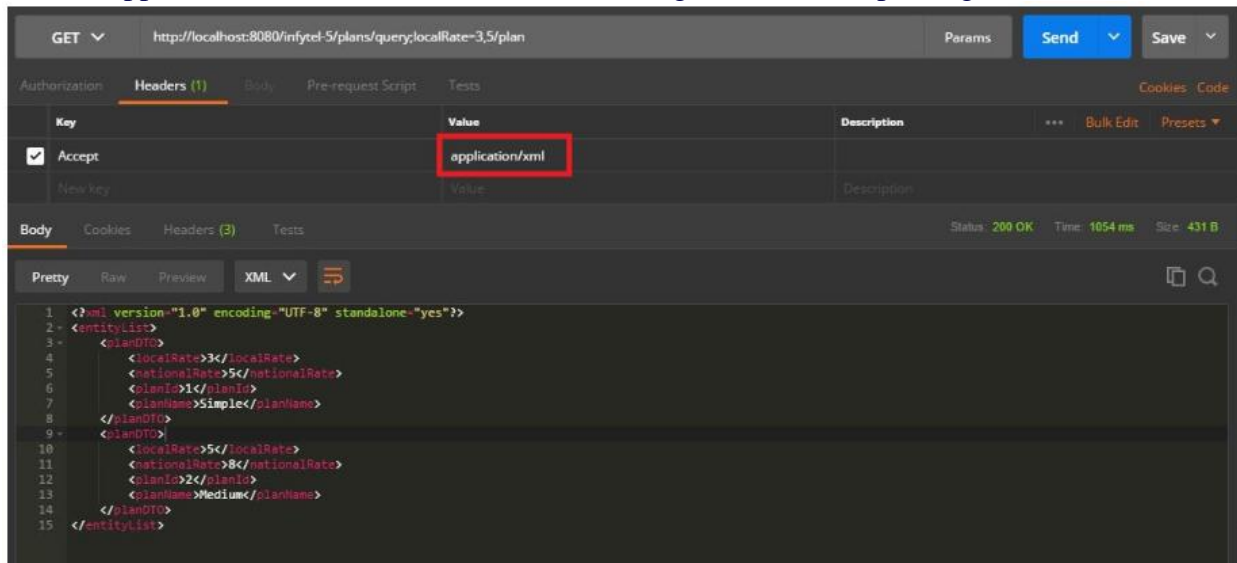
Testing Web Service using Postman

**Step 1:** Launch Postman.

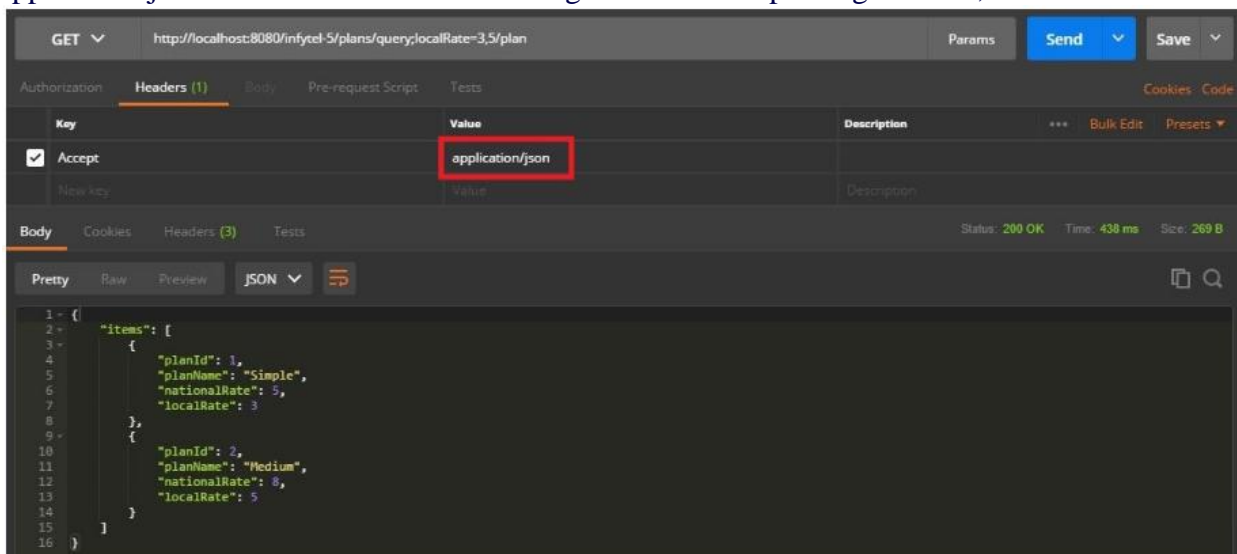
**Step 2:** Test this URL - **http://localhost:8080/infytel-5/plans/query;localRate=3,5/plan** using HTTP GET, where the URL reached out to plansLocalRate() method that works with matrix variables. Notice the way localrate is supplied in the URL. It appears in the middle of the URI path .

**Note:** There are two ways, the application can produce the data i.e., XML and JSON.

To get the response in XML format, enter the URL and select Headers and set the key as Accept and the value as application/xml and click on send. Following will be the response generated,



To get the response in JSON format, enter the URL, select Headers and set the key as Accept and value as application/json and click on send. Following will be the response generated,



Look at the records that are available in both the screenshots. Only the details of plans with localRate 3 and 5 are present.

### Exercise - Usage of @MatrixVariable

Write a RESTful endpoint for CookPick online grocery application that renders a list of products based on the vendor names.

RESTful URL	HTTP Action	Business Operation
/product/query;vendor=Sochsuper,Iravisupreme/	GET	getProducts()

**Method description as follows:**

```
public List<ProductDTO> getProducts(Map<String, List<String>> map)
```

This method,

- Takes a list of vendor names (matrix variables)
- Searches for the products of vendors whose names are received as matrix variables and renders the list of products in **XML** format

Note:

1. You can implement this requirement in the project that you used for previous exercise.
2. You can simply hardcode the product details. It is optional to use a database to retrieve the product details.

\*Time given for this exercise indicates the time required to create the REST endpoint and is excluding the time required to write the logic for retrieving the product details from database.

### Exception Handling - An Introduction

Handling exceptions will make sure that the entire stack trace is not thrown to the end-user which is very hard to read and, possesses a lot of security risks as well. With proper exception handling routine, it is possible for an application to send customized messages during failures and continue without being terminated abruptly.

In simple words, we can say that exception handling makes any application robust.

@ExceptionHandler is a Spring annotation that plays a vital role in handling exceptions thrown out of handler methods(Controller operations). This annotation can be used on the

- Methods of a controller class. Doing so will help handle exceptions thrown out of the methods of that specific controller alone.
- Methods of classes that are annotated with @RestControllerAdvice. Doing so will make exception handling global.

The most common way is applying @ExceptionHandler on methods of the class that is annotated with @RestControllerAdvice. This will make the exceptions that are thrown from the controllers of the application get handled in a centralized way. As well, there is no need for repeating the exception handling code in all the controllers, keeping the code more manageable.

### Example - Exception with no handler method

Example: An online telecom app called Infytel exposes its functionalities as RESTful resources. And, CustomerController is one among such resources that deal with the customers of Infytel. CustomerController has a method to delete a customer based on the phone number being passed.

Below is the code snippet for the same.

```
1. @RestController
2. @RequestMapping("/customers")
3. public class CustomerController
4. {
5.     // Deleting a customer
6.     @DeleteMapping(value =("/{phoneNumber}", produces = "text/html")
```

```
7. public String deleteCustomer(  
8.   @PathVariable("phoneNumber") long phoneNumber)  
9.   throws NoSuchCustomerException {  
10. // code goes here  
11. }  
12. }
```

What if the phoneNo does not exist and the code throws NoSuchCustomerException?

If no Exception handler is provided, Spring Boot provides a standard error message as shown below.

```
1. {  
2.   "timestamp": "2019-05-02T16:10:45.805+0000",  
3.   "status": 500,  
4.   "error": "Internal Server Error",  
5.   "message": "Customer does not exist :121",  
6.   "path": "/infytel-7/customers/121"  
7. }
```

### Example - Exception with handler method

In case, a customized error message that is easy to understand has to be provided, we need to

- Create a class annotated with @RestControllerAdvice
- Have methods annotated with @ExceptionHandler(value=NameoftheException) which takes the exception class as the value for which the method is the handler

We can have multiple methods in the Advice class to handle exceptions of different types and return the custom messages accordingly.

Example: Below is the RestControllerAdvice class that holds a method which handles NoSuchCustomerException by returning a customized error message.

```
1. @RestControllerAdvice  
2. public class ExceptionControllerAdvice {  
3.   @ExceptionHandler(NoSuchCustomerException.class)  
4.   public ResponseEntity<String> exceptionHandler2(NoSuchCustomerException ex) {  
5.     return new ResponseEntity<>(ex.getMessage(), HttpStatus.BAD_REQUEST);  
6.   }  
7. }
```

### Custom Error Message

It is important to map the exceptions to objects that can provide some specific information which allows the API clients to know, what has happened exactly. So, instead of returning a String, we can send an object that holds the error message and error code to the client back.

Let us look at an example:

The object which is going to hold a custom error message is **ErrorMessage** with two variables, errorcode and message.

```
1. public class ErrorMessage {
2.     private int errorCode;
3.     private String message;
4.     //getters and setters go here
5. }
```

Now, the `RestControllerAdvice` will be modified as shown below.

```
1. @RestControllerAdvice
2. public class ExceptionControllerAdvice {
3.     @ExceptionHandler(NoSuchCustomerException.class)
4.     public ResponseEntity<ErrorMessage> exceptionHandler2(NoSuchCustomerException ex) {
5.         ErrorMessage error = new ErrorMessage();
6.         error.setErrorCode(HttpStatus.BAD_GATEWAY.value());
7.         error.setMessage(ex.getMessage());
8.         return new ResponseEntity<>(error, HttpStatus.OK);
9.     }
10. }
```

Here, the handler method returns the instance of **ErrorMessage** that holds the error code and message, rather returning a String as the body of `ResponseEntity`.

Let us put all these together and look at an example.

## Video

### Demo: Handling Exceptions in a Spring REST Application

**Objectives:** To develop a Spring REST application that returns a graceful error message when the REST endpoints encounter exceptions. We will learn,

- How to handle Exceptions in a Spring REST application
- Usage of `@RestControllerAdvice` annotation
- Usage of `@ExceptionHandler` annotation

### Scenario:

An online telecom app called Infytel wishes its functionalities to get exposed as RESTful services. It has three controllers in total to deal with customer, plan and call details, among which, the focus will be on the controller named `CustomerController`. Following HTTP operation of the `CustomerController` will be taken for discussion.

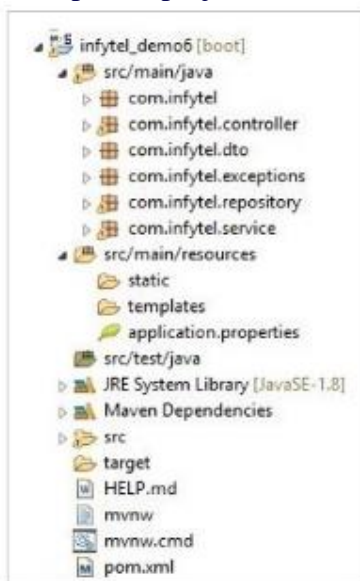
Controller Class	method	URI	HTTP method	Remarks
CustomerController	deleteCustomer() throws NoSuchCustomerException	/customers/{phoneNumber}	HTTP DELETE	Deletes an existing customer by phoneNumber. If the phoneNumber does not exist, the method

				throws NoSuchCustomerException
--	--	--	--	--------------------------------

**Steps:**

**Step 1:** Create a Maven project using Spring Initializer with Web dependency and import the same in STS.

**Step 2:** Modify the imported project according to the following project structure:



**Step 3:** Look at the class **InfytelDemo6Application** under **com.infytel** package that gets created automatically when the project is generated. We will modify this only when we need to go with custom configuration. Here, this class holds the code that is required to support matrix variables.

```

1. package com.infytel;
2. import org.springframework.boot.SpringApplication;
3. import org.springframework.boot.autoconfigure.SpringBootApplication;
4. import org.springframework.web.servlet.config.annotation.PathMatchConfigurer;
5. import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
6. import org.springframework.web.util.UrlPathHelper;
7. @SpringBootApplication
8. public class InfytelDemo6Application implements WebMvcConfigurer {
9.     public static void main(String[] args) {
10.         SpringApplication.run(InfytelDemo6Application.class, args);
11.     }
12.     // To support matrix parameters
13.     @Override
14.     public void configurePathMatch(PathMatchConfigurer configurer) {
15.         UrlPathHelper urlPathHelper = new UrlPathHelper();
16.         urlPathHelper.setRemoveSemicolonContent(false);
17.         configurer.setUrlPathHelper(urlPathHelper);

```

```
18. }  
19. }
```

**Step 4:** Create classes **CallDetailsController**, **CustomerController** and **PlanController** under **com.infytel.controller** package:

```
1. package com.infytel.controller;  
2. import java.time.LocalDate;  
3. import java.time.format.DateTimeFormatter;  
4. import java.util.List;  
5. import org.springframework.beans.factory.annotation.Autowired;  
6. import org.springframework.web.bind.annotation.GetMapping;  
7. import org.springframework.web.bind.annotation.RequestMapping;  
8. import org.springframework.web.bind.annotation.RequestParam;  
9. import org.springframework.web.bind.annotation.RestController;  
10. import com.infytel.dto.CallDetailsDTO;  
11. import com.infytel.service.CallDetailsService;  
12. @RestController  
13. @RequestMapping("/calldetails")  
14. public class CallDetailsController {  
15.     @Autowired  
16.     private CallDetailsService callDetailsService;  
17.     // Fetching call details based on the request parameters being passed along with  
18.     // the URI  
19.     @GetMapping(produces = "application/json")  
20.     public List<CallDetailsDTO> callDetails(@RequestParam("calledBy") long calledBy,  
21.     @RequestParam("calledOn") String calledOn) {  
22.         return callDetailsService.fetchCallDetails(calledBy,  
23.             LocalDate.parse(calledOn, DateTimeFormatter.ofPattern("MM-dd-yyyy")));  
24.     }  
25. }
```

```
1. package com.infytel.controller;  
2. import java.util.List;  
3. import org.springframework.beans.factory.annotation.Autowired;  
4. import org.springframework.http.ResponseEntity;  
5. import org.springframework.web.bind.annotation.DeleteMapping;  
6. import org.springframework.web.bind.annotation.GetMapping;  
7. import org.springframework.web.bind.annotation.PathVariable;  
8. import org.springframework.web.bind.annotation.PostMapping;  
9. import org.springframework.web.bind.annotation.PutMapping;  
10. import org.springframework.web.bind.annotation.RequestBody;  
11. import org.springframework.web.bind.annotation.RequestMapping;
```



```
12. import org.springframework.web.bind.annotation.RestController;
13. import com.infytel.dto.CustomerDTO;
14. import com.infytel.dto.PlanDTO;
15. import com.infytel.exceptions.NoSuchCustomerException;
16. import com.infytel.service.CustomerService;
17. @RestController
18. @RequestMapping("/customers")
19. public class CustomerController {
20. List<CustomerDTO> customers = null;
21. List<Long> friendFamily = null;
22. PlanDTO plan = null;
23. @Autowired
24. private CustomerService customerService;
25. // Fetching customer details
26. @GetMapping(produces = "application/json")
27. public List<CustomerDTO> fetchCustomer() {
28. return customerService.fetchCustomer();
29. }
30. // Adding a customer
31. @PostMapping(consumes = "application/json")
32. public ResponseEntity<String> createCustomer(@RequestBody CustomerDTO customerDTO) {
33. String response = "";
34. response = customerService.createCustomer(customerDTO);
35. return ResponseEntity.ok(response);
36. }
37. // Updating an existing customer
38. @PutMapping(value =("/{phoneNumber}", consumes = "application/json")
39. public String updateCustomer(@PathVariable("phoneNumber") long phoneNumber,
    @RequestBody CustomerDTO customerDTO) {
40. return customerService.updateCustomer(phoneNumber, customerDTO);
41. }
42. // Deleting a customer
43. @DeleteMapping(value =("/{phoneNumber}", produces = "text/plain")
44. public String deleteCustomer(@PathVariable("phoneNumber") long phoneNumber) throws
    NoSuchCustomerException {
45. return customerService.deleteCustomer(phoneNumber);
46. }
47. }
```

```
1. package com.infytel.controller;
2. import java.util.ArrayList;
3. import java.util.List;
```

```
4. import java.util.Map;
5. import java.util.Set;
6. import org.springframework.beans.factory.annotation.Autowired;
7. import org.springframework.web.bind.annotation.GetMapping;
8. import org.springframework.web.bind.annotation.MatrixVariable;
9. import org.springframework.web.bind.annotation.RequestMapping;
10. import org.springframework.web.bind.annotation.RestController;
11. import com.infytel.dto.EntityList;
12. import com.infytel.dto.PlanDTO;
13. import com.infytel.service.PlanService;
14. @RestController
15. @RequestMapping("/plans")
16. public class PlanController {
17.     private EntityList<PlanDTO> plans;
18.     @Autowired
19.     private PlanService planService;
20.     // Get all available plans
21.     @GetMapping(produces = { "application/json", "application/xml" })
22.     public EntityList<PlanDTO> fetchPlans() {
23.         plans = new EntityList<>(planService.fetchPlans());
24.         return plans;
25.     }
26.     // Gets plans based on localRate
27.     @GetMapping(value =("/{query}/plan", produces = { "application/json", "application/xml" })
28.     public EntityList<PlanDTO> plansLocalRate(@MatrixVariable(pathVar = "query") Map<String,
        List<Integer>> map) {
29.         Set<String> keysLocalRates = map.keySet();
30.         ArrayList localRates = new ArrayList();
31.         for (String key : keysLocalRates) {
32.             for (int i = 0; i < map.get(key).size(); i++) {
33.                 localRates.add(map.get(key).get(i));
34.             }
35.         }
36.         plans = new EntityList<>(planService.plansLocalRate(localRates));
37.         return plans;
38.     }
39. }
```

**Step 5:** Create classes **CallDetailsDTO**, **CustomerDTO**, **EntityList**, **ErrorMessage**, **FriendFamilyDTO** and **PlanDTO** under com.infytel.dto package:

```
1. package com.infytel.dto;
2. import java.time.LocalDate;
```

```
3. public class CallDetailsDTO {
4.     long calledBy;
5.     long calledTo;
6.     LocalDate calledOn;
7.     int duration;
8.     public long getCalledBy() {
9.         return calledBy;
10.    }
11.    public void setCalledBy(long calledBy) {
12.        this.calledBy = calledBy;
13.    }
14.    public long getCalledTo() {
15.        return calledTo;
16.    }
17.    public void setCalledTo(long calledTo) {
18.        this.calledTo = calledTo;
19.    }
20.    public LocalDate getCalledOn() {
21.        return calledOn;
22.    }
23.    public void setCalledOn(LocalDate calledOn) {
24.        this.calledOn = calledOn;
25.    }
26.    public int getDuration() {
27.        return duration;
28.    }
29.    public void setDuration(int duration) {
30.        this.duration = duration;
31.    }
32.    @Override
33.    public String toString() {
34.        return "CallDetailsDTO [calledBy=" + calledBy + ", calledTo=" + calledTo + ", calledOn=" +
35.            calledOn + ", duration=" + duration + "];"
36.    }
37. package com.infytel.dto;
38. import java.util.List;
39. public class CustomerDTO {
40.     long phoneNo;
41.     String name;
42.     String email;
43.     public String getEmail() {
```

```
44. return email;
45. }
46. public void setEmail(String email) {
47. this.email = email;
48. }
49. int age;
50. char gender;
51. List<FriendFamilyDTO> friendAndFamily;
52. String password;
53. String address;
54. PlanDTO currentPlan;
55. public PlanDTO getCurrentPlan() {
56. return currentPlan;
57. }
58. public void setCurrentPlan(PlanDTO currentPlan) {
59. this.currentPlan = currentPlan;
60. }
61. public String getPassword() {
62. return password;
63. }
64. public void setPassword(String password) {
65. this.password = password;
66. }
67. public String getAddress() {
68. return address;
69. }
70. public void setAddress(String address) {
71. this.address = address;
72. }
73. public List<FriendFamilyDTO> getFriendAndFamily() {
74. return friendAndFamily;
75. }
76. public void setFriendAndFamily(List<FriendFamilyDTO> friendAndFamily) {
77. this.friendAndFamily = friendAndFamily;
78. }
79. public long getPhoneNo() {
80. return phoneNo;
81. }
82. public void setPhoneNo(long phoneNo) {
83. this.phoneNo = phoneNo;
84. }
85. public String getName() {
```

```
86. return name;
87. }
88. public void setName(String name) {
89.     this.name = name;
90. }
91. public int getAge() {
92.     return age;
93. }
94. public void setAge(int age) {
95.     this.age = age;
96. }
97. public char getGender() {
98.     return gender;
99. }
100.     public void setGender(char gender) {
101.         this.gender = gender;
102.     }
103.     @Override
104.     public String toString() {
105.         return "CustomerDTO [phoneNo=" + phoneNo + ", name=" + name + ", age=" + age + ",
            gender=" + gender + ", friendAndFamily=" + friendAndFamily + ", password=" + password + ",
            address=" + address + "]";
106.     }
107. }
```

```
1. package com.infytel.dto;
2. import java.util.ArrayList;
3. import java.util.List;
4. import javax.xml.bind.annotation.XmlAnyElement;
5. import javax.xml.bind.annotation.XmlRootElement;
6. import javax.xml.bind.annotation.XmlSeeAlso;
7. @XmlRootElement
8. @XmlSeeAlso({ PlanDTO.class })
9. public class EntityList<T> {
10.     private List<T> listOfEntityObjects;
11.     public EntityList() {
12.         listOfEntityObjects = new ArrayList<>();
13.     }
14.     public EntityList(List<T> listOfEntityObjects) {
15.         this.listOfEntityObjects = listOfEntityObjects;
16.     }
17. @XmlAnyElement
```

```
18. public List<T> getItems() {  
19. return listOfEntityObjects;  
20. }  
21. }
```

```
1. package com.infytel.dto;  
2. public class ErrorMessage {  
3. private int errorCode;  
4. private String message;  
5. public int getErrorCode() {  
6. return errorCode;  
7. }  
8. public void setErrorCode(int errorCode) {  
9. this.errorCode = errorCode;  
10. }  
11. public String getMessage() {  
12. return message;  
13. }  
14. public void setMessage(String message) {  
15. this.message = message;  
16. }  
17. }
```

```
1. package com.infytel.dto;  
2. public class FriendFamilyDTO {  
3. long phoneNo;  
4. long friendAndFamily;  
5. public long getPhoneNo() {  
6. return phoneNo;  
7. }  
8. public void setPhoneNo(long phoneNo) {  
9. this.phoneNo = phoneNo;  
10. }  
11. public long getFriendAndFamily() {  
12. return friendAndFamily;  
13. }  
14. public void setFriendAndFamily(long friendAndFamily) {  
15. this.friendAndFamily = friendAndFamily;  
16. }  
17. public FriendFamilyDTO(long phoneNo, long friendAndFamily) {  
18. this();
```

```
19. this.phoneNo = phoneNo;
20. this.friendAndFamily = friendAndFamily;
21. }
22. public FriendFamilyDTO() {
23. super();
24. }
25. @Override
26. public String toString() {
27. return "FriendFamilyDTO [phoneNo=" + phoneNo + ", friendAndFamily=" + friendAndFamily +
    "]" ;
28. }
29. }
```

```
1. package com.infytel.dto;
2. import javax.xml.bind.annotation.XmlRootElement;
3. @XmlRootElement
4. public class PlanDTO {
5. Integer planId;
6. String planName;
7. Integer nationalRate;
8. Integer localRate;
9. public Integer getPlanId() {
10. return planId;
11. }
12. public void setPlanId(Integer planId) {
13. this.planId = planId;
14. }
15. public String getPlanName() {
16. return planName;
17. }
18. public void setPlanName(String planName) {
19. this.planName = planName;
20. }
21. public Integer getNationalRate() {
22. return nationalRate;
23. }
24. public void setNationalRate(Integer nationalRate) {
25. this.nationalRate = nationalRate;
26. }
27. public Integer getLocalRate() {
28. return localRate;
29. }
```

```
30. public void setLocalRate(Integer localRate) {
31. this.localRate = localRate;
32. }
33. public PlanDTO() {
34. super();
35. }
36. @Override
37. public String toString() {
38. return "PlanDTO [planId=" + planId + ", planName=" + planName + ", nationalRate=" +
    nationalRate + ", localRate=" + localRate + "]";
39. }
40. }
```

**Step6:** Create classes **ExceptionHandlerAdvice** and **NoSuchCustomerException** under com.infytel.exceptions package:

```
1. package com.infytel.exceptions;
2. import org.springframework.http.HttpStatus;
3. import org.springframework.http.ResponseEntity;
4. import org.springframework.web.bind.annotation.ExceptionHandler;
5. import org.springframework.web.bind.annotation.RestControllerAdvice;
6. import com.infytel.dto.ErrorMessage;
7. @RestControllerAdvice
8. public class ExceptionControllerAdvice {
9. @ExceptionHandler(Exception.class)
10. public String exceptionHandler(Exception ex) {
11. return ex.getMessage();
12. }
13. @ExceptionHandler(NoSuchCustomerException.class)
14. public ResponseEntity<ErrorMessage> exceptionHandler2(NoSuchCustomerException ex) {
15. ErrorMessage error = new ErrorMessage();
16. error.setErrorCode(HttpStatus.BAD_REQUEST.value());
17. error.setMessage(ex.getMessage());
18. return new ResponseEntity<>(error, HttpStatus.OK);
19. }
20. }
```

```
1. package com.infytel.exceptions;
2. public class NoSuchCustomerException extends Exception {
3. private static final long serialVersionUID = 1L;
4. public NoSuchCustomerException() {
5. super();
6. }
```



```
7. public NoSuchCustomerException(String errors) {  
8.     super(errors);  
9. }  
10. }
```

**Step7:** Create classes **CallDetailsRepository**, **CustomerRepository**, **PlanRepository** under com.infytel.repository package

```
1. package com.infytel.repository;  
2. import java.time.LocalDate;  
3. import java.util.ArrayList;  
4. import java.util.List;  
5. import javax.annotation.PostConstruct;  
6. import org.springframework.stereotype.Repository;  
7. import com.infytel.dto.CallDetailsDTO;  
8. @Repository  
9. public class CallDetailsRepository {  
10.     List<CallDetailsDTO> callDetails = null;  
11.     CallDetailsDTO callDetailsDTO = null;  
12.     CallDetailsDTO callDetailsDTO1 = null;  
13.     LocalDate calledOn = null;  
14.     // populates call details in hard-coded way  
15.     @PostConstruct  
16.     public void populatecalledOn() {  
17.         callDetails = new ArrayList<>();  
18.         callDetailsDTO = new CallDetailsDTO();  
19.         callDetailsDTO1 = new CallDetailsDTO();  
20.         calledOn = LocalDate.now();  
21.         callDetailsDTO.setCalledBy(88701064651);  
22.         callDetailsDTO.setCalledTo(99305084951);  
23.         callDetailsDTO.setCalledOn(calledOn);  
24.         callDetailsDTO.setDuration(3);  
25.         callDetailsDTO1.setCalledBy(88701064651);  
26.         callDetailsDTO1.setCalledTo(99305084951);  
27.         callDetailsDTO1.setCalledOn(calledOn);  
28.         callDetailsDTO1.setDuration(5);  
29.         callDetails.add(callDetailsDTO);  
30.         callDetails.add(callDetailsDTO1);  
31.     }  
32.     // fetching call details based on calledBy and calledOn attributes  
33.     public List<CallDetailsDTO> fetchCallDetails(long calledBy, LocalDate calledOn) {  
34.         List<CallDetailsDTO> callDetailsResultSet = new ArrayList<>();  
35.         for (CallDetailsDTO callDetail : callDetails) {  
36.             if (callDetail.getCalledBy() == calledBy && callDetail.getCalledOn().equals(calledOn))
```

```
37. callDetailsResultSet.add(callDetail);
38. }
39. return callDetailsResultSet;
40. }
41. }
```

```
1. package com.infytel.repository;
2. import java.util.ArrayList;
3. import java.util.List;
4. import javax.annotation.PostConstruct;
5. import org.springframework.stereotype.Repository;
6. import com.infytel.dto.CustomerDTO;
7. import com.infytel.dto.FriendFamilyDTO;
8. import com.infytel.dto.PlanDTO;
9. import com.infytel.exceptions.NoSuchCustomerException;
10. @Repository
11. public class CustomerRepository {
12.     List<CustomerDTO> customers = null;
13.     // Populates customer in hard-coded way
14.     @PostConstruct
15.     public void initializer() {
16.         CustomerDTO customerDTO = new CustomerDTO();
17.         PlanDTO planDTO = new PlanDTO();
18.         planDTO.setPlanId(1);
19.         planDTO.setPlanName("Simple");
20.         planDTO.setLocalRate(3);
21.         planDTO.setNationalRate(5);
22.         customerDTO.setAddress("Chennai");
23.         customerDTO.setAge(18);
24.         customerDTO.setCurrentPlan(planDTO);
25.         customerDTO.setGender('m');
26.         customerDTO.setName("Jack");
27.         customerDTO.setEmail("Jack@infy.com");
28.         customerDTO.setPassword("ABC@123");
29.         customerDTO.setPhoneNo(9951212222l);
30.         List<FriendFamilyDTO> friendAndFamily = new ArrayList<>();
31.         friendAndFamily.add(new FriendFamilyDTO(customerDTO.getPhoneNo(), 800000145));
32.         friendAndFamily.add(new FriendFamilyDTO(customerDTO.getPhoneNo(), 700000145));
33.         customerDTO.setFriendAndFamily(friendAndFamily);
34.         customers = new ArrayList<>();
35.         customers.add(customerDTO);
36.     }
```

```
37. // creates customer
38. public String createCustomer(CustomerDTO customerDTO) {
39. customers.add(customerDTO);
40. return "Customer with" + customerDTO.getPhoneNo() + "added successfully";
41. }
42. // fetches customer
43. public List<CustomerDTO> fetchCustomer() {
44. return customers;
45. }
46. // deletes customer - exception handling incorporated
47. public String deleteCustomer(long phoneNumber) throws NoSuchCustomerException {
48. boolean notfound = true;
49. String response = "Customer of:" + phoneNumber + "\t does not exist";
50. for (CustomerDTO customer : customers) {
51. if (customer.getPhoneNo() == phoneNumber) {
52. customers.remove(customer);
53. response = customer.getName() + " with phoneNumber " + customer.getPhoneNo() + " deleted
    successfully";
54. notfound = false;
55. break;
56. }
57. }
58. if (notfound)
59. throw new NoSuchCustomerException("Customer does not exist :" + phoneNumber);
60. return response;
61. }
62. // updates customer
63. public String updateCustomer(long phoneNumber, CustomerDTO customerDTO) {
64. String response = "Customer of:" + phoneNumber + "\t does not exist";
65. for (CustomerDTO customer : customers) {
66. if (customer.getPhoneNo() == phoneNumber) {
67. if (customerDTO.getName() != null)
68. customer.setName(customerDTO.getName());
69. if (customerDTO.getAddress() != null)
70. customer.setAddress(customerDTO.getAddress());
71. if (customerDTO.getPassword() != null)
72. customer.setPassword(customerDTO.getPassword());
73. customers.set(customers.indexOf(customer), customer);
74. response = "Customer of phoneNumber" + customer.getPhoneNo() + "\t got updated successfully";
75. break;
76. }
77. }
```

```
78. return response;
79. }
80. }
```

```
1. package com.infytel.repository;
2. import java.util.ArrayList;
3. import java.util.Iterator;
4. import java.util.List;
5. import javax.annotation.PostConstruct;
6. import org.springframework.stereotype.Repository;
7. import com.infytel.dto.PlanDTO;
8. @Repository
9. public class PlanRepository {
10. private List<PlanDTO> plans;
11. // Populating a list of plans
12. @PostConstruct
13. public void populatePlans() {
14. plans = new ArrayList<>();
15. PlanDTO plan1 = new PlanDTO();
16. plan1.setPlanId(1);
17. plan1.setPlanName("Simple");
18. plan1.setLocalRate(3);
19. plan1.setNationalRate(5);
20. plans.add(plan1);
21. PlanDTO plan2 = new PlanDTO();
22. plan2.setPlanId(2);
23. plan2.setPlanName("Medium");
24. plan2.setLocalRate(5);
25. plan2.setNationalRate(8);
26. plans.add(plan2);
27. }
28. // fetching all available plans
29. public List<PlanDTO> fetchPlans() {
30. return plans;
31. }
32. // fetching plans based on localRate
33. public List<PlanDTO> plansLocalRate(List localRates) {
34. List<PlanDTO> plansResponse = new ArrayList<>();
35. Iterator it = localRates.iterator();
36. while (it.hasNext()) {
37. int rate = Integer.parseInt((String) it.next());
38. for (PlanDTO plan : plans) {
```

```
39. if (rate == plan.getLocalRate())
40. plansResponse.add(plan);
41. }
42. }
43. return plansResponse;
44. }
45. }
```

**Step8:** Create classes **CallDetailsService**, **CustomerService** and **PlanService** under **com.infytel.service** package:

```
1. package com.infytel.service;
2. import java.time.LocalDate;
3. import java.util.List;
4. import org.springframework.beans.factory.annotation.Autowired;
5. import org.springframework.stereotype.Service;
6. import com.infytel.dto.CallDetailsDTO;
7. import com.infytel.repository.CallDetailsRepository;
8. @Service
9. public class CallDetailsService {
10. @Autowired
11. private CallDetailsRepository callDetailsRepository;
12. // contacts repository to fetch the call details
13. public List<CallDetailsDTO> fetchCallDetails(long calledBy, LocalDate calledOn) {
14. return callDetailsRepository.fetchCallDetails(calledBy, calledOn);
15. }
16. }
```

```
1. package com.infytel.service;
2. import java.util.List;
3. import org.springframework.beans.factory.annotation.Autowired;
4. import org.springframework.stereotype.Service;
5. import com.infytel.dto.CustomerDTO;
6. import com.infytel.exceptions.NoSuchCustomerException;
7. import com.infytel.repository.CustomerRepository;
8. @Service
9. public class CustomerService {
10. @Autowired
11. private CustomerRepository customerRepository;
12. // Contacts repository layer to add customer
13. public String createCustomer(CustomerDTO customerDTO) {
14. return customerRepository.createCustomer(customerDTO);
15. }
```

```
16. // Contacts repository layer to fetch customer
17. public List<CustomerDTO> fetchCustomer() {
18. return customerRepository.fetchCustomer();
19. }
20. // Contacts repository layer to delete customer
21. public String deleteCustomer(long phoneNumber) throws NoSuchCustomerException {
22. return customerRepository.deleteCustomer(phoneNumber);
23. }
24. // Contacts repository layer to update customer
25. public String updateCustomer(long phoneNumber, CustomerDTO customerDTO) {
26. return customerRepository.updateCustomer(phoneNumber, customerDTO);
27. }
28. }
```

```
1. package com.infytel.service;
2. import java.util.List;
3. import org.springframework.beans.factory.annotation.Autowired;
4. import org.springframework.stereotype.Service;
5. import com.infytel.dto.PlanDTO;
6. import com.infytel.repository.PlanRepository;
7. @Service
8. public class PlanService {
9. @Autowired
10. private PlanRepository planRepository;
11. // contacts repository to fetch plans
12. public List<PlanDTO> fetchPlans() {
13. return planRepository.fetchPlans();
14. }
15. // contacts repository to fetch plans by localRates
16. public List<PlanDTO> plansLocalRate(List localRates) {
17. return planRepository.plansLocalRate(localRates);
18. }
19. }
```

**Step 9:** Add the following content to **application.properties** file:

```
1. server.port = 8080
2. server.servlet.context-path=/infytel-6
```

**Step 10:** Make sure that the project's pom.xml looks similar to the one shown below.

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <project xmlns="http://maven.apache.org/POM/4.0.0"
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
4.  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
    4.0.0.xsd">
5.  <modelVersion>4.0.0</modelVersion>
6.  <parent>
7.  <groupId>org.springframework.boot</groupId>
8.  <artifactId>spring-boot-starter-parent</artifactId>
9.  <version>2.1.4.RELEASE</version>
10. <relativePath /> <!-- lookup parent from repository -->
11. </parent>
12. <groupId>infytel</groupId>
13. <artifactId>infytel_demo7</artifactId>
14. <version>0.0.1-SNAPSHOT</version>
15. <name>infytel_demo7</name>
16. <description>Demo project for Spring Boot</description>
17. <properties>
18. <java.version>1.8</java.version>
19. </properties>
20. <dependencies>
21. <dependency>
22. <groupId>org.springframework.boot</groupId>
23. <artifactId>spring-boot-starter-web</artifactId>
24. </dependency>
25. <dependency>
26. <groupId>org.springframework.boot</groupId>
27. <artifactId>spring-boot-starter-test</artifactId>
28. <scope>test</scope>
29. </dependency>
30. </dependencies>
31. <build>
32. <plugins>
33. <plugin>
34. <groupId>org.springframework.boot</groupId>
35. <artifactId>spring-boot-maven-plugin</artifactId>
36. </plugin>
37. </plugins>
38. </build>
39. </project>
```

**Step 11:** Deploy the application on the server by executing the class containing the main method.

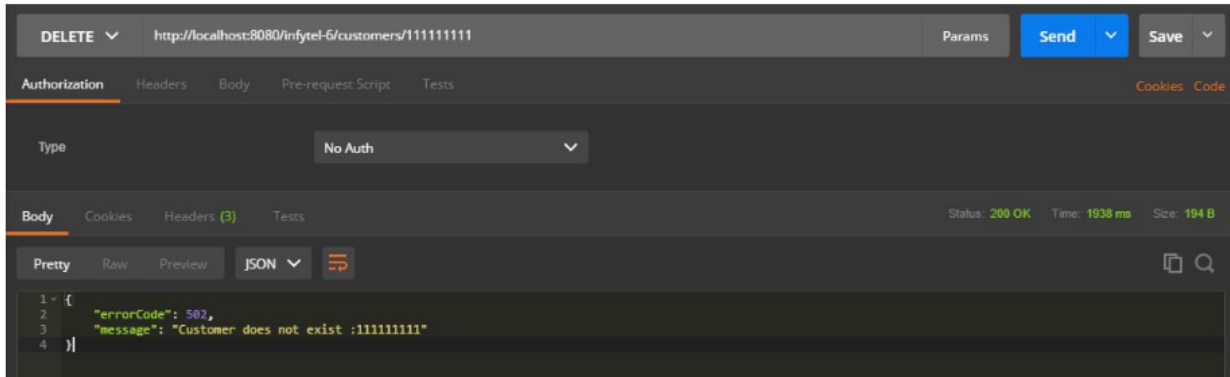
So, we have successfully created and deployed the REST endpoints. Now, let us see how can we test the same using Postman client.

## Output

Testing Web Service using Postman

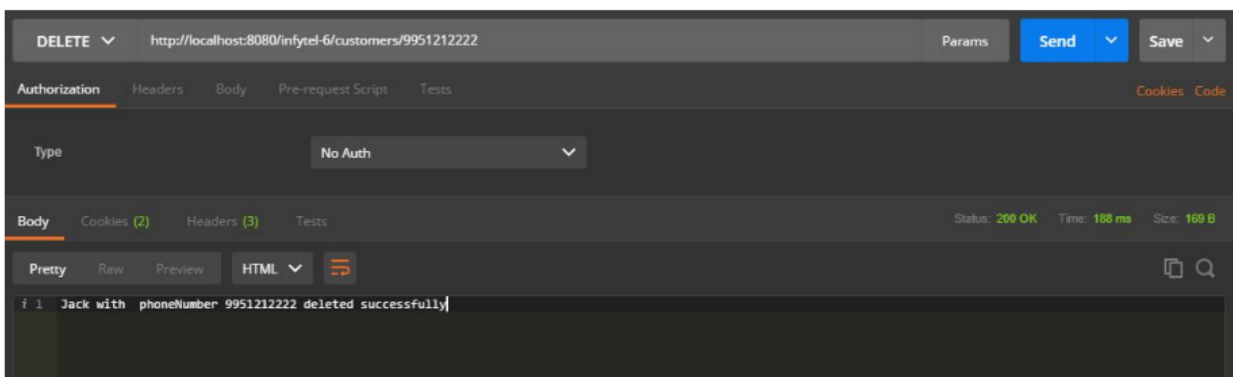
**Step 1:** Launch Postman.

**Step 2:** Test this URL - **http://localhost:8080/infytel-6/customers/111111111** using HTTP DELETE that will delete the customer based on the phone number (invalid phoneNumber that does not exist in Infytel) being passed.



Observe the message that is sent when we try to perform delete operation with an invalid phoneNumber.

**Step 3:** Test this URL - **http://localhost:8080/infytel-6/customers/9951212222** using HTTP DELETE that will delete the customer based on the phone number (valid phoneNumber that exists in Infytel) being passed.



The customer of valid phone number is deleted successfully and the same is intimated to the client back.

## Exercise - Exception Handling

CookPick online grocery application has a requirement where the admin should be able to delete a product based on its product code.

Write a RESTful endpoint that would serve this purpose. Also, handle the code in such a way that it renders a customized error message when the product is not found.

RESTful URL	HTTP Action	Business Operation
/product/{productCode}	DELETE	deleteProduct()

**Method description as follows:**

**public ResponseEntity deleteProduct(long productCode) throws ProductNotFoundException**

This method,



- Takes product code
- Renders a response entity with the appropriate status code and message

**Hint:** Status code-200 and message-“product deleted” if everything is ok

Status code-404 and message-“product is not found” otherwise

**Note:**

1. You can implement this requirement in the project that you used for previous exercise.
2. You can simply hardcode sample product details in a collection and delete the product details from the collection in your logic. It is optional to use a database to delete the product details.

\*Time given for this exercise indicates the time required to create the REST endpoint and is excluding the time required to write the logic for deleting the product details from database.

### Need for Data validation

Some times, the RESTful web service might need data in certain standard that should pass through specific constraints.

### What if the data is not in the format as needed by the RESTful service?

By default, Spring Boot provides some implementation for validation

- 415 - Unsupported Media Type : If the data is sent in a different format other than what is expected(eg:XML instead of JSON)
- 400 - Bad Request : If JSON data is not in proper format, for example.

Some times, we might send valid format and structure, but with missing data. Still, the request gets processed. In such situations, it becomes even more important to validate the data.

**Example:**

1. {
2. //empty JSON data
3. }

Hibernate Validator, is one of the implementations of the bean validation API. Let us see, how to use the Validation API to validate our incoming data.

Bean Validation API provides a number of annotations and most of these annotations are self explanatory.

- Digits
- Email
- Max
- Min
- NotEmpty
- NotNull
- Null
- Pattern

### Applying validation on the incoming data

Let us apply validation on the customer data that comes to the createCustomer() method of CustomerController. Infytel expects the name field not to be null and the email field to be of proper email format.

```
1. public class CustomerDTO {
2.     long phoneNo;
3.     @NotNull
4.     String name;
5.     @Email(message = "Email id is not in format, please check")
6.     String email;
7.     int age;
8.     char gender;
9.     List<FriendFamilyDTO> friendAndFamily;
10.    String password;
11.    String address;
12.    PlanDTO currentPlan;
13.    // getter and setters go here
14. }
```

Have a look at the usage of @NotNull and @Email on the fields, name and email respectively.

Adding the constraints simply on the bean will not carry out validation. In addition, we need to mention that validation is required while the JSON data is deserialized to CustomerDTO and get the validation error, if any, into org.springframework.validation.Errors object. This can be done by applying @Valid annotation on the handler method argument which captures the CustomerDTO object as shown below. We can inject the Errors object too into this method.

```
1. @PostMapping(consumes="application/json")
2. public ResponseEntity createCustomer(@Valid @RequestBody CustomerDTO customerDTO,
    Errors errors)
3. {
4.     String response = "";
5.     if (errors.hasErrors())
6.     {
7.         response = errors.getAllErrors().stream()
8.             .map(ObjectError::getDefaultMessage)
9.             .collect(Collectors.joining(", "));
10.        ErrorMessage error = new ErrorMessage();
11.        error.setErrorCode(HttpStatus.NOT_ACCEPTABLE.value());
12.        error.setMessage(response);
13.        return ResponseEntity.ok(error);
14.    }
15.    else
16.    {
17.        response = customerService.createCustomer(customerDTO);
18.        return ResponseEntity.ok(response);
    }
```

```
19. }  
20. }
```

Let us decode the above snippet.

### Discussion on how the validation takes place

**@Valid @RequestBody CustomerDTO customerDTO** - indicates that CustomerDTO should be validated before being taken.

Errors - indicates the violations, if any, to get stored in Errors object.

Finally, the business logic will be divided into two courses of actions. That is, what should be done when there are no validation errors. And, what should be done, otherwise.

**If there are errors:** Collect all the errors together as a String.

Set the String that contains errors in the message field and appropriate error code in the errorCode field of ErrorMessage.

Finally, return the ResponseEntity that is set with the ErrorMessage object.

```
1. if (errors.hasErrors())  
2. {  
3.     response = errors.getAllErrors().stream().  
4.     map(x->x.getDefaultMessage()).  
5.     collect(Collectors.joining(", "));  
6.     ErrorMessage error = new ErrorMessage();  
7.     error.setErrorCode(HttpStatus.NOT_ACCEPTABLE.value());  
8.     error.setMessage(response);  
9.     return ResponseEntity.ok(error);  
10. }
```

The ErrorMessage class:

```
1. public class ErrorMessage {  
2.     private int errorCode;  
3.     private String message;  
4.     //Getters and Setters goes here  
5. }
```

2. If there are no errors, invoke the service layer to create the customer and return the response back to the client.

```
1. response = customerService.createCustomer(customerDTO);  
2. return ResponseEntity.ok(response);
```

Let us put all these together and look at an example.

## Rich set of annotations

We are done dealing with validating the incoming objects/DTOs. Also, we finished traversing a set of annotations that impose validation on the individual fields/attributes of the DTO.

Apart from the annotations that we discussed, we have few other interesting annotations in place. For example,

**@PastOrPresent**

**@Past**

**@Future**

**@FutureOrPresent**

**@DateTimeFormat** - part of **org.springframework.format.annotation** unlike other annotations being mentioned here which are part of standard **javax.validation.constraints**

**These annotations can only be applied on the fields/attributes of type, date/time, for example, LocalDate/LocalTime.**

One more interesting annotation that needs a mention here is **@Positive** which is used to make sure a number should possess only positive values for example, cost of an article.

Last but not the least to be brought in for discussion is **@NotBlank** which makes sure that the string is not null and the trimmed length should be greater than zero.

**Further Reading :** Explore **javax.validations.constraints** API to get the complete list of validation annotations that can be applied on the fields/attributes.

## URI parameter validation

Like how it is important to validate the incoming objects/DTOs, it is essential to validate the incoming URI parameters as well. For example, when the customer details need to be deleted based on the phone number and if the same is reaching the REST endpoint as URI parameter (path variable, for example), it becomes essential that the URI parameter, phone number should also be validated (should be of 10 digits, for example).

The code snippet given below helps serving the purpose discussed above.

1. `public String deleteCustomer(@PathVariable("phoneNumber")`
2. `@Pattern(regexp = "[0-9]{10}",message="{customer.phoneNo.invalid}")`
3. `String phoneNumber) throws NoSuchCustomerException`

Here, the message that should be rendered during the validation failure need not be hard-coded always. It can be fetched from the external property file as well and the name of the file is **ValidationMessages.properties** that Spring Boot searches for. If the file name is other than **ValidationMessages.properties**, **MessageSource** and **LocalValidatorFactoryBean** need to be configured in the bootstrap class.

One important thing that needs attention while validating the URI parameters is, **@Validated**.

The controller where the URI parameter validation is carried out should be annotated with `@Validated`. The validation on URI parameters will not be triggered, otherwise. Here, the `CustomerController` has validations pertaining to `phoneNo` that is received as URI parameter (DELETE and UPDATE). So, `@Validated` annotation needs to be applied on the same as follows.

1. `@Validated`
2. `public class CustomerController`

### Handling validation failures in a centralized way

So, we know how to apply validation on the incoming objects and URI parameters.

Now, its time to handle the validation failures, if any. We have attempted to handle the validation exceptions with respect to the incoming DTOs in the earlier sample code. But, that was not in a centralized way. Only, the controller was coded to do so. This way of coding will lead to code duplication and the code will not be manageable as well.

So, the code that should be reached out during validation failures should also be made centralized as how we did for exceptions. We can enhance the **`ExceptionHandlerAdvice`** class which is annotated with `@RestControllerAdvice` by adding exception handlers for

- `MethodArgumentNotValidException` - Validation failures in DTOs
- `ConstraintViolationException` - Validation failures in URI parameters

Have a look at the code snippet below to understand the concept better.

```
1. //validation failures on DTOs
2. @ExceptionHandler(MethodArgumentNotValidException.class)
3. public ResponseEntity<ErrorMessage> handleValidationExceptions(
4.     MethodArgumentNotValidException ex) {
5.     ErrorMessage error = new ErrorMessage();
6.     error.setErrorCode(HttpStatus.BAD_REQUEST.value());
7.     error.setMessage(ex.getBindingResult().getAllErrors().stream()
8.         .map(ObjectError::getDefaultMessage)
9.         .collect(Collectors.joining(", ")));
10.    return new ResponseEntity<>(error, HttpStatus.BAD_REQUEST);
11. }
12. //Validation failures on URI parameters
13. @ExceptionHandler(ConstraintViolationException.class)
14. public ResponseEntity<ErrorMessage> handleConstraintValidationExceptions(
15.     ConstraintViolationException ex) {
16.     ErrorMessage error = new ErrorMessage();
17.     error.setErrorCode(HttpStatus.BAD_REQUEST.value());
18.     error.setMessage(ex.getConstraintViolations().stream()
19.         .map(ConstraintViolation::getMessage)
20.         .collect(Collectors.joining(", ")));
21.    return new ResponseEntity<>(error, HttpStatus.BAD_REQUEST);
22. }
```

### Demo-7: Validating Input Data

**Objectives:** To create a Spring REST controller, where the data being received is first validated, and then processed. If the data being received is not of the expected specifications, a Custom Error Message will be sent by the controller.

**Scenario:** An online telecom app called Infytel wishes its functionalities to get exposed as RESTful services. It has three controllers in total to deal with customer, plan and call details, among which, the focus will be on the controller named CustomerController that has a service method that validates the incoming data and sends customized error message to the client back, during validation failures. Here follows the method details,

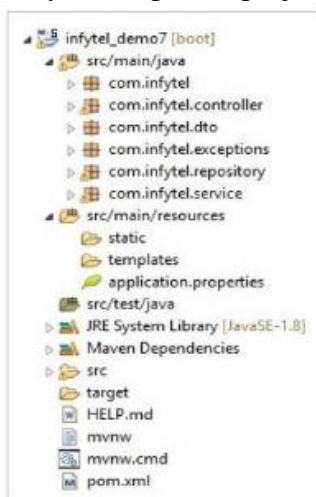
Controller class	method	URI	HTTP Method	Remarks
CustomerController	createCustomer	/customers	POST	This will create a customer in Infytel provided, the email address is in proper email format and the name is not null as well.

- CustomerDTO has used @Email and @NotNull annotations
- The special ErrorMessage POJO class

### Steps:

**Step 1:** Create a Maven project using Spring Initializer with web dependency and import the same in STS.

**Step 2:** Modify the imported project according to the following project structure:



**Step 3:** Look at the class **InfytelDemo7Application** under **com.infytel** package that gets created automatically when the project is generated. We will modify this only when we need to go with custom configuration. This class holds the code for enabling matrix variables as extra configuration.

```

1. package com.infytel;
2. import org.springframework.boot.SpringApplication;
3. import org.springframework.boot.autoconfigure.SpringBootApplication;
4. import org.springframework.web.servlet.config.annotation.PathMatchConfigurer;
5. import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
6. import org.springframework.web.util.UrlPathHelper;
7. @SpringBootApplication
8. public class InfytelDemo7Application implements WebMvcConfigurer {
9.     public static void main(String[] args) {

```

```
10. SpringApplication.run(InfytelDemo7Application.class, args);
11. }
12. // To support matrix parameters
13. @Override
14. public void configurePathMatch(PathMatchConfigurer configurer) {
15.     UrlPathHelper urlPathHelper = new UrlPathHelper();
16.     urlPathHelper.setRemoveSemicolonContent(false);
17.     configurer.setUrlPathHelper(urlPathHelper);
18. }
19. }
```

**Step 4:** Create classes **CallDetailsController**, **CustomerController** and **PlanController** under **com.infytel.controller** package:

```
1. package com.infytel.controller;
2. import java.time.LocalDate;
3. import java.time.format.DateTimeFormatter;
4. import java.util.List;
5. import org.springframework.beans.factory.annotation.Autowired;
6. import org.springframework.web.bind.annotation.GetMapping;
7. import org.springframework.web.bind.annotation.RequestMapping;
8. import org.springframework.web.bind.annotation.RequestParam;
9. import org.springframework.web.bind.annotation.RestController;
10. import com.infytel.dto.CallDetailsDTO;
11. import com.infytel.service.CallDetailsService;
12. @RestController
13. @RequestMapping("/calldetails")
14. public class CallDetailsController {
15.     @Autowired
16.     private CallDetailsService callDetailsService;
17.     // Fetching call details based on the request parameters being passed along with
18.     // the URI
19.     @GetMapping(produces = "application/json")
20.     public List<CallDetailsDTO> callDetails(@RequestParam("calledBy") long calledBy,
21.     @RequestParam("calledOn") String calledOn) {
22.         return callDetailsService.fetchCallDetails(calledBy,
23.         LocalDate.parse(calledOn, DateTimeFormatter.ofPattern("MM-dd-yyyy")));
24.     }
25. }
```

```
1. package com.infytel.controller;
2. import java.util.List;
3. import java.util.stream.Collectors;
```

```
4. import javax.validation.Valid;
5. import org.springframework.beans.factory.annotation.Autowired;
6. import org.springframework.http.HttpStatus;
7. import org.springframework.http.ResponseEntity;
8. import org.springframework.validation.Errors;
9. import org.springframework.validation.ObjectError;
10. import org.springframework.web.bind.annotation.DeleteMapping;
11. import org.springframework.web.bind.annotation.GetMapping;
12. import org.springframework.web.bind.annotation.PathVariable;
13. import org.springframework.web.bind.annotation.PostMapping;
14. import org.springframework.web.bind.annotation.PutMapping;
15. import org.springframework.web.bind.annotation.RequestBody;
16. import org.springframework.web.bind.annotation.RequestMapping;
17. import org.springframework.web.bind.annotation.RestController;
18. import com.infytel.dto.CustomerDTO;
19. import com.infytel.dto.ErrorMessage;
20. import com.infytel.exceptions.NoSuchCustomerException;
21. import com.infytel.service.CustomerService;
22. @RestController
23. @RequestMapping("/customers")
24. public class CustomerController {
25.     @Autowired
26.     private CustomerService customerService;
27.     // Fetching customer details
28.     @GetMapping(produces = "application/json")
29.     public List<CustomerDTO> fetchCustomer() {
30.         return customerService.fetchCustomer();
31.     }
32.     // Adding a customer
33.     @PostMapping(consumes = "application/json")
34.     public ResponseEntity createCustomer(@Valid @RequestBody CustomerDTO customerDTO,
        Errors errors) {
35.         String response = "";
36.         if (errors.hasErrors()) {
37.             // collecting the validation errors of all fields together in a String delimited
38.             // by commas
39.             response = errors.getAllErrors().stream().map(ObjectError::getDefaultMessage)
40.                 .collect(Collectors.joining(", "));
41.             ErrorMessage error = new ErrorMessage();
42.             error.setErrorCode(HttpStatus.NOT_ACCEPTABLE.value());
43.             error.setMessage(response);
44.             return ResponseEntity.ok(error);
```



```
45. } else {
46. response = customerService.createCustomer(customerDTO);
47. return ResponseEntity.ok(response);
48. }
49. }
50. // Updating an existing customer
51. @PutMapping(value =("/{phoneNumber}", consumes = "application/json")
52. public String updateCustomer(@PathVariable("phoneNumber") long phoneNumber,
    @RequestBody CustomerDTO customerDTO) {
53. return customerService.updateCustomer(phoneNumber, customerDTO);
54. }
55. // Deleting a customer
56. @DeleteMapping(value =("/{phoneNumber}", produces = "text/html")
57. public String deleteCustomer(@PathVariable("phoneNumber") long phoneNumber) throws
    NoSuchCustomerException {
58. return customerService.deleteCustomer(phoneNumber);
59. }
60. }
```

```
1. package com.infytel.controller;
2. import java.util.ArrayList;
3. import java.util.List;
4. import java.util.Map;
5. import java.util.Set;
6. import org.springframework.beans.factory.annotation.Autowired;
7. import org.springframework.web.bind.annotation.GetMapping;
8. import org.springframework.web.bind.annotation.PathVariable;
9. import org.springframework.web.bind.annotation.RequestMapping;
10. import org.springframework.web.bind.annotation.RestController;
11. import com.infytel.dto.EntityList;
12. import com.infytel.dto.PlanDTO;
13. import com.infytel.service.PlanService;
14. @RestController
15. @RequestMapping("/plans")
16. public class PlanController {
17. private EntityList<PlanDTO> plans;
18. @Autowired
19. private PlanService planService;
20. // Get all available plans
21. @GetMapping(produces = { "application/json", "application/xml" })
22. public EntityList<PlanDTO> fetchPlans() {
23. plans = new EntityList<>(planService.fetchPlans());
```

```
24. return plans;
25. }
26. // Gets plans based on localRate
27. @GetMapping(value = "{query}/plan", produces = { "application/json", "application/xml" })
28. public EntityList<PlanDTO> plansLocalRate(@MatrixVariable(pathVar = "query") Map<String,
    List<Integer>> map) {
29. Set<String> keysLocalRates = map.keySet();
30. ArrayList localRates = new ArrayList();
31. for (String key : keysLocalRates) {
32. for (int i = 0; i < map.get(key).size(); i++) {
33. localRates.add(map.get(key).get(i));
34. }
35. }
36. plans = new EntityList<>(planService.plansLocalRate(localRates));
37. return plans;
38. }
39. }
```

**Step5:** Create classes **CallDetailsDTO**, **CustomerDTO**, **EntityList**, **ErrorMessage**, **FriendFamilyDTO** and **PlanDTO** under com.infytel.dto package:

```
1. package com.infytel.dto;
2. import java.time.LocalDate;
3. public class CallDetailsDTO {
4. long calledBy;
5. long calledTo;
6. LocalDate calledOn;
7. int duration;
8. public long getCalledBy() {
9. return calledBy;
10. }
11. public void setCalledBy(long calledBy) {
12. this.calledBy = calledBy;
13. }
14. public long getCalledTo() {
15. return calledTo;
16. }
17. public void setCalledTo(long calledTo) {
18. this.calledTo = calledTo;
19. }
20. public LocalDate getCalledOn() {
21. return calledOn;
22. }
```

```
23. public void setCalledOn(LocalDate calledOn) {
24. this.calledOn = calledOn;
25. }
26. public int getDuration() {
27. return duration;
28. }
29. public void setDuration(int duration) {
30. this.duration = duration;
31. }
32. @Override
33. public String toString() {
34. return "CallDetailsDTO [calledBy=" + calledBy + ", calledTo=" + calledTo + ", calledOn=" +
    calledOn + ", duration=" + duration + "]";
35. }
36. }
```

```
1. package com.infytel.dto;
2. import java.util.List;
3. import javax.validation.constraints.Email;
4. import javax.validation.constraints.NotNull;
5. public class CustomerDTO {
6. long phoneNo;
7. @NotNull(message = "Name cannot be empty")
8. String name;
9. @Email(message = "Email id is not in format, please check")
10. String email;
11. public String getEmail() {
12. return email;
13. }
14. public void setEmail(String email) {
15. this.email = email;
16. }
17. int age;
18. char gender;
19. List<FriendFamilyDTO> friendAndFamily;
20. String password;
21. String address;
22. PlanDTO currentPlan;
23. public PlanDTO getCurrentPlan() {
24. return currentPlan;
25. }
26. public void setCurrentPlan(PlanDTO currentPlan) {
```

```
27. this.currentPlan = currentPlan;
28. }
29. public String getPassword() {
30. return password;
31. }
32. public void setPassword(String password) {
33. this.password = password;
34. }
35. public String getAddress() {
36. return address;
37. }
38. public void setAddress(String address) {
39. this.address = address;
40. }
41. public List<FriendFamilyDTO> getFriendAndFamily() {
42. return friendAndFamily;
43. }
44. public void setFriendAndFamily(List<FriendFamilyDTO> friendAndFamily) {
45. this.friendAndFamily = friendAndFamily;
46. }
47. public long getPhoneNo() {
48. return phoneNo;
49. }
50. public void setPhoneNo(long phoneNo) {
51. this.phoneNo = phoneNo;
52. }
53. public String getName() {
54. return name;
55. }
56. public void setName(String name) {
57. this.name = name;
58. }
59. public int getAge() {
60. return age;
61. }
62. public void setAge(int age) {
63. this.age = age;
64. }
65. public char getGender() {
66. return gender;
67. }
68. public void setGender(char gender) {
```

```
69. this.gender = gender;
70. }
71. @Override
72. public String toString() {
73. return "CustomerDTO [phoneNo=" + phoneNo + ", name=" + name + ", age=" + age + ", gender="
    + gender + ", friendAndFamily=" + friendAndFamily + ", password=" + password + ", address=" +
    address + "]";
74. }
75. }
```

```
1. package com.infytel.dto;
2. import java.util.ArrayList;
3. import java.util.List;
4. import javax.xml.bind.annotation.XmlAnyElement;
5. import javax.xml.bind.annotation.XmlRootElement;
6. import javax.xml.bind.annotation.XmlSeeAlso;
7. @XmlRootElement
8. @XmlSeeAlso({ PlanDTO.class })
9. public class EntityList<T> {
10. private List<T> listOfEntityObjects;
11. public EntityList() {
12. listOfEntityObjects = new ArrayList<>();
13. }
14. public EntityList(List<T> listOfEntityObjects) {
15. this.listOfEntityObjects = listOfEntityObjects;
16. }
17. @XmlAnyElement
18. public List<T> getItems() {
19. return listOfEntityObjects;
20. }
21. }
```

```
1. package com.infytel.dto;
2. public class ErrorMessage {
3. private int errorCode;
4. private String message;
5. public int getErrorCode() {
6. return errorCode;
7. }
8. public void setErrorCode(int errorCode) {
9. this.errorCode = errorCode;
```

```
10. }
11. public String getMessage() {
12. return message;
13. }
14. public void setMessage(String message) {
15. this.message = message;
16. }
17. }
```

```
1. package com.infytel.dto;
2. public class FriendFamilyDTO {
3. long phoneNo;
4. long friendAndFamily;
5. public long getPhoneNo() {
6. return phoneNo;
7. }
8. public void setPhoneNo(long phoneNo) {
9. this.phoneNo = phoneNo;
10. }
11. public long getFriendAndFamily() {
12. return friendAndFamily;
13. }
14. public void setFriendAndFamily(long friendAndFamily) {
15. this.friendAndFamily = friendAndFamily;
16. }
17. public FriendFamilyDTO(long phoneNo, long friendAndFamily) {
18. this();
19. this.phoneNo = phoneNo;
20. this.friendAndFamily = friendAndFamily;
21. }
22. public FriendFamilyDTO() {
23. super();
24. }
25. @Override
26. public String toString() {
27. return "FriendFamilyDTO [phoneNo=" + phoneNo + ", friendAndFamily=" + friendAndFamily +
    " ]";
28. }
29. }
```

```
1. package com.infytel.dto;
```

```
2. import javax.xml.bind.annotation.XmlRootElement;
3. @XmlRootElement
4. public class PlanDTO {
5.     Integer planId;
6.     String planName;
7.     Integer nationalRate;
8.     Integer localRate;
9.     public Integer getPlanId() {
10. return planId;
11. }
12. public void setPlanId(Integer planId) {
13. this.planId = planId;
14. }
15. public String getPlanName() {
16. return planName;
17. }
18. public void setPlanName(String planName) {
19. this.planName = planName;
20. }
21. public Integer getNationalRate() {
22. return nationalRate;
23. }
24. public void setNationalRate(Integer nationalRate) {
25. this.nationalRate = nationalRate;
26. }
27. public Integer getLocalRate() {
28. return localRate;
29. }
30. public void setLocalRate(Integer localRate) {
31. this.localRate = localRate;
32. }
33. public PlanDTO() {
34. super();
35. }
36. @Override
37. public String toString() {
38. return "PlanDTO [planId=" + planId + ", planName=" + planName + ", nationalRate=" +
    nationalRate + ", localRate=" + localRate + "]\n";
39. }
40. }
```

**Step6:** Create classes **ExceptionHandlerAdvice** and **NoSuchCustomerException** under **com.infytel.exception** package:

```
1. package com.infytel.exceptions;
2. import org.springframework.http.HttpStatus;
3. import org.springframework.http.ResponseEntity;
4. import org.springframework.web.bind.annotation.ExceptionHandler;
5. import org.springframework.web.bind.annotation.RestControllerAdvice;
6. import com.infytel.dto.ErrorMessage;
7. @RestControllerAdvice
8. public class ExceptionControllerAdvice {
9.     @ExceptionHandler(Exception.class)
10.    public String exceptionHandler(Exception ex) {
11.        return ex.getMessage();
12.    }
13.    @ExceptionHandler(NoSuchCustomerException.class)
14.    public ResponseEntity<ErrorMessage> exceptionHandler2(NoSuchCustomerException ex) {
15.        ErrorMessage error = new ErrorMessage();
16.        error.setErrorCode(HttpStatus.BAD_REQUEST.value());
17.        error.setMessage(ex.getMessage());
18.        return new ResponseEntity<>(error, HttpStatus.OK);
19.    }
20. }
```

```
1. package com.infytel.exceptions;
2. public class NoSuchCustomerException extends Exception {
3.     private static final long serialVersionUID = 1L;
4.     public NoSuchCustomerException() {
5.         super();
6.     }
7.     public NoSuchCustomerException(String errors) {
8.         super(errors);
9.     }
10. }
```

**Step 7:** Create classes **CallDetailsRepository**, **CustomerRepository** and **PlanRepository** under **com.infytel.repository** package:

```
1. package com.infytel.repository;
2. import java.time.LocalDate;
3. import java.util.ArrayList;
4. import java.util.List;
5. import javax.annotation.PostConstruct;
6. import org.springframework.stereotype.Repository;
```



```
7. import com.infytel.dto.CallDetailsDTO;
8. @Repository
9. public class CallDetailsRepository {
10. List<CallDetailsDTO> callDetails = null;
11. CallDetailsDTO callDetailsDTO = null;
12. CallDetailsDTO callDetailsDTO1 = null;
13. LocalDate calledOn = null;
14. // populates call details in hard-coded way
15. @PostConstruct
16. public void populatecalledOn() {
17. callDetails = new ArrayList<>();
18. callDetailsDTO = new CallDetailsDTO();
19. callDetailsDTO1 = new CallDetailsDTO();
20. calledOn = LocalDate.now();
21. callDetailsDTO.setCalledBy(88701064651);
22. callDetailsDTO.setCalledTo(99305084951);
23. callDetailsDTO.setCalledOn(calledOn);
24. callDetailsDTO.setDuration(3);
25. callDetailsDTO1.setCalledBy(88701064651);
26. callDetailsDTO1.setCalledTo(99305084951);
27. callDetailsDTO1.setCalledOn(calledOn);
28. callDetailsDTO1.setDuration(5);
29. callDetails.add(callDetailsDTO);
30. callDetails.add(callDetailsDTO1);
31. }
32. // fetching call details based on calledBy and calledOn attributes
33. public List<CallDetailsDTO> fetchCallDetails(long calledBy, LocalDate calledOn) {
34. List<CallDetailsDTO> callDetailsResultSet = new ArrayList<>();
35. for (CallDetailsDTO callDetail : callDetails) {
36. if (callDetail.getCalledBy() == calledBy && callDetail.getCalledOn().equals(calledOn))
37. callDetailsResultSet.add(callDetail);
38. }
39. return callDetailsResultSet;
40. }
41. }
```

```
1. package com.infytel.repository;
2. import java.util.ArrayList;
3. import java.util.List;
4. import javax.annotation.PostConstruct;
5. import org.springframework.stereotype.Repository;
6. import com.infytel.dto.CustomerDTO;
```

```
7. import com.infytel.dto.FriendFamilyDTO;
8. import com.infytel.dto.PlanDTO;
9. import com.infytel.exceptions.NoSuchCustomerException;
10. @Repository
11. public class CustomerRepository {
12. List<CustomerDTO> customers = null;
13. // Populates customer in hard-coded way
14. @PostConstruct
15. public void initializer() {
16. CustomerDTO customerDTO = new CustomerDTO();
17. PlanDTO planDTO = new PlanDTO();
18. planDTO.setPlanId(1);
19. planDTO.setPlanName("Simple");
20. planDTO.setLocalRate(3);
21. planDTO.setNationalRate(5);
22. customerDTO.setAddress("Chennai");
23. customerDTO.setAge(18);
24. customerDTO.setCurrentPlan(planDTO);
25. customerDTO.setGender('m');
26. customerDTO.setName("Jack");
27. customerDTO.setEmail("Jack@infy.com");
28. customerDTO.setPassword("ABC@123");
29. customerDTO.setPhoneNo(99512122221);
30. List<FriendFamilyDTO> friendAndFamily = new ArrayList<>();
31. friendAndFamily.add(new FriendFamilyDTO(customerDTO.getPhoneNo(), 800000145));
32. friendAndFamily.add(new FriendFamilyDTO(customerDTO.getPhoneNo(), 700000145));
33. customerDTO.setFriendAndFamily(friendAndFamily);
34. customers = new ArrayList<>();
35. customers.add(customerDTO);
36. }
37. // creates customer
38. public String createCustomer(CustomerDTO customerDTO) {
39. customers.add(customerDTO);
40. return "Customer with" + customerDTO.getPhoneNo() + "added successfully";
41. }
42. // fetches customer
43. public List<CustomerDTO> fetchCustomer() {
44. return customers;
45. }
46. // deletes customer - exception handling incorporated
47. public String deleteCustomer(long phoneNumber) throws NoSuchCustomerException {
48. boolean notfound = true;
```

```
49. String response = "Customer of:" + phoneNumber + "\t does not exist";
50. for (CustomerDTO customer : customers) {
51. if (customer.getPhoneNo() == phoneNumber) {
52. customers.remove(customer);
53. response = customer.getName() + " with  phoneNumber " + customer.getPhoneNo() + " deleted
    successfully";
54. notfound = false;
55. break;
56. }
57. }
58. if (notfound)
59. throw new NoSuchCustomerException("Customer does not exist :" + phoneNumber);
60. return response;
61. }
62. // updates customer
63. public String updateCustomer(long phoneNumber, CustomerDTO customerDTO) {
64. String response = "Customer of:" + phoneNumber + "\t does not exist";
65. for (CustomerDTO customer : customers) {
66. if (customer.getPhoneNo() == phoneNumber) {
67. if (customerDTO.getName() != null)
68. customer.setName(customerDTO.getName());
69. if (customerDTO.getAddress() != null)
70. customer.setAddress(customerDTO.getAddress());
71. if (customerDTO.getPassword() != null)
72. customer.setPassword(customerDTO.getPassword());
73. customers.set(customers.indexOf(customer), customer);
74. response = "Customer of phoneNumber" + customer.getPhoneNo() + "\t got updated successfully";
75. break;
76. }
77. }
78. return response;
79. }
80. }
```

```
1. package com.infytel.repository;
2. import java.util.ArrayList;
3. import java.util.Iterator;
4. import java.util.List;
5. import javax.annotation.PostConstruct;
6. import org.springframework.stereotype.Repository;
7. import com.infytel.dto.PlanDTO;
8. @Repository
```

```
9. public class PlanRepository {
10. private List<PlanDTO> plans;
11. // Populating a list of plans
12. @PostConstruct
13. public void populatePlans() {
14. plans = new ArrayList<>();
15. PlanDTO plan1 = new PlanDTO();
16. plan1.setPlanId(1);
17. plan1.setPlanName("Simple");
18. plan1.setLocalRate(3);
19. plan1.setNationalRate(5);
20. plans.add(plan1);
21. PlanDTO plan2 = new PlanDTO();
22. plan2.setPlanId(2);
23. plan2.setPlanName("Medium");
24. plan2.setLocalRate(5);
25. plan2.setNationalRate(8);
26. plans.add(plan2);
27. }
28. // fetching all available plans
29. public List<PlanDTO> fetchPlans() {
30. return plans;
31. }
32. // fetching plans based on localRate
33. public List<PlanDTO> plansLocalRate(List localRates) {
34. List<PlanDTO> plansResponse = new ArrayList<>();
35. Iterator it = localRates.iterator();
36. while (it.hasNext()) {
37. int rate = Integer.parseInt((String) it.next());
38. for (PlanDTO plan : plans) {
39. if (rate == plan.getLocalRate())
40. plansResponse.add(plan);
41. }
42. }
43. return plansResponse;
44. }
45. }
```

**Step 8:** Create classes **CallDetailsService**, **CustomerService** and **PlanService** under **com.infytel.service** package:

```
1. package com.infytel.service;
2. import java.time.LocalDate;
```

```
3. import java.util.List;
4. import org.springframework.beans.factory.annotation.Autowired;
5. import org.springframework.stereotype.Service;
6. import com.infytel.dto.CallDetailsDTO;
7. import com.infytel.repository.CallDetailsRepository;
8. @Service
9. public class CallDetailsService {
10. @Autowired
11. private CallDetailsRepository callDetailsRepository;
12. // contacts repository to fetch the call details
13. public List<CallDetailsDTO> fetchCallDetails(long calledBy, LocalDate calledOn) {
14. return callDetailsRepository.fetchCallDetails(calledBy, calledOn);
15. }
16. }
```

```
1. package com.infytel.service;
2. import java.util.List;
3. import org.springframework.beans.factory.annotation.Autowired;
4. import org.springframework.stereotype.Service;
5. import com.infytel.dto.CustomerDTO;
6. import com.infytel.exceptions.NoSuchCustomerException;
7. import com.infytel.repository.CustomerRepository;
8. @Service
9. public class CustomerService {
10. @Autowired
11. private CustomerRepository customerRepository;
12. // Contacts repository layer to add customer
13. public String createCustomer(CustomerDTO customerDTO) {
14. return customerRepository.createCustomer(customerDTO);
15. }
16. // Contacts repository layer to fetch customer
17. public List<CustomerDTO> fetchCustomer() {
18. return customerRepository.fetchCustomer();
19. }
20. // Contacts repository layer to delete customer
21. public String deleteCustomer(long phoneNumber) throws NoSuchCustomerException {
22. return customerRepository.deleteCustomer(phoneNumber);
23. }
24. // Contacts repository layer to update customer
25. public String updateCustomer(long phoneNumber, CustomerDTO customerDTO) {
26. return customerRepository.updateCustomer(phoneNumber, customerDTO);
27. }
```

```
28. }
```

```
1. package com.infytel.service;
2. import java.util.List;
3. import org.springframework.beans.factory.annotation.Autowired;
4. import org.springframework.stereotype.Service;
5. import com.infytel.dto.PlanDTO;
6. import com.infytel.repository.PlanRepository;
7. @Service
8. public class PlanService {
9.     @Autowired
10.    private PlanRepository planRepository;
11.    // contacts repository to fetch plans
12.    public List<PlanDTO> fetchPlans() {
13.        return planRepository.fetchPlans();
14.    }
15.    // contacts repository to fetch plans by localRates
16.    public List<PlanDTO> plansLocalRate(List localRates) {
17.        return planRepository.plansLocalRate(localRates);
18.    }
19. }
```

**Step 9:** Add the following content to **application.properties** file:

```
1. server.port = 8080
2. server.servlet.context-path=/infytel-7
```

**Step 10:** Make sure that the project's pom.xml looks similar to the one shown below.

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <project xmlns="http://maven.apache.org/POM/4.0.0"
3.    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
5.        4.0.0.xsd">
6.    <modelVersion>4.0.0</modelVersion>
7.    <parent>
8.        <groupId>org.springframework.boot</groupId>
9.        <artifactId>spring-boot-starter-parent</artifactId>
10.       <version>2.1.4.RELEASE</version>
11.       <relativePath /> <!-- lookup parent from repository -->
12.    </parent>
13.    <groupId>com.infytel</groupId>
14.    <artifactId>infytel_demo7</artifactId>
15.    <version>0.0.1-SNAPSHOT</version>
```

```
15. <name>infytel_demo7</name>
16. <description>Demo project for Spring Boot</description>
17. <properties>
18. <java.version>1.8</java.version>
19. </properties>
20. <dependencies>
21. <dependency>
22. <groupId>org.springframework.boot</groupId>
23. <artifactId>spring-boot-starter-web</artifactId>
24. </dependency>
25. <dependency>
26. <groupId>org.springframework.boot</groupId>
27. <artifactId>spring-boot-starter-test</artifactId>
28. <scope>test</scope>
29. </dependency>
30. </dependencies>
31. <build>
32. <plugins>
33. <plugin>
34. <groupId>org.springframework.boot</groupId>
35. <artifactId>spring-boot-maven-plugin</artifactId>
36. </plugin>
37. </plugins>
38. </build>
39. </project>
```

**Step 11:** Deploy the application on the server by executing the class containing the main method.

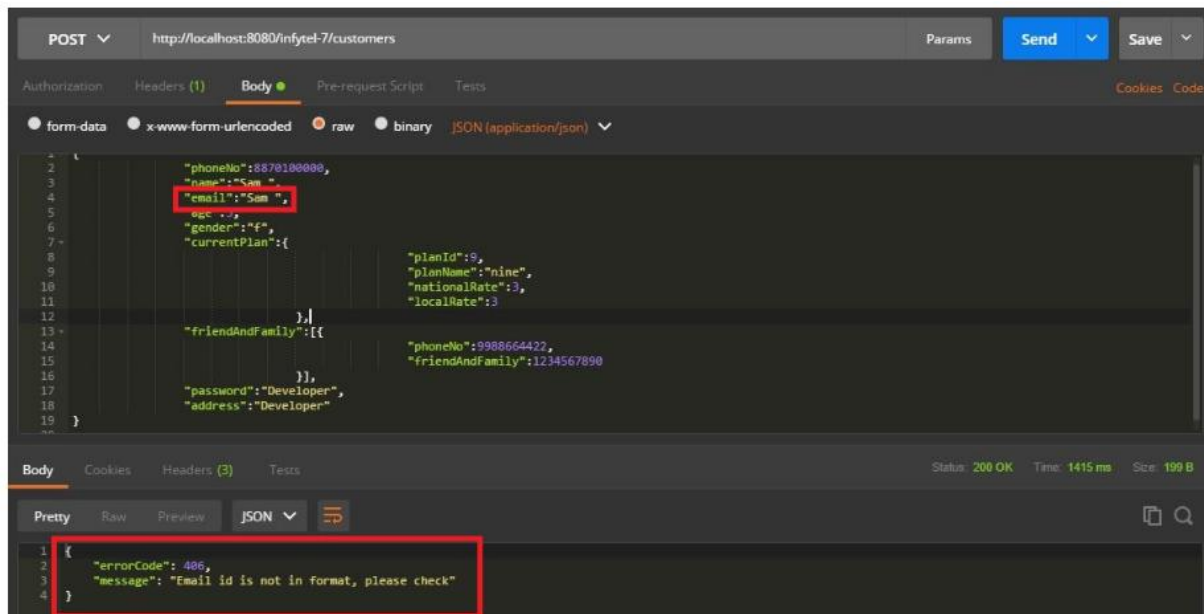
So, we have successfully created and deployed REST endpoints. Now, let us see how can we test the same using Postman client.

## Output

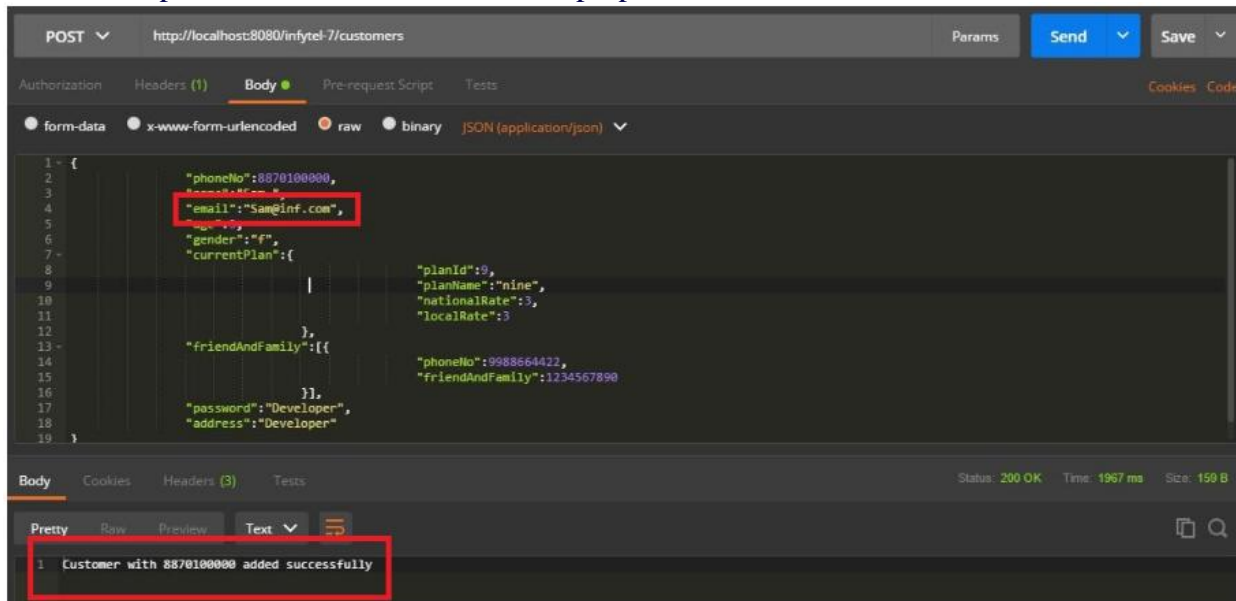
Testing Web Service using Postman

**Step 1:** Launch Postman.

**Step 2:** Test this URL - **http://localhost:8080/infytel-7/customers** using HTTP POST. Submit the request with customer data in JSON format. Try giving an invalid email id for the customer. The reason for failure along with the status code will be generated as response.



**Note:** Now, post another customer data with proper email id and feel the difference.



### Demo - Validation and exception handling

**Objectives:** To create a Spring REST controller, where the data being received is first validated, and then processed. If the data being received is not of the expected specifications, a custom error message will be sent to the client with the help of a centralized handler.

**Scenario:** An online telecom app called Infytel wishes its functionalities to get exposed as RESTful services. It has three controllers in total to deal with customer, plan and call details among which, the focus will be on the controller named CustomerController that has service methods that validate the incoming data and send customized error message back to the client during validation failures. Here follows the details of the methods,

Controller class	Method	URI	HTTP Method	Remarks
CustomerController	createCustomer	/customers	POST	This will create a customer in

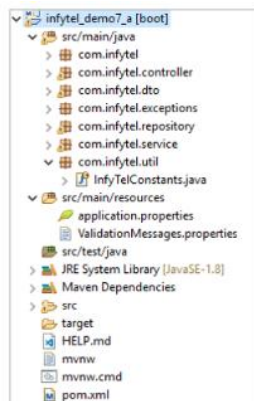


				Infytel provided, the fields/attributes of the CustomerDTO are valid. Have a look at the CustomerDTO for the complete set of validations that are applied on the incoming customer object.
CustomerController	deleteCustomer	/customers/{phoneNo}	DELETE	This method will delete the details of the customer based on the phoneNo, provided the phoneNo is of 10 digits
CustomerController	updateCustomer	/customers/{phoneNo}	PUT	This method will update the details (name, address and email) of the customer based on the phoneNo, provided the phoneNo is of 10 digits

**Steps:**

**Step 1:** Create a Maven project using Spring Initializer with web dependency and import the same in STS. Include spring-boot-starter-validation dependency in pom.xml if you use the recent version of Spring Boot to create the project.

**Step 2:** Modify the imported project according to the following project structure:



**Step 3:** Look at the class InfytelValidationApplication in com.infytel package that gets created automatically when the project is generated. We will modify this only when we need to go with custom configuration.

```

1. package com.infytel;
2. import org.springframework.boot.SpringApplication;
3. import org.springframework.boot.autoconfigure.SpringBootApplication;
4. @SpringBootApplication
5. public class InfytelValidationApplication {
6.     public static void main(String[] args) {
7.         SpringApplication.run(InfytelValidationApplication.class, args);
8.     }
9. }

```

**Step 4:** Create the class `CustomerController` under `com.infytel.controller` package:

```
1. package com.infytel.controller;
2. import java.util.List;
3. import javax.validation.Valid;
4. import javax.validation.constraints.Pattern;
5. import org.springframework.beans.factory.annotation.Autowired;
6. import org.springframework.http.ResponseEntity;
7. import org.springframework.validation.annotation.Validated;
8. import org.springframework.web.bind.annotation.DeleteMapping;
9. import org.springframework.web.bind.annotation.GetMapping;
10. import org.springframework.web.bind.annotation.PathVariable;
11. import org.springframework.web.bind.annotation.PostMapping;
12. import org.springframework.web.bind.annotation.PutMapping;
13. import org.springframework.web.bind.annotation.RequestBody;
14. import org.springframework.web.bind.annotation.RequestMapping;
15. import org.springframework.web.bind.annotation.RestController;
16. import com.infytel.dto.CustomerDTO;
17. import com.infytel.exceptions.NoSuchCustomerException;
18. import com.infytel.service.CustomerService;
19. @RestController
20. @RequestMapping("/customers")
21. @Validated //To trigger validation on URI parameters
22. public class CustomerController
23. {
24.     @Autowired
25.     private CustomerService customerService;
26.     //Fetching customer details
27.     @GetMapping(produces="application/json")
28.     public List<CustomerDTO> fetchCustomer()
29.     {
30.         return customerService.fetchCustomer();
31.     }
32.     //Adding a customer
33.     @PostMapping(consumes="application/json")
34.     public ResponseEntity<String> createCustomer(@Valid @RequestBody CustomerDTO
        customerDTO)
35.     {
36.         return ResponseEntity.ok(customerService.createCustomer(customerDTO));
37.     }
38.     //Updating an existing customer
39.     @PutMapping(value =("/{phoneNumber}", consumes = "application/json")
40.     public String updateCustomer(@PathVariable("phoneNumber")
```

```
41. @Pattern(regexp = "[0-9]{10}",message="{customer.phoneNo.invalid}")
42. String phoneNumber,@RequestBody CustomerDTO customerDTO) throws
    NoSuchCustomerException
43. {
44. return customerService.updateCustomer(Long.parseLong(phoneNumber), customerDTO);
45. }
46. // Deleting a customer
47. @DeleteMapping(value="/{phoneNumber}",produces="text/html")
48. public String deleteCustomer(@PathVariable("phoneNumber")
49. @Pattern(regexp = "[0-9]{10}",message="{customer.phoneNo.invalid}")
50. String phoneNumber) throws NoSuchCustomerException {
51. return customerService.deleteCustomer(Long.parseLong(phoneNumber));
52. }
53. }
```

**Step 5:** Create the DTO classes CustomerDTO, FriendFamilyDTO and PlanDTO under com.infytel.dto package

```
1. package com.infytel.dto;
2. import java.util.List;
3. import javax.validation.constraints.Email;
4. import javax.validation.constraints.Max;
5. import javax.validation.constraints.Min;
6. import javax.validation.constraints.NotBlank;
7. import javax.validation.constraints.NotEmpty;
8. import javax.validation.constraints.NotNull;
9. import javax.validation.constraints.Pattern;
10. public class CustomerDTO
11. {
12. @NotNull(message="{customer.phone.must}")
13. Long phoneNo;
14. @NotBlank(message="{customer.name.must}")
15. String name;
16. //Password should comprise of alphabets of both the cases and digits as well with a length of
    minimum 5
17. @NotEmpty(message="{customer.password.must}")
18. @Pattern(regexp="^(?=.*[0-9])(?=.*[a-z])(?=.*[A-Z])(?=.*\\S+$).{5,}$",message=
    "{customer.password.invalid}")
19. String password;
20. @NotNull(message="{customer.email.must}")
21. @Email(message="{customer.email.invalid}")
22. String email;
23. @Min(value=18, message="{customer.age.invalid}")
24. @Max(value=60, message="{customer.age.invalid}")
```

```
25. int age;
26. char gender;
27. List<FriendFamilyDTO> friendAndFamily;
28. String address;
29. @NotNull(message="{customer.plan.must}")
30. PlanDTO currentPlan;
31. public String getEmail() {
32. return email;
33. }
34. public void setEmail(String email) {
35. this.email = email;
36. }
37. public PlanDTO getCurrentPlan() {
38. return currentPlan;
39. }
40. public void setCurrentPlan(PlanDTO currentPlan) {
41. this.currentPlan = currentPlan;
42. }
43. public String getPassword() {
44. return password;
45. }
46. public void setPassword(String password) {
47. this.password = password;
48. }
49. public String getAddress() {
50. return address;
51. }
52. public void setAddress(String address) {
53. this.address = address;
54. }
55. public List<FriendFamilyDTO> getFriendAndFamily() {
56. return friendAndFamily;
57. }
58. public void setFriendAndFamily(List<FriendFamilyDTO> friendAndFamily) {
59. this.friendAndFamily = friendAndFamily;
60. }
61. public long getPhoneNo() {
62. return phoneNo;
63. }
64. public void setPhoneNo(long phoneNo) {
65. this.phoneNo = phoneNo;
66. }
```

```
67. public String getName() {
68. return name;
69. }
70. public void setName(String name) {
71. this.name = name;
72. }
73. public int getAge() {
74. return age;
75. }
76. public void setAge(int age) {
77. this.age = age;
78. }
79. public char getGender() {
80. return gender;
81. }
82. public void setGender(char gender) {
83. this.gender = gender;
84. }
85. @Override
86. public String toString() {
87. return "CustomerDTO [phoneNo=" + phoneNo + ", name=" + name + ", age=" + age + ", gender="
    + gender + ", friendAndFamily=" + friendAndFamily + ", password=" + password + ", address=" +
    address + "]\n";
88. }
89. }
90.
1. package com.infytel.dto;
2. public class FriendFamilyDTO {
3. long phoneNo;
4. long friendAndFamily;
5. public long getPhoneNo() {
6. return phoneNo;
7. }
8. public void setPhoneNo(long phoneNo) {
9. this.phoneNo = phoneNo;
10. }
11. public long getFriendAndFamily() {
12. return friendAndFamily;
13. }
14. public void setFriendAndFamily(long friendAndFamily) {
15. this.friendAndFamily = friendAndFamily;
16. }
```

```
17. public FriendFamilyDTO(long phoneNo, long friendAndFamily) {
18. this();
19. this.phoneNo = phoneNo;
20. this.friendAndFamily = friendAndFamily;
21. }
22. public FriendFamilyDTO() {
23. super();
24. }
25. @Override
26. public String toString() {
27. return "FriendFamilyDTO [phoneNo=" + phoneNo + ", friendAndFamily=" + friendAndFamily +
    "]\n";
28. }
29. }
30.
1. package com.infytel.dto;
2. import javax.xml.bind.annotation.XmlRootElement;
3. @XmlRootElement
4. public class PlanDTO {
5. Integer planId;
6. String planName;
7. Integer nationalRate;
8. Integer localRate;
9. public Integer getPlanId() {
10. return planId;
11. }
12. public void setPlanId(Integer planId) {
13. this.planId = planId;
14. }
15. public String getPlanName() {
16. return planName;
17. }
18. public void setPlanName(String planName) {
19. this.planName = planName;
20. }
21. public Integer getNationalRate() {
22. return nationalRate;
23. }
24. public void setNationalRate(Integer nationalRate) {
25. this.nationalRate = nationalRate;
26. }
27. public Integer getLocalRate() {
```

```
28. return localRate;
29. }
30. public void setLocalRate(Integer localRate) {
31. this.localRate = localRate;
32. }
33. public PlanDTO() {
34. super();
35. }
36. @Override
37. public String toString() {
38. return "PlanDTO [planId=" + planId + ", planName=" + planName + ", nationalRate=" +
    nationalRate + ", localRate=" + localRate + "]";
39. }
40. }
```

**Step 6:** Create the class `CustomerRepository` under `com.infytel.repository` package

```
1. package com.infytel.repository;
2. import java.util.ArrayList;
3. import java.util.List;
4. import javax.annotation.PostConstruct;
5. import org.springframework.stereotype.Repository;
6. import com.infytel.dto.CustomerDTO;
7. import com.infytel.dto.FriendFamilyDTO;
8. import com.infytel.dto.PlanDTO;
9. @Repository
10. public class CustomerRepository
11. {
12. List<CustomerDTO> customers = null;
13. //Populates customer in hard-coded way
14. @PostConstruct
15. public void initializer()
16. {
17. CustomerDTO customerDTO = new CustomerDTO();
18. PlanDTO planDTO = new PlanDTO();
19. planDTO.setPlanId(1);
20. planDTO.setPlanName("Simple");
21. planDTO.setLocalRate(3);
22. planDTO.setNationalRate(5);
23. customerDTO.setAddress("Chennai");
24. customerDTO.setAge(18);
25. customerDTO.setCurrentPlan(planDTO);
26. customerDTO.setGender('m');
27. customerDTO.setName("Jack");
```

```
28. customerDTO.setEmail("Jack@infy.com");
29. customerDTO.setPassword("ABC@123");
30. customerDTO.setPhoneNo(99512122221);
31. List<FriendFamilyDTO> friendAndFamily = new ArrayList<>();
32. friendAndFamily.add(new FriendFamilyDTO(customerDTO.getPhoneNo(),800000145));
33. friendAndFamily.add(new FriendFamilyDTO(customerDTO.getPhoneNo(),700000145));
34. customerDTO.setFriendAndFamily(friendAndFamily);
35. customers = new ArrayList<>();
36. customers.add(customerDTO);
37. }
38. //creates customer
39. public String createCustomer(CustomerDTO customerDTO)
40. {
41. customers.add(customerDTO);
42. return "Customer details got added successfully";
43. //since this code deals with a hard-coded list and not the actual repository, return string gets hard-
    coded here.
44. //The service layer will deal with the exception or the response, otherwise.
45. //In such cases, its preferred to make the exception and the success message available in the
    ValidationMessages.properties
46. //And, these messages can be taken as how the methods of CustomerService do
47. }
48. //fetches customer
49. public List<CustomerDTO> fetchCustomer()
50. {
51. return customers;
52. }
53. //deletes customer - exception handling incorporated
54. public boolean deleteCustomer(long phoneNumber)
55. {
56. boolean responseDelete=false;
57. for(CustomerDTO customer : customers)
58. {
59. if(customer.getPhoneNo() == phoneNumber)
60. {
61. customers.remove(customer);
62. responseDelete=true;
63. break;
64. }
65. }
66. return responseDelete;
67. }
```



```
68. //finds the customer based on phoneNumber and updates the details of the same
69. public boolean updateCustomer(long phoneNumber, CustomerDTO customerDTO)
70. {
71. boolean responseUpdate=false;
72. for(CustomerDTO customer : customers)
73. {
74. if(customer.getPhoneNo() == phoneNumber)
75. {
76. if(customerDTO.getAddress()!=null)
77. {
78. customer.setAddress(customerDTO.getAddress());
79. }
80. if(customerDTO.getEmail()!=null)
81. {
82. customer.setEmail(customerDTO.getEmail());
83. }
84. responseUpdate = true;
85. break;
86. }
87. }
88. return responseUpdate;
89. }
90. }
```

**Step 7:** Create the class CustomerService under com.infytel.service package

```
1. package com.infytel.service;
2. import java.util.List;
3. import java.util.stream.Collectors;
4. import org.springframework.beans.factory.annotation.Autowired;
5. import org.springframework.context.annotation.PropertySource;
6. import org.springframework.core.env.Environment;
7. import org.springframework.stereotype.Service;
8. import com.infytel.dto.CustomerDTO;
9. import com.infytel.exceptions.NoSuchCustomerException;
10. import com.infytel.repository.CustomerRepository;
11. import com.infytel.util.InfyTelConstants;
12. @Service
13. @PropertySource("classpath:ValidationMessages.properties")
14. public class CustomerService
15. {
16. @Autowired
17. private Environment environment;
18. @Autowired
```

```
19. private CustomerRepository customerRepository;
20. //Contacts repository layer to add customer
21. public String createCustomer(CustomerDTO customerDTO)
22. {
23. return customerRepository.createCustomer(customerDTO);
24. }
25. //makes a call to repository method for returning a list of customers
26. public List<CustomerDTO> fetchCustomer()
27. {
28. List<CustomerDTO> customers = customerRepository.fetchCustomer();
29. //code that iterates through customers and set the password to *
30. return customers.stream().map(c->{c.setPassword("*");return c;}).collect(Collectors.toList());
31. }
32. //Contacts repository layer to delete customer
33. public String deleteCustomer(long phoneNumber)throws NoSuchCustomerException
34. {
35. boolean response = customerRepository.deleteCustomer(phoneNumber);
36. if(!response)
37. throw new
    NoSuchCustomerException(environment.getProperty(InfyTelConstants.CUSTOMER_NOT_FOU
    ND.toString()));
38. return environment.getProperty(InfyTelConstants.CUSTOMER_DELETE_SUCCESS.toString());
39. }
40. //Contacts repository layer to update customer
41. public String updateCustomer(long phoneNumber, CustomerDTO customerDTO) throws
    NoSuchCustomerException
42. {
43. boolean response = customerRepository.updateCustomer(phoneNumber,customerDTO);
44. if(!response)
45. throw new
    NoSuchCustomerException(environment.getProperty(InfyTelConstants.CUSTOMER_NOT_FOU
    ND.toString()));
46. return environment.getProperty(InfyTelConstants.CUSTOMER_UPDATE_SUCCESS.toString());
47. }
48. }
```

**Step 8:** Add the following content to application.properties file

1. server.port = 8080
2. server.servlet.context-path=/infytel-7a

**Step 9:** Add the following content to ValidationMessages.properties file

1. #validation related messages
2. customer.phone.must = Phone number should be present, please check

3. customer.name.must = Name should be present, please check
4. customer.password.invalid = Password is not in format, please check
5. customer.password.must = Password should be present, please check
6. customer.email.invalid = Email-Id is not in format, please check
7. customer.email.must = Email-Id should be present, please check
8. customer.age.invalid = Age is not in range, please check
9. customer.plan.must = Plan-id should be present, please check
10. customer.phoneNo.invalid = Phone number not in format, please check
11. call.calledon.invalid = Called date should be present or past, please check
12. call.duration.invalid = Call duration can either be zero or positive, please check
13. #exception related messages
14. customer.not.found = Customer does not exist for this phone number
15. general.exception = Oops! something went wrong, please try again!
16. #response related messages
17. customer.delete.success = Customer details got deleted successfully
18. customer.update.success = Customer details got updated successfully

**Step 10:** Create `Infytelconstants.java` under `com.infytel.util` package

```
1. package com.infytel.util;
2. /**
3.  * The Enum ExceptionConstants.
4.  */
5. public enum InfyTelConstants {
6.     //Exception message constants
7.     CUSTOMER_NOT_FOUND("customer.not.found"),
8.     GENERAL_EXCEPTION_MESSAGE("general.exception"),
9.     //Success message constants
10.    CUSTOMER_UPDATE_SUCCESS("customer.update.success"),
11.    CUSTOMER_DELETE_SUCCESS("customer.delete.success");
12.    private final String type;
13.    private InfyTelConstants(String type) {
14.        this.type = type;
15.    }
16.    @Override
17.    public String toString() {
18.        return this.type;
19.    }
20. }
```

**Step 10:** Create `ExceptionHandlerAdvice` under `com.infytel.exceptions`.

```
1. package com.infytel.exceptions;
2. import java.util.stream.Collectors;
3. import javax.validation.ConstraintViolation;
```

```
4. import javax.validation.ConstraintViolationException;
5. import org.springframework.beans.factory.annotation.Autowired;
6. import org.springframework.core.env.Environment;
7. import org.springframework.http.HttpStatus;
8. import org.springframework.http.ResponseEntity;
9. import org.springframework.validation.ObjectError;
10. import org.springframework.web.bind.MethodArgumentNotValidException;
11. import org.springframework.web.bind.annotation.ExceptionHandler;
12. import org.springframework.web.bind.annotation.RestControllerAdvice;
13. import com.infytel.dto.ErrorMessage;
14. import com.infytel.util.InfyTelConstants;
15. @RestControllerAdvice
16. public class ExceptionControllerAdvice {
17. //this helps receiving the message/value related to the general exception from the
    ValidationMessages.properties
18. @Autowired
19. private Environment environment;
20. //Handler for exceptions other than NoSuchCustomerException and validation exceptions
21. @ExceptionHandler(Exception.class)
22. public ResponseEntity<ErrorMessage> exceptionHandler2(Exception ex)
23. {
24. ErrorMessage error = new ErrorMessage();
25. error.setErrorCode(HttpStatus.INTERNAL_SERVER_ERROR.value());
26. error.setMessage(environment.getProperty(InfyTelConstants.GENERAL_EXCEPTION_MESSA
    GE.toString()));
27. return new ResponseEntity<>(error, HttpStatus.OK);
28. }
29. //Handler for NoSuchCustomerException
30. @ExceptionHandler(NoSuchCustomerException.class)
31. public ResponseEntity<ErrorMessage> exceptionHandler2(NoSuchCustomerException ex)
32. {
33. ErrorMessage error = new ErrorMessage();
34. error.setErrorCode(HttpStatus.BAD_REQUEST.value());
35. error.setMessage(ex.getMessage());
36. return new ResponseEntity<>(error, HttpStatus.BAD_REQUEST);
37. }
38. //Handler that handles the exception raised because of invalid data that is received as method
    argument (DTO)
39. @ExceptionHandler(MethodArgumentNotValidException.class)
40. public ResponseEntity<ErrorMessage>
    handleValidationExceptions(MethodArgumentNotValidException ex)
41. {
```

```
42. ErrorMessage error = new ErrorMessage();
43. error.setErrorCode(HttpStatus.BAD_REQUEST.value());
44. error.setMessage(ex.getBindingResult().getAllErrors()
45. .stream().map(ObjectError::getDefaultMessage)//lambda equivalent -> x->x.getDefaultMessage()
46. .collect(Collectors.joining(", ")));
47. return new ResponseEntity<>(error, HttpStatus.BAD_REQUEST);
48. }
49. //Handler that handles the exception raised because of invalid data that is received as
50. //URI parameter (path variables, request parameters)
51. @ExceptionHandler(ConstraintViolationException.class)
52. public ResponseEntity<ErrorMessage>
    handleConstraintValidationExceptions(ConstraintViolationException ex)
53. {
54. ErrorMessage error = new ErrorMessage();
55. error.setErrorCode(HttpStatus.BAD_REQUEST.value());
56. error.setMessage(ex.getConstraintViolations()
57. .stream().map(ConstraintViolation::getMessage)//lambda equivalent -> x->x.getMessage()
58. .collect(Collectors.joining(", ")));
59. return new ResponseEntity<>(error, HttpStatus.BAD_REQUEST);
60. }
61. }
```

**Step 11:** Deploy the application by executing the class that contains the main method.

So, we have successfully created and deployed the REST endpoints with the provision for validating the incoming data. Now, let us see how to test the same using Postman client.

## Output Screenshots

### Testing Web Service using Postman

**Step 1:** Launch Postman.

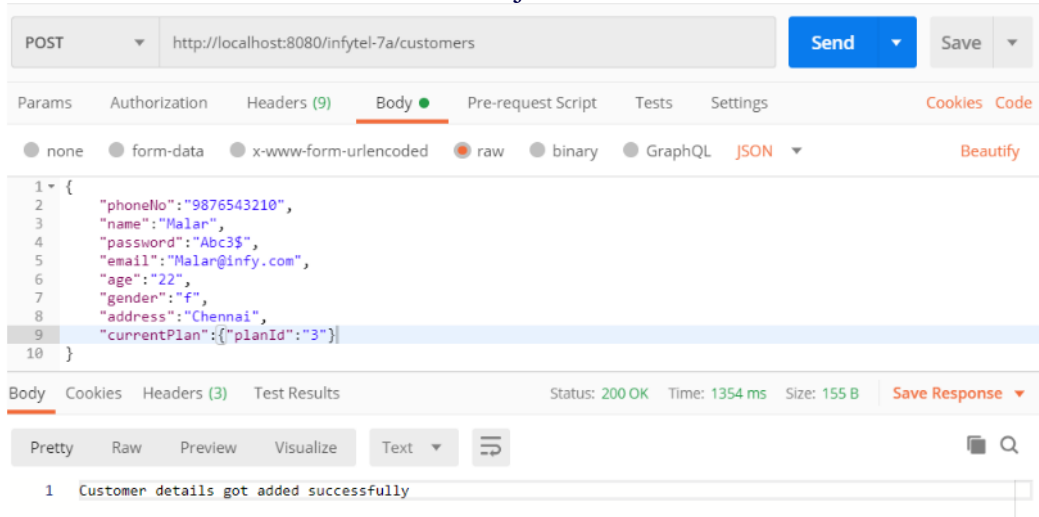
**Step 2:** Test POST request

Select POST from the drop-down and enter `http://localhost:8080/infytel-7a/customers` into the URL field. Then, click on body to enter the following JSON data. Finally, click on Send. This action will create/add a customer in Infytel.

```
1. {
2.   "phoneNo":"9876543210",
3.   "name":"Malar",
4.   "password":"Abc3$",
5.   "email":"Malar@infy.com",
6.   "age":"22",
```

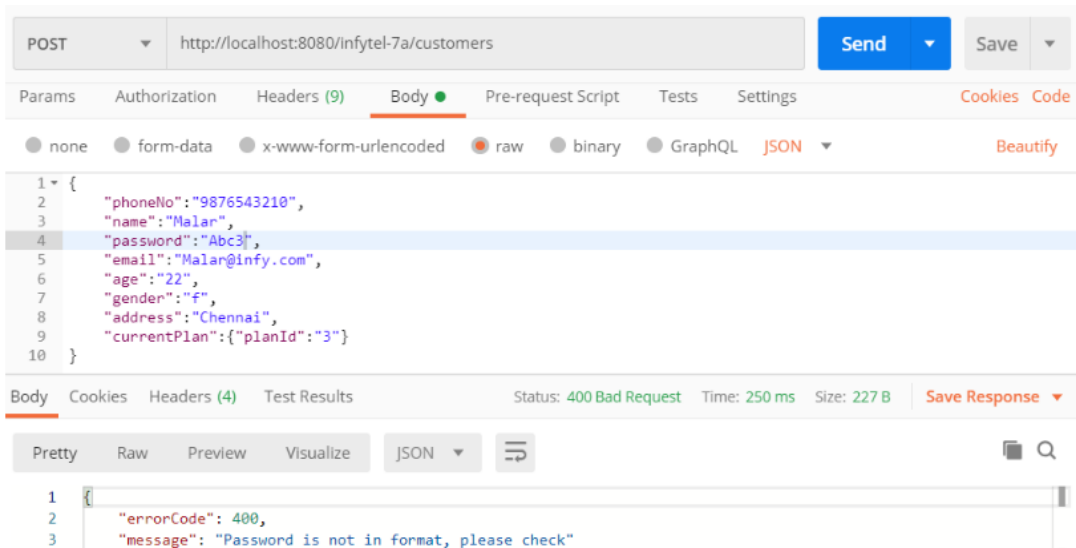
```
7. "gender":"f",
8. "address":"Chennai",
9. "currentPlan":
10. {
11. "planId":"3"
12. }
13. }
```

The JSON data has to match the DTO object to which it is converted.

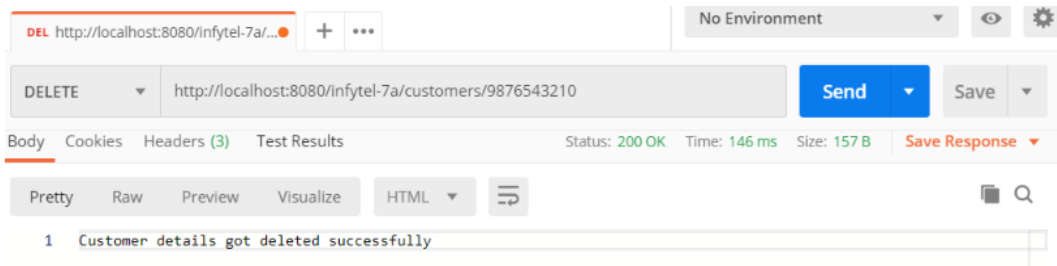


The data provided here in the above step is all valid.

**Step 3:** Now, have a look at the CustomerDTO to know what can be the valid set of values for the fields of the same and try sending an invalid value and see what happens. For example, let the password not be in format.

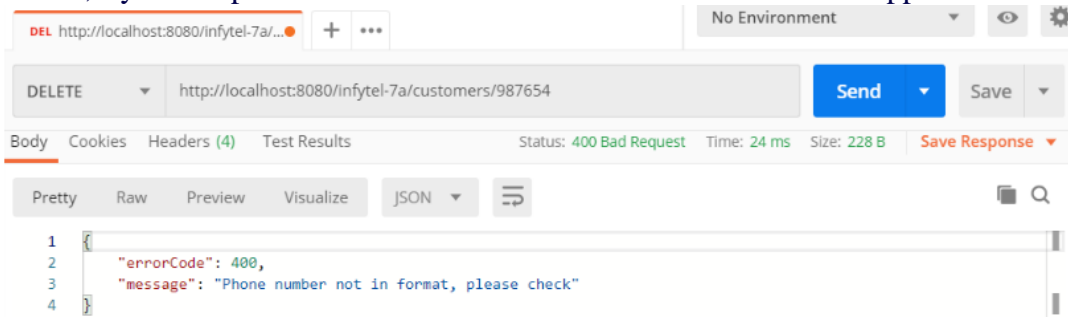


**Step 4 :** Now, choose DELETE from the drop-down and enter `http://localhost:8080/infytel-7a/customers/9876543210` into the URL field where phone number is given as path-variable.



The phone number being passed in the above step was valid.

**Step 5:** Now, try with a phone number that is not in format and see what happens.



## Creating a Rest Client

### RestTemplate

There are situations where a Spring REST endpoint might be in need of contacting other RESTful resources. Situations like this can be handled with the help of REST client that is available in the Spring framework. And, the name of this REST client is RestTemplate.

RestTemplate has a wonderful support for standard HTTP methods.

The methods of the RestTemplate need to be provided with the absolute URI where the service can be found, input data, if any and, the response type that is expected out of the service.

### Calling a RESTful service of HTTP request method Get:

The following code snippet shows how to consume a RESTful service exposed by url :<http://localhost:8080/infytel/customers> using HTTP GET.

```
1. RestTemplate restTemplate = new RestTemplate();
2. String url="http://localhost:8080/infytel/customers";
3. List<CustomerDTO> customers = (List<CustomerDTO>) restTemplate.getForObject(url,
   List.class);
4. for(CustomerDTO customer:customers)
5. {
6. System.out.println("Customer Name: "+customer.getName());
7. System.out.println("Customer Phone: "+customer.getPhoneNo());
8. System.out.println("Email Id: "+customer.getEmail());
9. }
```

## Calling a RESTful service of HTTP request method Post:

URI of the service: `http://localhost:8080/infytel/customers`

```
1. CustomerDTO customerrequest= new CustomerDTO();
2. //populate CustomerDTO object
3. //....
4. String url="http://localhost:8080/infytel/customers"
5. RestTemplate restTemplate = new RestTemplate();
6. ResponseEntity<String> response =
    restTemplate.postForEntity(url,customerrequest,String.class);
7. System.out.println("status code :" + response.getStatusCodeValue());
8. System.out.println("Info      :" + response.getBody());
```

Similarly, RESTful services of various other HTTP methods can be called using RestTemplate.

## Demo 8a - Creating a REST endpoint exposing plan details

**Objectives:** To create a Spring REST endpoint and expose the same to the clients for consumption.

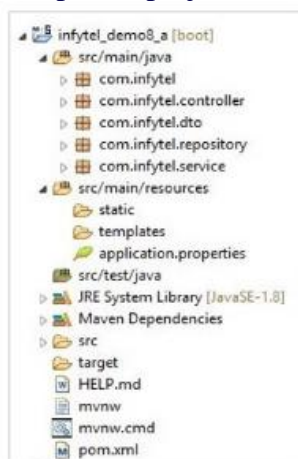
**Scenario:** An online telecom app called Infytel is exposing its telephone tariff plans as a RESTful service. This is a simple RESTful endpoint which will be consumed by another application, Demo 8b.

Controller Class	method	URI	HTTP Method	Remarks
PlanController	fetchPlanById()	/plans/{planId}	GET	Will return the complete plan details for a given planId

### Steps:

**Step 1:** Create a Maven project using Spring Initializer with web dependency and import the same in STS.

**Step 2:** Modify the imported project according to the following project structure:



**Step 3:** Look at the class **InfytelDemo8AApplication** under **com.infytel** package that gets created automatically when the project is generated. We will modify this only when we need to go with custom configuration.

```
1. package com.infytel;
2. import org.springframework.boot.SpringApplication;
```



```
3. import org.springframework.boot.autoconfigure.SpringBootApplication;
4. @SpringBootApplication
5. public class InfytelDemo8AApplication {
6.     public static void main(String[] args) {
7.         SpringApplication.run(InfytelDemo8AApplication.class, args);
8.     }
9. }
```

**Step 4:** Create class **PlanController** under **com.infytel.controller** package.

```
1. package com.infytel.controller;
2. import org.springframework.beans.factory.annotation.Autowired;
3. import org.springframework.web.bind.annotation.GetMapping;
4. import org.springframework.web.bind.annotation.PathVariable;
5. import org.springframework.web.bind.annotation.RequestMapping;
6. import org.springframework.web.bind.annotation.RestController;
7. import com.infytel.dto.PlanDTO;
8. import com.infytel.service.PlanService;
9. @RestController
10. @RequestMapping("/plans")
11. public class PlanController {
12.     @Autowired
13.     private PlanService planService;
14.     // method for getting all the available plans will go here
15.     // and
16.     // method for getting the plans based on local rate will go here
17.     // fetching the plans by id
18.     @GetMapping("/{planId}")
19.     public PlanDTO fetchPlanById(@PathVariable("planId") int planId) {
20.         return planService.fetchPlanById(planId);
21.     }
22. }
```

**Step 5:** Create class **PlanDTO** under **com.infytel.dto** package:

```
1. package com.infytel.dto;
2. public class PlanDTO {
3.     Integer planId;
4.     String planName;
5.     Integer nationalRate;
6.     Integer localRate;
7.     public Integer getPlanId() {
8.         return planId;
9.     }
10.     public void setPlanId(Integer planId) {
```

```
11. this.planId = planId;
12. }
13. public String getPlanName() {
14. return planName;
15. }
16. public void setPlanName(String planName) {
17. this.planName = planName;
18. }
19. public Integer getNationalRate() {
20. return nationalRate;
21. }
22. public void setNationalRate(Integer nationalRate) {
23. this.nationalRate = nationalRate;
24. }
25. public Integer getLocalRate() {
26. return localRate;
27. }
28. public void setLocalRate(Integer localRate) {
29. this.localRate = localRate;
30. }
31. public PlanDTO() {
32. super();
33. }
34. @Override
35. public String toString() {
36. return "PlanDTO [planId=" + planId + ", planName=" + planName + ", nationalRate=" +
    nationalRate + ", localRate=" + localRate + "]";
37. }
38. }
```

**Step 6:** Create class **PlanRepository** under com.infytel.repository package.

```
1. package com.infytel.repository;
2. import java.util.ArrayList;
3. import java.util.List;
4. import java.util.Optional;
5. import javax.annotation.PostConstruct;
6. import org.springframework.stereotype.Repository;
7. import com.infytel.dto.PlanDTO;
8. @Repository
9. public class PlanRepository {
10. private List<PlanDTO> plans;
11. // Populating a list of plans
```

```
12. @PostConstruct
13. public void populatePlans() {
14. plans = new ArrayList<>();
15. PlanDTO plan1 = new PlanDTO();
16. plan1.setPlanId(1);
17. plan1.setPlanName("Simple");
18. plan1.setLocalRate(3);
19. plan1.setNationalRate(5);
20. plans.add(plan1);
21. PlanDTO plan2 = new PlanDTO();
22. plan2.setPlanId(2);
23. plan2.setPlanName("Medium");
24. plan2.setLocalRate(5);
25. plan2.setNationalRate(8);
26. plans.add(plan2);
27. }
28. // methods fetchPlans() and plansLocalRate() go here
29. // fetching plan by id
30. public PlanDTO fetchPlanById(int planId) {
31. Optional<PlanDTO> optionalPlanDTO = plans.stream().filter(x -> x.getPlanId() ==
    planId).findFirst();
32. return optionalPlanDTO.orElse(plans.get(0)); // if the optional contains a value, fine. Else the first
    plan
33. // available in the list will be set
34. }
35. }
```

**Step 7:** Create class **PlanService** under com.infytel.service package.

```
1. package com.infytel.service;
2. import org.springframework.beans.factory.annotation.Autowired;
3. import org.springframework.stereotype.Service;
4. import com.infytel.dto.PlanDTO;
5. import com.infytel.repository.PlanRepository;
6. @Service
7. public class PlanService {
8. @Autowired
9. private PlanRepository planRepository;
10. // methods fetchPlans() and plansLocalRate() go here
11. public PlanDTO fetchPlanById(int planId) {
12. return planRepository.fetchPlanById(planId);
13. }
14. }
```

**Step 8:** Add the following content to **application.properties** file:

1. server.port=8080
2. server.servlet.context-path=/infytel-8a

**Step 9:** Make sure that the project's pom.xml looks similar to the one shown below.

1. <?xml version="1.0" encoding="UTF-8"?>
2. <project xmlns="http://maven.apache.org/POM/4.0.0"
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
5. <modelVersion>4.0.0</modelVersion>
6. <parent>
7. <groupId>org.springframework.boot</groupId>
8. <artifactId>spring-boot-starter-parent</artifactId>
9. <version>2.1.4.RELEASE</version>
10. <relativePath /> <!-- lookup parent from repository -->
11. </parent>
12. <groupId>infytel</groupId>
13. <artifactId>infytel\_demo8\_a</artifactId>
14. <version>0.0.1-SNAPSHOT</version>
15. <name>infytel\_demo8\_a</name>
16. <description>REST client demo A</description>
17. <properties>
18. <java.version>1.8</java.version>
19. </properties>
20. <dependencies>
21. <dependency>
22. <groupId>org.springframework.boot</groupId>
23. <artifactId>spring-boot-starter-web</artifactId>
24. </dependency>
25. <dependency>
26. <groupId>org.springframework.boot</groupId>
27. <artifactId>spring-boot-starter-test</artifactId>
28. <scope>test</scope>
29. </dependency>
30. </dependencies>
31. <build>
32. <plugins>
33. <plugin>
34. <groupId>org.springframework.boot</groupId>
35. <artifactId>spring-boot-maven-plugin</artifactId>
36. </plugin>

```
37. </plugins>
38. </build>
39. </project>
```

**Step 10:** Deploy the application on the server by executing the class containing the main method.

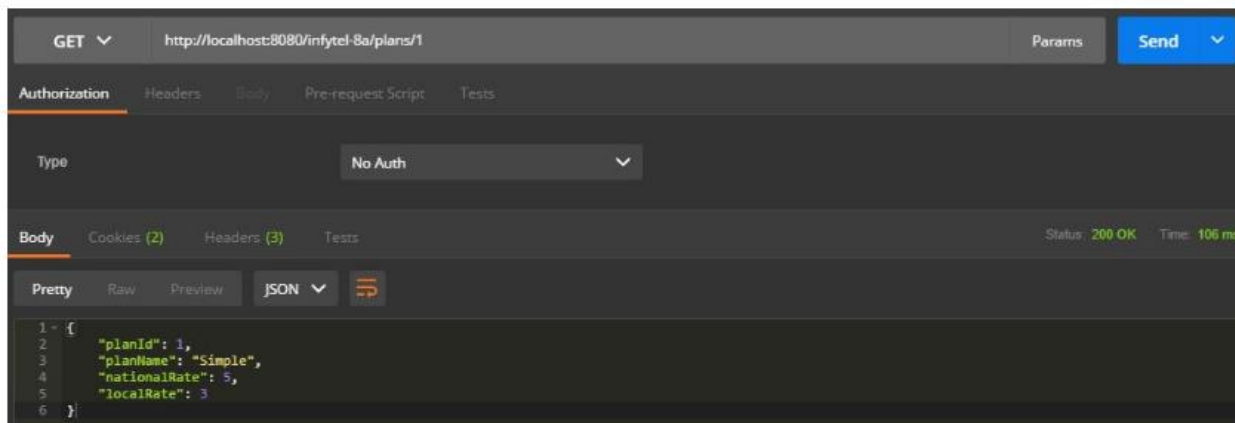
So, we have successfully created and exposed a REST endpoint and made it available on port number 8080. Now, let us test the same using Postman client.

## Output

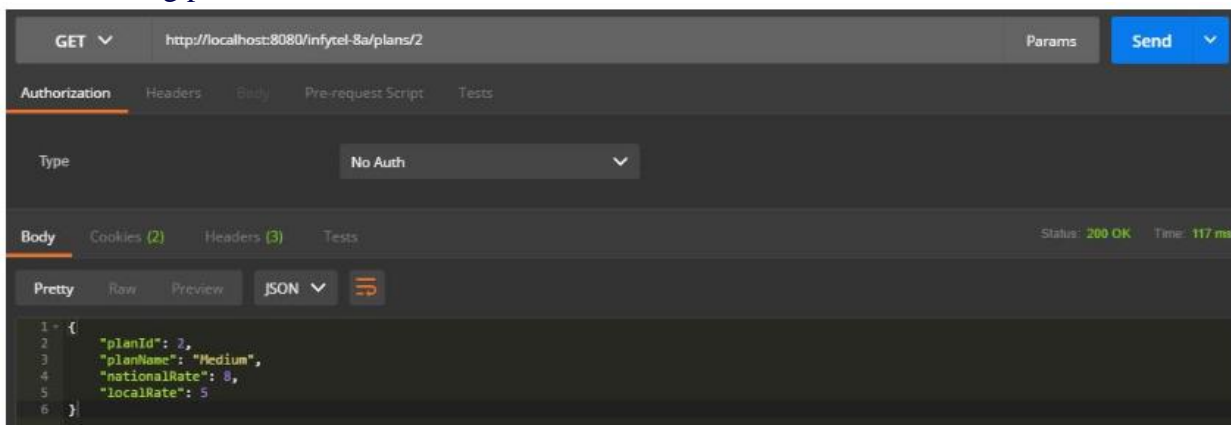
Testing Web Service using Postman

**Step 1:** Launch Postman.

**Step 2:** Test this URL - **http://localhost:8080/infytel-8a/plans/1** using HTTP GET to get the details of the existing planId, 1.



**Step 3:** Testing this URL - **http://localhost:8080/infytel-8a/plans/2** using HTTP GET to get the details of the existing planId, 2.



## Demo 8b - REST client consuming an existing endpoint

**Objectives:** To create a Spring REST application which will consume a REST endpoint that is being exposed by another REST application. We will learn,

- How to consume a RESTful endpoint using RestTemplate.

**Scenario:** An online telecom app called Infytel wishes its functionalities to get exposed as RESTful services. It has three controllers in total to deal with customer, plan and call details, among which, the

focus will be on the controller named `CustomerController` that consumes the plan details from an existing endpoint of the application, `infytel_demo_8a`.

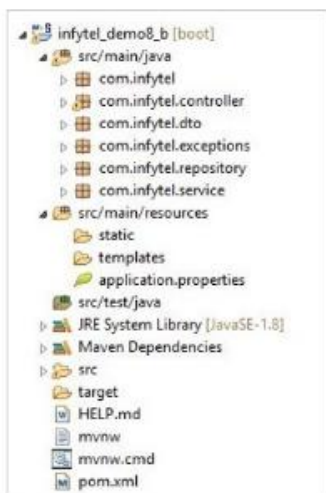
Following is the method that uses `RestTemplate` for the consumption of plan details.

Controller Class	method	URI	HTTP method	Remarks
<code>CustomerController</code>	<code>createCustomer()</code>	<code>/customers</code>	POST	To create a customer

### Steps:

**Step 1:** Create a Maven project using Spring Initializer with web dependency and import the same in STS.

**Step 2:** Modify the imported project according to the following project structure:



**Step 3:** Look at the class `InfytelDemo8BApplication` under `com.infytel` package that gets created automatically when the project is generated. We will modify this only when we need to go with custom configuration.

```

1. package com.infytel;
2. import org.springframework.boot.SpringApplication;
3. import org.springframework.boot.autoconfigure.SpringBootApplication;
4. @SpringBootApplication
5. public class InfytelDemo8BApplication {
6.     public static void main(String[] args) {
7.         SpringApplication.run(InfytelDemo8BApplication.class, args);
8.     }
9. }

```

**Step 4:** Create class `CustomerController` under `com.infytel.controller` package:

```

1. package com.infytel.controller;
2. import java.util.List;
3. import java.util.stream.Collectors;
4. import javax.validation.Valid;
5. import org.springframework.beans.factory.annotation.Autowired;
6. import org.springframework.http.HttpStatus;
7. import org.springframework.http.ResponseEntity;

```

```
8. import org.springframework.validation.Errors;
9. import org.springframework.validation.ObjectError;
10. import org.springframework.web.bind.annotation.DeleteMapping;
11. import org.springframework.web.bind.annotation.GetMapping;
12. import org.springframework.web.bind.annotation.PathVariable;
13. import org.springframework.web.bind.annotation.PostMapping;
14. import org.springframework.web.bind.annotation.PutMapping;
15. import org.springframework.web.bind.annotation.RequestBody;
16. import org.springframework.web.bind.annotation.RequestMapping;
17. import org.springframework.web.bind.annotation.RestController;
18. import org.springframework.web.client.RestTemplate;
19. import com.infytel.dto.CustomerDTO;
20. import com.infytel.dto.ErrorMessage;
21. import com.infytel.dto.PlanDTO;
22. import com.infytel.exceptions.NoSuchCustomerException;
23. import com.infytel.service.CustomerService;
24. @RestController
25. @RequestMapping("/customers")
26. public class CustomerController {
27.     @Autowired
28.     private CustomerService customerService;
29.     // Fetching customer details
30.     @GetMapping(produces = "application/json")
31.     public List<CustomerDTO> fetchCustomer() {
32.         return customerService.fetchCustomer();
33.     }
34.     // Adding a customer
35.     @PostMapping(consumes = "application/json")
36.     public ResponseEntity createCustomer(@Valid @RequestBody CustomerDTO customerDTO,
        Errors errors) {
37.         String response = "";
38.         if (errors.hasErrors()) {
39.             response = errors.getAllErrors().stream().map(ObjectError::getDefaultMessage)
40.                 .collect(Collectors.joining(", "));
41.             ErrorMessage error = new ErrorMessage();
42.             error.setErrorCode(HttpStatus.NOT_ACCEPTABLE.value());
43.             error.setMessage(response);
44.             return ResponseEntity.ok(error);
45.         } else {
46.             PlanDTO planDTOReceived = new RestTemplate().getForObject(
47.                 "http://localhost:8080/infytel-8a/plans/" + customerDTO.getCurrentPlan().getPlanId(),
                PlanDTO.class);
```

```
48. customerDTO.setCurrentPlan(planDTOReceived);
49. response = customerService.createCustomer(customerDTO);
50. return ResponseEntity.ok(response);
51. }
52. }
53. // Updating an existing customer
54. @PutMapping(value =("/{phoneNumber}", consumes = "application/json")
55. public String updateCustomer(@PathVariable("phoneNumber") long phoneNumber,
    @RequestBody CustomerDTO customerDTO) {
56. return customerService.updateCustomer(phoneNumber, customerDTO);
57. }
58. // Deleting a customer
59. @DeleteMapping(value =("/{phoneNumber}", produces = "text/html")
60. public String deleteCustomer(@PathVariable("phoneNumber") long phoneNumber) throws
    NoSuchCustomerException {
61. return customerService.deleteCustomer(phoneNumber);
62. }
63. }
```

**Step 5:** Create classes **CustomerDTO**, **ErrorMessage**, **FriendFamilyDTO** and **PlanDTO** under **com.infytel.dto** package:

```
1. package com.infytel.dto;
2. import java.util.List;
3. import javax.validation.constraints.Email;
4. public class CustomerDTO {
5. long phoneNo;
6. String name;
7. @Email(message = "Email id is not in format, please check")
8. String email;
9. public String getEmail() {
10. return email;
11. }
12. public void setEmail(String email) {
13. this.email = email;
14. }
15. int age;
16. char gender;
17. List<FriendFamilyDTO> friendAndFamily;
18. String password;
19. String address;
20. PlanDTO currentPlan;
21. public PlanDTO getCurrentPlan() {
```



```
22. return currentPlan;
23. }
24. public void setCurrentPlan(PlanDTO currentPlan) {
25. this.currentPlan = currentPlan;
26. }
27. public String getPassword() {
28. return password;
29. }
30. public void setPassword(String password) {
31. this.password = password;
32. }
33. public String getAddress() {
34. return address;
35. }
36. public void setAddress(String address) {
37. this.address = address;
38. }
39. public List<FriendFamilyDTO> getFriendAndFamily() {
40. return friendAndFamily;
41. }
42. public void setFriendAndFamily(List<FriendFamilyDTO> friendAndFamily) {
43. this.friendAndFamily = friendAndFamily;
44. }
45. public long getPhoneNo() {
46. return phoneNo;
47. }
48. public void setPhoneNo(long phoneNo) {
49. this.phoneNo = phoneNo;
50. }
51. public String getName() {
52. return name;
53. }
54. public void setName(String name) {
55. this.name = name;
56. }
57. public int getAge() {
58. return age;
59. }
60. public void setAge(int age) {
61. this.age = age;
62. }
63. public char getGender() {
```

```
64. return gender;
65. }
66. public void setGender(char gender) {
67. this.gender = gender;
68. }
69. @Override
70. public String toString() {
71. return "CustomerDTO [phoneNo=" + phoneNo + ", name=" + name + ", age=" + age + ", gender="
    + gender + ", friendAndFamily=" + friendAndFamily + ", password=" + password + ", address=" +
    address + "]";
72. }
73. }
```

```
1. package com.infytel.dto;
2. public class ErrorMessage {
3. private int errorCode;
4. private String message;
5. public int getErrorCode() {
6. return errorCode;
7. }
8. public void setErrorCode(int errorCode) {
9. this.errorCode = errorCode;
10. }
11. public String getMessage() {
12. return message;
13. }
14. public void setMessage(String message) {
15. this.message = message;
16. }
17. }
```

```
1. package com.infytel.dto;
2. public class FriendFamilyDTO {
3. long phoneNo;
4. long friendAndFamily;
5. public long getPhoneNo() {
6. return phoneNo;
7. }
8. public void setPhoneNo(long phoneNo) {
9. this.phoneNo = phoneNo;
10. }
```

```
11. public long getFriendAndFamily() {
12. return friendAndFamily;
13. }
14. public void setFriendAndFamily(long friendAndFamily) {
15. this.friendAndFamily = friendAndFamily;
16. }
17. public FriendFamilyDTO(long phoneNo, long friendAndFamily) {
18. this();
19. this.phoneNo = phoneNo;
20. this.friendAndFamily = friendAndFamily;
21. }
22. public FriendFamilyDTO() {
23. super();
24. }
25. @Override
26. public String toString() {
27. return "FriendFamilyDTO [phoneNo=" + phoneNo + ", friendAndFamily=" + friendAndFamily +
    "]\n";
28. }
29. }
```

```
1. package com.infytel.dto;
2. import javax.xml.bind.annotation.XmlRootElement;
3. @XmlRootElement
4. public class PlanDTO {
5. Integer planId;
6. String planName;
7. Integer nationalRate;
8. Integer localRate;
9. public Integer getPlanId() {
10. return planId;
11. }
12. public void setPlanId(Integer planId) {
13. this.planId = planId;
14. }
15. public String getPlanName() {
16. return planName;
17. }
18. public void setPlanName(String planName) {
19. this.planName = planName;
20. }
21. public Integer getNationalRate() {
```

```
22. return nationalRate;
23. }
24. public void setNationalRate(Integer nationalRate) {
25. this.nationalRate = nationalRate;
26. }
27. public Integer getLocalRate() {
28. return localRate;
29. }
30. public void setLocalRate(Integer localRate) {
31. this.localRate = localRate;
32. }
33. public PlanDTO() {
34. super();
35. }
36. @Override
37. public String toString() {
38. return "PlanDTO [planId=" + planId + ", planName=" + planName + ", nationalRate=" +
    nationalRate + ", localRate=" + localRate + "]";
39. }
40. }
```

**Step6:** Create classes **ExceptionHandlerAdvice** and **NoSuchCustomerException** under **com.infytel.exception** package:

```
1. package com.infytel.exceptions;
2. import org.springframework.http.HttpStatus;
3. import org.springframework.http.ResponseEntity;
4. import org.springframework.web.bind.annotation.ExceptionHandler;
5. import org.springframework.web.bind.annotation.RestControllerAdvice;
6. import com.infytel.dto.ErrorMessage;
7. @RestControllerAdvice
8. public class ExceptionControllerAdvice {
9.     @ExceptionHandler(Exception.class)
10.    public String exceptionHandler(Exception ex) {
11.    return ex.getMessage();
12.    }
13.    @ExceptionHandler(NoSuchCustomerException.class)
14.    public ResponseEntity<ErrorMessage> exceptionHandler2(NoSuchCustomerException ex) {
15.    ErrorMessage error = new ErrorMessage();
16.    error.setErrorCode(HttpStatus.BAD_GATEWAY.value());
17.    error.setMessage(ex.getMessage());
18.    return new ResponseEntity<>(error, HttpStatus.OK);
19.    }
20. }
```

```
1. package com.infytel.exceptions;
2. public class NoSuchCustomerException extends Exception {
3.     private static final long serialVersionUID = 1L;
4.     public NoSuchCustomerException() {
5.         super();
6.     }
7.     public NoSuchCustomerException(String errors) {
8.         super(errors);
9.     }
10. }
```

**Step 7:** Create class **CustomerRepository** under com.infytel.repository package:

```
1. package com.infytel.repository;
2. import java.util.ArrayList;
3. import java.util.List;
4. import javax.annotation.PostConstruct;
5. import org.springframework.stereotype.Repository;
6. import com.infytel.dto.CustomerDTO;
7. import com.infytel.dto.FriendFamilyDTO;
8. import com.infytel.dto.PlanDTO;
9. import com.infytel.exceptions.NoSuchCustomerException;
10. @Repository
11. public class CustomerRepository {
12.     List<CustomerDTO> customers = null;
13.     // Populates customer in hard-coded way
14.     @PostConstruct
15.     public void initializer() {
16.         CustomerDTO customerDTO = new CustomerDTO();
17.         PlanDTO planDTO = new PlanDTO();
18.         planDTO.setPlanId(1);
19.         planDTO.setPlanName("Simple");
20.         planDTO.setLocalRate(3);
21.         planDTO.setNationalRate(5);
22.         customerDTO.setAddress("Chennai");
23.         customerDTO.setAge(18);
24.         customerDTO.setCurrentPlan(planDTO);
25.         customerDTO.setGender('m');
26.         customerDTO.setName("Jack");
27.         customerDTO.setEmail("Jack@infy.com");
28.         customerDTO.setPassword("ABC@123");
29.         customerDTO.setPhoneNo(99512122221);
```

```
30. List<FriendFamilyDTO> friendAndFamily = new ArrayList<>();
31. friendAndFamily.add(new FriendFamilyDTO(customerDTO.getPhoneNo(), 800000145));
32. friendAndFamily.add(new FriendFamilyDTO(customerDTO.getPhoneNo(), 700000145));
33. customerDTO.setFriendAndFamily(friendAndFamily);
34. customers = new ArrayList<>();
35. customers.add(customerDTO);
36. }
37. // creates customer
38. public String createCustomer(CustomerDTO customerDTO) {
39. customers.add(customerDTO);
40. return "Customer with " + customerDTO.getPhoneNo() + " added successfully";
41. }
42. // fetches customer
43. public List<CustomerDTO> fetchCustomer() {
44. return customers;
45. }
46. // deletes customer - exception handling incorporated
47. public String deleteCustomer(long phoneNumber) throws NoSuchCustomerException {
48. boolean notfound = true;
49. String response = "Customer of:" + phoneNumber + "\t does not exist";
50. for (CustomerDTO customer : customers) {
51. if (customer.getPhoneNo() == phoneNumber) {
52. customers.remove(customer);
53. response = customer.getName() + " with phoneNumber " + customer.getPhoneNo() + " deleted
    successfully";
54. notfound = false;
55. break;
56. }
57. }
58. if (notfound)
59. throw new NoSuchCustomerException("Customer does not exist :" + phoneNumber);
60. return response;
61. }
62. // updates customer
63. public String updateCustomer(long phoneNumber, CustomerDTO customerDTO) {
64. String response = "Customer of:" + phoneNumber + "\t does not exist";
65. for (CustomerDTO customer : customers) {
66. if (customer.getPhoneNo() == phoneNumber) {
67. if (customerDTO.getName() != null)
68. customer.setName(customerDTO.getName());
69. if (customerDTO.getAddress() != null)
70. customer.setAddress(customerDTO.getAddress());
```

```
71. if (customerDTO.getPassword() != null)
72. customer.setPassword(customerDTO.getPassword());
73. customers.set(customers.indexOf(customer), customer);
74. response = "Customer of phoneNo" + customer.getPhoneNo() + "\t got updated successfully";
75. break;
76. }
77. }
78. return response;
79. }
80. }
```

**Step 8:** Create class **CustomerService** under **com.infytel.service** package:

```
1. package com.infytel.service;
2. import java.util.List;
3. import org.springframework.beans.factory.annotation.Autowired;
4. import org.springframework.stereotype.Service;
5. import com.infytel.dto.CustomerDTO;
6. import com.infytel.exceptions.NoSuchCustomerException;
7. import com.infytel.repository.CustomerRepository;
8. @Service
9. public class CustomerService {
10. @Autowired
11. private CustomerRepository customerRepository;
12. // Contacts repository layer to add customer
13. public String createCustomer(CustomerDTO customerDTO) {
14. return customerRepository.createCustomer(customerDTO);
15. }
16. // Contacts repository layer to fetch customer
17. public List<CustomerDTO> fetchCustomer() {
18. return customerRepository.fetchCustomer();
19. }
20. // Contacts repository layer to delete customer
21. public String deleteCustomer(long phoneNo) throws NoSuchCustomerException {
22. return customerRepository.deleteCustomer(phoneNo);
23. }
24. // Contacts repository layer to update customer
25. public String updateCustomer(long phoneNo, CustomerDTO customerDTO) {
26. return customerRepository.updateCustomer(phoneNo, customerDTO);
27. }
28. }
```

**Step 9:** Add the following content to **application.properties** file:

1. server.port=8090
2. server.servlet.context-path=/infytel-8b

**Step 10:** Make sure that the project's pom.xml looks similar to the one that is shown below.

1. <?xml version="1.0" encoding="UTF-8"?>
2. <project xmlns="http://maven.apache.org/POM/4.0.0"
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
5. <modelVersion>4.0.0</modelVersion>
6. <parent>
7. <groupId>org.springframework.boot</groupId>
8. <artifactId>spring-boot-starter-parent</artifactId>
9. <version>2.1.4.RELEASE</version>
10. <relativePath /> <!-- lookup parent from repository -->
11. </parent>
12. <groupId>infytel</groupId>
13. <artifactId>infytel\_demo8\_b</artifactId>
14. <version>0.0.1-SNAPSHOT</version>
15. <name>infytel\_demo8\_b</name>
16. <description>REST client demo B</description>
17. <properties>
18. <java.version>1.8</java.version>
19. </properties>
20. <dependencies>
21. <dependency>
22. <groupId>org.springframework.boot</groupId>
23. <artifactId>spring-boot-starter-web</artifactId>
24. </dependency>
25. <dependency>
26. <groupId>org.springframework.boot</groupId>
27. <artifactId>spring-boot-starter-test</artifactId>
28. <scope>test</scope>
29. </dependency>
30. </dependencies>
31. <build>
32. <plugins>
33. <plugin>
34. <groupId>org.springframework.boot</groupId>
35. <artifactId>spring-boot-maven-plugin</artifactId>
36. </plugin>
37. </plugins>
38. </build>



39. </project>

**Step 11:** Deploy the application on the server by executing the class containing the main method.

So, we have successfully created a REST client that consumes the REST endpoint that exposes the plan details.

Now, let us see how can we test the same using Postman client.

### Output

Testing Web Service using Postman

**Step 1:** Launch Postman.

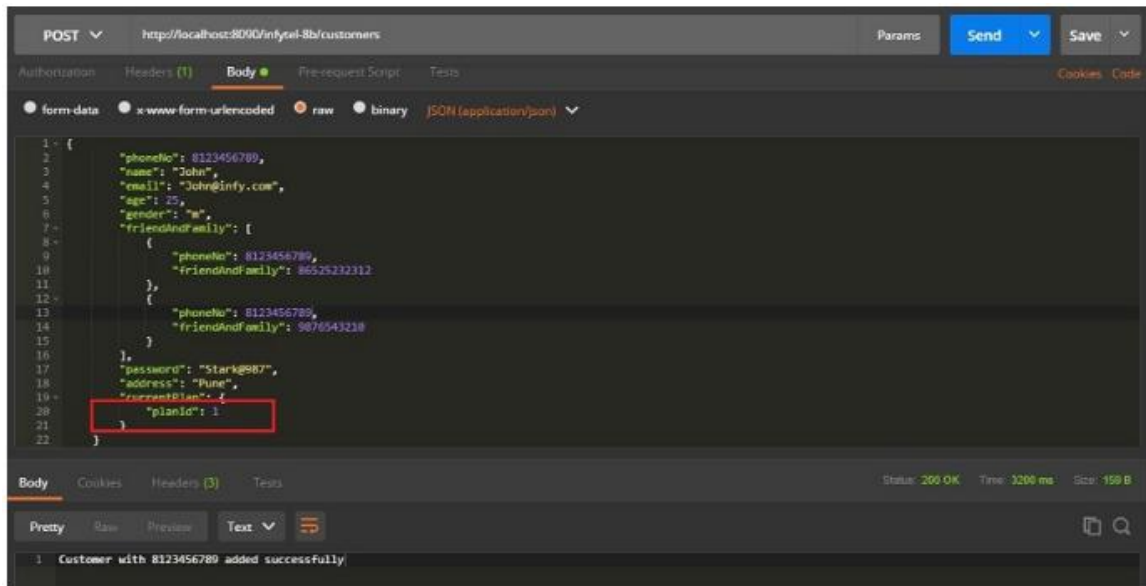
**Step 2:** Test this URL - **http://localhost:8090/infytel-8b/customers** using HTTP GET to check the existing customers in infytel.



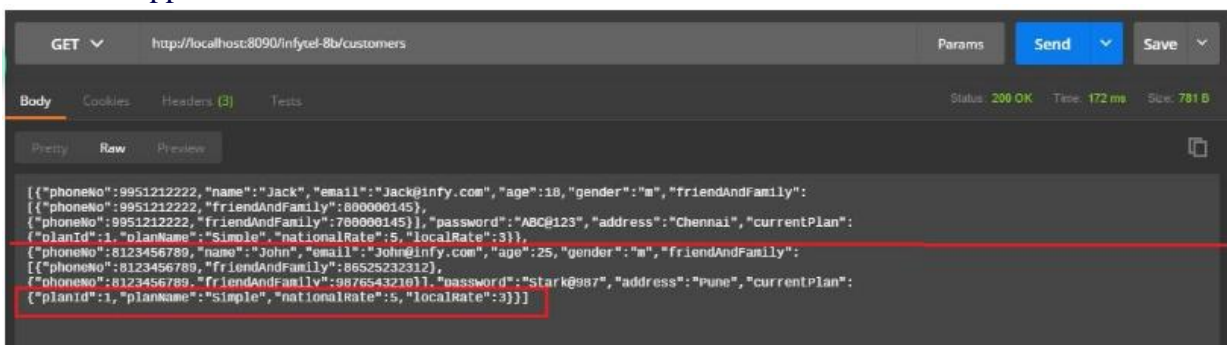
**Step 3:** Adding a New Customer Using **http://localhost:8090/infytel-8b/customers** and the sample data is as given below

```
1. {
2.   "phoneNo": 8123456789,
3.   "name": "John",
4.   "email": "John@infy.com",
5.   "age": 25,
6.   "gender": "m",
7.   "friendAndFamily": [
8.     {
9.       "phoneNo": 8123456789,
10.      "friendAndFamily": 86525232312
11.    },
12.    {
13.      "phoneNo": 8123456789,
14.      "friendAndFamily": 9876543210
15.    }
16.  ],
17.  "password": "Stark@987",
18.  "address": "Pune",
19.  "currentPlan": {
20.    "planId": 1
21.  }
22. }
```

You will get the following response.



**Step 4:** Verify if the new customer is added and also check the complete plan details chosen by the customers appears or not.



Observe that in the customer profile, the complete plan details are appearing as the customer REST resource consumed the plan REST resource to populate the customer details while creating a new Customer.

### Versioning a REST endpoint

Versioning a service becomes important when we want to create a new version of an already existing service. But at the same time, we need to support the earlier version as well, to ensure backward compatibility.

There are different ways of achieving API versioning such as

- URI versioning
- **Request Parameter Versioning**
- Custom Header Versioning
- Accept Header versioning

Let us take a look at each one of them.

**Note:** Request Parameter Versioning is used in the demos

#### URI versioning

URI versioning involves different URI for every newer version of the API.

**Example:** There is a need for two versions of functionality that fetches the plan details of Infytel telecom application. And, this fetch operation is based on the planId being passed as a path variable.

**version-1:** Fetches the complete plan details that include plainId, planName, localRate and nationalRate

**version-2:** Fetches only the localRate and nationalRate.

Below is the URI proposed to be used

1. GET http://localhost:8080/infytel/plans/v1/{planId}
2. eg:http://localhost:8080/infytel/plans/v1/1
3. GET http://localhost:8080/infytel/plans/v2/{planId}
4. eg:http://localhost:8080/infytel/plans/v2/1

Now, let us look at how the handler method can be written.

For Version-1

1. @GetMapping(value = "v1/{planId}")
2. public ResponseEntity<PlanDTO> getPlan(@PathVariable("planId")String planId)
3. {
4. //Return the complete plan details
5. }

For Version-2

1. @GetMapping(value = "v2/{planId}")
2. public ResponseEntity<PlanDTO> getPlanv2(@PathVariable("planId")String planId)
3. {
4. //Returns only the localRate and nationalRate
5. }

Observe, how a functionality with two different versions or approaches is made available in a REST resource. Also, focus on how the consumers call the different versions of the API method using the URIs that carry information about versioning.

### Versioning with Custom Headers

This type of versioning requires the inclusion of custom headers in the request URI to map to the correct version of the API endpoint.

Let us take the service that fetches plan details, here as well.

Below is the URI proposed to be used

1. GET http://localhost:8080/infytel/plans/{planId}
2. headers=[X-API-VERSION=1]
3. eg:http://localhost:8080/infytel/plans/1 and pass the header information headers=[X-API-VERSION=1]
4. GET http://localhost:8080/infytel/plans/{planId}
5. headers=[X-API-VERSION=2]

- eg:http://localhost:8080/infytel/plans/1 and pass the header information headers=[X-API-VERSION=2]

Now, let us look at how the handler method can be written

For Version-1

1. @GetMapping(value = "{planId}",headers = "X-API-VERSION=1")
2. public ResponseEntity<PlanDTO> getPlan(@PathVariable("planId")String planId)
3. {
4. //Return the complete plan details
5. }

For Version-2

1. @GetMapping(value = "{planId}",headers = "X-API-VERSION=2")
2. public ResponseEntity<PlanDTO> getPlanv2(@PathVariable("planId")String planId)
3. {
4. //Returns only the localRate and nationalRate
5. }

Observe, how a functionality with two different versions or approaches is made available in a REST resource. Also, focus on how the consumers call the different versions of the API method using the URIs that carry information about versioning as part of the request headers.

### Accept Header Versioning

Accept Header Versioning approach is one other way of versioning the API methods that insists the usage of Accept Header in the request.

1. GET http://localhost:8080/infytel/plans/{planId}
2. headers=[Accept=application/vnd.plans.app-v1+json]
3. eg:http://localhost:8080/infytel/plans/1 and pass the header information headers=[Accept=application/vnd.plans.app-v1+json]
4. GET http://localhost:8080/infytel/plans/{planId}
5. headers=[Accept=application/vnd.plans.app-v2+json]
6. eg:http://localhost:8080/infytel/plans/1 and pass the header information headers=[Accept=application/vnd.plans.app-v2+json]

Now, let us look at how the handler method can be written

For Version-1

1. @GetMapping(value = "{planId}", produces = "application/vnd.plans.app-v1+json")
2. public ResponseEntity<PlanDTO> getPlan(@PathVariable("planId")String planId)
3. {
4. //Return the complete plan details
5. }

**For Version-2**

```
1. @GetMapping(value =("/{planId}", produces = "application/vnd.plans.app-v2+json")
2. public ResponseEntity<PlanDTO> getPlanv2(@PathVariable("planId")String planId)
3. {
4. //Returns only the localRate and nationalRate
5. }
```

Observe, how a functionality with two different versions or approaches is made available in a REST resource. Also, focus on how the consumers call the different versions of the API method using the URIs that carry information about versioning as part of the request headers named, **accept**. One more important thing to be noted here is the inclusion of custom MIME type.

**Request Parameter versioning**

Request parameter versioning can be achieved by providing the request parameter in the URI that holds the version details.

```
1. GET http://localhost:8080/infytel/plans/{planId}?version=
2. eg:http://localhost:8080/infytel/plans/1?version=1
3. GET http://localhost:8080/infytel/plans/{planId}?version=
4. eg:http://localhost:8080/infytel/plans/1?version=2
```

Now, let us look at how the handler method can be written

**For Version-1**

```
1. @GetMapping(value =("/{planId}", params = "version=1")
2. public ResponseEntity<PlanDTO> getPlan(@PathVariable("planId")String planId)
3. {
4. //Return the complete plan details
5. }
```

**For Version-2**

```
6. @GetMapping(value =("/{planId}", params = "version=2")
7. public ResponseEntity<PlanDTO> getPlanv2(@PathVariable("planId")String planId)
8. {
9. //Returns only the localRate and nationalRate
10. }
```

Observe, how a functionality with two different versions or approaches is made available in a REST resource. Also, focus on how the consumers call the different versions of the API method using the URIs that carry information about versioning in the form of request parameter.

**Demo - Versioning a Spring REST endpoint**

**Objectives:** To create a Spring REST API with multiple versions of the same functionality. We will learn,

- To create different versions of a REST method which can be targetted with the same URI that holds, request parameter to differentiate among the versions.

**Scenario:**An online telecom app called Infytel wishes its functionalities to get exposed as RESTful services. It has three controllers in total to deal with customer, plan and call details, among which, the focus will be on the controller named PlanDetailsController.

And, below is the method taken for versioning.

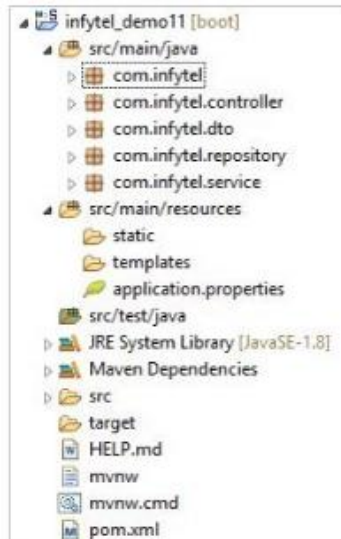
Controller class	method	URI	HTTP Method	Remarks
PlanController	fetchPlanById()	/plans/{planId}?version=	GET	If version query string is supplied with value 1 enters this method
PlanController	fetchPlanByIdV2()	/plans/{planId}?version=	GET	If version query string is supplied with value 2 enters this method

Notice that the URI is the same for both the versions, except the value that will be mentioned for the request parameter.

### Steps:

**Step 1:** Create a Maven project using Spring Initializer with web dependency and import the same in STS.

**Step 2:** Modify the imported project according to the following project structure:



**Step 3:** Look at the class **InfytelDemo11Application** under **com.infytel** package that gets created automatically when the project is generated. We will modify this only when we need to go with custom configuration.

```
1. package com.infytel;
2. import org.springframework.boot.SpringApplication;
3. import org.springframework.boot.autoconfigure.SpringBootApplication;
4. @SpringBootApplication
5. public class InfytelDemo11Application {
6.     public static void main(String[] args) {
```

```
7. SpringApplication.run(InfytelDemo11Application.class, args);
8. }
9. }
```

**Step 4:** Create class **PlanController** under com.infytel.controller package:

```
1. package com.infytel.controller;
2. import java.util.Map;
3. import org.springframework.beans.factory.annotation.Autowired;
4. import org.springframework.web.bind.annotation.GetMapping;
5. import org.springframework.web.bind.annotation.PathVariable;
6. import org.springframework.web.bind.annotation.RequestMapping;
7. import org.springframework.web.bind.annotation.RestController;
8. import com.infytel.dto.PlanDTO;
9. import com.infytel.service.PlanService;
10. @RestController
11. @RequestMapping("/plans")
12. public class PlanController {
13.     @Autowired
14.     private PlanService planService;
15.     // method for getting all the available plans will go here
16.     // and
17.     // method for getting the plans based on local rate will go here
18.     // fetching the plans by id
19.     @GetMapping(value =("/{planId}", params = "version=1")
20.     public PlanDTO fetchPlanById(@PathVariable("planId") int planId) {
21.         return planService.fetchPlanById(planId);
22.     }
23.     @GetMapping(value =("/{planId}", params = "version=2")
24.     public Map<String, Integer> fetchPlanByIdv2(@PathVariable("planId") int planId) {
25.         return planService.fetchPlanByIdv2(planId);
26.     }
27. }
```

**Step 5:** Create class **PlanDTO** under com.infytel.dto package:

```
1. package com.infytel.dto;
2. public class PlanDTO {
3.     Integer planId;
4.     String planName;
5.     Integer nationalRate;
6.     Integer localRate;
7.     public Integer getPlanId() {
8.         return planId;
9.     }
```

```
10. public void setPlanId(Integer planId) {
11. this.planId = planId;
12. }
13. public String getPlanName() {
14. return planName;
15. }
16. public void setPlanName(String planName) {
17. this.planName = planName;
18. }
19. public Integer getNationalRate() {
20. return nationalRate;
21. }
22. public void setNationalRate(Integer nationalRate) {
23. this.nationalRate = nationalRate;
24. }
25. public Integer getLocalRate() {
26. return localRate;
27. }
28. public void setLocalRate(Integer localRate) {
29. this.localRate = localRate;
30. }
31. public PlanDTO() {
32. super();
33. }
34. @Override
35. public String toString() {
36. return "PlanDTO [planId=" + planId + ", planName=" + planName + ", nationalRate=" +
    nationalRate + ", localRate=" + localRate + "]";
37. }
38. }
```

**Step 6:** Create class **PlanRepository** under com.infytel.repository package:

```
1. package com.infytel.repository;
2. import java.util.ArrayList;
3. import java.util.HashMap;
4. import java.util.List;
5. import java.util.Map;
6. import java.util.Optional;
7. import javax.annotation.PostConstruct;
8. import org.springframework.stereotype.Repository;
9. import com.infytel.dto.PlanDTO;
10. @Repository
```



```
11. public class PlanRepository {
12. private List<PlanDTO> plans;
13. // Populating a list of plans
14. @PostConstruct
15. public void populatePlans() {
16. plans = new ArrayList<>();
17. PlanDTO plan1 = new PlanDTO();
18. plan1.setPlanId(1);
19. plan1.setPlanName("Simple");
20. plan1.setLocalRate(3);
21. plan1.setNationalRate(5);
22. plans.add(plan1);
23. PlanDTO plan2 = new PlanDTO();
24. plan2.setPlanId(2);
25. plan2.setPlanName("Medium");
26. plan2.setLocalRate(5);
27. plan2.setNationalRate(8);
28. plans.add(plan2);
29. }
30. // methods fetchPlans() and plansLocalRate() go here
31. // fetching plan by id
32. public PlanDTO fetchPlanById(int planId) {
33. Optional<PlanDTO> optionalPlanDTO = plans.stream().filter(x -> x.getPlanId() ==
    planId).findFirst();
34. return optionalPlanDTO.orElse(plans.get(0)); // if the optional contains a value, fine. Else the first
    plan
35. // available in the list will be set
36. }
37. public Map<String, Integer> fetchPlanByIdv2(int planId) {
38. Map<String, Integer> rates = new HashMap<>();
39. Optional<PlanDTO> optionalPlanDTO = plans.stream().filter(x -> x.getPlanId() ==
    planId).findFirst();
40. rates.put("localRate", optionalPlanDTO.orElse(plans.get(0)).getLocalRate());
41. rates.put("nationalRate", optionalPlanDTO.orElse(plans.get(0)).getNationalRate());
42. return rates;
43. }
44. }
```

**Step 7:** Create class **PlanService** under com.infytel.service package:

```
1. package com.infytel.service;
2. import java.util.Map;
3. import org.springframework.beans.factory.annotation.Autowired;
```

```
4. import org.springframework.stereotype.Service;
5. import org.springframework.web.bind.annotation.PathVariable;
6. import com.infytel.dto.PlanDTO;
7. import com.infytel.repository.PlanRepository;
8. @Service
9. public class PlanService {
10. @Autowired
11. private PlanRepository planRepository;
12. // methods fetchPlans() and plansLocalRate() go here
13. public PlanDTO fetchPlanById(int planId) {
14. return planRepository.fetchPlanById(planId);
15. }
16. public Map<String, Integer> fetchPlanByIdv2(@PathVariable("planId") int planId) {
17. return planRepository.fetchPlanByIdv2(planId);
18. }
19. }
```

**Step 8:** Add the following content to **application.properties** file:

```
1. server.port=8080
2. server.servlet.context-path=/infytel-11
```

**Step 9:** Make sure that the project's **pom.xml** looks similar to the one shown below.

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <project xmlns="http://maven.apache.org/POM/4.0.0"
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
5. <modelVersion>4.0.0</modelVersion>
6. <parent>
7. <groupId>org.springframework.boot</groupId>
8. <artifactId>spring-boot-starter-parent</artifactId>
9. <version>2.1.4.RELEASE</version>
10. <relativePath /> <!-- lookup parent from repository -->
11. </parent>
12. <groupId>infytel</groupId>
13. <artifactId>infytel_demo11</artifactId>
14. <version>0.0.1-SNAPSHOT</version>
15. <name>infytel_demo11</name>
16. <description>Versioning Spring REST API</description>
17. <properties>
18. <java.version>1.8</java.version>
19. </properties>
20. <dependencies>
```

```
21. <dependency>
22. <groupId>org.springframework.boot</groupId>
23. <artifactId>spring-boot-starter-web</artifactId>
24. </dependency>
25. <dependency>
26. <groupId>org.springframework.boot</groupId>
27. <artifactId>spring-boot-starter-test</artifactId>
28. <scope>test</scope>
29. </dependency>
30. </dependencies>
31. <build>
32. <plugins>
33. <plugin>
34. <groupId>org.springframework.boot</groupId>
35. <artifactId>spring-boot-maven-plugin</artifactId>
36. </plugin>
37. </plugins>
38. </build>
39. </project>
```

**Step 10:** Deploy the application on the server by executing the class containing the main method.

So, we have successfully created and deployed the REST endpoints. Now, let us see how the same can be tested using any standard web browser.

## Output

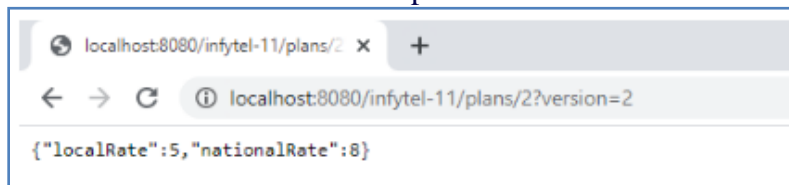
### Testing Web Service

**Step 1:** Open any standard web browser.

**Step 2:** Test this URL - **http://localhost:8080/infytel-11/plans/2?version=1**. Following will be the response that contains the plan details with all the fields.



**Step 3:** Test this URL - **http://localhost:8080/infytel-11/plans/2?version=2**. Only the local and national rates of plan details will be rendered as output.



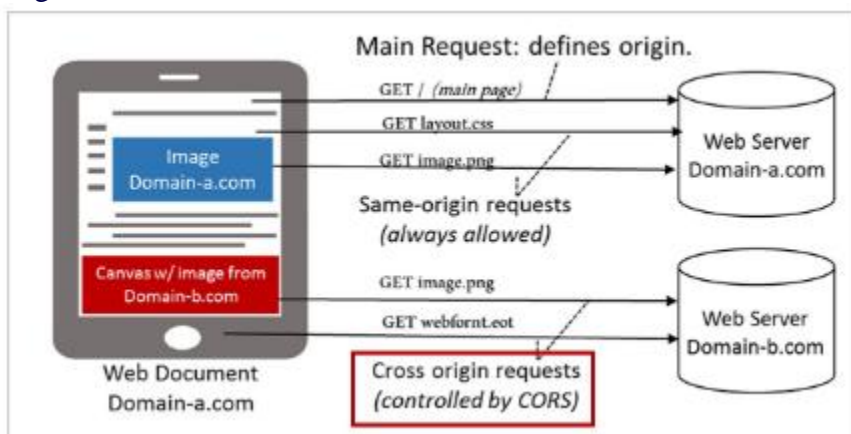
## Enabling CORS in Spring REST

## Why CORS?

Typically, scripts running in a web application of one origin (domain) cannot have access to the resources of a server available at a different origin. If the clients of different origins try to access the REST endpoints of an application, CORS error will be thrown simply. And, below is a sample of the same.

1. Cross-Origin Request Blocked: The Same Origin Policy disallows
2. reading the remote resource at https://domain-1.com

Look at the image below, which depicts the request for resources by the clients of the same origin and different origins as well.



Request for resources from a different domain will encounter CORS error.

To overcome this error, we need to enable cross-origin requests at the server-side so that the browsers which run the client code can allow scripts to make a Cross-origin call.

Let us see, how can it be done in Spring REST.

## How to enable CORS?

Cross-origin calls can be enabled for Spring REST resources in three ways by applying:

- CORS configuration on the REST controller methods
- CORS configuration on the REST controller itself
- CORS configuration globally (common to the entire application)

### CORS configuration on controller methods:

In this style, we need to apply `@CrossOrigin` on the handler method.

Here, the annotation, `@CrossOrigin` enables cross-origin requests only for the specific handler method.

By default, it allows all origins, all headers, all HTTP methods specified in the `@RequestMapping` annotation. Also, a lot more other defaults are available.

We can customize this behavior by specifying the value for the following attributes that are available with Http method mapping annotations (`@GetMapping`, `@PostMapping`, etc.,)

- origins - list of allowed origins to access the method

- methods - list of supported HTTP request methods
- allowedHeaders - list of request headers which can be used during the request
- exposedHeaders - list of response headers which the browser can allow the clients to access
- allowCredentials - determines whether the browser can include cookies that are associated with the request

**Example:**

```
1. @Controller
2. @RequestMapping(path="/customers")
3. public class CustomerController
4. {
5.     @CrossOrigin(origins = "*", allowedHeaders = "*")
6.     @GetMapping()
7.     public String homeInit(Model model) {
8.         return "home";
9.     }
10. }
```

**How to enable CORS?****CORS configuration on the controller:**

It is possible to add @CrossOrigin at the controller level as well, to enable CORS for all the handler methods of that particular controller.

```
1. @Controller
2. @CrossOrigin(origins = "*", allowedHeaders = "*")
3. @RequestMapping(path="/customers")
4. public class CustomerController
5. {
6.     @GetMapping()
7.     public String homeInit(Model model) {
8.         return "home";
9.     }
10. }
```

**Global CORS configuration:**

As an alternative to fine-grained, annotation-based configuration, we can also go for global CORS configuration as well.

Look at the code below. The starter class has been modified to implement WebMvcConfigurer and override addCorsMappings() method that takes the CorsRegistry object as argument using which we can configure the allowed set of domains and HTTP methods as well.

```
1. @SpringBootApplication
2. public class InfytelApplication implements WebMvcConfigurer {
3.     public static void main(String[] args) {
4.         SpringApplication.run(InfytelDemo8BApplication.class, args);
```

```

5.  }
6.  @Override
7.  public void addCorsMappings(CorsRegistry registry) {
8.      registry.addMapping("/**").allowedMethods("GET", "POST");
9.  }
10. }

```

In our demo, we will enable CORS at the global level.

### Demo - Enabling CORS in a Spring REST application

**Objectives:** To enable CORS in a Spring REST application. We will learn,

- How to enable CORS at the global level to make all the REST endpoints available in an application, have support for CORS.

**Scenario:** An online telecom app called Infytel is exposing its "customer management profile" as a RESTful resource that provides provisions to create, fetch, delete and update customer details.

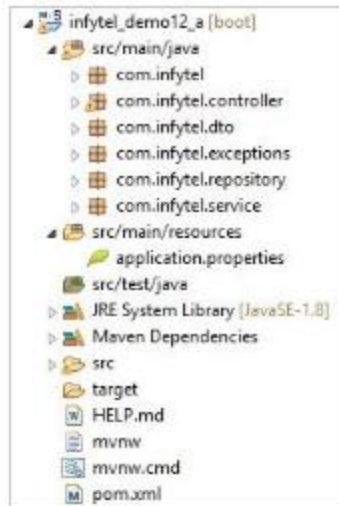
Controller Class	methods	URI	HTTP Method	Remarks
CustomerController	fetchCustomer()	/customers	GET	To fetch all the Infytel customers
CustomerController	createCustomer()	/customers	POST	To create a new customer in Infytel
CustomerController	deleteCustomer()	/customers/{phoneNumber}	DELETE	To delete an existing customer of Infytel identified by phoneNumber
CustomerController	updateCustomer()	/customers/{phoneNumber}	PUT	To update an existing customer of Infytel identified by phoneNumber

- Concentrate on InfytelDemo12AApplication.java class which is our starter class annotated with @SpringBootApplication. Look at the interface it implements and the method it overrides to enable CORS support. With this support, all origins are allowed to access the REST resources of Infytel that are tagged with GET and POST mappings.

### Steps:

**Step 1:** Create a Maven project using Spring Initializer with web dependency and import the same in STS.

**Step 2:** Modify the imported project according to the following project structure:



**Step 3:** Look at the class **InfytelDemo12AApplication** under **com.infytel** package that gets created automatically when the project is generated. We will modify this only when we need to go with custom configuration. And, CORS configuration is included here.

```
1. package com.infytel;
2. import org.springframework.boot.SpringApplication;
3. import org.springframework.boot.autoconfigure.SpringBootApplication;
4. import org.springframework.web.servlet.config.annotation.CorsRegistry;
5. import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
6. @SpringBootApplication
7. public class InfytelDemo12AApplication implements WebMvcConfigurer {
8.     public static void main(String[] args) {
9.         SpringApplication.run(InfytelDemo12AApplication.class, args);
10.    }
11.    //Code for CORS configuration
12.    @Override
13.    public void addCorsMappings(CorsRegistry registry) {
14.        registry.addMapping("/*").allowedMethods("GET", "POST");
15.    }
16. }
```

**Step 4:** Create class **CustomerController** under **com.infytel.controller** package:

```
1. package com.infytel.controller;
2. import java.util.List;
3. import java.util.stream.Collectors;
4. import javax.validation.Valid;
5. import org.springframework.beans.factory.annotation.Autowired;
6. import org.springframework.http.HttpStatus;
7. import org.springframework.http.ResponseEntity;
8. import org.springframework.validation.Errors;
9. import org.springframework.validation.ObjectError;
10. import org.springframework.web.bind.annotation.DeleteMapping;
```

```
11. import org.springframework.web.bind.annotation.GetMapping;
12. import org.springframework.web.bind.annotation.PathVariable;
13. import org.springframework.web.bind.annotation.PostMapping;
14. import org.springframework.web.bind.annotation.PutMapping;
15. import org.springframework.web.bind.annotation.RequestBody;
16. import org.springframework.web.bind.annotation.RequestMapping;
17. import org.springframework.web.bind.annotation.RestController;
18. import com.infytel.dto.CustomerDTO;
19. import com.infytel.dto.ErrorMessage;
20. import com.infytel.exceptions.NoSuchCustomerException;
21. import com.infytel.service.CustomerService;
22. @RestController
23. @RequestMapping("/customers")
24. public class CustomerController {
25.     @Autowired
26.     private CustomerService customerService;
27.     // Fetching customer details
28.     @GetMapping(produces = "application/json")
29.     public List<CustomerDTO> fetchCustomer() {
30.         return customerService.fetchCustomer();
31.     }
32.     // Adding a customer
33.     @PostMapping(consumes = "application/json")
34.     public ResponseEntity createCustomer(@Valid @RequestBody CustomerDTO customerDTO,
        Errors errors) {
35.         String response = "";
36.         if (errors.hasErrors()) {
37.             response = errors.getAllErrors().stream().map(ObjectError::getDefaultMessage)
38.                 .collect(Collectors.joining(", "));
39.             ErrorMessage error = new ErrorMessage();
40.             error.setErrorCode(HttpStatus.NOT_ACCEPTABLE.value());
41.             error.setMessage(response);
42.             return ResponseEntity.ok(error);
43.         } else {
44.             response = customerService.createCustomer(customerDTO);
45.             return ResponseEntity.ok(response);
46.         }
47.     }
48.     // Updating an existing customer
49.     @PutMapping(value =("/{phoneNumber}", consumes = "application/json")
50.     public String updateCustomer(@PathVariable("phoneNumber") long phoneNumber,
        @RequestBody CustomerDTO customerDTO) {
```



```
51. return customerService.updateCustomer(phoneNumber, customerDTO);
52. }
53. // Deleting a customer
54. @DeleteMapping(value =("/{phoneNumber}", produces = "text/html")
55. public String deleteCustomer(@PathVariable("phoneNumber") long phoneNumber) throws
    NoSuchCustomerException {
56. return customerService.deleteCustomer(phoneNumber);
57. }
58. }
```

**Step 5:** Create classes **CustomerDTO**, **ErrorMessage**, **FriendFamilyDTO** and **PlanDTO** under **com.infytel.dto** package:

```
1. package com.infytel.dto;
2. import java.util.List;
3. import javax.validation.constraints.Email;
4. public class CustomerDTO {
5. long phoneNo;
6. String name;
7. @Email(message = "Email id is not in format, please check")
8. String email;
9. public String getEmail() {
10. return email;
11. }
12. public void setEmail(String email) {
13. this.email = email;
14. }
15. int age;
16. char gender;
17. List<FriendFamilyDTO> friendAndFamily;
18. String password;
19. String address;
20. PlanDTO currentPlan;
21. public PlanDTO getCurrentPlan() {
22. return currentPlan;
23. }
24. public void setCurrentPlan(PlanDTO currentPlan) {
25. this.currentPlan = currentPlan;
26. }
27. public String getPassword() {
28. return password;
29. }
30. public void setPassword(String password) {
```

```
31. this.password = password;
32. }
33. public String getAddress() {
34. return address;
35. }
36. public void setAddress(String address) {
37. this.address = address;
38. }
39. public List<FriendFamilyDTO> getFriendAndFamily() {
40. return friendAndFamily;
41. }
42. public void setFriendAndFamily(List<FriendFamilyDTO> friendAndFamily) {
43. this.friendAndFamily = friendAndFamily;
44. }
45. public long getPhoneNo() {
46. return phoneNo;
47. }
48. public void setPhoneNo(long phoneNo) {
49. this.phoneNo = phoneNo;
50. }
51. public String getName() {
52. return name;
53. }
54. public void setName(String name) {
55. this.name = name;
56. }
57. public int getAge() {
58. return age;
59. }
60. public void setAge(int age) {
61. this.age = age;
62. }
63. public char getGender() {
64. return gender;
65. }
66. public void setGender(char gender) {
67. this.gender = gender;
68. }
69. @Override
70. public String toString() {
```

```
71. return "CustomerDTO [phoneNo=" + phoneNo + ", name=" + name + ", age=" + age + ", gender="
    + gender + ", friendAndFamily=" + friendAndFamily + ", password=" + password + ", address=" +
    address + "]";
72. }
73. }
```

```
1. package com.infytel.dto;
2. public class ErrorMessage {
3.     private int errorCode;
4.     private String message;
5.     public int getErrorCode() {
6.         return errorCode;
7.     }
8.     public void setErrorCode(int errorCode) {
9.         this.errorCode = errorCode;
10.    }
11.    public String getMessage() {
12.        return message;
13.    }
14.    public void setMessage(String message) {
15.        this.message = message;
16.    }
17. }

1. package com.infytel.dto;
2. public class FriendFamilyDTO {
3.     long phoneNo;
4.     long friendAndFamily;
5.     public long getPhoneNo() {
6.         return phoneNo;
7.     }
8.     public void setPhoneNo(long phoneNo) {
9.         this.phoneNo = phoneNo;
10.    }
11.    public long getFriendAndFamily() {
12.        return friendAndFamily;
13.    }
14.    public void setFriendAndFamily(long friendAndFamily) {
15.        this.friendAndFamily = friendAndFamily;
16.    }
17.    public FriendFamilyDTO(long phoneNo, long friendAndFamily) {
18.        this();
19.        this.phoneNo = phoneNo;
```

```
20. this.friendAndFamily = friendAndFamily;
21. }
22. public FriendFamilyDTO() {
23. super();
24. }
25. @Override
26. public String toString() {
27. return "FriendFamilyDTO [phoneNo=" + phoneNo + ", friendAndFamily=" + friendAndFamily +
    "]" ;
28. }
29. }
```

```
1. package com.infytel.dto;
2. import javax.xml.bind.annotation.XmlRootElement;
3. @XmlRootElement
4. public class PlanDTO {
5. Integer planId;
6. String planName;
7. Integer nationalRate;
8. Integer localRate;
9. public Integer getPlanId() {
10. return planId;
11. }
12. public void setPlanId(Integer planId) {
13. this.planId = planId;
14. }
15. public String getPlanName() {
16. return planName;
17. }
18. public void setPlanName(String planName) {
19. this.planName = planName;
20. }
21. public Integer getNationalRate() {
22. return nationalRate;
23. }
24. public void setNationalRate(Integer nationalRate) {
25. this.nationalRate = nationalRate;
26. }
27. public Integer getLocalRate() {
28. return localRate;
29. }
30. public void setLocalRate(Integer localRate) {
```

```
31. this.localRate = localRate;
32. }
33. public PlanDTO() {
34. super();
35. }
36. @Override
37. public String toString() {
38. return "PlanDTO [planId=" + planId + ", planName=" + planName + ", nationalRate=" +
    nationalRate + ", localRate=" + localRate + "]";
39. }
40. }
```

**Step 6:** Create classes **ExceptionHandlerAdvice** and **NoSuchCustomerException** under **com.infytel.exception** package:

```
1. package com.infytel.exceptions;
2. import org.springframework.http.HttpStatus;
3. import org.springframework.http.ResponseEntity;
4. import org.springframework.web.bind.annotation.ExceptionHandler;
5. import org.springframework.web.bind.annotation.RestControllerAdvice;
6. import com.infytel.dto.ErrorMessage;
7. @RestControllerAdvice
8. public class ExceptionControllerAdvice {
9. @ExceptionHandler(Exception.class)
10. public String exceptionHandler(Exception ex) {
11. return ex.getMessage();
12. }
13. @ExceptionHandler(NoSuchCustomerException.class)
14. public ResponseEntity<ErrorMessage> exceptionHandler2(NoSuchCustomerException ex) {
15. ErrorMessage error = new ErrorMessage();
16. error.setErrorCode(HttpStatus.BAD_GATEWAY.value());
17. error.setMessage(ex.getMessage());
18. return new ResponseEntity<>(error, HttpStatus.OK);
19. }
20. }
```

```
1. package com.infytel.exceptions;
2. public class NoSuchCustomerException extends Exception {
3. private static final long serialVersionUID = 1L;
4. public NoSuchCustomerException() {
5. super();
6. }
7. public NoSuchCustomerException(String errors) {
```

```
8. super(errors);
9. }
10. }
```

**Step 7: Create class `CustomerRepository` under `com.infytel.repository` package:**

```
1. package com.infytel.repository;
2. import java.util.ArrayList;
3. import java.util.List;
4. import javax.annotation.PostConstruct;
5. import org.springframework.stereotype.Repository;
6. import com.infytel.dto.CustomerDTO;
7. import com.infytel.dto.FriendFamilyDTO;
8. import com.infytel.dto.PlanDTO;
9. import com.infytel.exceptions.NoSuchCustomerException;
10. @Repository
11. public class CustomerRepository {
12.     List<CustomerDTO> customers = null;
13.     // Populates customer in hard-coded way
14.     @PostConstruct
15.     public void initializer() {
16.         CustomerDTO customerDTO = new CustomerDTO();
17.         PlanDTO planDTO = new PlanDTO();
18.         planDTO.setPlanId(1);
19.         planDTO.setPlanName("Simple");
20.         planDTO.setLocalRate(3);
21.         planDTO.setNationalRate(5);
22.         customerDTO.setAddress("Chennai");
23.         customerDTO.setAge(18);
24.         customerDTO.setCurrentPlan(planDTO);
25.         customerDTO.setGender('m');
26.         customerDTO.setName("Jack");
27.         customerDTO.setEmail("Jack@infy.com");
28.         customerDTO.setPassword("ABC@123");
29.         customerDTO.setPhoneNo(99512122221);
30.         List<FriendFamilyDTO> friendAndFamily = new ArrayList<>();
31.         friendAndFamily.add(new FriendFamilyDTO(customerDTO.getPhoneNo(), 800000145));
32.         friendAndFamily.add(new FriendFamilyDTO(customerDTO.getPhoneNo(), 700000145));
33.         customerDTO.setFriendAndFamily(friendAndFamily);
34.         customers = new ArrayList<>();
35.         customers.add(customerDTO);
36.     }
37.     // creates customer
```

```
38. public String createCustomer(CustomerDTO customerDTO) {
39. customers.add(customerDTO);
40. return "Customer with" + customerDTO.getPhoneNo() + "added successfully";
41. }
42. // fetches customer
43. public List<CustomerDTO> fetchCustomer() {
44. return customers;
45. }
46. // deletes customer - exception handling incorporated
47. public String deleteCustomer(long phoneNumber) throws NoSuchCustomerException {
48. boolean notfound = true;
49. String response = "Customer of:" + phoneNumber + "\t does not exist";
50. for (CustomerDTO customer : customers) {
51. if (customer.getPhoneNo() == phoneNumber) {
52. customers.remove(customer);
53. response = customer.getName() + " with phoneNumber " + customer.getPhoneNo() + " deleted
    successfully";
54. notfound = false;
55. break;
56. }
57. }
58. if (notfound)
59. throw new NoSuchCustomerException("Customer does not exist :" + phoneNumber);
60. return response;
61. }
62. // updates customer
63. public String updateCustomer(long phoneNumber, CustomerDTO customerDTO) {
64. String response = "Customer of:" + phoneNumber + "\t does not exist";
65. for (CustomerDTO customer : customers) {
66. if (customer.getPhoneNo() == phoneNumber) {
67. if (customerDTO.getName() != null)
68. customer.setName(customerDTO.getName());
69. if (customerDTO.getAddress() != null)
70. customer.setAddress(customerDTO.getAddress());
71. if (customerDTO.getPassword() != null)
72. customer.setPassword(customerDTO.getPassword());
73. customers.set(customers.indexOf(customer), customer);
74. response = "Customer of phoneNumber" + customer.getPhoneNo() + "\t got updated successfully";
75. break;
76. }
77. }
78. return response;
```

```
79. }  
80. }
```

**Step 8:** Create class **CustomerService** under **com.infytel.service** package:

```
1. package com.infytel.service;  
2. import java.util.List;  
3. import org.springframework.beans.factory.annotation.Autowired;  
4. import org.springframework.stereotype.Service;  
5. import com.infytel.dto.CustomerDTO;  
6. import com.infytel.exceptions.NoSuchCustomerException;  
7. import com.infytel.repository.CustomerRepository;  
8. @Service  
9. public class CustomerService {  
10. @Autowired  
11. private CustomerRepository customerRepository;  
12. // Contacts repository layer to add customer  
13. public String createCustomer(CustomerDTO customerDTO) {  
14. return customerRepository.createCustomer(customerDTO);  
15. }  
16. // Contacts repository layer to fetch customer  
17. public List<CustomerDTO> fetchCustomer() {  
18. return customerRepository.fetchCustomer();  
19. }  
20. // Contacts repository layer to delete customer  
21. public String deleteCustomer(long phoneNumber) throws NoSuchCustomerException {  
22. return customerRepository.deleteCustomer(phoneNumber);  
23. }  
24. // Contacts repository layer to update customer  
25. public String updateCustomer(long phoneNumber, CustomerDTO customerDTO) {  
26. return customerRepository.updateCustomer(phoneNumber, customerDTO);  
27. }  
28. }
```

**Step 9:** Add the following content to **application.properties** file:

```
1. server.port=8090  
2. server.servlet.context-path=/infytel-12a
```

**Step 10:** Make sure that the project's **pom.xml** looks similar to the one shown here.

```
1. <?xml version="1.0" encoding="UTF-8"?>  
2. <project xmlns="http://maven.apache.org/POM/4.0.0"  
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
4. xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-  
4.0.0.xsd">
```



```
5. <modelVersion>4.0.0</modelVersion>
6. <parent>
7. <groupId>org.springframework.boot</groupId>
8. <artifactId>spring-boot-starter-parent</artifactId>
9. <version>2.1.4.RELEASE</version>
10. <relativePath /> <!-- lookup parent from repository -->
11. </parent>
12. <groupId>com.infytel</groupId>
13. <artifactId>infytel_demo12_a</artifactId>
14. <version>0.0.1-SNAPSHOT</version>
15. <name>infytel_demo12_a</name>
16. <description>Demo project for Spring Boot</description>
17. <properties>
18. <java.version>1.8</java.version>
19. </properties>
20. <dependencies>
21. <dependency>
22. <groupId>org.springframework.boot</groupId>
23. <artifactId>spring-boot-starter-web</artifactId>
24. </dependency>
25. <dependency>
26. <groupId>org.springframework.boot</groupId>
27. <artifactId>spring-boot-starter-test</artifactId>
28. <scope>test</scope>
29. </dependency>
30. </dependencies>
31. <build>
32. <plugins>
33. <plugin>
34. <groupId>org.springframework.boot</groupId>
35. <artifactId>spring-boot-maven-plugin</artifactId>
36. </plugin>
37. </plugins>
38. </build>
39. </project>
```

**Step 11:** Deploy the application on the server by executing the class containing the main method.

So, we have successfully created and deployed the REST endpoints. Now, let us see how can we test the same using a browser.

### Output

### Testing Web Service

**Step 1:** Open any standard web browser.

**Step 2:** Test this URL - <http://localhost:8090/infytel-12a/customers>. Various customers' info will be listed out, which is in JSON form.



### Demo - UI application to test the CORS enabled Spring RESTful endpoint.

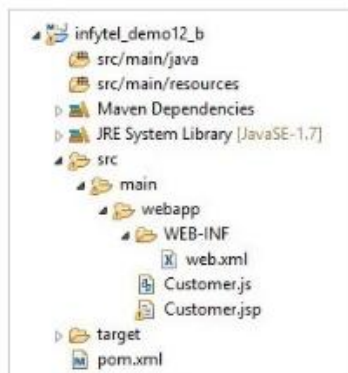
**Objectives:** A Simple UI application to test the CORS enabled Spring RESTful endpoint.

**Scenario:** This is a simple UI application developed using Javascript and JSP to test if we are able to access the REST endpoints that are enabled with CORS. Observe the AJAX call that is placed from the .js file.

#### Steps:

**Step 1:** Create a Maven web project.

**Step 2:** Modify the project according to the following project structure:



**Step 3:** Create **web.xml** file under src/main/webapp/WEB-INF package:

1. `<!DOCTYPE web-app PUBLIC`
2. `"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"`
3. `"http://java.sun.com/dtd/web-app_2_3.dtd" >`
4. `<web-app>`
5. `<display-name>Archetype Created Web Application</display-name>`
6. `</web-app>`

**Step 4:** Create **Customer.js** file under src/main/webapp package:

1. `/**`
2. `*`
3. `*/`
4. `$(document).ready(function() {`
5. `function tableCreate(el, data) {`

```
6. var tbl = document.createElement("table");
7. tbl.border = 1;
8. tbl.style.width = "70%";
9. var tr1 = tbl.insertRow();
10. tr1.bgColor = "blue";
11. tr1.insertCell().appendChild(document.createTextNode("phoneNo"));
12. tr1.insertCell().appendChild(document.createTextNode("Name"));
13. tr1.insertCell().appendChild(document.createTextNode("Email"));
14. for (var i = 0; i < data.length; ++i) {
15. var tr = tbl.insertRow();
16. for (var j = 0; j < data[i].length; ++j) {
17. var td = tr.insertCell();
18. td.appendChild(document.createTextNode(data[i][j].toString()));
19. }
20. }
21. $('table').attr('border', '0');
22. $('#customer-id').append(tbl);
23. }
24. $.ajax({
25. url : "http://localhost:8090/infytel-12a/customers",
26. headers : {
27. "Authorization" : "Basic " + btoa("admin" + ":" + "infytel")
28. }
29. }).then(function(data, status, jqxhr) {
30. let rows = [];
31. for (var i = 0; i < data.length; ++i) {
32. let cells = [];
33. cells.push(data[i].phoneNo);
34. cells.push(data[i].name);
35. cells.push(data[i].email);
36. rows.push(cells);
37. }
38. tableCreate($("#customer-id"), rows);
39. console.log(jqxhr);
40. });
41. });
```

**Step 5: Create Customer.jsp** file under src/main/webapp package:

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4. <title>Hello CORS</title>
```

```
5. <script
6. src="https://ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js"></script>
7. <script src="Customer.js"></script>
8. </head>
9. <body>
10. <Center>
11. <h1>Infytel Customer List</h1>
12. </Center>
13. <div>
14. <center>
15. <div class="customer-id"></div>
16. </center>
17. </div>
18. </body>
19. </html>
```

**Step 6:** Make sure that the project's pom.xml looks similar to the one shown here.

```
1. <project xmlns="http://maven.apache.org/POM/4.0.0"
2. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3. xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-
  v4_0_0.xsd">
4. <modelVersion>4.0.0</modelVersion>
5. <groupId>com.infytel</groupId>
6. <artifactId>infytel_demo12_b</artifactId>
7. <packaging>war</packaging>
8. <version>0.0.1-SNAPSHOT</version>
9. <name>infytel_demo12_b</name>
10. <url>http://maven.apache.org</url>
11. <properties>
12. <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
13. <maven.compiler.source>1.8</maven.compiler.source>
14. <maven.compiler.target>1.8</maven.compiler.target>
15. </properties>
16. <dependencies>
17. <!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
18. <dependency>
19. <groupId>javax.servlet</groupId>
20. <artifactId>javax.servlet-api</artifactId>
21. <version>4.0.1</version>
22. <scope>provided</scope>
23. </dependency>
24. </dependencies>
```

```
25. <build>
26. <finalName>infytel_demo12_b</finalName>
27. <pluginManagement><!-- lock down plugins versions to avoid using Maven
28. defaults (may be moved to parent pom) -->
29. <plugins>
30. <plugin>
31. <artifactId>maven-clean-plugin</artifactId>
32. <version>3.1.0</version>
33. </plugin>
34. <!-- see http://maven.apache.org/ref/current/maven-core/default-
    bindings.html#Plugin_bindings_for_war_packaging -->
35. <plugin>
36. <artifactId>maven-resources-plugin</artifactId>
37. <version>3.0.2</version>
38. </plugin>
39. <plugin>
40. <artifactId>maven-compiler-plugin</artifactId>
41. <version>3.8.0</version>
42. </plugin>
43. <plugin>
44. <artifactId>maven-surefire-plugin</artifactId>
45. <version>2.22.1</version>
46. </plugin>
47. <plugin>
48. <artifactId>maven-war-plugin</artifactId>
49. <version>3.2.2</version>
50. </plugin>
51. <plugin>
52. <artifactId>maven-install-plugin</artifactId>
53. <version>2.5.2</version>
54. </plugin>
55. <plugin>
56. <artifactId>maven-deploy-plugin</artifactId>
57. <version>2.8.2</version>
58. </plugin>
59. <plugin>
60. <groupId>org.apache.tomcat.maven</groupId>
61. <artifactId>tomcat7-maven-plugin</artifactId>
62. <version>2.2</version>
63. <configuration>
64. </configuration>
65. </plugin>
```

```
66. </plugins>
67. </pluginManagement>
68. </build>
69. </project>
```

**Step 7:** Execute the applications 12a and 12b and make them available in different port numbers, say 8090 and 8080.

Now let us go for testing them both.

## Output

Testing Web Service using a Web browser and Postman.

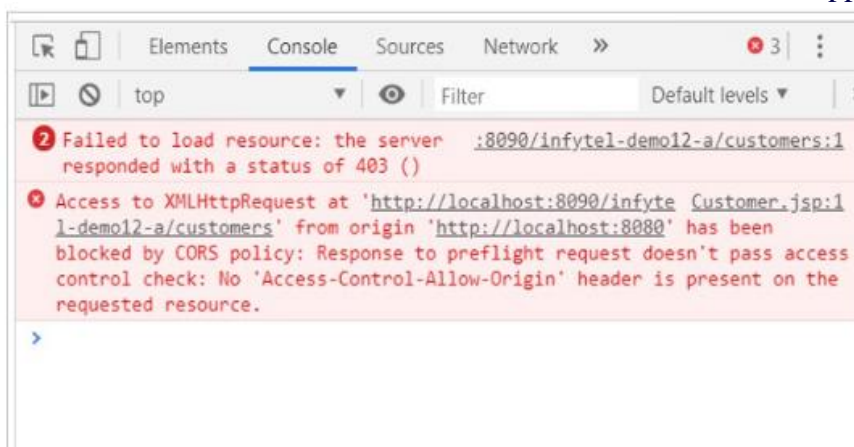
**Step 1:** Right click on the project. Choose Run As->Maven build... option. Specify the goal, tomcat7:run. Click on Apply and Run.

**Step 2:** Open any standard browser.

**Step 3:** Before doing so, just disable CORS config that is there available in the starter code of demo 12a. Enter this URL [http://localhost:8080/infytel\\_demo12\\_b/Customer.jsp](http://localhost:8080/infytel_demo12_b/Customer.jsp). Following will be the response.

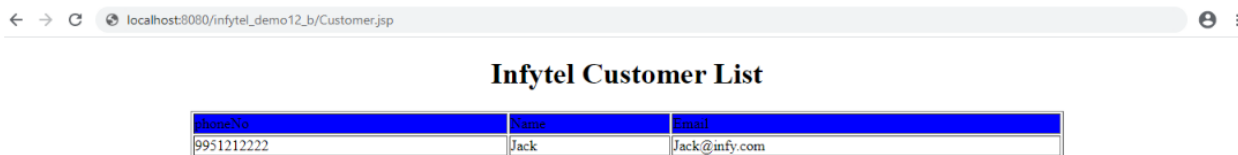


Now Press ctrl+shift+J. The reason behind this failure is the lack of CORS support.



Now, enable CORS config that is there available in the starter code of demo 12a and follow the next step to get the desired output.

**Step 4:** Test this URL - [http://localhost:8080/infytel\\_demo12\\_b/Customer.jsp](http://localhost:8080/infytel_demo12_b/Customer.jsp). Following will be the response generated.



The screenshot shows a web browser window with the address bar displaying 'localhost:8090/infytel\_demo12\_b/Customer.jsp'. The page title is 'Infytel Customer List'. Below the title is a table with three columns: 'phoneNo', 'Name', and 'Email'. The table contains one row of data.

phoneNo	Name	Email
9951212222	Jack	Jack@infy.com

**Step**

**5:** Now try posting another customer object with postman client.

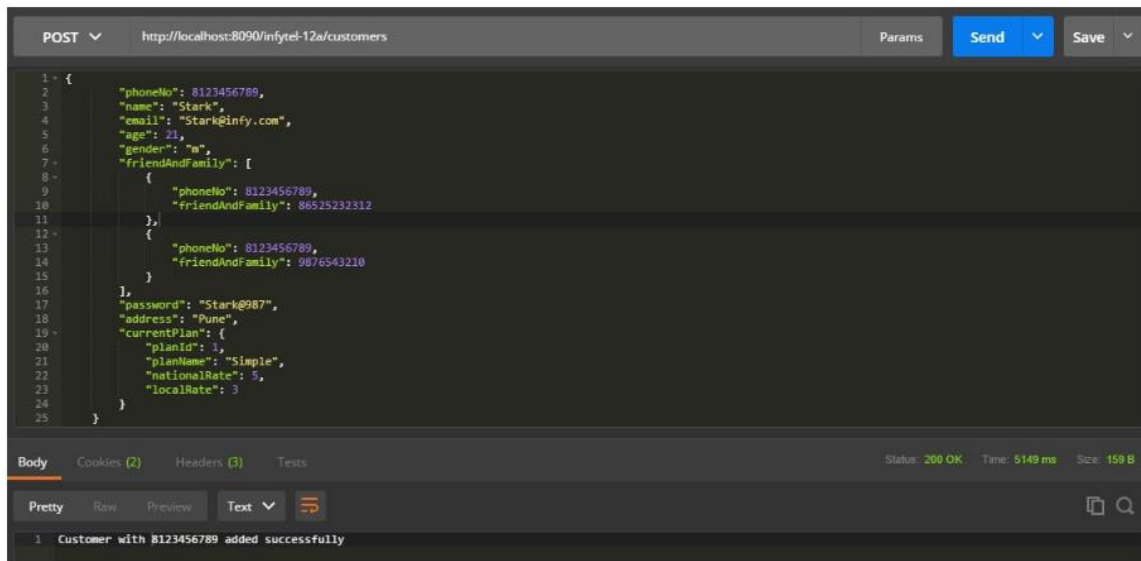
From the drop-down, select POST and enter **http://localhost:8090/infytel-12a/customers** into the URL field to test the service. And, click on the body to enter the following JSON data. The below data is used to create a new customer in infytel.

The JSON data has to exactly match the DTO object, to which it is converted.

```
1. {  
2.   "phoneNo": 8123456789,  
3.   "name": "Stark",  
4.   "email": "Stark@infy.com",  
5.   "age": 21,  
6.   "gender": "m",  
7.   "friendAndFamily": [  
8.     {  
9.       "phoneNo": 8123456789,  
10.      "friendAndFamily": 86525232312  
11.     },  
12.     {  
13.       "phoneNo": 8123456789,  
14.       "friendAndFamily": 9876543210  
15.     }  
16.   ],  
17.   "password": "Stark@987",  
18.   "address": "Pune",  
19.   "currentPlan": {  
20.     "planId": 1,  
21.     "planName": "Simple",  
22.     "nationalRate": 5,  
23.     "localRate": 3  
24.   }  
25. }
```

**Note:** Observe the CustomerDTO's variable names. It is important that the variable name in CustomerDTO and JSON key has to match exactly, for the correct mapping to happen.

Click on Send and, observe the response.



Now try refreshing the browser window with URL, [http://localhost:8080/infytel\\_demo12\\_b/Customer.jsp](http://localhost:8080/infytel_demo12_b/Customer.jsp). Following will be the response that includes the recently added customer as well.

localhost:8080/infytel\_demo12\_b/Customer.jsp

### Infytel Customer List

phoneNo	Name	Email
9951212222	Jack	Jack@infy.com
8123456789	Stark	Stark@infy.com

## Securing Spring REST endpoints

### Securing a Spring Rest API With Spring Security

Need for securing a RESTful service arises when the authorized entities alone have to access to the end points.

Securing REST APIs here, follows the same steps involved in securing any other web application that is developed using Spring Boot. As we all know, Spring Boot minimizes configuration and that goes good with security as well.

We can secure the Spring REST endpoints using,

- **Basic Authentication** (one that will be used in our course)
- OAUTH2 authentication

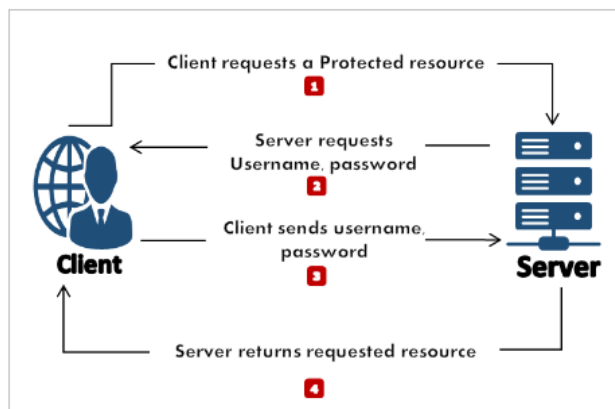
### Basic Authentication:

When client requests for a protected resource, server demands the client to supply the authentication credentials.

Client will get the requested resource only when the credentials pass authentication at server side.

Steps involved in basic authentication are shown below.





With Basic Authentication in place, clients send their Base64 encoded credentials with each request using HTTP Authorization header.

This type of authentication does not mandate us to have any Login page which is the very standard way of authenticating the client requests in any web application. In addition, there is no need to store the credentials or any related info in the session.

Each request here, is treated as an independent request which in turn promotes scalability.

Our example for implementing security will follow this approach.

### Enabling Basic Authentication

Enabling security in a Spring Boot application needs the inclusion of the following dependency as the very first step.

1. <dependency>
2. <groupId>org.springframework.boot</groupId>
3. <artifactId>spring-boot-starter-security</artifactId>
4. </dependency>

This will include the SecurityAutoConfiguration class – containing the initial or default security configuration.

By default, authentication gets enabled for the whole application.

There are some predefined properties as shown below which needs to be configured in the application.properties file.

- spring.security.user.name
- spring.security.user.password

If there is no password configured using the predefined property **spring.security.user.password**, Spring Boot generates a password in a random fashion during the application start. And, the same will be found in the console log.

1. Using default security password: c8be15de-4488-4490-9dc6-fab3f9

Following steps are required only when we need to customize the security related aspects.

```

1. @Configuration
2. @EnableWebSecurity
3. public class SecurityConfiguration extends WebSecurityConfigurerAdapter {
4. }

```

Need to override the below methods

```

1. //Configuring username and password and authorities
2. @Autowired
3. public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
4. auth.inMemoryAuthentication()
5. .withUser("admin").password(passwordEncoder().encode("infytel"))
6. .authorities("ROLE_USER");
7. }
8. //Configuring the HttpSecurity
9. @Override
10. protected void configure(HttpSecurity http) throws Exception {
11. http.authorizeRequests()
12. .antMatchers("/securityNone").permitAll()//URLS starting with securityNone need not be
    authenticated
13. .anyRequest().authenticated() //rest of the URL's should be authenticated
14. .and().
15. httpBasic().and()//Every request should be authenticated it should not store in a session
16. sessionManagement().sessionCreationPolicy(SessionCreationPolicy.NEVER).and().csrf().disable()
    ;
17. }
18. //The passwordEncoder to be used.
19. @Bean
20. public PasswordEncoder passwordEncoder() {
21. return new BCryptPasswordEncoder();
22. }

```

Let us put all these together and look at an example.

### Demo - Securing Spring Rest APIs

**Objectives:** To enable basic authentication for a Spring REST API using Spring Security

**Scenario:** An online telecom app called Infytel is exposing its customer management profile as a RESTful resource. The customer resource titled, CustomerController allows the clients to create, fetch, delete and update the customer details. Following are the HTTP operation that will come to our focus, here.

Controller Class	method name	URI	HTTP Method	Remarks
CustomerController	fetchCustomer()	/customers	GET	Fetch all the customers
CustomerController	createCustomer()	/customers	POST	Create a new Customer

Controller Class	method name	URI	HTTP Method	Remarks
CustomerController	deleteCustomer()	/customers/{phoneNumber}	Delete	Delete an existing Customer identified by phoneNumber
CustomerController	updateCustomer()	/customers/{phoneNumber}	PUT	Update an existing Customer identified by phoneNumber

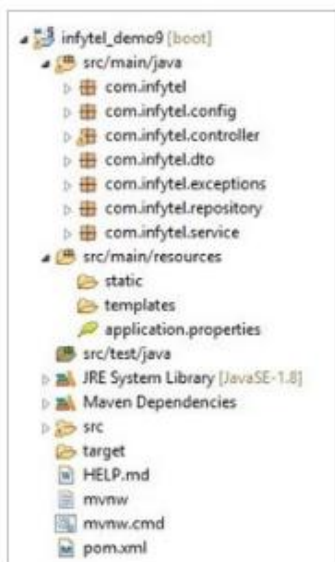
Observe the following things:

- SecurityConfiguration class that holds configuration pertaining to security.
- pom.xml with an additional dependency for security.

#### Steps:

**Step 1:** Create a Maven project using Spring Initializer with web dependency and import the same in STS.

**Step 2:** Modify the imported project according to the following project structure:



**Step 3:** Look at the class **InfytelDemo9Application** under **com.infytel** package that gets created automatically when the project is generated. We will modify this only when we need to go with custom configuration.

```

1. package com.infytel;
2. import org.springframework.boot.SpringApplication;
3. import org.springframework.boot.autoconfigure.SpringBootApplication;
4. @SpringBootApplication
5. public class InfytelDemo9Application {
6.     public static void main(String[] args) {
7.         SpringApplication.run(InfytelDemo9Application.class, args);
8.     }
9. }

```

**Step 4:** Create class **SecurityConfiguration** under **com.infytel.config** package:

```
1. package com.infytel.config;
2. import org.springframework.beans.factory.annotation.Autowired;
3. import org.springframework.context.annotation.Bean;
4. import org.springframework.context.annotation.Configuration;
5. import
    org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
6. import org.springframework.security.config.annotation.web.builders.HttpSecurity;
7. import
    org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
8. import org.springframework.security.config.http.SessionCreationPolicy;
9. import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
10. import org.springframework.security.crypto.password.PasswordEncoder;
11. @Configuration
12. public class SecurityConfiguration extends WebSecurityConfigurerAdapter {
13.     @Autowired
14.     public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
15.         auth.inMemoryAuthentication().withUser("admin").password(passwordEncoder().encode("infytel"
16.         ))
17.     }
18.     @Override
19.     protected void configure(HttpSecurity http) throws Exception {
20.         http.authorizeRequests().antMatchers("/").permitAll().anyRequest().authenticated().and().httpBasic().and()
21.         .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.NEVER).and().csrf().disable(
22.         );
23.     }
24.     @Bean
25.     public PasswordEncoder passwordEncoder() {
26.         return new BCryptPasswordEncoder();
27.     }
```

**Step 5:** Create class **CustomerController** under com.infytel.controller package:

```
1. package com.infytel.controller;
2. import java.util.List;
3. import java.util.stream.Collectors;
4. import javax.validation.Valid;
5. import org.springframework.beans.factory.annotation.Autowired;
6. import org.springframework.http.HttpStatus;
```

```
7. import org.springframework.http.ResponseEntity;
8. import org.springframework.validation.Errors;
9. import org.springframework.validation.ObjectError;
10. import org.springframework.web.bind.annotation.DeleteMapping;
11. import org.springframework.web.bind.annotation.GetMapping;
12. import org.springframework.web.bind.annotation.PathVariable;
13. import org.springframework.web.bind.annotation.PostMapping;
14. import org.springframework.web.bind.annotation.PutMapping;
15. import org.springframework.web.bind.annotation.RequestBody;
16. import org.springframework.web.bind.annotation.RequestMapping;
17. import org.springframework.web.bind.annotation.RestController;
18. import org.springframework.web.client.RestTemplate;
19. import com.infytel.dto.CustomerDTO;
20. import com.infytel.dto.ErrorMessage;
21. import com.infytel.dto.PlanDTO;
22. import com.infytel.exceptions.NoSuchCustomerException;
23. import com.infytel.service.CustomerService;
24. @RestController
25. @RequestMapping("/customers")
26. public class CustomerController {
27.     @Autowired
28.     private CustomerService customerService;
29.     // Fetching customer details
30.     @GetMapping(produces = "application/json")
31.     public List<CustomerDTO> fetchCustomer() {
32.         return customerService.fetchCustomer();
33.     }
34.     // Adding a customer
35.     @PostMapping(consumes = "application/json")
36.     public ResponseEntity createCustomer(@Valid @RequestBody CustomerDTO customerDTO,
        Errors errors) {
37.         String response = "";
38.         if (errors.hasErrors()) {
39.             response = errors.getAllErrors().stream().map(ObjectError::getDefaultMessage)
40.                 .collect(Collectors.joining(", "));
41.             ErrorMessage error = new ErrorMessage();
42.             error.setErrorCode(HttpStatus.NOT_ACCEPTABLE.value());
43.             error.setMessage(response);
44.             return ResponseEntity.ok(error);
45.         } else {
46.             PlanDTO planDTOReceived = new RestTemplate().getForObject(
47.                 "http://localhost:8080/plans/" + customerDTO.getCurrentPlan().getPlanId(), PlanDTO.class);
```

```
48. customerDTO.setCurrentPlan(planDTOReceived);
49. response = customerService.createCustomer(customerDTO);
50. return ResponseEntity.ok(response);
51. }
52. }
53. // Updating an existing customer
54. @PutMapping(value = "/{phoneNumber}", consumes = "application/json")
55. public String updateCustomer(@PathVariable("phoneNumber") long phoneNumber,
    @RequestBody CustomerDTO customerDTO) {
56. return customerService.updateCustomer(phoneNumber, customerDTO);
57. }
58. // Deleting a customer
59. @DeleteMapping(value = "/{phoneNumber}", produces = "text/html")
60. public String deleteCustomer(@PathVariable("phoneNumber") long phoneNumber) throws
    NoSuchCustomerException {
61. return customerService.deleteCustomer(phoneNumber);
62. }
63. }
```

**Step 6:** Create classes **CustomerDTO**, **ErrorMessage**, **FriendFamilyDTO** and **PlanDTO** under **com.infytel.dto** package:

```
1. package com.infytel.dto;
2. import java.util.List;
3. import javax.validation.constraints.Email;
4. public class CustomerDTO {
5. long phoneNo;
6. String name;
7. @Email(message = "Email id is not in format, please check")
8. String email;
9. public String getEmail() {
10. return email;
11. }
12. public void setEmail(String email) {
13. this.email = email;
14. }
15. int age;
16. char gender;
17. List<FriendFamilyDTO> friendAndFamily;
18. String password;
19. String address;
20. PlanDTO currentPlan;
21. public PlanDTO getCurrentPlan() {
22. return currentPlan;
23. }
```

```
23. }
24. public void setCurrentPlan(PlanDTO currentPlan) {
25. this.currentPlan = currentPlan;
26. }
27. public String getPassword() {
28. return password;
29. }
30. public void setPassword(String password) {
31. this.password = password;
32. }
33. public String getAddress() {
34. return address;
35. }
36. public void setAddress(String address) {
37. this.address = address;
38. }
39. public List<FriendFamilyDTO> getFriendAndFamily() {
40. return friendAndFamily;
41. }
42. public void setFriendAndFamily(List<FriendFamilyDTO> friendAndFamily) {
43. this.friendAndFamily = friendAndFamily;
44. }
45. public long getPhoneNo() {
46. return phoneNo;
47. }
48. public void setPhoneNo(long phoneNo) {
49. this.phoneNo = phoneNo;
50. }
51. public String getName() {
52. return name;
53. }
54. public void setName(String name) {
55. this.name = name;
56. }
57. public int getAge() {
58. return age;
59. }
60. public void setAge(int age) {
61. this.age = age;
62. }
63. public char getGender() {
64. return gender;
```

```
65. }
66. public void setGender(char gender) {
67. this.gender = gender;
68. }
69. @Override
70. public String toString() {
71. return "CustomerDTO [phoneNo=" + phoneNo + ", name=" + name + ", age=" + age + ", gender="
    + gender + ", friendAndFamily=" + friendAndFamily + ", password=" + password + ", address=" +
    address + "]";
72. }
73. }
```

```
1. package com.infytel.dto;
2. public class ErrorMessage {
3. private int errorCode;
4. private String message;
5. public int getErrorCode() {
6. return errorCode;
7. }
8. public void setErrorCode(int errorCode) {
9. this.errorCode = errorCode;
10. }
11. public String getMessage() {
12. return message;
13. }
14. public void setMessage(String message) {
15. this.message = message;
16. }
17. }
1. package com.infytel.dto;
2. public class FriendFamilyDTO {
3. long phoneNo;
4. long friendAndFamily;
5. public long getPhoneNo() {
6. return phoneNo;
7. }
8. public void setPhoneNo(long phoneNo) {
9. this.phoneNo = phoneNo;
10. }
11. public long getFriendAndFamily() {
12. return friendAndFamily;
13. }
```



```
14. public void setFriendAndFamily(long friendAndFamily) {
15. this.friendAndFamily = friendAndFamily;
16. }
17. public FriendFamilyDTO(long phoneNo, long friendAndFamily) {
18. this();
19. this.phoneNo = phoneNo;
20. this.friendAndFamily = friendAndFamily;
21. }
22. public FriendFamilyDTO() {
23. super();
24. }
25. @Override
26. public String toString() {
27. return "FriendFamilyDTO [phoneNo=" + phoneNo + ", friendAndFamily=" + friendAndFamily +
    "]\n";
28. }
29. }
```

```
1. package com.infytel.dto;
2. import javax.xml.bind.annotation.XmlRootElement;
3. @XmlRootElement
4. public class PlanDTO {
5.     Integer planId;
6.     String planName;
7.     Integer nationalRate;
8.     Integer localRate;
9.     public Integer getPlanId() {
10. return planId;
11. }
12. public void setPlanId(Integer planId) {
13. this.planId = planId;
14. }
15. public String getPlanName() {
16. return planName;
17. }
18. public void setPlanName(String planName) {
19. this.planName = planName;
20. }
21. public Integer getNationalRate() {
22. return nationalRate;
23. }
24. public void setNationalRate(Integer nationalRate) {
```

```
25. this.nationalRate = nationalRate;
26. }
27. public Integer getLocalRate() {
28. return localRate;
29. }
30. public void setLocalRate(Integer localRate) {
31. this.localRate = localRate;
32. }
33. public PlanDTO() {
34. super();
35. }
36. @Override
37. public String toString() {
38. return "PlanDTO [planId=" + planId + ", planName=" + planName + ", nationalRate=" +
    nationalRate + ", localRate=" + localRate + "]";
39. }
40. }
```

**Step 7:** Create classes **ExceptionHandlerAdvice** and **NoSuchCustomerException** under **com.infytel.exception** package:

```
1. package com.infytel.exceptions;
2. import org.springframework.http.HttpStatus;
3. import org.springframework.http.ResponseEntity;
4. import org.springframework.web.bind.annotation.ExceptionHandler;
5. import org.springframework.web.bind.annotation.RestControllerAdvice;
6. import com.infytel.dto.ErrorMessage;
7. @RestControllerAdvice
8. public class ExceptionControllerAdvice {
9. @ExceptionHandler(Exception.class)
10. public String exceptionHandler(Exception ex) {
11. return ex.getMessage();
12. }
13. @ExceptionHandler(NoSuchCustomerException.class)
14. public ResponseEntity<ErrorMessage> exceptionHandler2(NoSuchCustomerException ex) {
15. ErrorMessage error = new ErrorMessage();
16. error.setErrorCode(HttpStatus.BAD_GATEWAY.value());
17. error.setMessage(ex.getMessage());
18. return new ResponseEntity<>(error, HttpStatus.OK);
19. }
20. }
```

```
1. package com.infytel.exceptions;
```

```
2. public class NoSuchCustomerException extends Exception {
3.     private static final long serialVersionUID = 1L;
4.     public NoSuchCustomerException() {
5.         super();
6.     }
7.     public NoSuchCustomerException(String errors) {
8.         super(errors);
9.     }
10. }
```

**Step 8:** Create class **CustomerRepository** under com.infytel.repository package:

```
1. package com.infytel.repository;
2. import java.util.ArrayList;
3. import java.util.List;
4. import javax.annotation.PostConstruct;
5. import org.springframework.stereotype.Repository;
6. import com.infytel.dto.CustomerDTO;
7. import com.infytel.dto.FriendFamilyDTO;
8. import com.infytel.dto.PlanDTO;
9. import com.infytel.exceptions.NoSuchCustomerException;
10. @Repository
11. public class CustomerRepository {
12.     List<CustomerDTO> customers = null;
13.     // Populates customer in hard-coded way
14.     @PostConstruct
15.     public void initializer() {
16.         CustomerDTO customerDTO = new CustomerDTO();
17.         PlanDTO planDTO = new PlanDTO();
18.         planDTO.setPlanId(1);
19.         planDTO.setPlanName("Simple");
20.         planDTO.setLocalRate(3);
21.         planDTO.setNationalRate(5);
22.         customerDTO.setAddress("Chennai");
23.         customerDTO.setAge(18);
24.         customerDTO.setCurrentPlan(planDTO);
25.         customerDTO.setGender('m');
26.         customerDTO.setName("Jack");
27.         customerDTO.setEmail("Jack@infy.com");
28.         customerDTO.setPassword("ABC@123");
29.         customerDTO.setPhoneNo(99512122221);
30.         List<FriendFamilyDTO> friendAndFamily = new ArrayList<>();
31.         friendAndFamily.add(new FriendFamilyDTO(customerDTO.getPhoneNo(), 800000145));
32.         friendAndFamily.add(new FriendFamilyDTO(customerDTO.getPhoneNo(), 700000145));
```

```
33. customerDTO.setFriendAndFamily(friendAndFamily);
34. customers = new ArrayList<>();
35. customers.add(customerDTO);
36. }
37. // creates customer
38. public String createCustomer(CustomerDTO customerDTO) {
39. customers.add(customerDTO);
40. return "Customer with" + customerDTO.getPhoneNo() + "added successfully";
41. }
42. // fetches customer
43. public List<CustomerDTO> fetchCustomer() {
44. return customers;
45. }
46. // deletes customer - exception handling incorporated
47. public String deleteCustomer(long phoneNumber) throws NoSuchCustomerException {
48. boolean notfound = true;
49. String response = "Customer of:" + phoneNumber + "\t does not exist";
50. for (CustomerDTO customer : customers) {
51. if (customer.getPhoneNo() == phoneNumber) {
52. customers.remove(customer);
53. response = customer.getName() + " with  phoneNumber " + customer.getPhoneNo() + " deleted
    successfully";
54. notfound = false;
55. break;
56. }
57. }
58. if (notfound)
59. throw new NoSuchCustomerException("Customer does not exist ." + phoneNumber);
60. return response;
61. }
62. // updates customer
63. public String updateCustomer(long phoneNumber, CustomerDTO customerDTO) {
64. String response = "Customer of:" + phoneNumber + "\t does not exist";
65. for (CustomerDTO customer : customers) {
66. if (customer.getPhoneNo() == phoneNumber) {
67. if (customerDTO.getName() != null)
68. customer.setName(customerDTO.getName());
69. if (customerDTO.getAddress() != null)
70. customer.setAddress(customerDTO.getAddress());
71. if (customerDTO.getPassword() != null)
72. customer.setPassword(customerDTO.getPassword());
73. customers.set(customers.indexOf(customer), customer);
```

```
74. response = "Customer of phoneNumber" + customer.getPhoneNo() + "\t got updated successfully";
75. break;
76. }
77. }
78. return response;
79. }
80. }
```

**Step 9:** Create class **CustomerService** under **com.infytel.service** package:

```
1. package com.infytel.service;
2. import java.util.List;
3. import org.springframework.beans.factory.annotation.Autowired;
4. import org.springframework.stereotype.Service;
5. import com.infytel.dto.CustomerDTO;
6. import com.infytel.exceptions.NoSuchCustomerException;
7. import com.infytel.repository.CustomerRepository;
8. @Service
9. public class CustomerService {
10. @Autowired
11. private CustomerRepository customerRepository;
12. // Contacts repository layer to add customer
13. public String createCustomer(CustomerDTO customerDTO) {
14. return customerRepository.createCustomer(customerDTO);
15. }
16. // Contacts repository layer to fetch customer
17. public List<CustomerDTO> fetchCustomer() {
18. return customerRepository.fetchCustomer();
19. }
20. // Contacts repository layer to delete customer
21. public String deleteCustomer(long phoneNumber) throws NoSuchCustomerException {
22. return customerRepository.deleteCustomer(phoneNumber);
23. }
24. // Contacts repository layer to update customer
25. public String updateCustomer(long phoneNumber, CustomerDTO customerDTO) {
26. return customerRepository.updateCustomer(phoneNumber, customerDTO);
27. }
28. }
```

**Step 10:** Add the following content to **application.properties** file:

```
1. server.port=8080
2. server.servlet.context-path=/infytel-9
```

**Step 11:** Make sure that the project's **pom.xml** looks similar to the one that is shown below.

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <project xmlns="http://maven.apache.org/POM/4.0.0"
```

```
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
  4.0.0.xsd">
5. <modelVersion>4.0.0</modelVersion>
6. <parent>
7. <groupId>org.springframework.boot</groupId>
8. <artifactId>spring-boot-starter-parent</artifactId>
9. <version>2.1.4.RELEASE</version>
10. <relativePath /> <!-- lookup parent from repository -->
11. </parent>
12. <groupId>infytel</groupId>
13. <artifactId>infytel_demo9</artifactId>
14. <version>0.0.1-SNAPSHOT</version>
15. <name>infytel_demo9</name>
16. <description>REST with Security</description>
17. <properties>
18. <java.version>1.8</java.version>
19. </properties>
20. <dependencies>
21. <dependency>
22. <groupId>org.springframework.boot</groupId>
23. <artifactId>spring-boot-starter-web</artifactId>
24. </dependency>
25. <!-- for security -->
26. <dependency>
27. <groupId>org.springframework.boot</groupId>
28. <artifactId>spring-boot-starter-security</artifactId>
29. </dependency>
30. <dependency>
31. <groupId>org.springframework.boot</groupId>
32. <artifactId>spring-boot-starter-test</artifactId>
33. <scope>test</scope>
34. </dependency>
35. </dependencies>
36. <build>
37. <plugins>
38. <plugin>
39. <groupId>org.springframework.boot</groupId>
40. <artifactId>spring-boot-maven-plugin</artifactId>
41. </plugin>
42. </plugins>
43. </build>
```

44. </project>

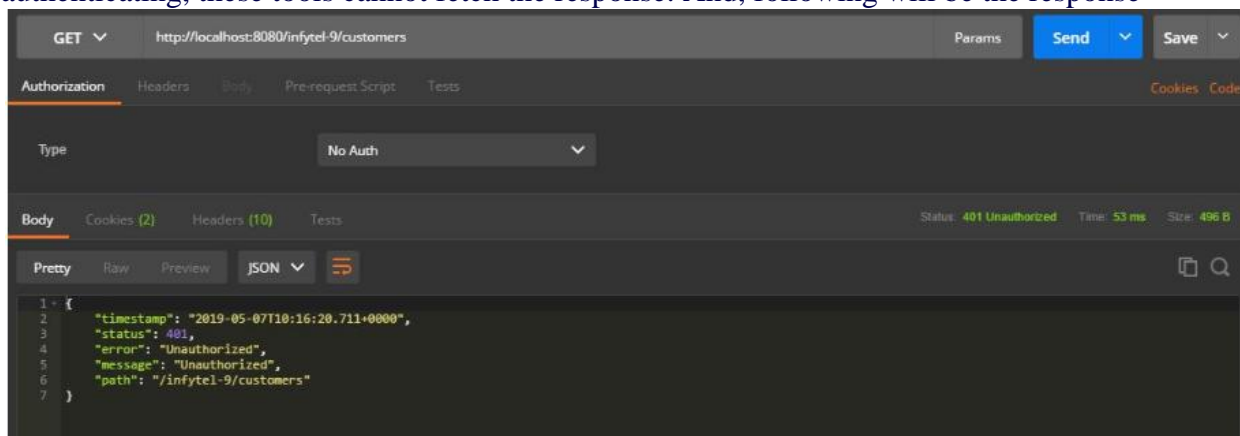
**Step 12:** Deploy the application on the server by executing the class containing the main method. So, we have successfully created and deployed the REST endpoints. Now, let us see how can we test the same using Postman client.

## Output

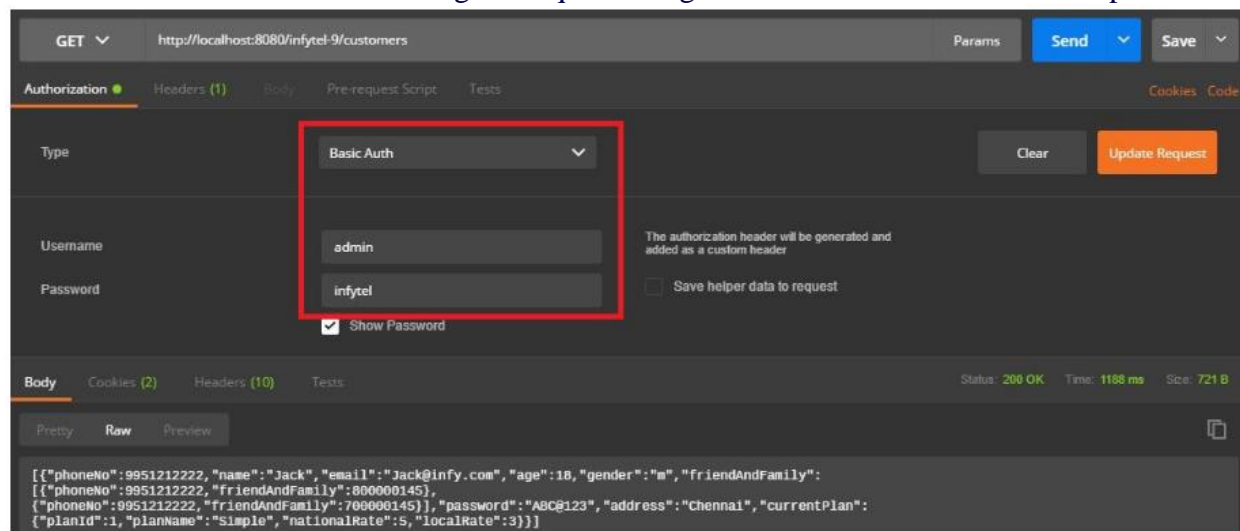
Testing Web Service using Postman

**Step 1:** Launch Postman.

**Step 2:** Test this URL - `http://localhost:8080/infytel-9/customers` using HTTP GET. Without authenticating, these tools cannot fetch the response. And, following will be the response



Set the credentials before submitting the request and get the customer details as the response.



**Note:** We can even test this URL - `http://localhost:8080/infytel-9/customers` using any standard web browser. No customer details will be rendered and a pop up will appear that asks for the credentials. Provide the username and password configured that are configured in our Security config class.

