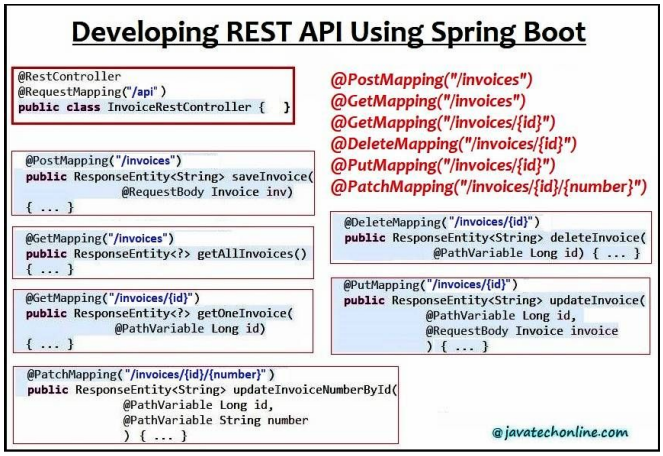
How To Develop REST CRUD API Using Spring Boot ?

java Spring Boot Spring Boot REST *by devs5003 - October 15, 2020* 11



It will not be an overstatement if I say that we can't develop an enterprise application without using webservices as an integration layer. Generally, we develop webservices in the form of either a producer or a consumer or both. However Producer is very important for us because we develop it in Java only. Also, we have full control of database interaction logic implementation with us. Now you might have interpreted the significance of our article 'How to develop REST CRUD API using Spring Boot?'. On the other hand, Consumer can be Angular Application, ReactJS Application, Android Device, iOS Device and many others or even our favorite java based RestTemplate(Spring Boot REST Client).

Our focus in this article is on developing producer API(REST API) using Spring Boot. Now let's discuss on 'How to develop REST CRUD API using Spring Boot ?' without leaving any theoretical concept which are very essential to know.

Table of Contents (Click on links below to navigate) [hide]

What will you learn from this article?

Once you complete going through all points of this article, You will be able to answer :

1) What is REST and REST API in the context of Web services?

TOP

2) How to create a Spring Boot REST application that incorporates industry level project design ?

3) How to develop CRUD (Create, Retrieve, Update, Delete) operations that can be used by any other even non-java application ?

4) How to write bug free CRUD operations, including exceptions & exception handlers?

5) Equally important, How to use annotations @RestController, @RequestMapping, @GetMapping, @PostMapping, @PutMapping, @DeleteMapping, @PatchMapping, @Modifying, @Query, @Transactional, @RestControllerAdvice, @ExceptionHandler, @ControllerAdvice, @ResponseBody, @RequestBody, @PathVariable, @Data, @NoArgsConstructor, @AllArgsConstructor, @Entity, @Component, @Service, @Autowired ?

6) How to work with Spring Boot Data JPA repository interface?

7) How to write modular & reusable code?

8) Moreover, How to implement dynamic code with minimal changes, keeping future change requests in mind?

9) How to develop an integration layer to get interoperability ?

10) How to test REST Application by supplying & receiving JSON data ?

11) Last but not the least you will learn “How to develop REST CRUD API using Spring Boot ?”

What is REST?

REST stands for **Re**presentational **S**tate **T**ransfer. It transfers state(data) in global format(representational) between two different applications running on different servers. In the process of data transfer, who requests data is called Consumer/Client application and who provides data is called a producer application. REST is an architectural style that follows a set of rules to create webservices. Webservices provide reusable data to multiple applications and interoperability between them on the internet. Web services that conform to the REST architectural style, called RESTful Web services.

How will you define a development of REST API ?

Developing a Rest API is nothing but creating classes & methods in a specific architectural style so that data can be reused between interoperable applications. More or less we create RestController and respective CRUD operations in the process of development of the REST API.

What is RestController ?

In Spring Boot REST programming RestController is a mandatory class which acts as a front controller. It contains several methods that return Body and Status as a ResponseEntity object. Body refers to data in form of String, Object, Collections etc. Whereas Status refers to the HttpResponse Status (200, 404,405, 500 etc.).

Software used in this Project ?

–STS (Spring Tool Suite) : Version-> 4.7.1.RELEASE
–MySQL Database : Version ->8.0.19 MySQL Community Server
–JDK8 or later versions (Extremely tested on JDK8, JDK9 and JDK14)

Coding Steps

Step#1 : Create Project in STS

If you are new to Spring Boot, visit Internal Link to create a sample project in spring boot. While creating project in STS add 4 starters ‘MySqL Driver’, ‘Spring Data JPA’, ‘Spring Web’ and ‘Lombok’. You can also add ‘Spring Boot DevTools’ optionally. If you are new to ‘Lombok’, kindly visit ‘How to configure Lombok‘ and to know all about it in detail.

Step#2 : Create Database

You can use database software of your own choice and create a Database accordingly. We have used MySQL database software. Login to your MySQL and execute a Database creation query. In our case DB Name is ‘REST_INVOICE’, so our query will be ‘ CREATE DATABSE REST_INVOICE’.

Step#3 : Update application.properties and Write Classes & methods(REST API)

TOP

We are considering 'Invoice' as a model to develop the REST API. Invoice will have many fields such as invoiceNumber, invoiceName, invoiceAmount etc.. Below Table will show the list of classes & other files used in the project accordingly.

Package/Location	Class/Interface/file name	Purpose
src/main/resources	application.properties	properties file to declare common properties in the project
com.dev.invoice.rest.entity	Invoice.java	Model/Entity class with Database table mapping
com.dev.invoice.rest.repo	InvoiceRepository.java	Repository Interface which extends JpaRepository interface
com.dev.invoice.rest.service	IInvoiceService.java	Service interface with Database related methods
com.dev.invoice.rest.service.impl	InvoiceServiceImpl.java	Service class contains implementations of methods declared in Service interface for Database related operations
com.dev.invoice.rest.entity	ErrorType.java	Helper class used in InvoiceErrorHandler.java
com.dev.invoice.rest.exception	InvoiceNotFoundException.java	To define custom Exception if any Invoice not found
com.dev.invoice.rest.exception.handler	InvoiceErrorHandler.java	To handle the error In case InvoiceNotFoundException is thrown from any controller method
com.dev.invoice.rest.util	InvoiceUtil.java	Utility class to maximize code reusability & minimize code redundancy.
com.dev.invoice.rest.controller	InvoiceRestController.java	Rest Controller the backbone of REST API, accepts all requests coming from client and handover to respective method for processing.

application.properties

```
# DB Connection Properties
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/REST_INVOICE
spring.datasource.username=root
spring.datasource.password=devs

# JPA Properites
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect
```

Invoice.java

TOP

```java
package com.dev.invoice.rest.entity;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
@Entity
public class Invoice {

    @Id
    @GeneratedValue
    private Long id;
    private String name;
    private Double amount;
    private Double finalAmount;
    private String number;
    private String receivedDate;
    private String type;
    private String vendor;
    private String comments;
}
```

InvoiceRepository.java

```java
package com.dev.invoice.rest.repo;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Modifying;
import org.springframework.data.jpa.repository.Query;

import com.dev.invoice.rest.entity.Invoice;

public interface InvoiceRepository extends JpaRepository<Invoice, Long>{

    // Update is Non-Select Operation, so @Modifying is used
    @Modifying
    @Query("UPDATE Invoice SET number=:number WHERE id=:id")
    Integer updateInvoiceNumberById(String number,Long id);
}
```

Service Interface IInvoiceService.java

```java
package com.dev.invoice.rest.service;

import java.util.List;

import com.dev.invoice.rest.entity.Invoice;

public interface IInvoiceService {

    /**
     * Takes Invoice Object as input and returns PK generated
     */
    Long saveInvoice(Invoice inv);

    /**
     * Takes existing Invoice data as input and updates values
     */
    void updateInvoice(Invoice e);

    /**
     * Takes PK(ID) as input and deletes Invoice Object data
     */
    void deleteInvoice(Long id);

    /**
     * Takes id as input and returns one row as one object
     */
    Invoice getOneInvoice(Long id);  //used in RestController

    /**
     * select all rows and provides result as a List<Invoice>
```

TOP

```
    */
   List<Invoice> getAllInvoices();

   /**
    * Takes Id as input,checks if record exists returns true, else false
    *
    */
   boolean isInvoiceExist(Long id);

   /**
    * Takes 2 fields as input, updates Invoice data as provided where clause
    * like 'UPDATE Invoice SET number=:number WHERE id=:id'
    */
   Integer updateInvoiceNumberById(String number,Long id);
}
```

InvoiceServiceImpl.java

```
package com.dev.invoice.rest.service.impl;

import java.util.List;
import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.dev.invoice.rest.entity.Invoice;
import com.dev.invoice.rest.exception.InvoiceNotFoundException;
import com.dev.invoice.rest.repo.InvoiceRepository;
import com.dev.invoice.rest.service.</yoastmark>IInvoiceService;
import com.dev.invoice.rest.util.InvoiceUtil;

@Service
public class InvoiceServiceImpl implements IInvoiceService {

   @Autowired
   private InvoiceRepository repo;

   @Autowired
   private InvoiceUtil util;

   @Override
   public Long saveInvoice(Invoice inv) {
           util.CalculateFinalAmountIncludingGST(inv);
           Long id = repo.save(inv).getId();
           return id;
   }

   @Override
   public void updateInvoice(Invoice inv) {
           util.CalculateFinalAmountIncludingGST(inv);
           repo.save(inv);
   }

   @Override
   public void deleteInvoice(Long id) {
           Invoice inv= getOneInvoice(id);
           repo.delete(inv);
   }

   public Optional<Invoice> getSingleInvoice(Long Id) {
           return repo.findById(Id);
   }

   @Override
   public Invoice getOneInvoice(Long id) {

           Invoice inv = repo.findById(id)
                           .orElseThrow(()->new InvoiceNotFoundException(
                                           new StringBuffer().append("Product  '")
                                           .append(id)
                                           .append("' not exist")
                                           .toString())
                                           );
           return inv;
   }

   @Override
   public List<Invoice> getAllInvoices() {
           List<Invoice> list = repo.findAll();
           //JDK 1.8 List Sort (using Comparator)
                           list.sort((ob1,ob2)->ob1.getId().intValue()-ob2.getId().intValue());
                           //list.sort((ob1,ob2)->ob1.getAmount().compareTo(ob2.getAmount())); //ASC
                           //list.sort((ob1,ob2)->ob2.getAmount().compareTo(ob1.getAmount())); // DESC
           return list;
```

TOP

```
    }

    @Override
    public boolean isInvoiceExist(Long id) {

            return repo.existsById(id);
    }

    @Override
    @Transactional
    public Integer updateInvoiceNumberById(
                    String number, Long id)
    {
            if(!repo.existsById(id)) {
                    throw new InvoiceNotFoundException(
                                    new StringBuffer()
                                    .append("Invoice '")
                                    .append(id)
                                    .append("' not exist")
                                    .toString());
            }
            return repo.updateInvoiceNumberById(number, id);
    }

}
```

ErrorType.java

```
package com.dev.invoice.rest.entity;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class ErrorType {

    private String time;
    private String status;
    private String message;

}
```

InvoiceNotFoundException.java

```
package com.dev.invoice.rest.exception;

//Custom Exception
public class InvoiceNotFoundException extends RuntimeException{

    private static final long serialVersionUID = 1L;

    public InvoiceNotFoundException() {
            super();
    }

    public InvoiceNotFoundException(String message) {
            super(message);
    }

}
```

InvoiceErrorHandler.java

```
package com.dev.invoice.rest.exception.handler;

import java.util.Date;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.RestControllerAdvice;
```

TOP

```
 9
10 import com.dev.invoice.rest.entity.ErrorType;
11 import com.dev.invoice.rest.exception.InvoiceNotFoundException;
12
13 //@ControllerAdvice
14 @RestControllerAdvice
15 public class InvoiceErrorHandler {
16     /**
17      * In case of InvoiceNotFoundException is thrown
18      * from any controller method, this logic gets
19      * executed which behaves like re-usable and
20      * clear code (Code Modularity)
21      * @param nfe
22      * @return ResponseEntity
23      */
24     //@ResponseBody
25     @ExceptionHandler(InvoiceNotFoundException.class)
26     public ResponseEntity<ErrorType> handleNotFound(InvoiceNotFoundException nfe){
27
28             return new ResponseEntity<ErrorType>(
29                             new ErrorType(
30                                             new Date(System.currentTimeMillis()).toString(),
31                                             "404- NOT FOUND",
32                                             nfe.getMessage()),
33                             HttpStatus.</yoastmark>NOT_FOUND);
34     }
35 }
```

Utility class InvoiceUtil.java

```
1  package com.dev.invoice.rest.util;
2
3  import org.springframework.stereotype.Component;
4
5  import com.dev.invoice.rest.entity.Invoice;
6
7  @Component
8  public class InvoiceUtil {
9
10     public Invoice  CalculateFinalAmountIncludingGST (Invoice inv) {
11             var amount=inv.getAmount();
12             var gst= 0.1;
13             var finalAmount=amount+(amount*gst);
14             inv.setFinalAmount(finalAmount);
15             return inv;
16     }
17
18     public void copyNonNullValues(Invoice req, Invoice db) {
19
20             if(req.getName() !=null) {
21                     db.setName(req.getName());
22             }
23
24             if(req.getAmount() !=null) {
25                     db.setAmount(req.getAmount());
26             }
27
28             if(req.getNumber() !=null) {
29                     db.setNumber(req.getNumber());
30             }
31
32             if(req.getReceivedDate() !=null) {
33                     db.setReceivedDate(req.getReceivedDate());
34             }
35
36             if(req.getType() !=null) {
37                     db.setType(req.getType());
38             }
39
40             if(req.getVendor() !=null) {
41                     db.setVendor(req.getVendor());
42             }
43
44             if(req.getComments() !=null) {
45                     db.setComments(req.getComments());
46             }
47     }
48 }
```

InvoiceRestController.java

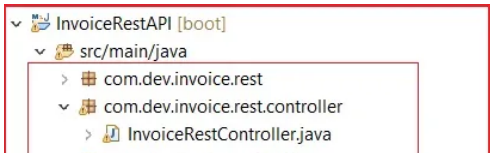
TOP

```java
package com.dev.invoice.rest.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PatchMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.dev.invoice.rest.entity.Invoice;
import com.dev.invoice.rest.exception.InvoiceNotFoundException;
import com.dev.invoice.rest.service.IInvoiceService;
import com.dev.invoice.rest.util.InvoiceUtil;

@RestController
@RequestMapping("/api")
//@CrossOrigin(origins = "http://localhost:4200") //Required in case of Angular Client
public class InvoiceRestController {

    @Autowired
    private IInvoiceService service;

    @Autowired
    private InvoiceUtil util;

    /**
     * Takes Invoice Object as input and returns save Status as ResponseEntity<String>
     */
    @PostMapping("/invoices")
    public ResponseEntity<String> saveInvoice(@RequestBody Invoice inv){
            ResponseEntity<String> resp = null;
            try{
                    Long id = service.saveInvoice(inv);
                    resp= new ResponseEntity<String>(
                                    "Invoice '"+id+"' created",HttpStatus.CREATED); //201-created
            } catch (Exception e) {
                    e.printStackTrace();
                    resp = new ResponseEntity<String>(
                                    "Unable to save Invoice",
                                    HttpStatus.INTERNAL_SERVER_ERROR); //500-Internal Server Error
            }
            return resp;
    }

    /**
     * To retrieve all Invoices, returns data retrieval Status as ResponseEntity<?>
     */
    @GetMapping("/invoices")
    public ResponseEntity<?> getAllInvoices() {
            ResponseEntity<?> resp=null;
            try {
                    List<Invoice> list= service.getAllInvoices();
                    resp= new ResponseEntity<List<Invoice>>(list,HttpStatus.OK);
            } catch (Exception e) {
                    e.printStackTrace();
                    resp = new ResponseEntity<String>(
                                    "Unable to get Invoice",
                                    HttpStatus.INTERNAL_SERVER_ERROR);
            }
            return resp;
    }

    /**
     * To retrieve one Invoice by providing id, returns Invoice object & Status as ResponseEntity<?>
     */
    @GetMapping("/invoices/{id}")
    public ResponseEntity<?> getOneInvoice(@PathVariable Long id){
            ResponseEntity<?> resp= null;
            try {
                    Invoice inv= service.getOneInvoice(id);
                    resp= new ResponseEntity<Invoice>(inv,HttpStatus.OK);
            }catch (InvoiceNotFoundException nfe) {
                    throw nfe;
            }catch (Exception e) {
                    e.printStackTrace();
                    resp = new ResponseEntity<String>(
                                    "Unable to find Invoice",
                                    HttpStatus.INTERNAL_SERVER_ERROR);
            }
            return resp;
    }

```

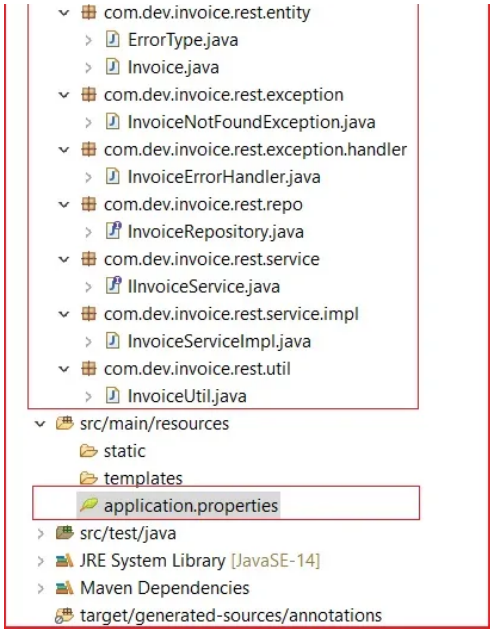
TOP

```
92    /**
93     * To delete one Invoice by providing id, returns Status as ResponseEntity<String>
94     */
95    @DeleteMapping("/invoices/{id}")
96    public ResponseEntity<String> deleteInvoice(@PathVariable Long id){
97
98            ResponseEntity<String> resp= null;
99            try {
100                   service.deleteInvoice(id);
101                   resp= new ResponseEntity<String> (
102                                   "Invoice '"+id+"' deleted",HttpStatus.OK);
103
104           } catch (InvoiceNotFoundException nfe) {
105                   throw nfe;
106           } catch (Exception e) {
107                   e.printStackTrace();
108                   resp= new ResponseEntity<String>(
109                                   "Unable to delete Invoice", HttpStatus.INTERNAL_SERVER_ERROR);
110           }
111
112           return resp;
113   }
114
115   /**
116    * To modify one Invoice by providing id, updates Invoice object & returns Status as ResponseEntity<String>
117    */
118   @PutMapping("/invoices/{id}")
119   public ResponseEntity<String> updateInvoice(@PathVariable Long id, @RequestBody Invoice invoice){
120
121           ResponseEntity<String> resp = null;
122           try {
123                   //db Object
124                   Invoice inv= service.getOneInvoice(id);
125                   //copy non-null values from request to Database object
126                   util.copyNonNullValues(invoice, inv);
127                   //finally update this object
128                   service.updateInvoice(inv);
129                   resp = new ResponseEntity<String>(
130                                   //"Invoice '"+id+"' Updated",
131                                   HttpStatus.RESET_CONTENT); //205- Reset-Content(PUT)
132
133           } catch (InvoiceNotFoundException nfe) {
134                   throw nfe; // re-throw exception to handler
135           } catch (Exception e) {
136                   e.printStackTrace();
137                   resp = new ResponseEntity<String>(
138                                   "Unable to Update Invoice",
139                                   HttpStatus.INTERNAL_SERVER_ERROR); //500-ISE
140           }
141           return resp;
142   }
143
144   /**
145    * To update one Invoice just like where clause condition, updates Invoice object & returns Status as ResponseEntity
146    */
147   @PatchMapping("/invoices/{id}/{number}")
148   public ResponseEntity<String> updateInvoiceNumberById(
149                   @PathVariable Long id,
150                   @PathVariable String number
151                   )
152   {
153           ResponseEntity<String> resp = null;
154           try {
155                   service.updateInvoiceNumberById(number, id);
156                   resp = new ResponseEntity<String>(
157                                   "Invoice '"+number+"' Updated",
158                                   HttpStatus.PARTIAL_CONTENT); //206- Reset-Content(PUT)
159
160           } catch(InvoiceNotFoundException pne) {
161                   throw pne; // re-throw exception to handler
162           } catch (Exception e) {
163                   e.printStackTrace();
164                   resp = new ResponseEntity<String>(
165                                   "Unable to Update Invoice",
166                                   HttpStatus.INTERNAL_SERVER_ERROR); //500-ISE
167           }
168           return resp;
169   }
170   }
```

Finally your project structure would look like below screen.



TOP



How to run the application ?

Further to run the application for testing purpose, right click on Project and then select Run As >> Spring Boot App. You can also package it into a jar & run it accordingly. Additionally, to test the app you need to have a Client application/software.

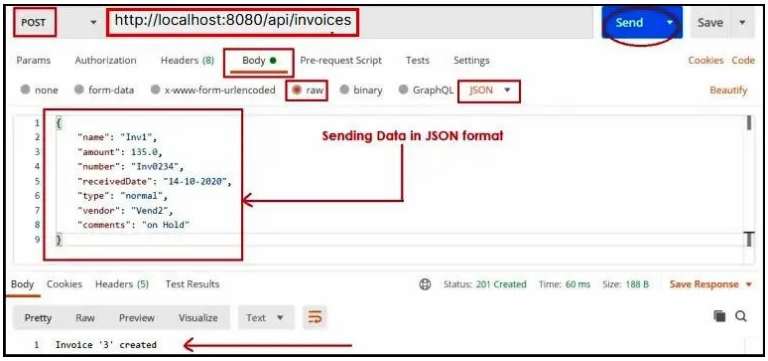
How to test the application ?

As discussed in introduction part, there are multiple ways to test the REST application. At this point, we will suggest you to use the most popular tool 'POSTMAN'. You can also download it from here.

♠ Testing saveInvoice() method : [http://localhost:8080/api/invoices]

Open Postman software, Select method 'POST' from dropdown, enter below URL, select 'Body' then click on 'raw', select 'JSON" from dropdown. All the above selections are highlighted in the below screenshot. Now Enter data in JSON format. If you want to know more about json, visit a separate article on 'how to write data in JSON format'. Once you enter the JSON data, click on 'Send' button & check the successful message in the lower box.

http://localhost:8080/api/invoices



♦ Testing getAllInvoices() method : [http://localhost:8080/api/invoices]

Select method 'GET' from dropdown, then enter below URL. Now click on 'Send' button & check the list of all invoices as a JSON format in lower box.

http://localhost:8080/api/invoices

♠ Testing getOneInvoice() method : [http://localhost:8080/api/invoices/{id}]

{id} represents dynamic data. Suppose we want to retrieve Invoice whose id is 2. Select method 'GET' from dropdown, then e
below URL. Then click on 'Send' button & check the invoice with given id as a JSON format in lower box.

TOP

http://localhost:8080/api/invoices/2

♦ Testing updateInvoice() method : [http://localhost:8080/api/invoices/{id}]

Select method 'PUT' from dropdown, enter below URL, select 'Body' then click on 'raw', select 'JSON" from dropdown. Then Enter data in JSON format to be modified. Then click on 'Send' button and check the successful message in lower box.

http://localhost:8080/api/invoices/2

♠ Testing deleteInvoice() method : [http://localhost:8080/api/invoices/{id}]

Suppose you want to remove an Invoice where id is 2. Then use below pattern URL. Select method 'DELETE' from dropdown, then enter below URL. Then click on 'Send' button and check the successful message in lower box.

http://localhost:8080/api/invoice/2

♦ Testing updateInvoiceNumberById() method : [http://localhost:8080/api/invoices/{id}/{number}]

Suppose you want to modify name field of an Invoice where id is 1. Then use below pattern URL. Select method 'PATCH' from dropdown, enter below URL, select 'Body' then click on 'raw', select 'JSON" from dropdown. Then Enter data in JSON format to be modified. Finally click on 'Send' button & check the successful message in lower box.

http://localhost:8080/api/invoices/1/Inv02345

Can we use this REST API in the real project ?

Of course. You have to change all occurrences of Entity Name as per your real project, then you may use it accordingly.

Summary

Almost every REST application will have these operations we learnt in this article. In other words, no REST API can be developed without these operations. So, you have learnt the mandatory concepts of 'How to develop REST CRUD API using Spring Boot ?'. Furthermore, you can go through other article on How to consume REST API using RestTemplate using Spring Boot.

Tagged @ControllerAdvice @DeleteMapping @deletemapping example in spring boot @ExceptionHandler @GetMapping @getmapping annotation used on @getmapping spring boot example @Modifying @PatchMapping @patchmapping example @PathVariable @PostMapping @PutMapping @Query @RequestBody @RequestMapping @ResponseBody @RestController @RestControllerAdvice @Transactional api rest spring boot api rest spring boot mysql Building a RESTful Web Service - Spring create rest api in java create rest api in java with spring boot crud api CRUD example spring boot with mySql and postman crud interface java crud rest api example exception handling spring boot rest api how to create a rest api in java How to create a REST API using Java Spring Boot How to Create a REST API With Spring Boot How to develop CRUD REST API using Spring Boot ? How to develop REST API using Spring Boot ? How to develop REST CRUD API using Spring Boot ? how to implement rest api in java java spring boot rest api rest api example java rest api using spring boot rest api with spring boot rest crud rest crud operations restful web services in spring boot restful web services spring boot spring boot crud example spring boot crud operations spring boot crud rest api spring boot rest api spring boot rest api best practices spring boot rest api crud example spring boot rest api crud example with mysql database spring boot rest api example spring boot rest api tutorial spring boot rest controller exception handling spring boot rest example spring boot rest web service spring boot rest web service example spring boot rest web services spring boot restful web services crud example spring boot restful web services example step by step spring boot restful web services json example spring boot web services spring rest api crud example spring rest api tutorial spring rest example spring rest web services example spring web services spring web services rest

‹ **Previous article**
What is Java JSON

Next article ›
How to reduce boilerplate code in REST API Spring Data REST

11 thoughts on "How to develop REST CRUD API using Spring Boot ?"

Pingback: How to reduce boilerplate code in REST API : Spring Data REST - Making Java easy to learn

Devs5003
October 26, 2020 at 2:16 AM
ok

Reply

Pingback: How to write REST Consumer API using Spring Boot : RestTemplate - Making Java easy to learn

↑ TOP

October 30, 2020 at 3:30 AM
Devs5003
ok

Reply

Pingback: How to handle Exceptions & Errors in Spring Boot? | Making Java easy to learn

Pingback: How to develop a Reactive CRUD REST API with Spring WebFlux? | Making Java easy to learn

Pingback: How to work with JSON in Java? | Making Java easy to learn

Pingback: Microservices Architecture | Making Java easy to learn

April 11, 2021 at 10:58 PM
Lehel
I want to say that this post is awesome, great written and include almost all necessary information.

I would like to see more posts like this .

Reply

April 18, 2021 at 1:59 AM
Delila Dorey
This is a topic that's near to my heart... Cheers! Exactly where are your contact details though?

Reply

April 27, 2021 at 3:47 PM
Devs5003
Always click on 'Contact Us' to connect with us.

Reply

Leave a Reply

Name *

Email Address *

Website

Comment *

☐ Yes, add me to your mailing list

Post Comment

FOLLOW US

RECENTLY PUBLISHED POSTS

- **Spring Boot Interview Questions**
- **Java Interface**
- **Spring Security Without WebSecurityConfigurerAdapter**
- **Spring Boot Batch Example CSV to MySQL Using JPA**
- **Spring Batch Tutorial**
- **Java 14 Features**

TOP

- **Java Features After Java 8**
- **Java Core Tutorial**
- **Spring Cloud Annotations With Examples**
- **How To Add JDK 18 Support in Eclipse**

DO YOU HAVE A QUERY ?

Submit your Query

Email *

Subscribe to receive Updates !