PROJECT

SOFTWARE CONSTRUCTION AND DEVELOPMENT

BSE-5(A)

SUBMITTED TO: SIR SHAHZAD

SUBMITTED BY:

Laraib Tanveer (2022-BSE-016)
Muneeba Bibi (2022-BSE-023)
Alishbah Waseem Malik (2022-BSE-003)

DEPARTMENT:

Software Engineering

Date: December 23, 2024

Title: SPRING BOOT REST API DEVELOPMENT

AIM

To implement and master the development of Java REST APIs using Spring Boot through step-by-step sessions covering foundational concepts, CRUD operations, exception handling, JSON responses, and unit testing.

Objectives

1. To set up a Spring Boot application for creating REST APIs.
2. To implement CRUD operations using Spring Boot, Spring Data JPA, and MySQL.
3. To handle exceptions effectively in a Spring Boot REST API.
4. To produce structured JSON responses in REST APIs.
5. To master unit testing of Spring Boot REST APIs.

Introduction

This project is a comprehensive exploration of Java REST API development using Spring Boot. It focuses on building scalable and maintainable APIs, starting from the basics of setup to advanced concepts such as exception handling and unit testing. Through a series of five sessions, the project delves into CRUD operations, structured JSON responses, and robust testing strategies. This practical approach equips developers with hands-on experience in API development and deployment.

Sessions Overview

Session 1: Java REST API with Spring Boot

Session 2: Creating Java REST API with Spring Boot, Spring Data JPA and MySQL | REST API CRUD Operations

Session 3: Exception Handling in Spring Boot REST API

Session 4: Java Spring Boot REST API JSON Response

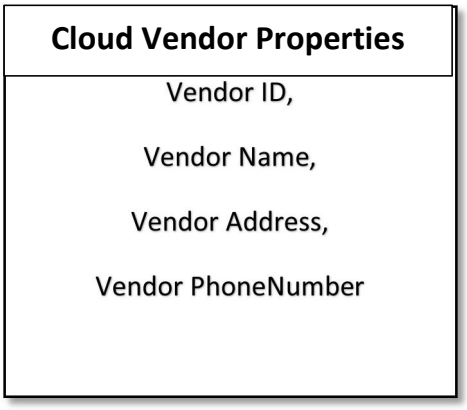
Session 5: Master Unit Testing Java Spring Boot REST API Application in One Shot

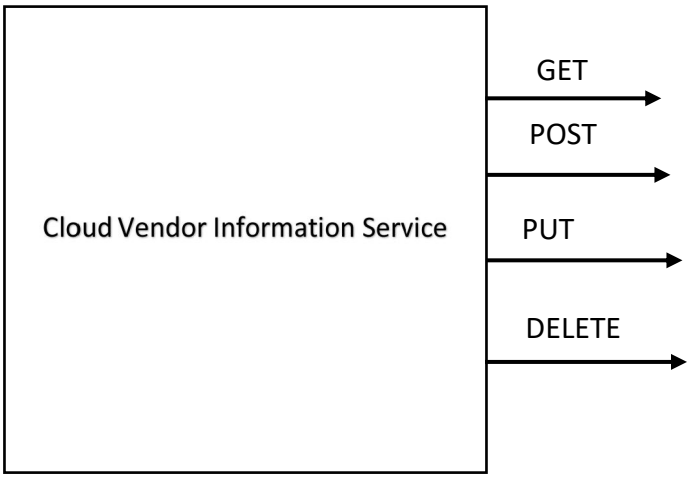
We will learn how to create REST APIs from scratch using Spring Boot. The process will start with setting up a basic Spring Boot application through Spring Initializer. Then, we will develop the required code to convert it into a fully functional REST API. Finally, we will test the REST API using the Postman tool, covering all four CRUD operations—Create, Read, Update, and Delete—implemented with Spring Boot.

➢ Spring Boot Project Architecture

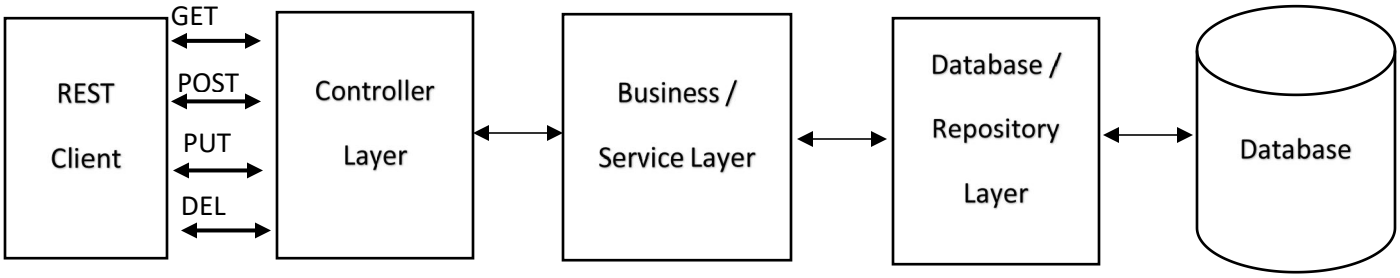
CLOUD VENDOR API SERVICE

Cloud Vendor class





CLOUD VENDOR SERVICE INFO LAYERS



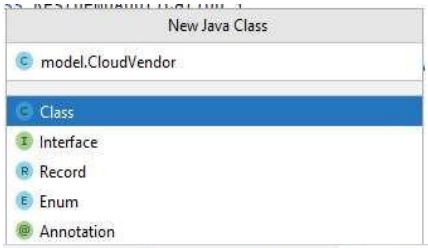
- In this instance, the database is "***MySQL workbench***," and the Rest Client is "***POSTMAN***."
- The "***Controller Layer***" communicates with the external environment and the REST Client. receives and responds to REST requests.
- The database and the "***Database/Repository Layer***" communicate.
- As the name suggests, any business logic you create for your project should belong in the intermediate tier, known as the "***Business/Service Layer***."

➢ Creating spring boot Application

⇒ SESSION # 01

➢ RestDemoApplication.java

➢ CloudVendor.java



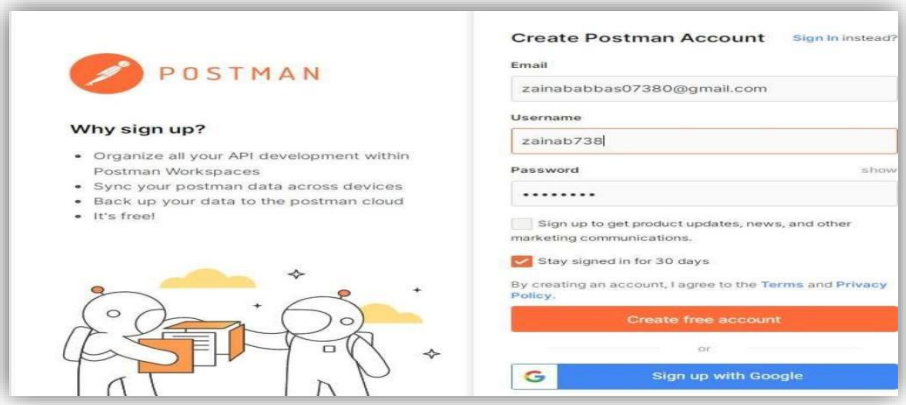
xml (rest-demo) | RestDemoApplication.java | CloudVendor.java

```
package com.thinkconstructive.restdemo.model;

public class CloudVendor
{
    3 usages
    private String vendorId;
    3 usages
    private String vendorName;
    3 usages
    private String vendorAddress;
    3 usages
    private String vendorPhoneNumber;

    public CloudVendor()
    {
    }
public CloudVendor(String vendorId, String vendorName, String vendorAddress, String vendorPhoneNumber) {
    this.vendorId = vendorId;
    this.vendorName = vendorName;
    this.vendorAddress = vendorAddress;
    this.vendorPhoneNumber = vendorPhoneNumber;
}
public String getVendorId() {
    return vendorId;
}

public void setVendorId(String vendorId) {
    this.vendorId = vendorId;
}
 public void setVendorName(String vendorName) {
     this.vendorName = vendorName;
 }

 public String getVendorAddress() {
     return vendorAddress;
 }

 public void setVendorAddress(String vendorAddress) {
     this.vendorAddress = vendorAddress;
 }

 public String getVendorPhoneNumber() {
     return vendorPhoneNumber;
 }
    public void setVendorPhoneNumber(String vendorPhoneNumber) {
        this.vendorPhoneNumber = vendorPhoneNumber;
    }

}
```

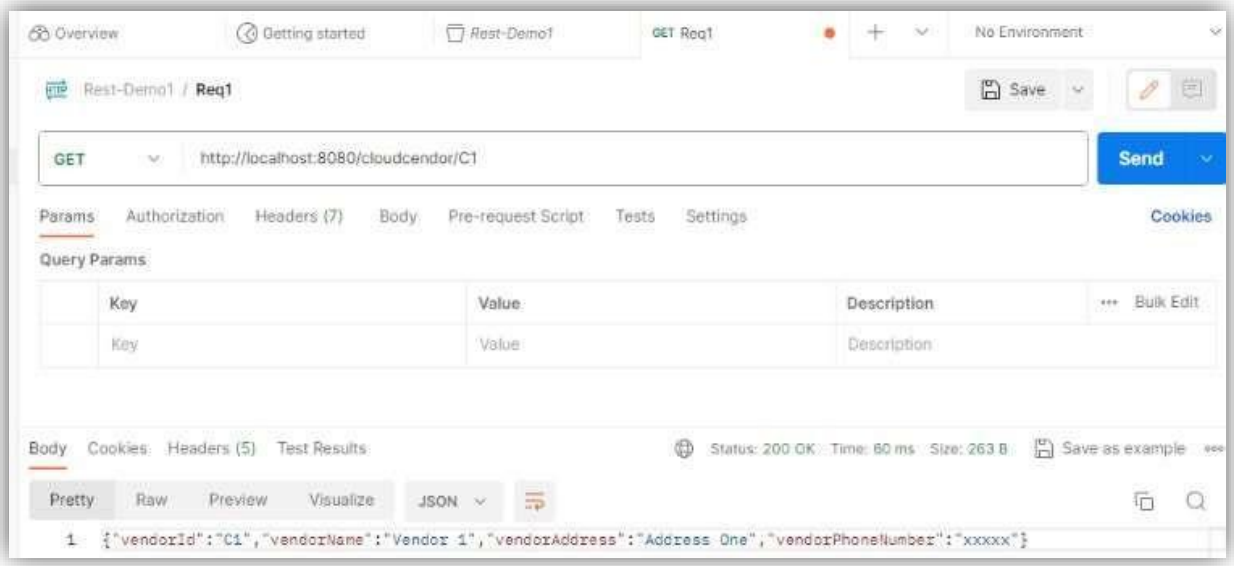
➢ CloudApiService.java


```
ml (rest-demo) × RestDemoApplication.java × CloudVendor.java × CloudVendorAPIService.java ×
package com.thinkconstructive.restdemo.controller;

import com.thinkconstructive.restdemo.model.CloudVendor;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/cloudcendor")
public class CloudVendorAPIService
{
    @GetMapping("{vendorId}")
    public CloudVendor getCloudVendorDetails(String vendorId)
    {
        return new CloudVendor( vendorId: "C1", vendorName: "Vendor 1",
                vendorAddress: "Address One", vendorPhoneNumber: "xxxxx");
    }
}
```

➢ <u>Postman</u>



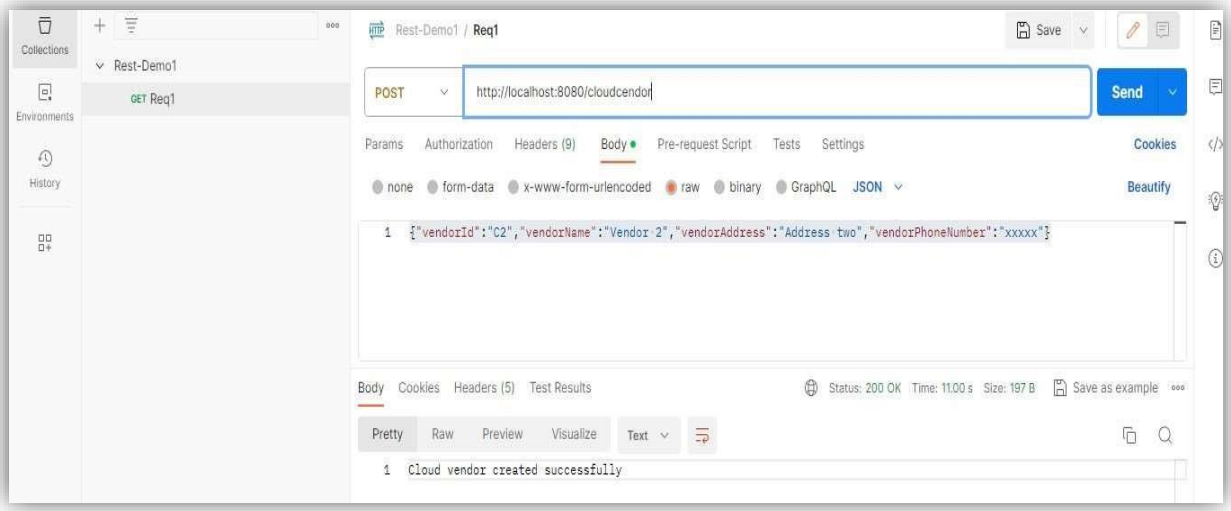
- **Accessing C1 through postman**



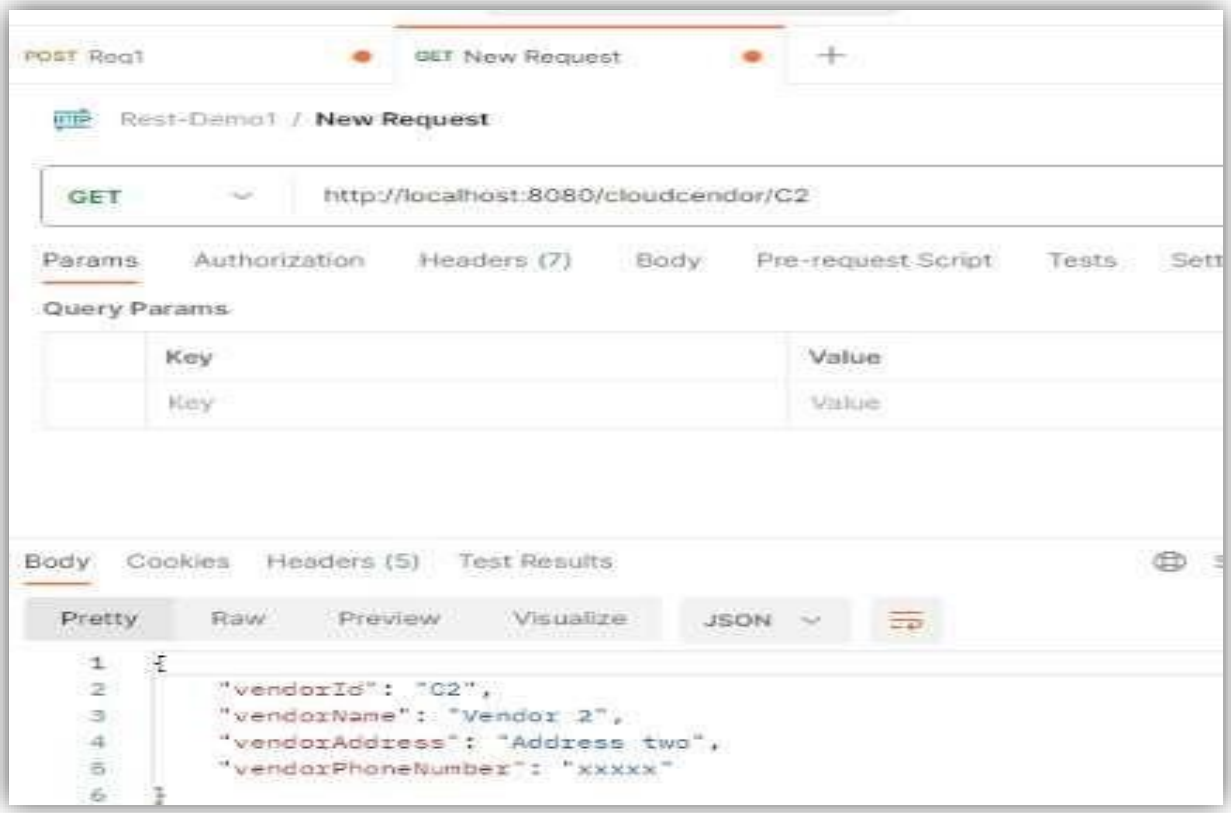
- **Adding create cloud vendor function in CloudVendorAPIService.java**

```
.xml (rest-demo)   RestDemoApplication.java   CloudVendor.java   CloudVendorAPIService.java
package com.thinkconstructive.restdemo.controller;

import com.thinkconstructive.restdemo.model.CloudVendor;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/cloudcendor")
public class CloudVendorAPIService
{
    2 usages
    CloudVendor cloudVendor;
    @GetMapping("{vendorId}")
    public CloudVendor getCloudVendorDetails(String vendorId)
    {
        return cloudVendor;
    }
    @PostMapping
    public String createCloudVendorDetails(@RequestBody CloudVendor cloudVendor)
    {
        this.cloudVendor=cloudVendor;
        return "Cloud vendor created successfully";
    }
}
```

- **Making a new Cloud vendor “C2” using Postman**



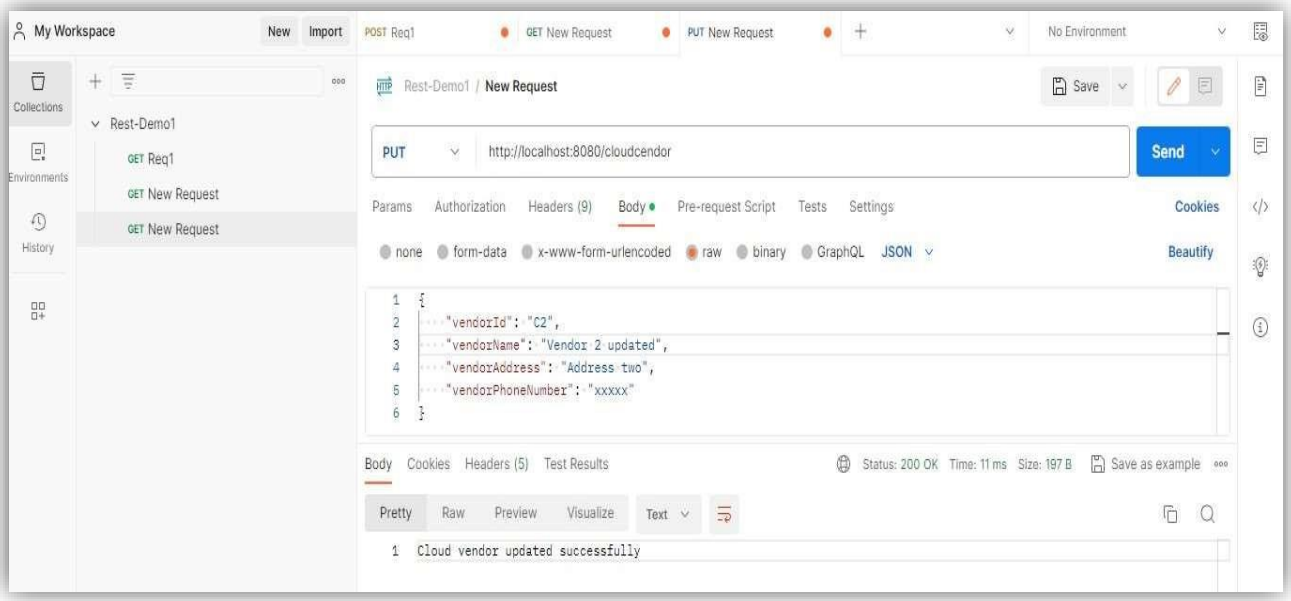
- **Accessing C2 through postman**



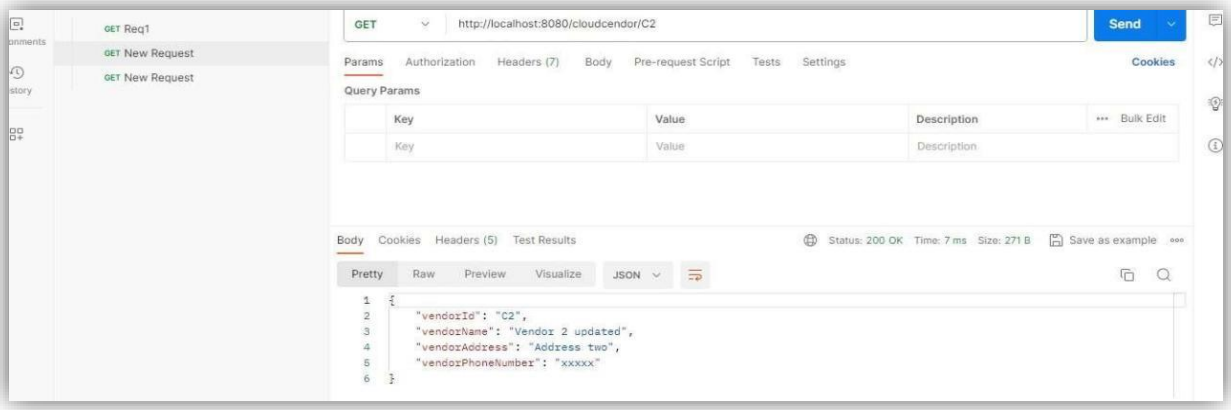
- **Add update cloud vendor function in CloudVendorAPIService.java**

```
@GetMapping("{vendorId}")
public CloudVendor getCloudVendorDetails(String vendorId)
{
    return cloudVendor;
}
@PostMapping
public String createCloudVendorDetails(@RequestBody CloudVendor cloudVendor)
{
    this.cloudVendor=cloudVendor;
    return "Cloud vendor created successfully";
}
@PutMapping
public String updateCloudVendorDetails(@RequestBody CloudVendor cloudVendor)
{
    this.cloudVendor=cloudVendor;
    return "Cloud vendor updated successfully";
}
}
```

- **Updating C2 using Postman**

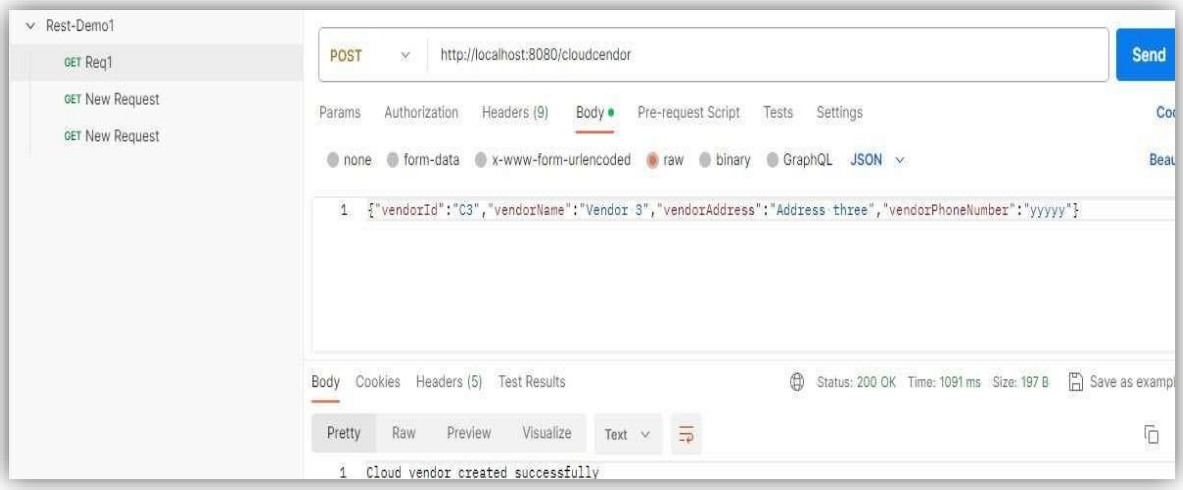


- **Assessing updated C2 using Postman**

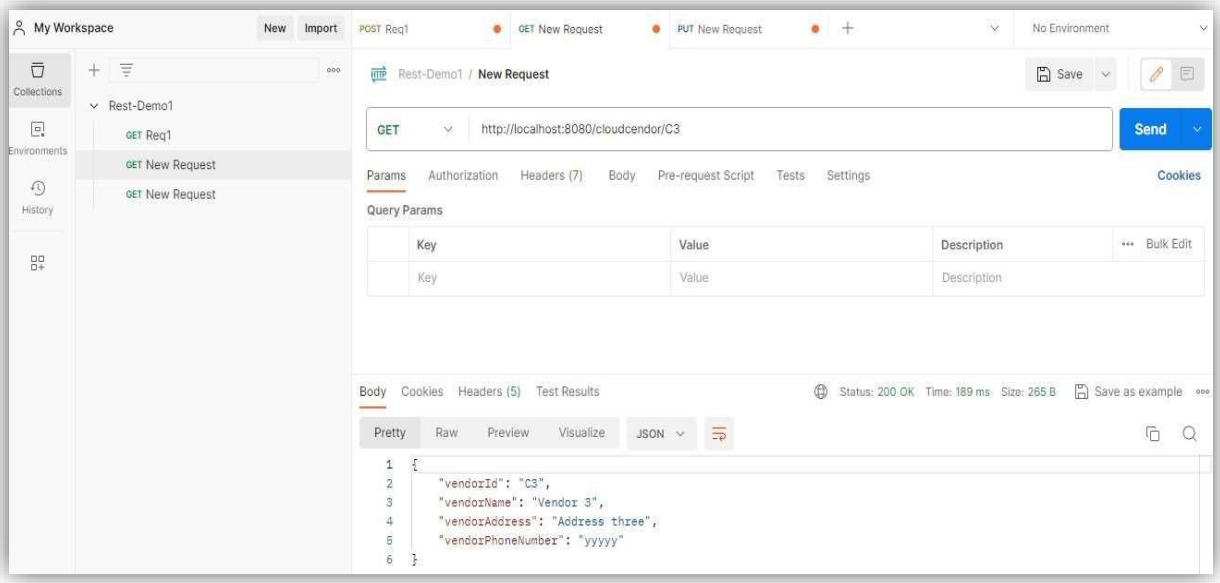


- **Add delete cloud vendor function in CloudServiceApiService.java**

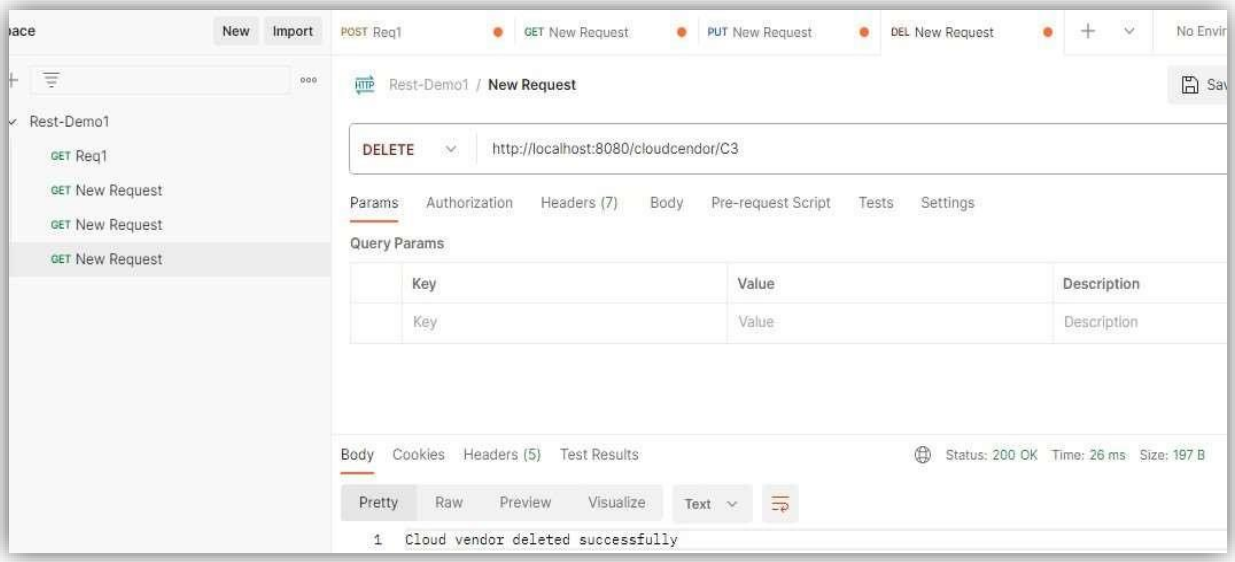
- **Making a new cloud vendor "C3" using Postman**



- **Accessing C3 using Postman**

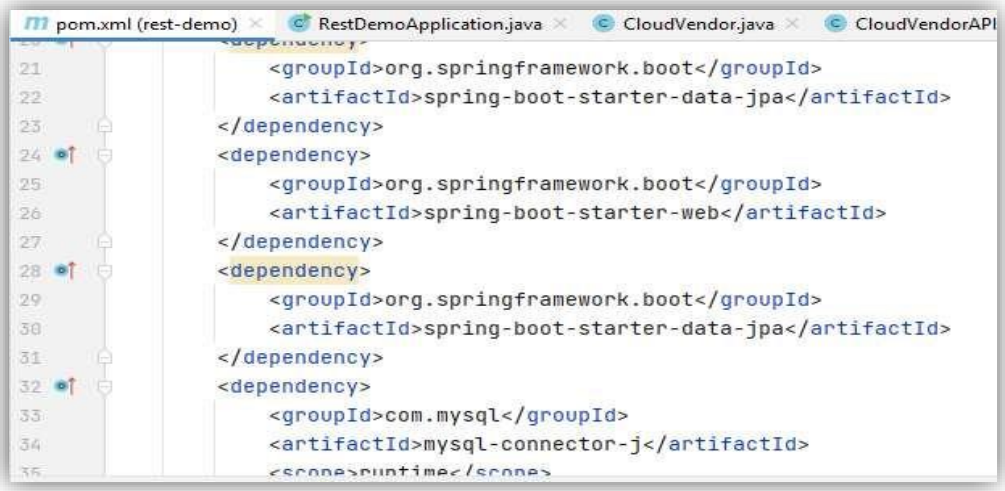


➢ Deleting C3 using Postman



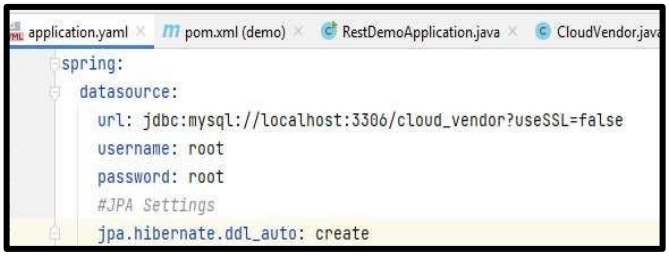
We will discover how to use the ***application.yaml*** file to set up a MySQL database for a Spring Boot REST API. In order to facilitate smooth communication between the MySQL database and the Spring Boot REST API, we will also apply the Service and Repository patterns. Lastly, we will use MySQL Workbench and Postman to test the application.

➢ Configuring pom.xml



⇒ SESSION # 02

- **Creating a new file named "Application.yaml"**

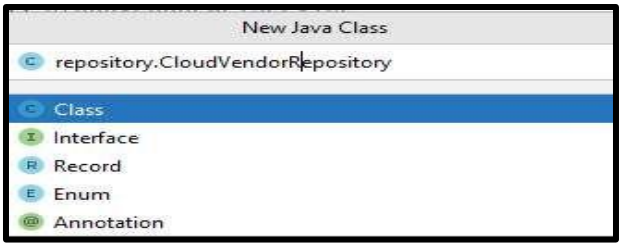


- **Creating cloud_vendor schema**

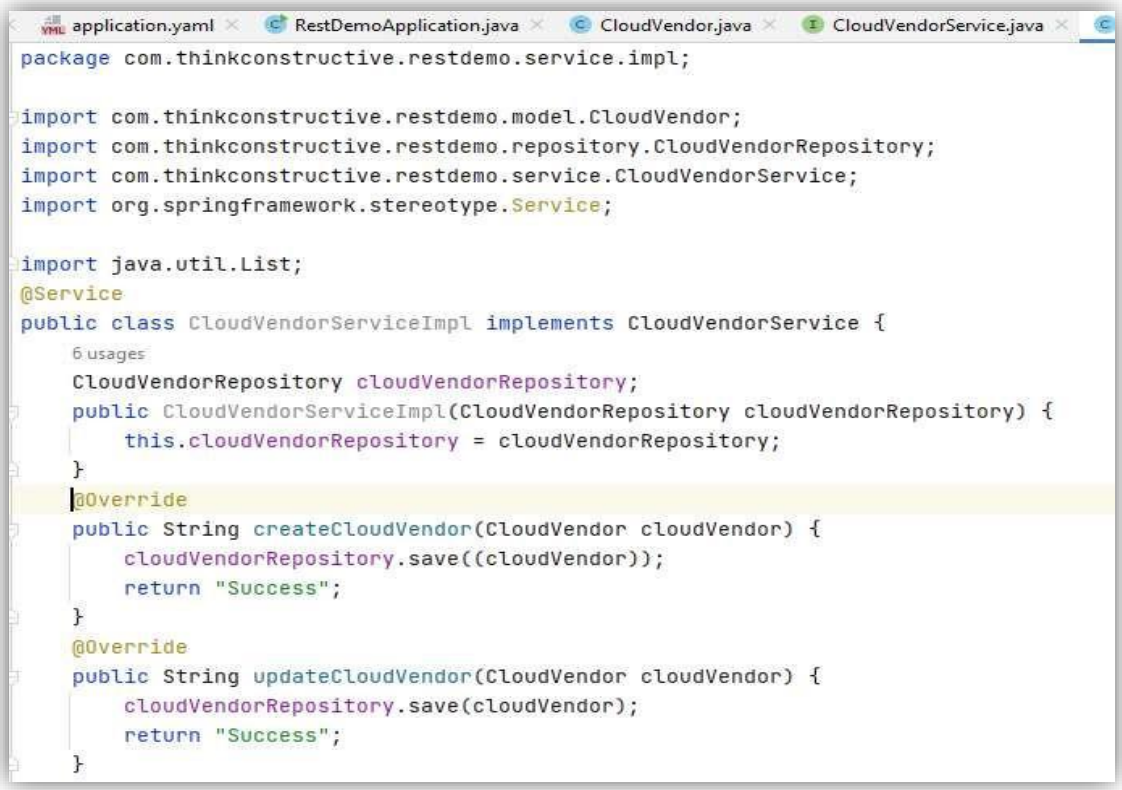
- **Modifying CloudVendor.java**

```java
import jakarta.persistence.Id;
import jakarta.persistence.Table;

7 usages
@Entity
@Table(name = "cloud_vendor_info")
public class CloudVendor
{
    3 usages
    @Id
    private String vendorId;
    3 usages
    private String vendorName;
    3 usages
    private String vendorAddress;
    3 usages
    private String vendorPhoneNumber;

    public CloudVendor()
    {
    }
    public CloudVendor(String vendorId, String vendorName, String vendorAddress, String vendorPhoneNumber) {
        this.vendorId = vendorId;
        this.vendorName = vendorName;
        this.vendorAddress = vendorAddress;
        this.vendorPhoneNumber = vendorPhoneNumber;
    }
```

- **Creating repository named "CloudVendorRepository"**




```java
@Override
public String deleteCloudVendor(String cloudVendorId) {
    cloudVendorRepository.deleteById(cloudVendorId);
    return "Success";
}
@Override
public CloudVendor getCloudVendor(String cloudVendorId) {
    return cloudVendorRepository.findById(cloudVendorId).get();
}

@Override
public List<CloudVendor> getAllCloudVendors() {
    return cloudVendorRepository.findAll();
}
```



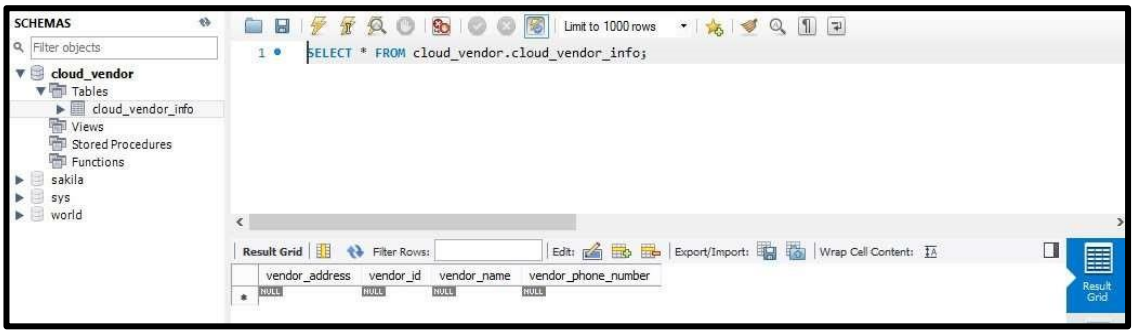
CloudVendorServiceImpl

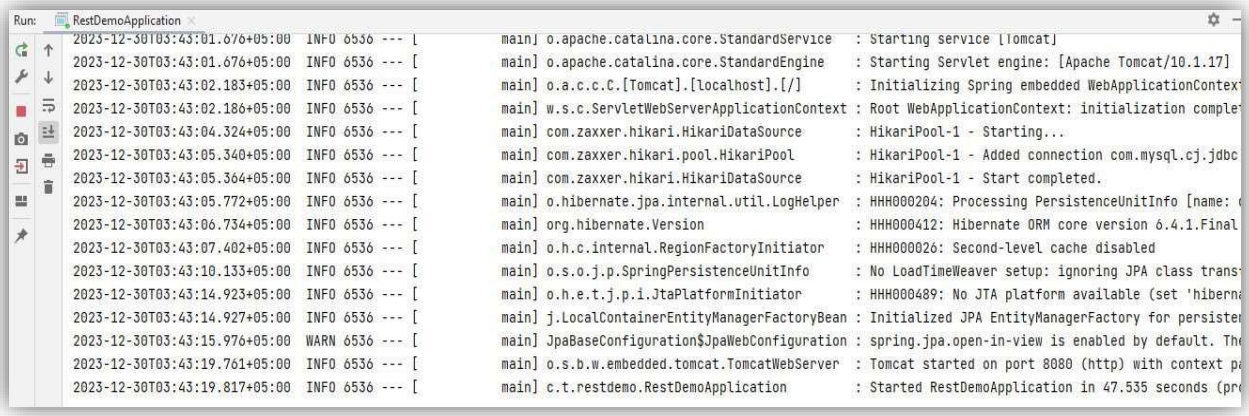
- **CloudVendorController.java**

```
package com.thinkconstructive.restdemo.model.controller;

import com.thinkconstructive.restdemo.model.CloudVendor;
import com.thinkconstructive.restdemo.service.CloudVendorService;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/cloudcendor")
public class CloudVendorController
{
    6 usages
    CloudVendorService cloudVendorService;
    public CloudVendorController(CloudVendorService cloudVendorService) {
        this.cloudVendorService = cloudVendorService;
    }

    @GetMapping("{vendorId}")
    public CloudVendor getCloudVendorDetails(@PathVariable("vendorId") String vendorId)
    {
        return cloudVendorService.getCloudVendor(vendorId);
    }
```

```java
@GetMapping()
public List<CloudVendor> getAllCloudVendorDetails()
{
    return cloudVendorService.getAllCloudVendors();
}
@PostMapping
public String createCloudVendorDetails(@RequestBody CloudVendor cloudVendor)
{
    cloudVendorService.createCloudVendor(cloudVendor);
    return "Cloud vendor created successfully";
}
@PutMapping
public String updateCloudVendorDetails(@RequestBody CloudVendor cloudVendor)
{
    cloudVendorService.updateCloudVendor(cloudVendor);
    return "Cloud vendor updated successfully";
}
@DeleteMapping("{vendorId}")
public String deleteCloudVendorDetails(@PathVariable("vendorId") String vendorId)
{
    cloudVendorService.deleteCloudVendor(vendorId);
    return "Cloud vendor deleted successfully";
}
```

➢ Application Started

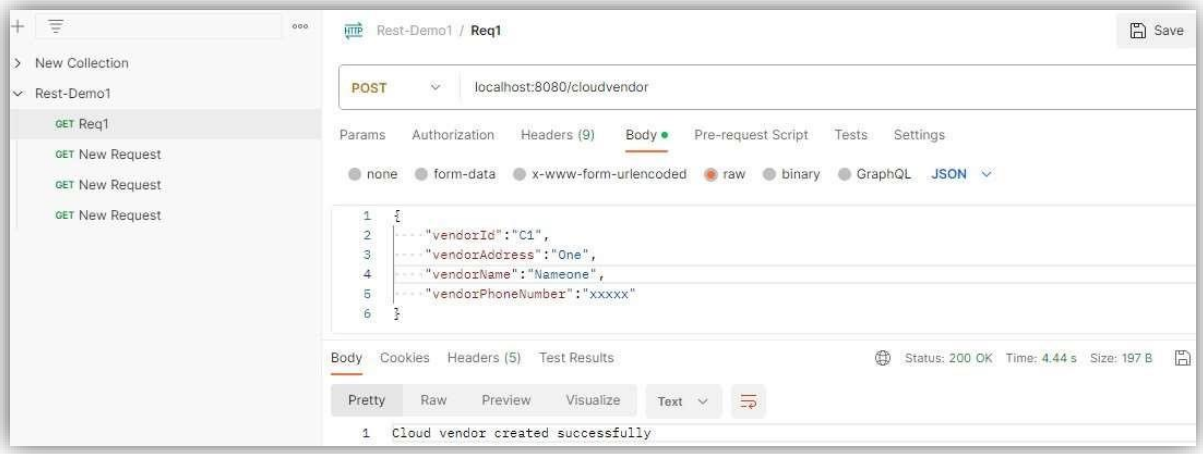




Open Mysql Workbench

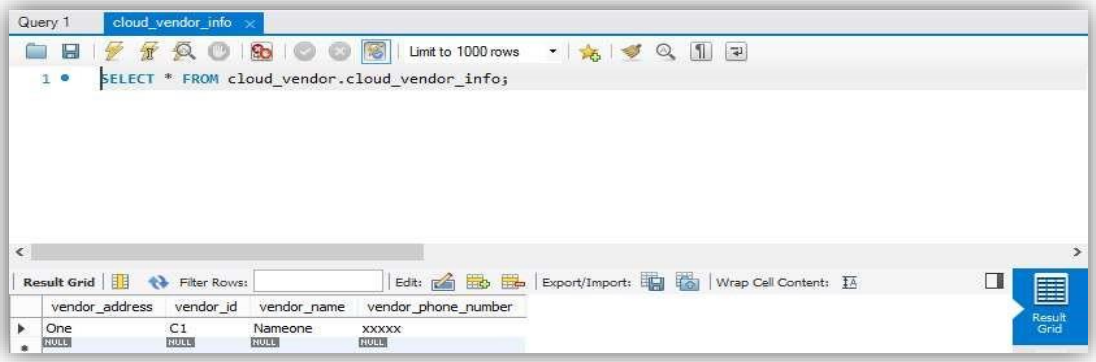
- **Table cloud_vendor_info is automatically created after running the code**

Create a cloud vendor using Postman

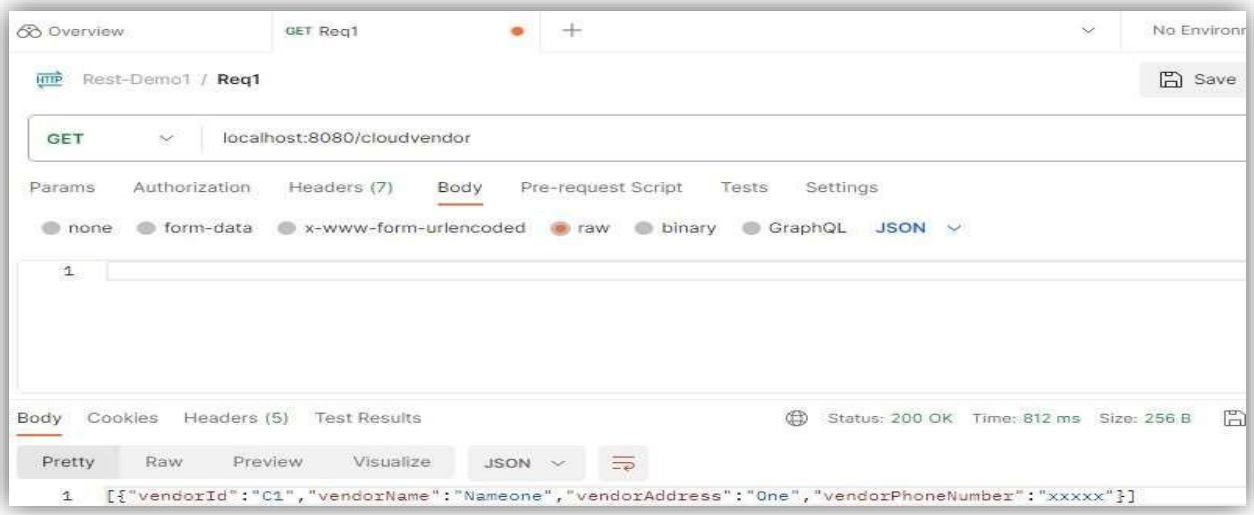


Select table row from mysql workbench

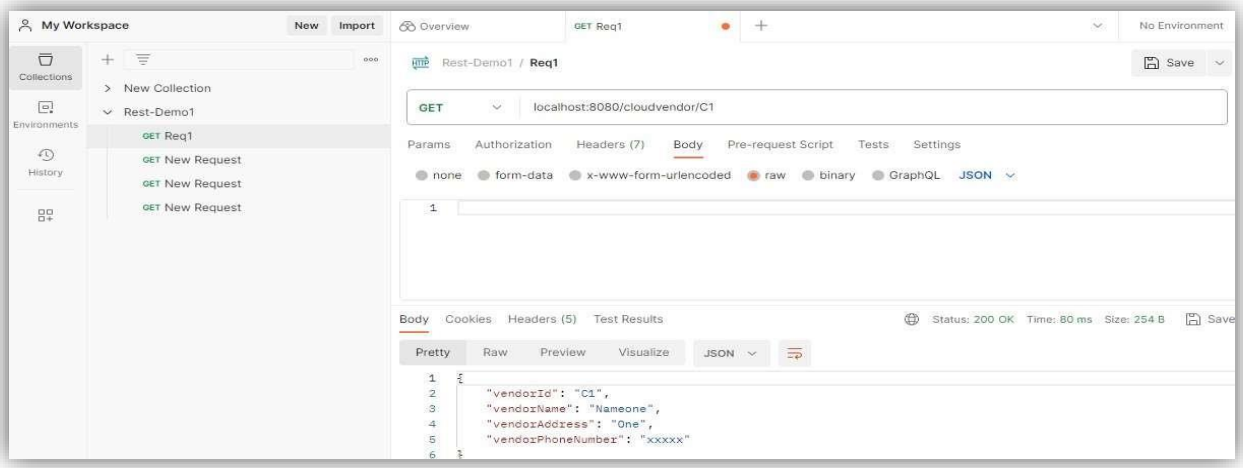
- **Table cloud_vendor_info is updated and a new row is created**



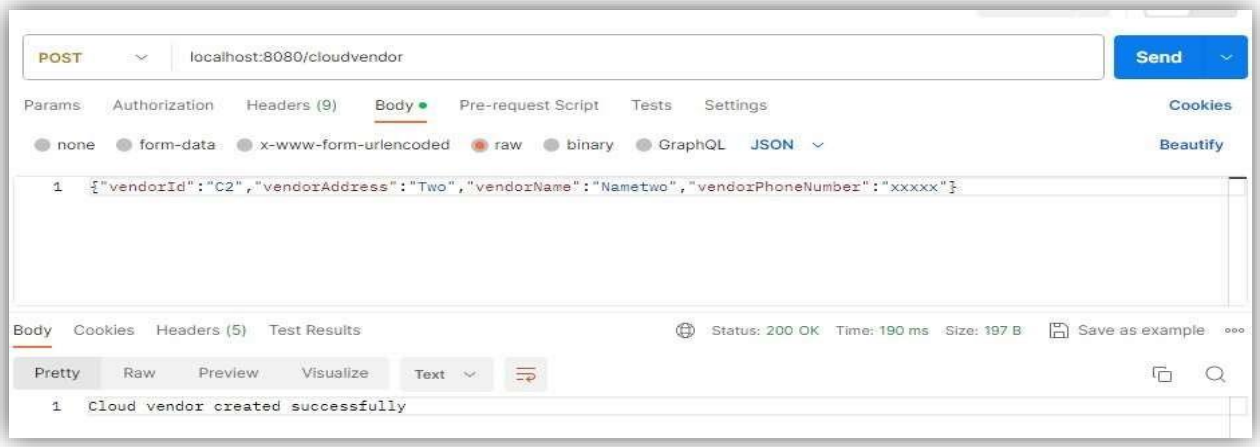
Use GetAll method in Postman



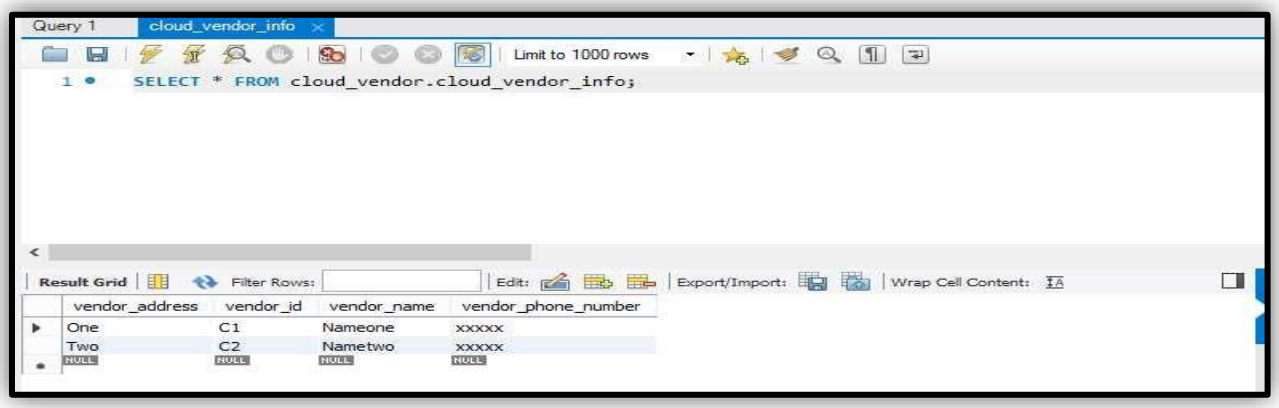
Using GetById method in Postman



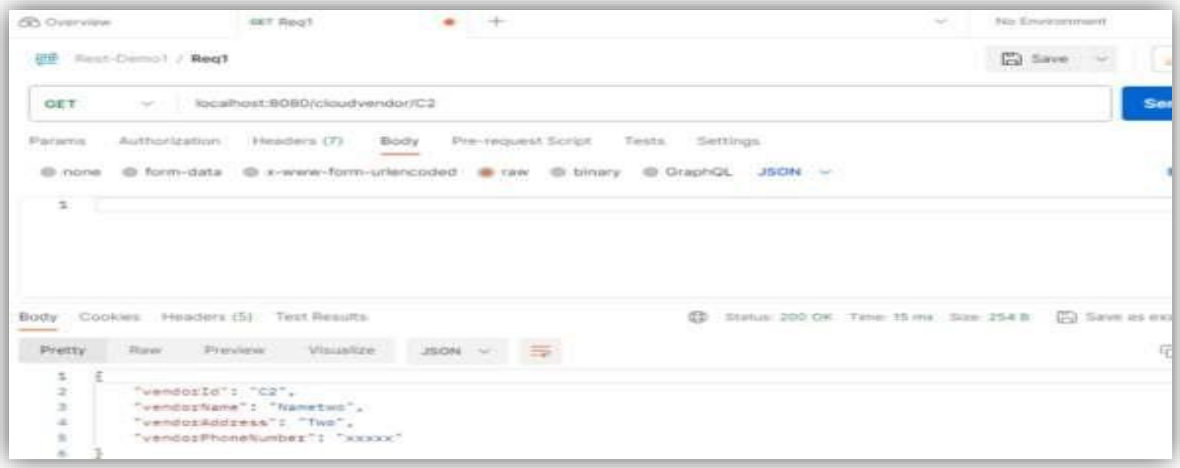
- **Create more Vendors using Postman**



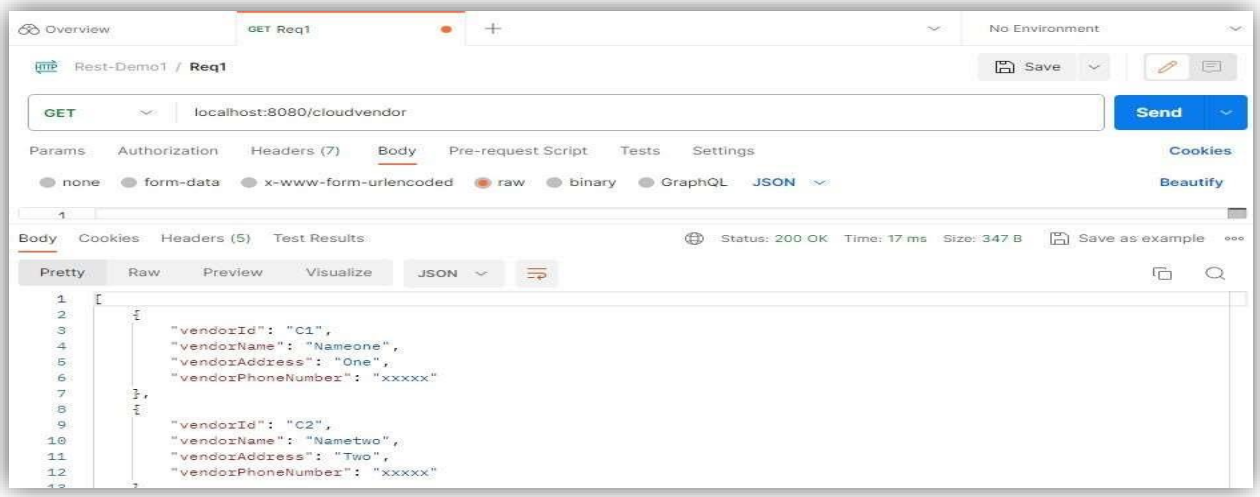
Selecting all rows using mysql workbench



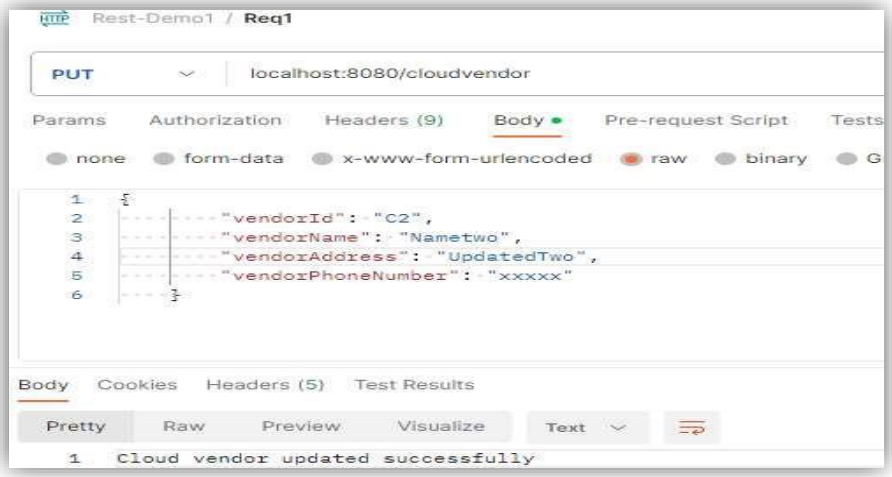
- **Now accessing C2 using GetById using Postman**



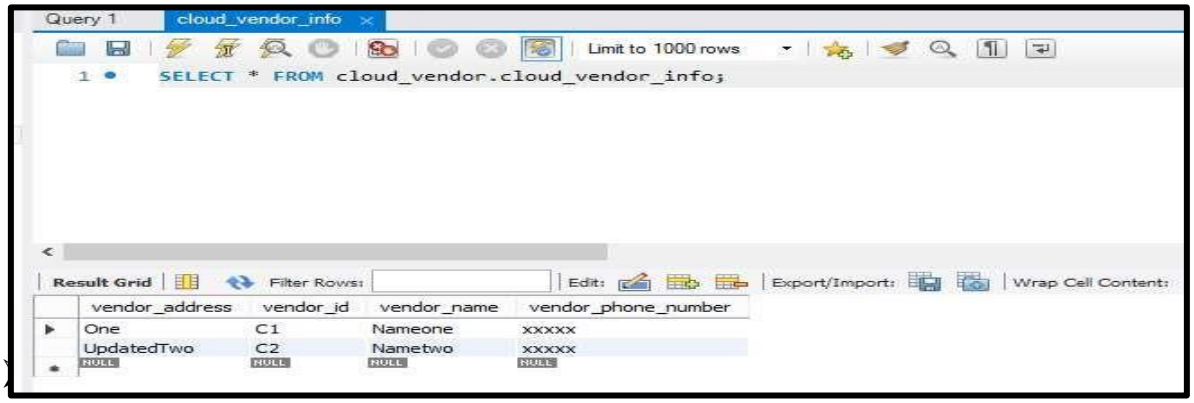
- **Again using GetAll method in Postman**

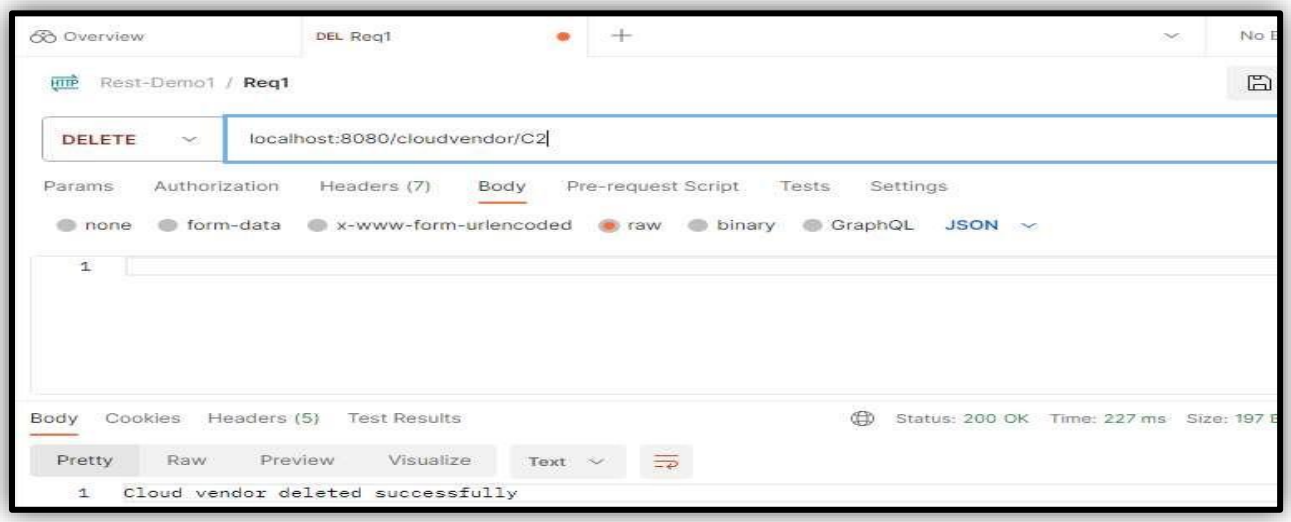


Updating Vendor Address of C2

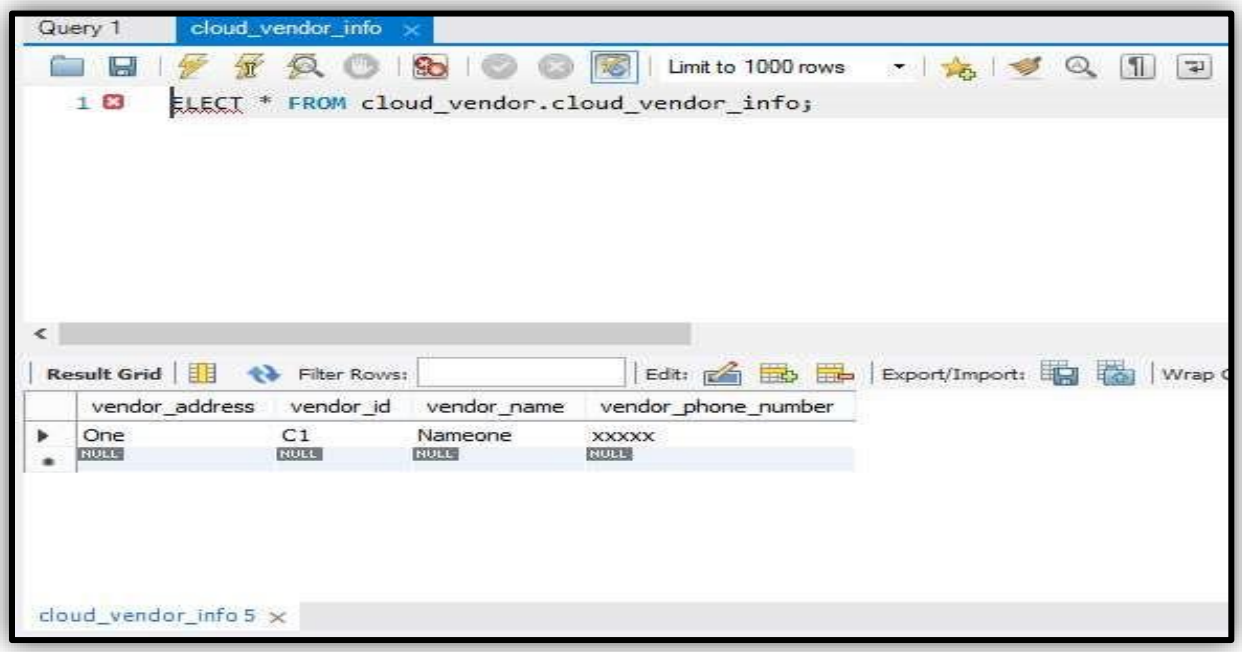


Mysql workbench





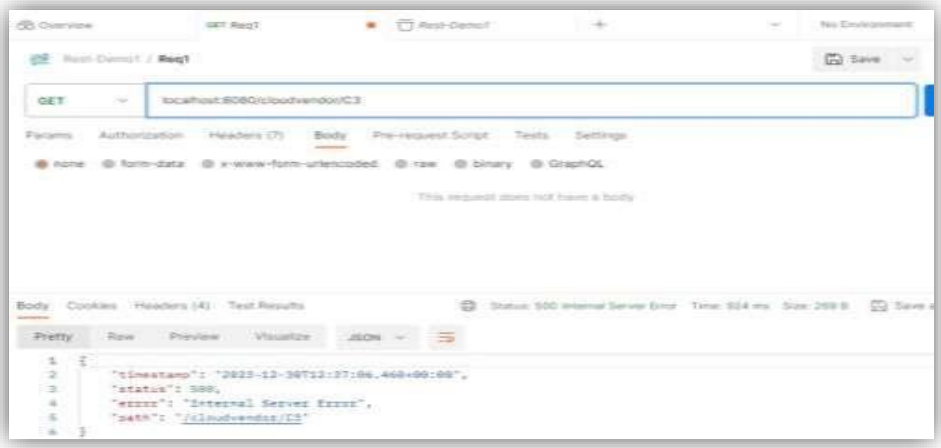
Mysql Workbench



In order to learn how to properly handle exceptions in a Spring Boot REST API project, we will focusing on exception handling in a Spring Boot REST API Java application.

⇒ SESSION # 03

- **GetbyId method using postman**



The error in this case arises from the nonexistence of C3. However, the root cause is not always immediately apparent when this problem is encountered. As a result, we will modify our code in this session to throw exceptions that provide greater clarity and insight into the true problem.

➢ CloudVendorNotFoundException.java

```
lication.properties  RestDemoApplication.java  CloudVendorNotFoundException.java  CloudVendorExce
package com.thinkconstructive.restdemo.exception;

public class CloudVendorNotFoundException extends RuntimeException{

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

    public CloudVendorNotFoundException(String message, Throwable cause) {
        super(message, cause);
    }
}
```

CloudVendorException.java

```
RestDemoApplication.java  CloudVendorNotFoundException.java  CloudVendorException.java  CloudVendorExceptionHandler.java
1   package com.thinkconstructive.restdemo.exception;
2
3   import org.springframework.http.HttpStatus;
4
5   public class CloudVendorException {
        2 usages
6       private final String message;
        2 usages
7       private final String throwable;
        2 usages
8       private final HttpStatus httpStatus;
9       public String getMessage() {
10          return message;
11      }
12
13      public String getThrowable() {
14          return throwable;
15      }
16
17      public HttpStatus getHttpStatus() {
18          return httpStatus;
19      }
20
21
22      public CloudVendorException(String message, String throwable, HttpStatus httpStatus) {
23          this.message = message;
```

```
        return message;
    }

    public String getThrowable() {
        return throwable;
    }

    public HttpStatus getHttpStatus() {
        return httpStatus;
    }


    public CloudVendorException(String message, String throwable, HttpStatus httpStatus) {
        this.message = message;
        this.throwable = throwable;
        this.httpStatus = httpStatus;
    }


}
```

CloudVendorExceptionHandler.java

```java
package com.thinkconstructive.restdemo.exception;

import org.aspectj.apache.bcel.Repository;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
@ControllerAdvice
public class CloudVendorExceptionHandler {
    @ExceptionHandler(value = {CloudVendorNotFoundException.class})
    public ResponseEntity<Object> handleCloudVendorNotFoundException
            (CloudVendorNotFoundException cloudVendorNotFoundException)
    {
        CloudVendorException cloudVendorException= new CloudVendorException(
                cloudVendorNotFoundException.getMessage(),cloudVendorNotFoundException.getCause(),
                HttpStatus.NOT_FOUND
        );
        return new ResponseEntity<>(cloudVendorException, HttpStatus.NOT_FOUND);
    }
}
```

CloudVendorServiceimpl.java

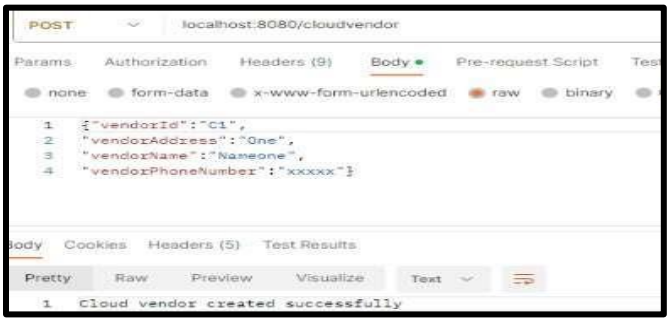
```java
1 usage
@Override
public String updateCloudVendor(CloudVendor cloudVendor) {
    cloudVendorRepository.save(cloudVendor);
    return "Success";
}
1 usage
@Override
public String deleteCloudVendor(String cloudVendorId) {
    cloudVendorRepository.deleteById(cloudVendorId);
    return "Success";
}
1 usage
@Override
public CloudVendor getCloudVendor(String cloudVendorId) {
    if (cloudVendorRepository.findById(cloudVendorId).isEmpty())
    throw new CloudVendorNotFoundException("Requested Cloud Vendor doesnot exists");
        return cloudVendorRepository.findById(cloudVendorId).get();
}

1 usage
@Override
public List<CloudVendor> getAllCloudVendors() { return cloudVendorRepository.findAll(); }
}
```

- **Now check error**

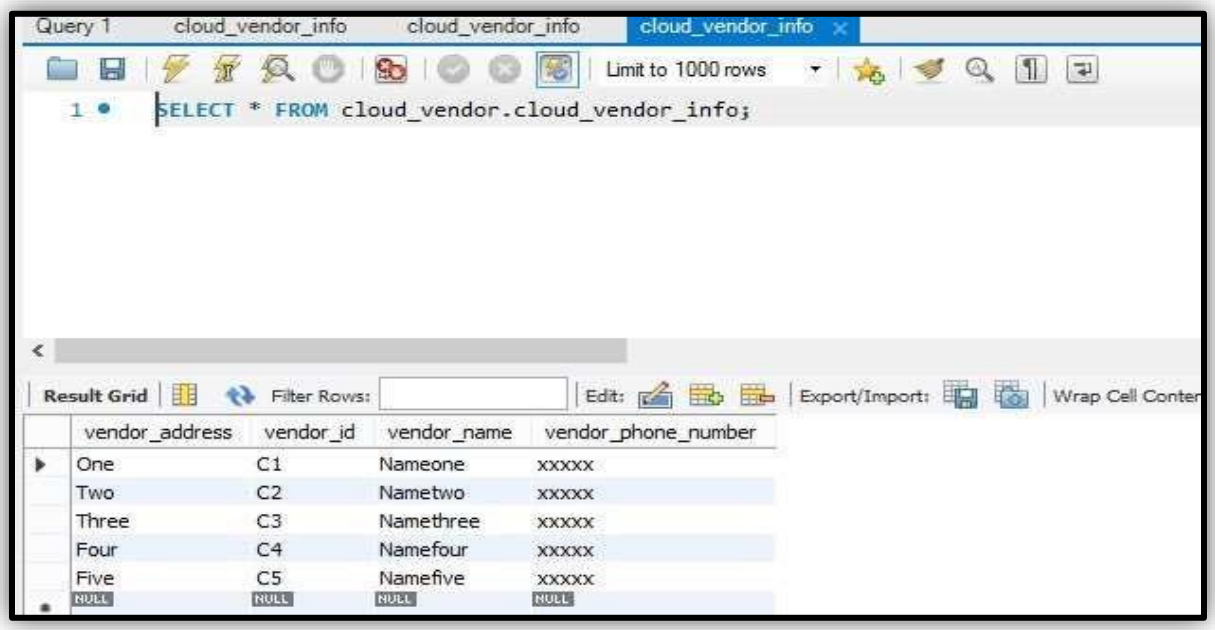
Now the error message clearly tells us that our cloud vendor doesn't exists

The CloudVendorAPI application will be enhanced to generate and return customized REST API responses to clients using the ***ResponseEntity*** class.

Populating table

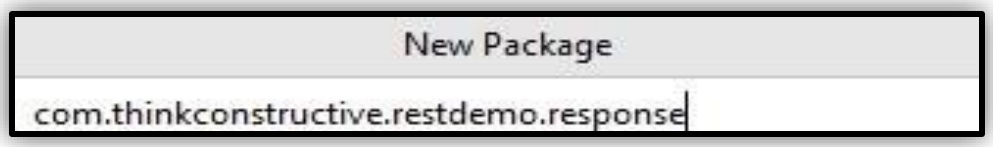


Mysql workbench



⇒ SESSION # 04

Creating a response



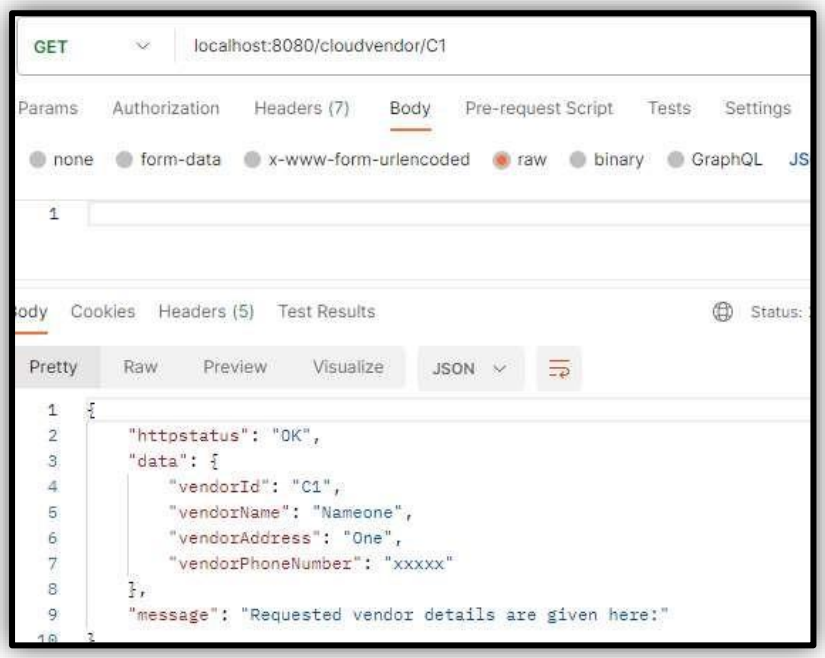
ResponseHandler.java

```
package com.thinkconstructive.restdemo.response;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;

import java.util.HashMap;
import java.util.Map;

public class ResponseHandler {
    public static ResponseEntity<Object>responseBuilder(
            String message, HttpStatus httpStatus, Object responseObject
            )
    {
        Map<String, Object> response=new HashMap<>();
        response.put("message",message);
        response.put("httpstatus",httpStatus);
        response.put("data",responseObject);
        return new ResponseEntity<>(response,httpStatus);
    }
}
```

CloudVendorController

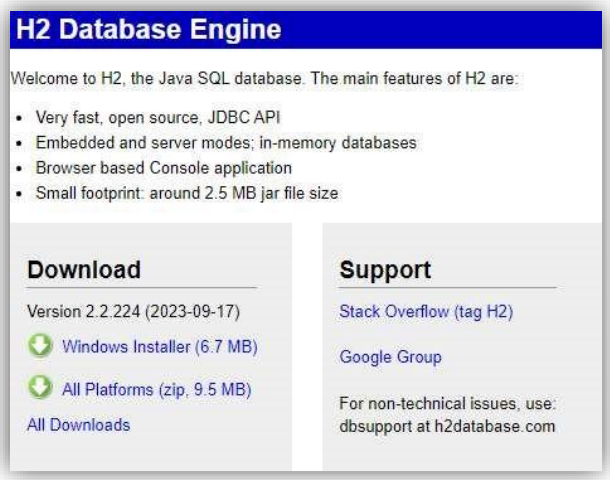
```java
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/cloudvendor")
public class CloudVendorController
{
    6 usages
    CloudVendorService cloudVendorService;
    public CloudVendorController(CloudVendorService cloudVendorService) {
        this.cloudVendorService = cloudVendorService;
    }

    @GetMapping("{vendorId}")
    public ResponseEntity<Object> getCloudVendorDetails(@PathVariable("vendorId") String vendorId)
    {
        return ResponseHandler.responseBuilder( message: "Requested vendor details are given here:", HttpStatus.
        cloudVendorService.getCloudVendor(vendorId));

    }
    @GetMapping()
    public List<CloudVendor> getAllCloudVendorDetails()
```

GetById method using Postman



The Cloud Vendor API was further enhanced by implementing JUnit test cases for all three layers: the controller layer, service layer, and repository layer, ensuring comprehensive unit testing of each layer.

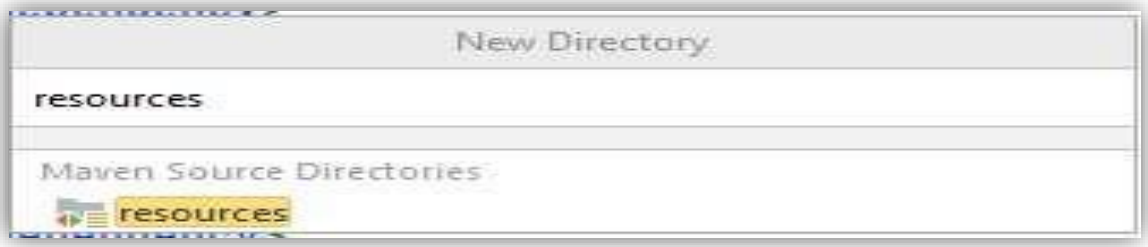
- **Downloading h2 database**



Configuring pom.xml

```xml
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>test</scope>
</dependency>
```

Making resources directory in test



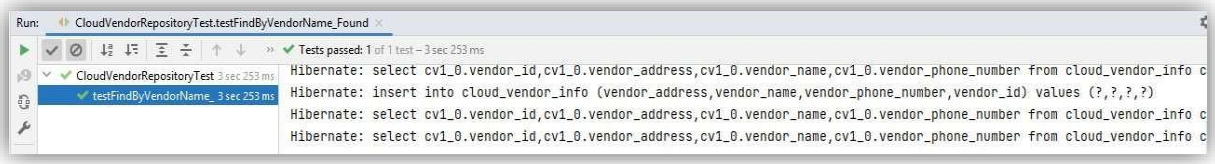
- **Creating Application.yaml file in resources**

Application.yaml

```yaml
spring:
  datasource:
    url: jdbc:h2://mem:db;DB_CLOSE_DELAY=-1
    username: sa
    password: sa
    driver-class-name: org.h2.Driver
    jpa:
      properties:
        hibernate:
          dialect: org.hibernate.dialect.H2Dialect
      show-sql: true

      jpa.hibernate.ddl-auto: create-drop
```

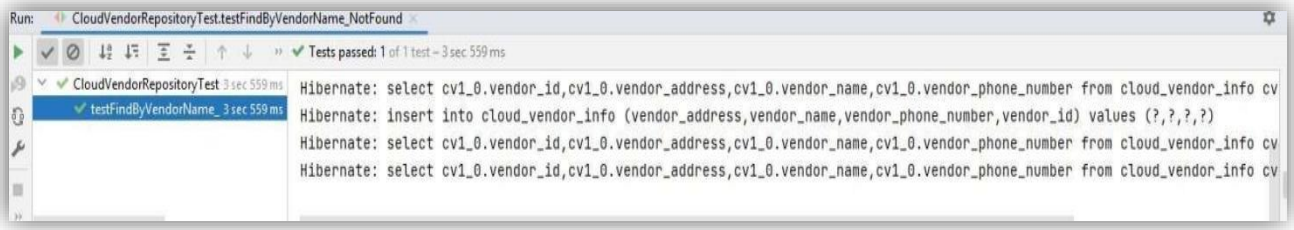
Testing Success case

```
//Success
@Test
void testFindByVendorName_Found()
{
    List<CloudVendor> cloudVendorList=
            cloudVendorRepository.findByVendorName("Amazon");
    assertThat(cloudVendorList.get(0).getVendorId()).isEqualTo(cloudVendor.getVendorId());
    assertThat(cloudVendorList.get(0).getVendorAddress())
            .isEqualTo(cloudVendor.getVendorAddress());
}
```



Testing Fail Test Case

```
    //Fail
    @Test
    void testFindByVendorName_NotFound()
    {
        List<CloudVendor> cloudVendorList=
                cloudVendorRepository.findByVendorName("GCP");
        assertThat(cloudVendorList.isEmpty()).isTrue();
    }
}
```



Run CloudVendorRepositoryTest

CloudVendorServiceImplTest

```java
package com.thinkconstructive.restdemo.service.impl;

import com.thinkconstructive.restdemo.model.CloudVendor;
import com.thinkconstructive.restdemo.repository.CloudVendorRepository;
import com.thinkconstructive.restdemo.service.CloudVendorService;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.mockito.Answers;
import org.mockito.Mock;
import org.mockito.Mockito;
import org.mockito.MockitoAnnotations;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Optional;

import static org.assertj.core.api.AssertionsForClassTypes.assertThat;
import static org.mockito.Mockito.*;

class CloudVendorServiceImplTest {
    7 usages
    @Mock
    private CloudVendorRepository cloudVendorRepository;
    7 usages
    private CloudVendorService cloudVendorService;
```

```java
    AutoCloseable autoCloseable;
    13 usages
    CloudVendor cloudVendor;
    @BeforeEach
    void setUp() {
        autoCloseable= MockitoAnnotations.openMocks( testClass: this);
        cloudVendorService=new CloudVendorServiceImpl(cloudVendorRepository);
        cloudVendor=new CloudVendor( vendorId: "1", vendorName: "Amazon"
                , vendorAddress: "USA", vendorPhoneNumber: "xxxx");
    }

    @AfterEach
    void tearDown() throws Exception{
        autoCloseable.close();
    }

    @Test
    void testCreateCloudVendor() {
        mock(CloudVendor.class);
        mock(CloudVendorRepository.class);
        when(cloudVendorRepository.save(cloudVendor)).thenReturn(cloudVendor);
        assertThat(cloudVendorService.createCloudVendor(cloudVendor)).isEqualTo( expected: "Success");
    }
```

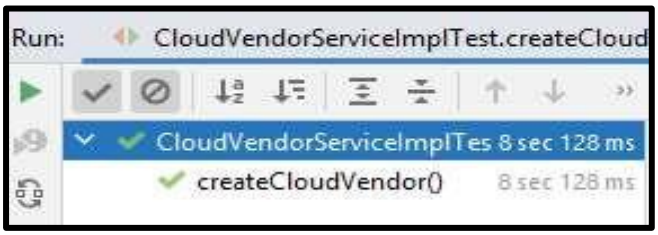
```
@Test
void testUpdateCloudVendor() {
    mock(CloudVendor.class);
    mock(CloudVendorRepository.class);
    when(cloudVendorRepository.save(cloudVendor)).thenReturn(cloudVendor);
    assertThat(cloudVendorService.updateCloudVendor(cloudVendor)).isEqualTo( expected: "Success");
}

@Test
void testDeleteCloudVendor() {
    mock(CloudVendor.class);
    mock(CloudVendorRepository.class, Mockito.CALLS_REAL_METHODS);

    doAnswer(Answers.CALLS_REAL_METHODS).when(cloudVendorRepository)
            .deleteById(any());
    assertThat(cloudVendorService.deleteCloudVendor( cloudVendorId: "1")).isEqualTo( expected: "Success"
}

@Test
void testGetCloudVendor() {
    mock(CloudVendor.class);
    mock(CloudVendorRepository.class);
    when(cloudVendorRepository.findById("1")).thenReturn(Optional.ofNullable(cloudVendor));
    assertThat(cloudVendorService.getCloudVendor( cloudVendorId: "1").getVendorName())
            .isEqualTo(cloudVendor.getVendorName());
}
```

```
@Test
void testGetByVendorName() {
    mock(CloudVendor.class);
    mock(CloudVendorRepository.class);
    when(cloudVendorRepository.findByVendorName("Amazon")).thenReturn(
            new ArrayList<CloudVendor>(Collections.singleton(cloudVendor))
    );
    assertThat(cloudVendorService.getByVendorName("Amazon").get(0).getVendorId())
            .isEqualTo(cloudVendor.getVendorId());
}
@Test
void testGetAllCloudVendors() {
    mock(CloudVendor.class);
    mock(CloudVendorRepository.class);
    when(cloudVendorRepository.findAll()).thenReturn(
            new ArrayList<CloudVendor>(Collections.singleton(cloudVendor))
    );
    assertThat(cloudVendorService.getAllCloudVendors().get(0).getVendorPhoneNumber())
            .isEqualTo(cloudVendor.getVendorPhoneNumber());
}
```

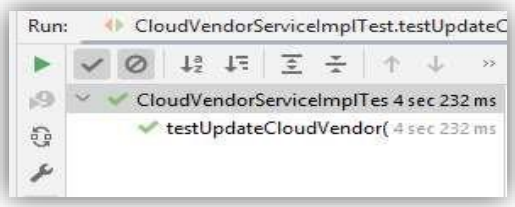
Running CreateCloudVendor Test

```
@Test
void createCloudVendor() {
    mock(CloudVendor.class);
    mock(CloudVendorRepository.class);
    when(cloudVendorRepository.save(cloudVendor)).thenReturn(cloudVendor);
    assertThat(cloudVendorService.createCloudVendor(cloudVendor)).isEqualTo( expected: "Success");
}
```

TestUpdateCloudVendor

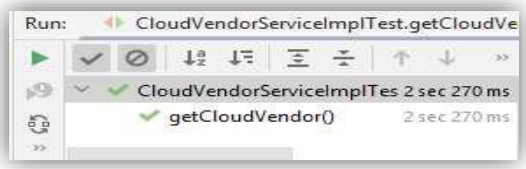



```
@Test
void testUpdateCloudVendor() {
    mock(CloudVendor.class);
    mock(CloudVendorRepository.class);
    when(cloudVendorRepository.save(cloudVendor)).thenReturn(cloudVendor);
    assertThat(cloudVendorService.updateCloudVendor(cloudVendor)).isEqualTo( expected: "Success");
}
```



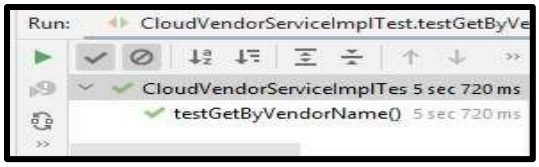
getCloudVendor

```
@Test
void getCloudVendor() {
    mock(CloudVendor.class);
    mock(CloudVendorRepository.class);
    when(cloudVendorRepository.findById("1")).thenReturn(Optional.ofNullable(cloudVendor));
    assertThat(cloudVendorService.getCloudVendor( cloudVendorId: "1").getVendorName())
            .isEqualTo(cloudVendor.getVendorName());
}
```



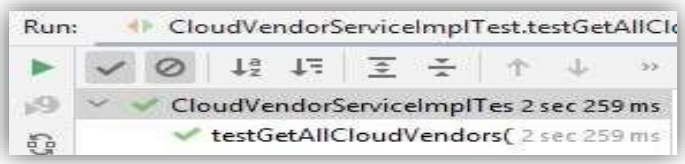
- **testGetByVendorName**

```
void testGetByVendorName() {
    mock(CloudVendor.class);
    mock(CloudVendorRepository.class);
    when(cloudVendorRepository.findByVendorName("Amazon")).thenReturn(
            new ArrayList<CloudVendor>(Collections.singleton(cloudVendor))
    );
    assertThat(cloudVendorService.getByVendorName("Amazon").get(0).getVendorId())
            .isEqualTo(cloudVendor.getVendorId());
}
```



testGetAllCloudVendors

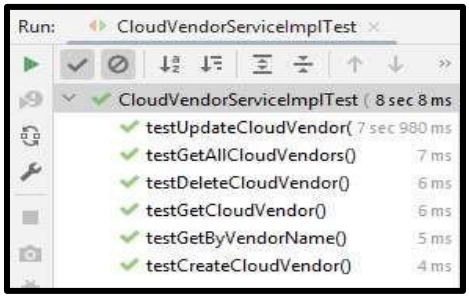
```
@Test
void testGetAllCloudVendors() {
    mock(CloudVendor.class);
    mock(CloudVendorRepository.class);
    when(cloudVendorRepository.findAll()).thenReturn(
            new ArrayList<CloudVendor>(Collections.singleton(cloudVendor))
    );
    assertThat(cloudVendorService.getAllCloudVendors().get(0).getVendorPhoneNumber())
            .isEqualTo(cloudVendor.getVendorPhoneNumber());
}
```



testDeleteCloudVendor

```
@Test
void deleteCloudVendor() {
    mock(CloudVendor.class);
    mock(CloudVendorRepository.class, Mockito.CALLS_REAL_METHODS);

    doAnswer(Answers.CALLS_REAL_METHODS).when(cloudVendorRepository)
            .deleteById(any());
    assertThat(cloudVendorService.deleteCloudVendor( cloudVendorId: "1")).isEqualTo( expected: "Success");
}
```

- **Running CloudVendorServiceImplTest**



⇒ SESSION # 05

CloudVendorControllerTest

```
package com.thinkconstructive.restdemo.model.controller;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.ObjectWriter;
import com.fasterxml.jackson.databind.SerializationFeature;
import com.thinkconstructive.restdemo.model.CloudVendor;
import com.thinkconstructive.restdemo.model.controller.CloudVendorController;
import com.thinkconstructive.restdemo.service.CloudVendorService;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.MockMvcAutoConfiguration;
import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;
import org.springframework.boot.test.mock.mockito.MockBean;
import org.springframework.http.MediaType;
import org.springframework.test.web.servlet.MockMvc;

import java.util.ArrayList;
import java.util.List;

import static org.mockito.Mockito.when;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.*;
import static org.springframework.test.web.servlet.result.MockMvcResultHandlers.print;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;
```

```
@WebMvcTest(CloudVendorController.class)
class CloudVendorControllerTest {

    5 usages
    @Autowired
    private MockMvc mockMvc;
    5 usages
    @MockBean
    private CloudVendorService cloudVendorService;
    7 usages
    CloudVendor cloudVendorOne;
    2 usages
    CloudVendor cloudVendorTwo;
    3 usages
    List<CloudVendor> cloudVendorList= new ArrayList<>();

    @BeforeEach
    void setUp() {
        cloudVendorOne = new CloudVendor( vendorId: "1", vendorName: "Amazon",
                vendorAddress: "USA", vendorPhoneNumber: "xxxxx");
        cloudVendorTwo = new CloudVendor( vendorId: "2", vendorName: "GCP",
                vendorAddress: "UK", vendorPhoneNumber: "yyyyy");
        cloudVendorList.add(cloudVendorOne);
        cloudVendorList.add(cloudVendorTwo);
    }
```

```
@AfterEach
void tearDown() {
}

@Test
void getCloudVendorDetails() throws Exception {
    when(cloudVendorService.getCloudVendor( cloudVendorId: "1")).thenReturn(cloudVendorOne);
    this.mockMvc.perform(get( urlTemplate: "/cloudvendor/" + "1")).andDo(print()).andExpect(sta
}

@Test
void getAllCloudVendorDetails() throws  Exception {
    when(cloudVendorService.getAllCloudVendors()).thenReturn(cloudVendorList);
    this.mockMvc.perform(get( urlTemplate: "/cloudvendor"))
            .andDo(print()).andExpect(status().isOk());
}

@Test
void testCreateCloudVendorDetails() throws Exception {
    ObjectMapper mapper = new ObjectMapper();
    mapper.configure(SerializationFeature.WRAP_ROOT_VALUE,  state: false);
    ObjectWriter ow = mapper.writer().withDefaultPrettyPrinter();
    String requestJson=ow.writeValueAsString(cloudVendorOne);

    when(cloudVendorService.createCloudVendor(cloudVendorOne)).thenReturn( value: "Success");
    this.mockMvc.perform(post( urlTemplate: "/cloudvendor")
```

```
                    .contentType(MediaType.APPLICATION_JSON)
                    .content(requestJson))
            .andDo(print()).andExpect(status().isOk());
}

@Test
void testUpdateCloudVendorDetails() throws Exception {
    ObjectMapper mapper = new ObjectMapper();
    mapper.configure(SerializationFeature.WRAP_ROOT_VALUE,  state: false);
    ObjectWriter ow = mapper.writer().withDefaultPrettyPrinter();
    String requestJson=ow.writeValueAsString(cloudVendorOne);

    when(cloudVendorService.updateCloudVendor(cloudVendorOne))
            .thenReturn( value: "Cloud Vendor Updated Successfully");
    this.mockMvc.perform(put( urlTemplate: "/cloudvendor")
                    .contentType(MediaType.APPLICATION_JSON)
                    .content(requestJson))
            .andDo(print()).andExpect(status().isOk());
}

@Test
void testDeleteCloudVendorDetails() throws Exception {
    when(cloudVendorService.deleteCloudVendor( cloudVendorId: "1"))
            .thenReturn( value: "Cloud Vendor Deleted Successfully");
    this.mockMvc.perform(delete( urlTemplate: "/cloudvendor/" + "1"))
            .andDo(print()).andExpect(status().isOk());
```

```
@Test
void testGetCloudVendorDetails() throws Exception{
    when(cloudVendorService.getCloudVendor( cloudVendorId: "1"))
            .thenReturn(cloudVendorOne);
    this.mockMvc.perform(get( urlTemplate: "/cloudvendor/1"))
            .andDo(print()).andExpect(status().isOk());
}
```

```
Run:  CloudVendorControllerTest.testGetCloudVendorDetails
Tests passed: 1 of 1 test – 1 sec 597 ms
CloudVendorControllerTest 1 sec 597 ms
testGetCloudVendorDeta 1 sec 597 ms
MockHttpServletRequest:
      HTTP Method = GET
      Request URI = /cloudvendor/1
       Parameters = {}
          Headers = []
             Body = null
    Session Attrs = {}

Handler:
```

➢ testGetCloudVendorDetails

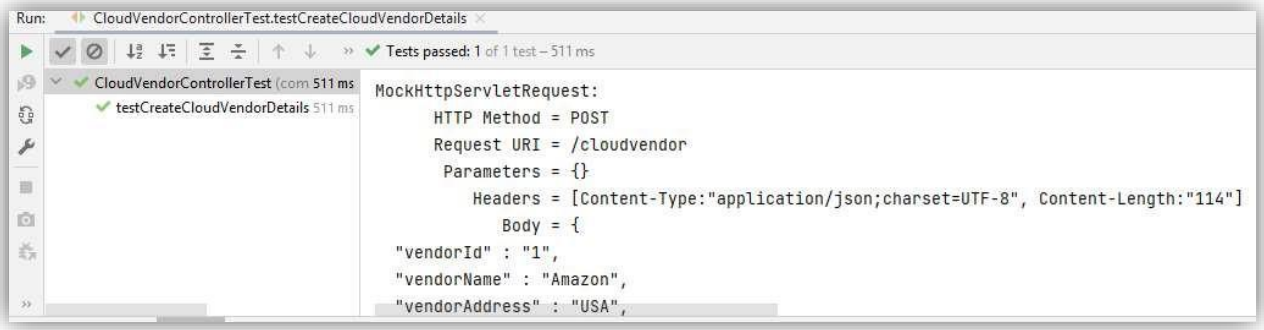
testGetAllCloudVendorDetails

```
@Test
void testGetAllCloudVendorDetails() throws Exception {
    when(cloudVendorService.getAllCloudVendors())
            .thenReturn(cloudVendorList);
    this.mockMvc.perform(get( urlTemplate: "/cloudvendor"))
            .andDo(print()).andExpect(status().isOk());
}
```

```
Run:  CloudVendorControllerTest.testGetAllCloudVendorDetails
Tests passed: 1 of 1 test – 494 ms
CloudVendorControllerTest (com 494 ms
testGetAllCloudVendorDetails 494 ms
MockHttpServletRequest:
      HTTP Method = GET
      Request URI = /cloudvendor
       Parameters = {}
          Headers = []
             Body = null
    Session Attrs = {}
```

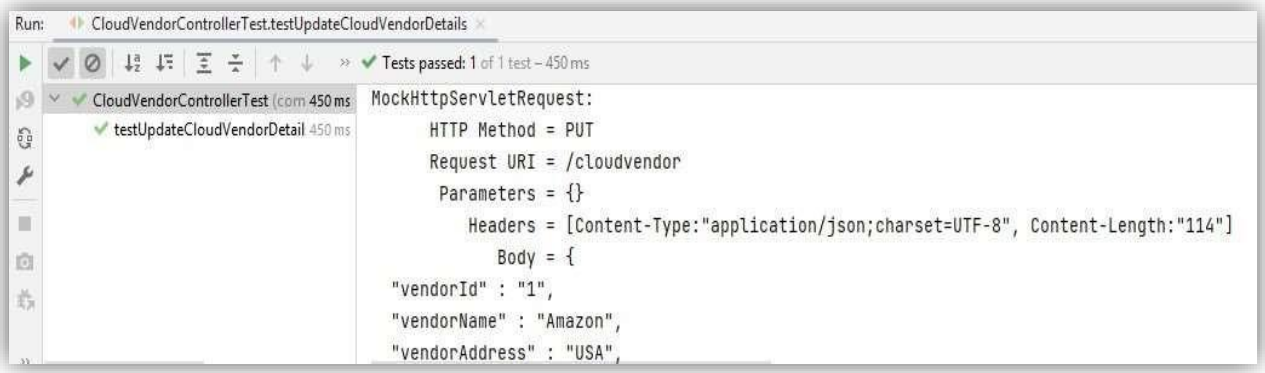
testCreateCloudVendorDetails

```
void testCreateCloudVendorDetails() throws Exception {
    ObjectMapper mapper = new ObjectMapper();
    mapper.configure(SerializationFeature.WRAP_ROOT_VALUE, state: false);
    ObjectWriter ow = mapper.writer().withDefaultPrettyPrinter();
    String requestJson=ow.writeValueAsString(cloudVendorOne);

    when(cloudVendorService.createCloudVendor(cloudVendorOne)).thenReturn( value: "Success");
    this.mockMvc.perform(post( urlTemplate: "/cloudvendor")
                    .contentType(MediaType.APPLICATION_JSON)
                    .content(requestJson))
            .andDo(print()).andExpect(status().isOk());
}
```



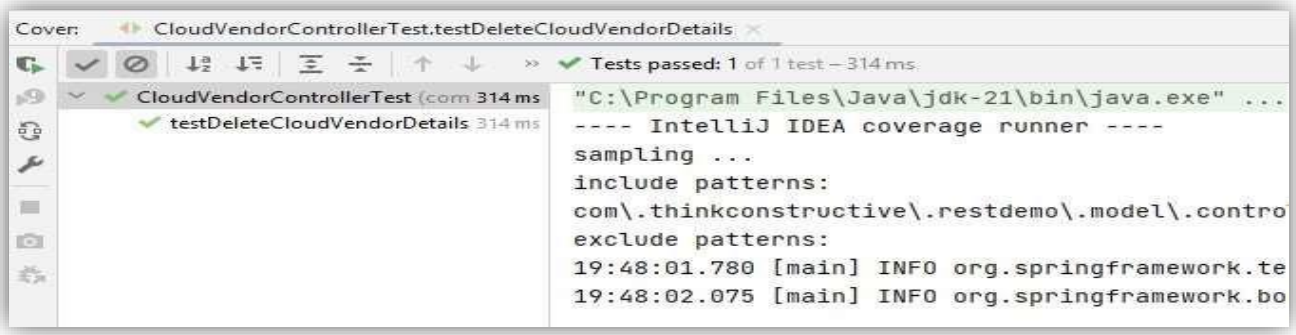
- **testUpdateCloudVendorDetail**

```
void testUpdateCloudVendorDetails() throws Exception {
    ObjectMapper mapper = new ObjectMapper();
    mapper.configure(SerializationFeature.WRAP_ROOT_VALUE, state: false);
    ObjectWriter ow = mapper.writer().withDefaultPrettyPrinter();
    String requestJson=ow.writeValueAsString(cloudVendorOne);

    when(cloudVendorService.updateCloudVendor(cloudVendorOne))
            .thenReturn( value: "Cloud Vendor Updated Successfully");
    this.mockMvc.perform(put( urlTemplate: "/cloudvendor")
                    .contentType(MediaType.APPLICATION_JSON)
                    .content(requestJson))
            .andDo(print()).andExpect(status().isOk());
}
```

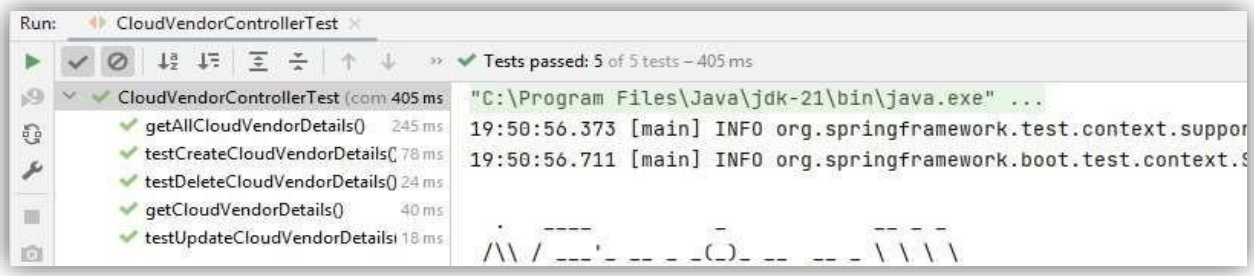


➢ testDeleteCloudVendorDetails

```
@Test
void testDeleteCloudVendorDetails() throws Exception {
    when(cloudVendorService.deleteCloudVendor( cloudVendorId: "1"))
            .thenReturn( value: "Cloud Vendor Deleted Successfully");
    this.mockMvc.perform(delete( urlTemplate: "/cloudvendor/" + "1"))
            .andDo(print()).andExpect(status().isOk());

}
```



➢ CloudVendorControllerTest



CONCLUSION

This project provided an in-depth understanding of Spring Boot REST API development, covering every aspect from architecture design to practical implementation. The flowchart offers a clear visualization of the project's three-layered architecture, demonstrating how the Controller, Service, and Repository layers interact with REST clients like Postman and databases such as MySQL Workbench. The screenshots and code implementation validated the functionality of CRUD operations, exception handling, and structured JSON responses. Robust testing practices, including unit tests, ensured application reliability. This hands-on experience solidified key concepts, making this project an excellent foundation for enterprise-level API development, ensuring scalability, maintainability, and readiness for real-world applications.

<><><><><><><><><><> *** (THE END) *** <><><><><><><><><><>