



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

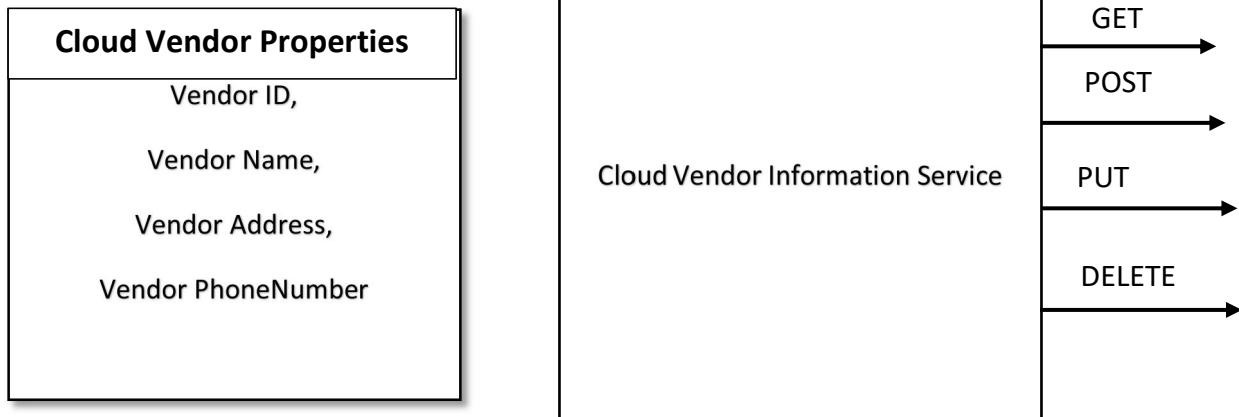
Session 5: Master Unit Testing Java Spring Boot REST API Application in One Shot

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.

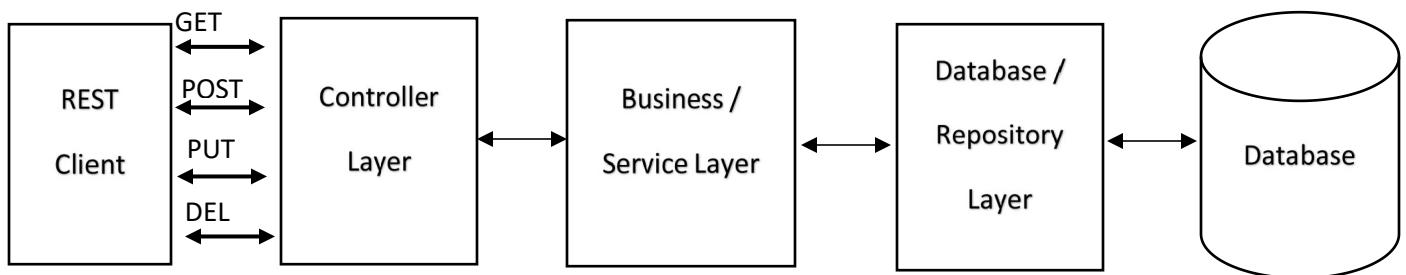
➤ **Spring Boot Project Architecture**

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

The screenshot shows the Spring Initializr web interface. In the 'Project' section, 'Maven' is selected. Under 'Language', 'Java' is chosen. In the 'Dependencies' section, 'Spring Web' is selected. Below it, 'Spring Boot' is set to version 3.2.1. The 'Project Metadata' section shows 'Group' as com.thinkconstructive and 'Artifact' as rest-demo. At the bottom, there are three buttons: 'GENERATE' (CTRL + D), 'EXPLORE' (CTRL + SPACE), and 'SHARE...'.

Project
○ Gradle - Groovy
○ Gradle - Kotlin Maven
○ Groovy
Language
 Java Kotlin
Spring Boot
○ 3.2.2 (SNAPSHOT) 3.2.1 3.1.8 (SNAPSHOT)
○ 3.1.7
Project Metadata
Group: com.thinkconstructive
Artifact: rest-demo
Dependencies
Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
Project Metadata
Group: com.thinkconstructive
Artifact: rest-demo
Name: rest-demo
Description: Demo project for Spring Boot
Package name: com.thinkconstructive.rest-demo
Packaging: Jar War
Java: ○ 21 17
GENERATE CTRL + D **EXPLORE** CTRL + SPACE **SHARE...**

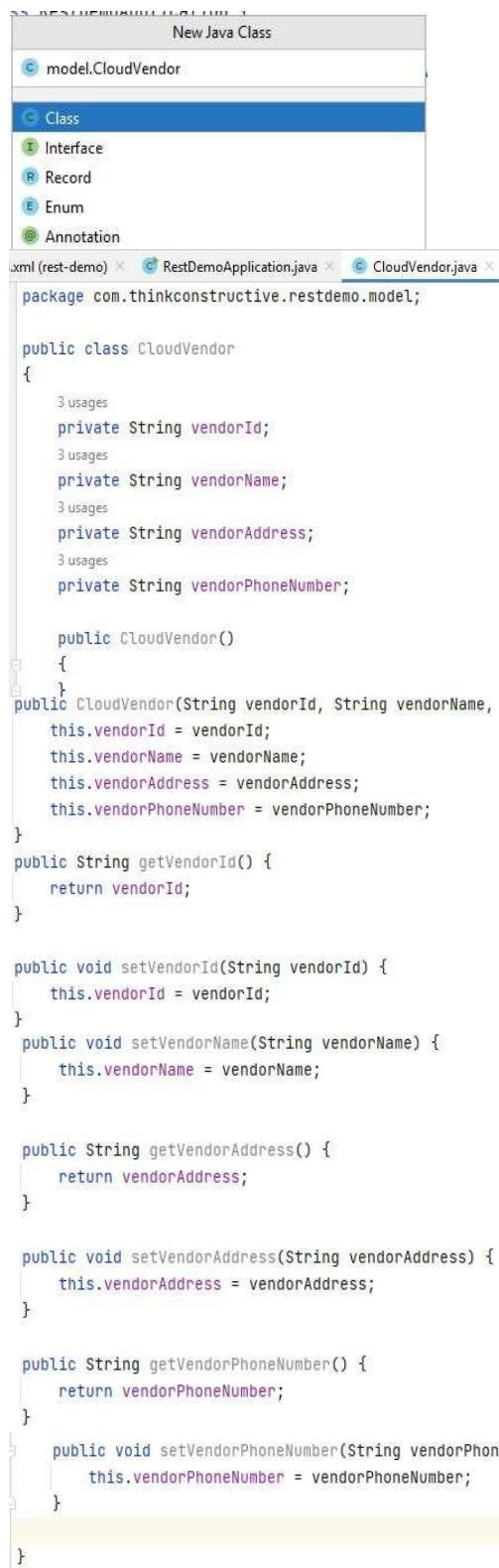
⇒ SESSION # 01

➤ RestDemoApplication.java

The screenshot shows a code editor with two tabs: 'pom.xml (rest-demo)' and 'RestDemoApplication.java'. The code in 'RestDemoApplication.java' is:

```
1 package com.thinkconstructive.restdemo;  
2  
3 import org.springframework.boot.SpringApplication;  
4 import org.springframework.boot.autoconfigure.SpringBootApplication;  
5  
6 @SpringBootApplication  
7 public class RestDemoApplication {  
8  
9     public static void main(String[] args) { SpringApplication.run(RestDemoApplication.class, args); }  
10 }  
11  
12 }
```

➤ CloudVendor.java



The screenshot shows a Java code editor interface. At the top, there is a dropdown menu labeled "New Java Class" with options: "model.CloudVendor", "Class" (which is selected), "Interface", "Record", "Enum", and "Annotation". Below the menu, the file tab bar shows "xml (rest-demo) x", "RestDemoApplication.java x", and "CloudVendor.java x". The main code area contains the following Java code:

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

public class CloudVendor {
    private String vendorId;
    private String vendorName;
    private String vendorAddress;
    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;
    }

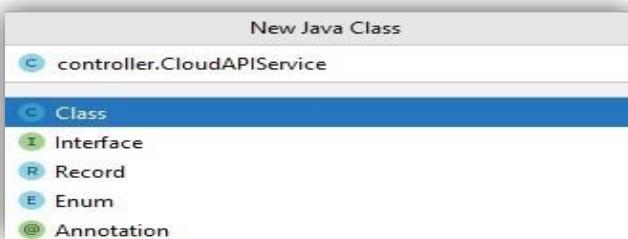
    public String getVendorAddress() {
        return vendorAddress;
    }

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

    public String getVendorPhoneNumber() {
        return vendorPhoneNumber;
    }

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

➤ Cloud ApiService.java

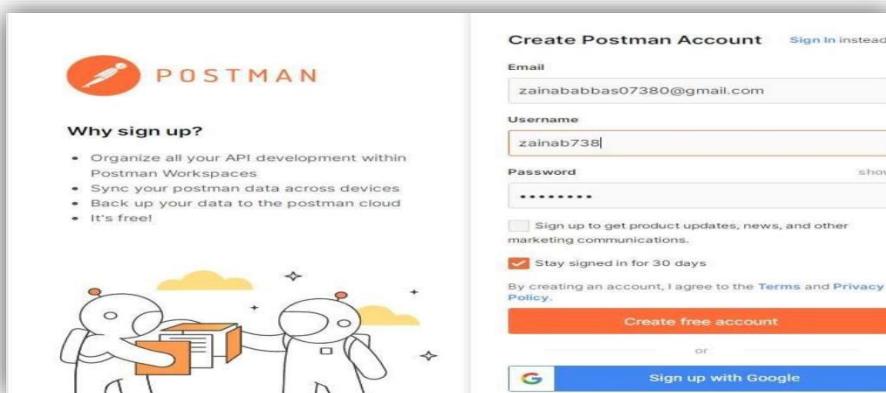


```
ml (rest-demo) × RestDemoApplication.java × CloudVendor.java × CloudVendorAPIService.java ×
package com.thinkconstructive.restdemo.controller;

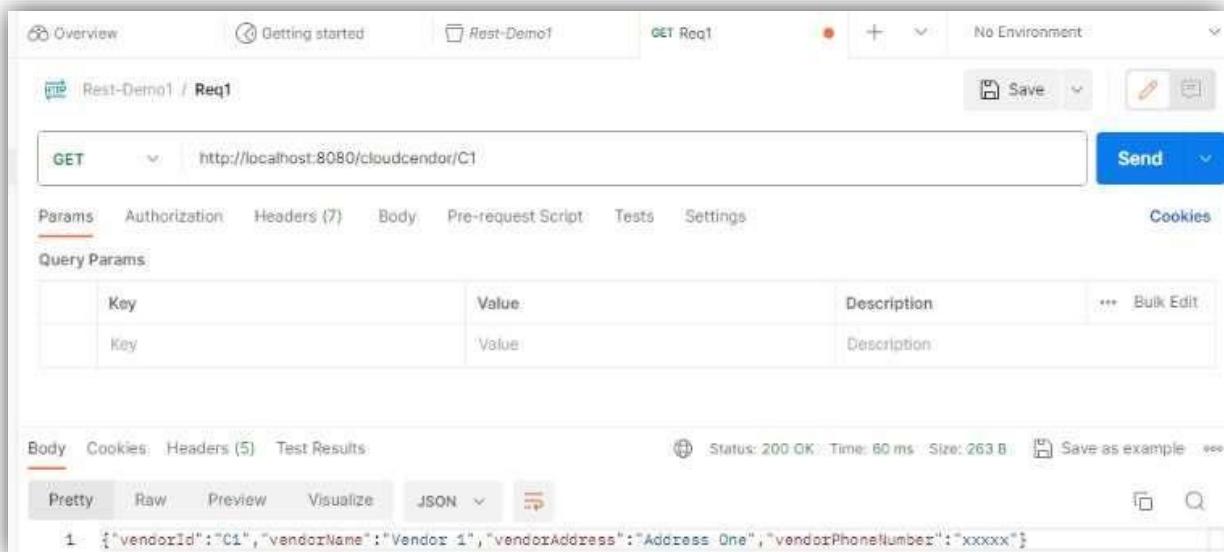
import com.thinkconstructive.restdemo.model.CloudVendor;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

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

➤ Postman



➤ Accessing C1 through postman



The screenshot shows the Postman application interface. At the top, there are tabs for Overview, Getting started, Rest-Demo1 (which is selected), and GET Req1. Below the tabs, the URL is set to `http://localhost:8080/cloudcendor/C1`. The method is set to GET. On the right side, there are buttons for Save, Edit, and Delete. Below the URL input, there are tabs for Params, Authorization, Headers (7), Body, Pre-request Script, Tests, Settings, and Cookies. The Params tab is selected. Under Query Params, there is a table with columns Key, Value, Description, and Bulk Edit. There are two rows: one with Key 'key' and Value 'Value', and another with Key 'Value' and Description 'Description'. At the bottom, there are tabs for Body, Cookies, Headers (5), Test Results, and a status bar showing Status: 200 OK, Time: 60 ms, Size: 263 B. Below the status bar, there are buttons for Pretty, Raw, Preview, Visualize, and JSON. The JSON preview shows the following response:

```
1. {"vendorId": "C1", "vendorName": "Vendor 1", "vendorAddress": "Address One", "vendorPhoneNumber": "xxxxx"}
```

➤ Adding create cloud vendor function in CloudVendorAPIService.java



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

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

import java.util.List;

@RestController
@RequestMapping("/cloudcendor")
public class CloudVendorAPIService {

    @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

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'Collections' (Rest-Demo1), 'Environments', and 'History'. The main area shows a 'POST' request titled 'Req1' with the URL 'http://localhost:8080/cloudcendor/'. The 'Body' tab is selected, showing a JSON payload:

```
1 {"vendorId": "C2", "vendorName": "Vendor 2", "vendorAddress": "Address two", "vendorPhoneNumber": "xxxxx"}
```

Below the body, the status bar indicates 'Status: 200 OK Time: 11.00 s Size: 197 B'. The 'Test Results' section shows the response: 'Cloud vendor created successfully'.

➤ Accessing C2 through postman

The screenshot shows the Postman application interface. The title bar says 'POST Req1' and 'GET New Request'. The main area shows a 'GET' request titled 'New Request' with the URL 'http://localhost:8080/cloudcendor/C2'. The 'Params' tab is selected, showing a table for 'Query Params' with two rows:

Key	Value
Key	Value

Below the params, the 'Body' tab is selected, showing the JSON response from the previous request:

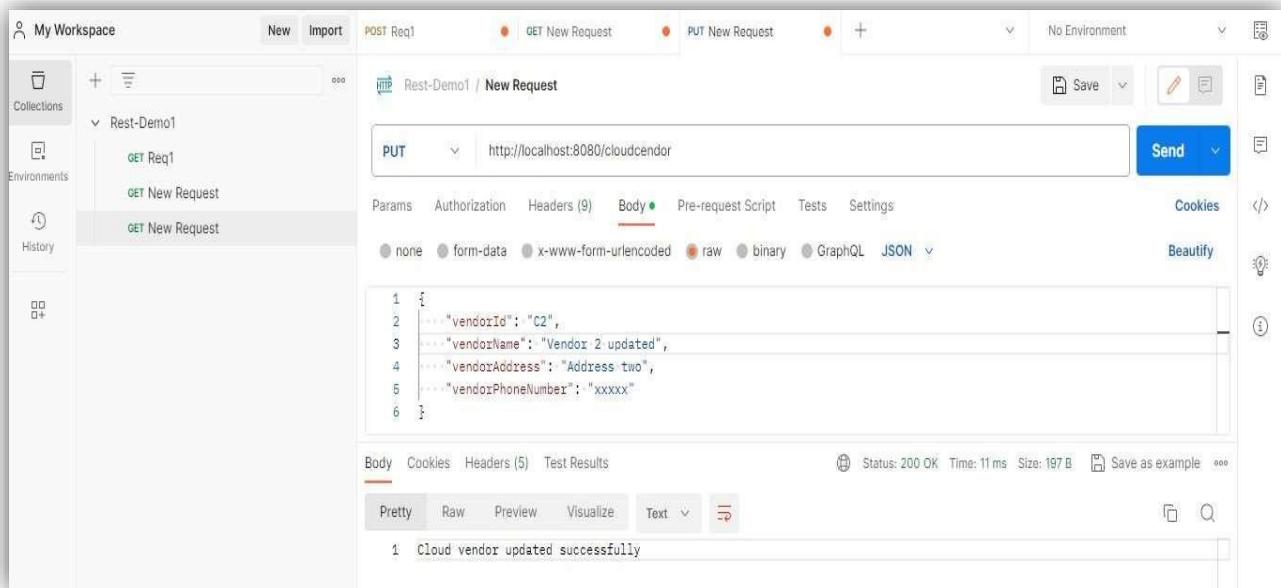
```
1 {"vendorId": "C2", "vendorName": "Vendor 2", "vendorAddress": "Address two", "vendorPhoneNumber": "xxxxx"}
```

➤ Add update cloud vendor function in CloudVendorAPIService.java



```
xml (rest-demo)  C RestDemoApplication.java  C CloudVendor.java  C 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



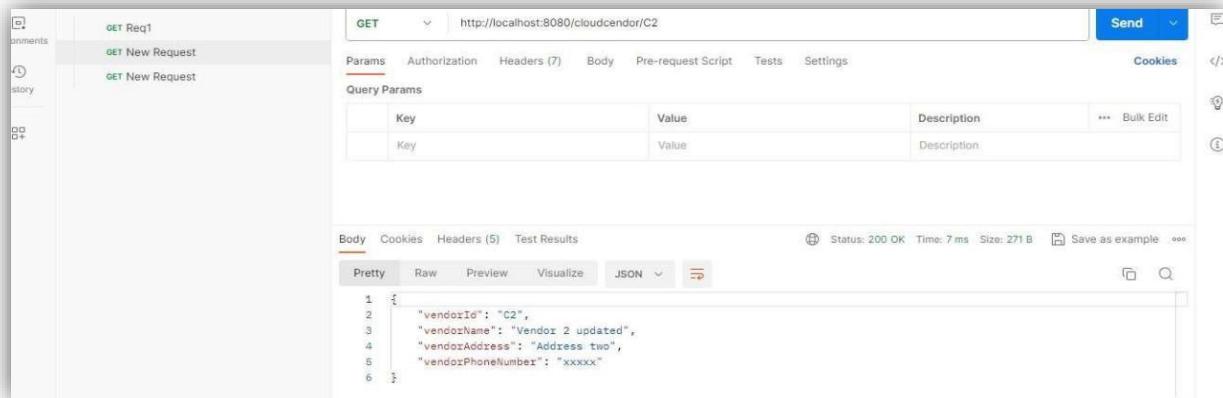
PUT http://localhost:8080/cloudendor

```
1 {
2   ...
3   "vendorId": "C2",
4   ...
5   "vendorName": "Vendor 2 updated",
6   ...
7   "vendorAddress": "Address two",
8   ...
9   "vendorPhoneNumber": "xxxxx"
10 }
```

Status: 200 OK Time: 11 ms Size: 197 B

1 Cloud vendor updated successfully

➤ Assessing updated C2 using Postman



GET http://localhost:8080/cloudendor/C2

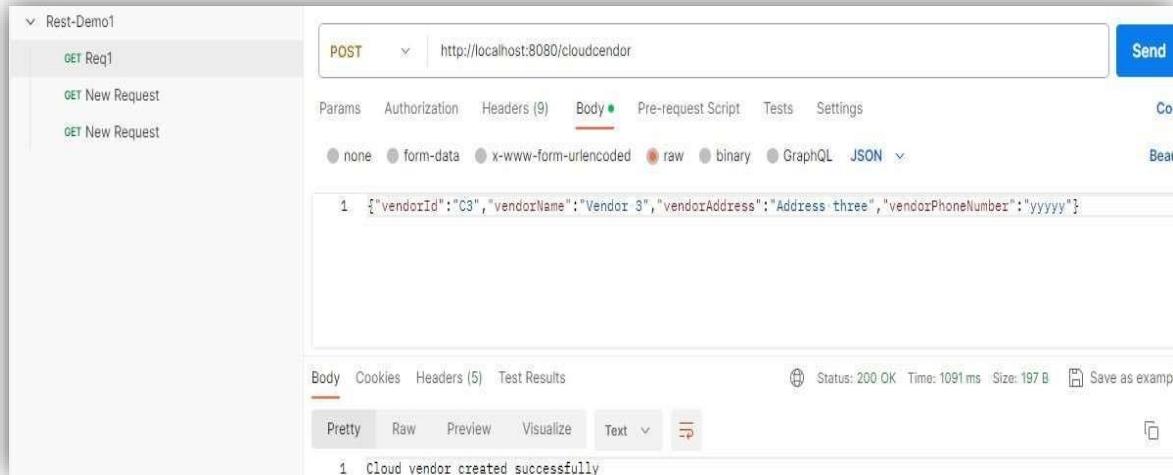
```
1 {
2   ...
3   "vendorId": "C2",
4   ...
5   "vendorName": "Vendor 2 updated",
6   ...
7   "vendorAddress": "Address two",
8   ...
9   "vendorPhoneNumber": "xxxxx"
10 }
```

➤ Add delete cloud vendor function in CloudService ApiService.java



```
xml (rest-demo) × RestDemoApplication.java × CloudVendor.java × CloudVendorAPIService.java ×
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";
}
@DeleteMapping("{vendorId}")
public String deleteCloudVendorDetails(String vendorId)
{
    this.cloudVendor=null;
    return "Cloud vendor deleted successfully";
}
```

➤ Making a new cloud vendor “C3” using Postman



Rest-Demo1

GET Req1

POST http://localhost:8080/cloudcendor

Params Authorization Headers (9) Body Pre-request Script Tests Settings

Body (radio) none (radio) form-data (radio) x-www-form-urlencoded (radio) raw (radio) binary (radio) GraphQL (radio) JSON

1 {"vendorId": "C3", "vendorName": "Vendor 3", "vendorAddress": "Address three", "vendorPhoneNumber": "yyyyy"}

Status: 200 OK Time: 1091 ms Size: 197 B Save as example

Pretty Raw Preview Visualize Text

1 Cloud vendor created successfully

➤ Accessing C3 using Postman

The screenshot shows the Postman interface with a collection named "Rest-Demo1". A "GET New Request" is selected. The request URL is set to `http://localhost:8080/cloudcendor/C3`. The response body is displayed in JSON format:

```

1 {
2   "vendorId": "C3",
3   "vendorName": "Vendor 3",
4   "vendorAddress": "Address three",
5   "vendorPhoneNumber": "yyyyy"
6 }

```

➤ Deleting C3 using Postman

The screenshot shows the Postman interface with a collection named "Rest-Demo1". A "DELETE New Request" is selected. The request URL is set to `http://localhost:8080/cloudcendor/C3`. The response body is displayed in Text format:

```

1 Cloud vendor deleted successfully

```

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

```
pom.xml (rest-demo) <dependency>
21   <groupId>org.springframework.boot</groupId>
22   <artifactId>spring-boot-starter-data-jpa</artifactId>
23 </dependency>
24 <dependency>
25   <groupId>org.springframework.boot</groupId>
26   <artifactId>spring-boot-starter-web</artifactId>
27 </dependency>
28 <dependency>
29   <groupId>org.springframework.boot</groupId>
30   <artifactId>spring-boot-starter-data-jpa</artifactId>
31 </dependency>
32 <dependency>
33   <groupId>com.mysql</groupId>
34   <artifactId>mysql-connector-j</artifactId>
35 </dependency>
```

⇒ SESSION # 02

➤ Creating a new file named “Application.yaml”



```
application.yaml <spring:
  datasource:
    url: jdbc:mysql://localhost:3306/cloud_vendor?useSSL=false
    username: root
    password: root
    #JPA Settings
    jpa.hibernate.ddl_auto: create>
```

➤ Creating cloud_vendor schema



➤ Modifying CloudVendor.java

```

import jakarta.persistence.Id;
import jakarta.persistence.Table;

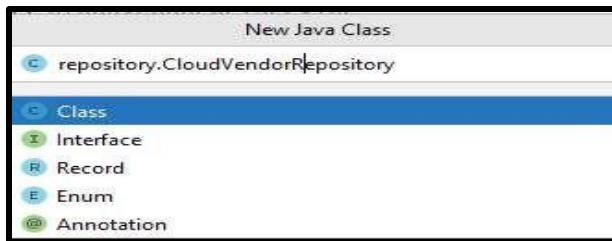
7 usages
@Entity
@Table(name = "cloud_vendor_info")
public class CloudVendor
{
    3 usages
    @Id
    private String vendorId;
    3 usages
    private String vendorName;
    3 usages
    private String vendorAddress;
    3 usages
    private String vendorPhoneNumber;

    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”



```

@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();
}

```

The screenshot shows a Java code editor with several tabs at the top: application.yaml, RestDemoApplication.java, CloudVendor.java, CloudVendorService.java, and CloudVendorServiceImpl.java. The CloudVendorServiceImpl.java tab is active, displaying the following code:

```
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.springframework.stereotype.Service;

import java.util.List;
@Service
public class CloudVendorServiceImpl implements CloudVendorService {
    6 usages
    CloudVendorRepository cloudVendorRepository;
    public CloudVendorServiceImpl(CloudVendorRepository cloudVendorRepository) {
        this.cloudVendorRepository = cloudVendorRepository;
    }
    @Override
    public String createCloudVendor(CloudVendor cloudVendor) {
        cloudVendorRepository.save((cloudVendor));
        return "Success";
    }
    @Override
    public String updateCloudVendor(CloudVendor cloudVendor) {
        cloudVendorRepository.save(cloudVendor);
        return "Success";
    }
}
```

CloudVendorServiceImpl

➤ CloudVendorController.java

The screenshot shows a Java code editor with several tabs at the top: application.yaml, RestDemoApplication.java, CloudVendor.java, CloudVendorService.java, and CloudVendorController.java. The CloudVendorController.java tab is active, displaying the following code:

```
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("/cloudendor")
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);
    }
}
```

```

    @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

The screenshot shows the MySQL Workbench interface. On the left, the 'SCHEMAS' tree view is open, showing the 'cloud_vendor' schema selected. Under 'Tables', there is a single table named 'cloud_vendor_info'. On the right, a 'Result Grid' window displays the results of the query 'SELECT * FROM cloud_vendor.cloud_vendor_info'. The grid has four columns: 'vendor_address', 'vendor_id', 'vendor_name', and 'vendor_phone_number'. All four rows in the grid show 'NULL' values.

The screenshot shows the 'Run' window of a Spring Boot application named 'RestDemoApplication'. The window displays the application's startup logs. Key log entries include:

- INFO 6536 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
- INFO 6536 --- [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.17]
- INFO 6536 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
- INFO 6536 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1 ms
- INFO 6536 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
- INFO 6536 --- [main] com.zaxxer.hikari.pool.HikariPool : HikariPool-1 - Added connection com.mysql.cj.jdbc
- INFO 6536 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
- INFO 6536 --- [main] o.hibernate.jpa.internal.util.LogHelper : HHH000204: Processing PersistenceUnitInfo [name: default]
- INFO 6536 --- [main] o.hibernate.core.Version : HHH000412: Hibernate ORM core version 6.4.1.Final
- INFO 6536 --- [main] o.h.c.internal.RegionFactoryInitiator : HHH000026: Second-level cache disabled
- INFO 6536 --- [main] o.s.o.j.p.SpringPersistenceUnitInfo : No LoadTimeWeaver setup: ignoring JPA class trans
- INFO 6536 --- [main] o.h.e.t.j.p.i.JtaPlatformInitiator : HHH000489: No JTA platform available (set 'hibernate.jta.platform')
- INFO 6536 --- [main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persisten
- INFO 6536 --- [main] JpaBaseConfiguration\$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. The
- INFO 6536 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context pa
- INFO 6536 --- [main] c.t.restdemo.RestDemoApplication : Started RestDemoApplication in 47.535 seconds (pr

Open MySql Workbench

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

Create a cloud vendor using Postman

The screenshot shows the Postman interface with a successful API call. The URL is `localhost:8080/cloudvendor`. The request method is `POST`. The body contains the following JSON:

```
1 {
2   "vendorId": "C1",
3   "vendorAddress": "One",
4   "vendorName": "Nameone",
5   "vendorPhoneNumber": "xxxxx"
6 }
```

The response status is `200 OK` with a message: `Cloud vendor created successfully`.

Select table row from mysql workbench

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

The screenshot shows the MySQL Workbench Query Editor with the following SQL query:

```
1 •  SELECT * FROM cloud_vendor.cloud_vendor_info;
```

The result grid displays one row of data:

	vendor_address	vendor_id	vendor_name	vendor_phone_number
▶	One	C1	Nameone	xxxxx
*	NULL	NULL	NULL	NULL

Use GetAll method in Postman

The screenshot shows the Postman interface with a collection named "Rest-Demo1". A request titled "GET Req1" is selected, which performs a GET operation on the URL "localhost:8080/cloudvendor". The "Body" tab is active, showing the response body as JSON:

```
1 [{"vendorId": "C1", "vendorName": "Nameone", "vendorAddress": "One", "vendorPhoneNumber": "xxxxxx"}]
```

Using GetById method in Postman

The screenshot shows the Postman interface with a collection named "Rest-Demo1". A request titled "GET Req1" is selected, which performs a GET operation on the URL "localhost:8080/cloudvendor/C1". The "Body" tab is active, showing the response body as JSON:

```
1 {"vendorId": "C1", "vendorName": "Nameone", "vendorAddress": "One", "vendorPhoneNumber": "xxxxxx"}
```

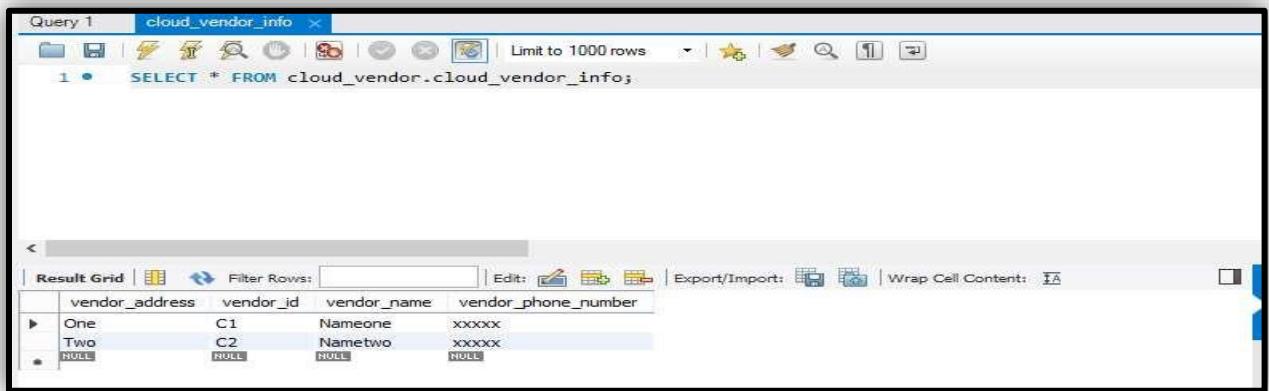
➤ Create more Vendors using Postman

The screenshot shows the Postman interface with a collection named "Rest-Demo1". A request is being prepared to perform a POST operation on the URL "localhost:8080/cloudvendor". The "Body" tab is active, showing the request body as JSON:

```
1 {"vendorId": "C2", "vendorAddress": "Two", "vendorName": "Nametwo", "vendorPhoneNumber": "xxxxxx"}
```

The response status is 200 OK, and the message "Cloud vendor created successfully" is displayed in the "Test Results" section.

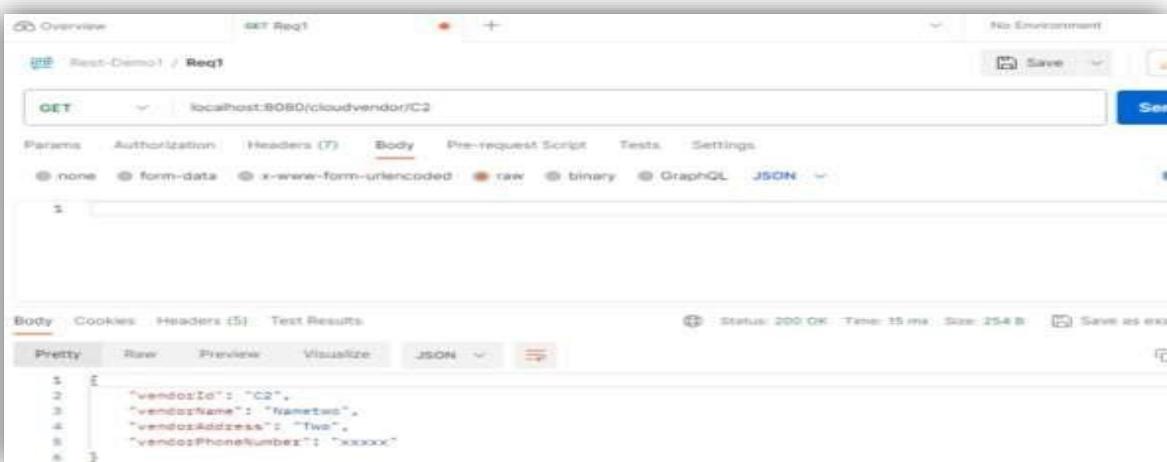
Selecting all rows using mysql workbench



A screenshot of the MySQL Workbench interface. The top bar shows "Query 1" and the database "cloud_vendor_info". The main area contains a SQL query: "SELECT * FROM cloud_vendor.cloud_vendor_info;". Below the query is a "Result Grid" showing two rows of data:

vendor_address	vendor_id	vendor_name	vendor_phone_number
One	C1	Nameone	xxxxx
Two	C2	Nametwo	xxxxx
NULL	NULL	NULL	NULL

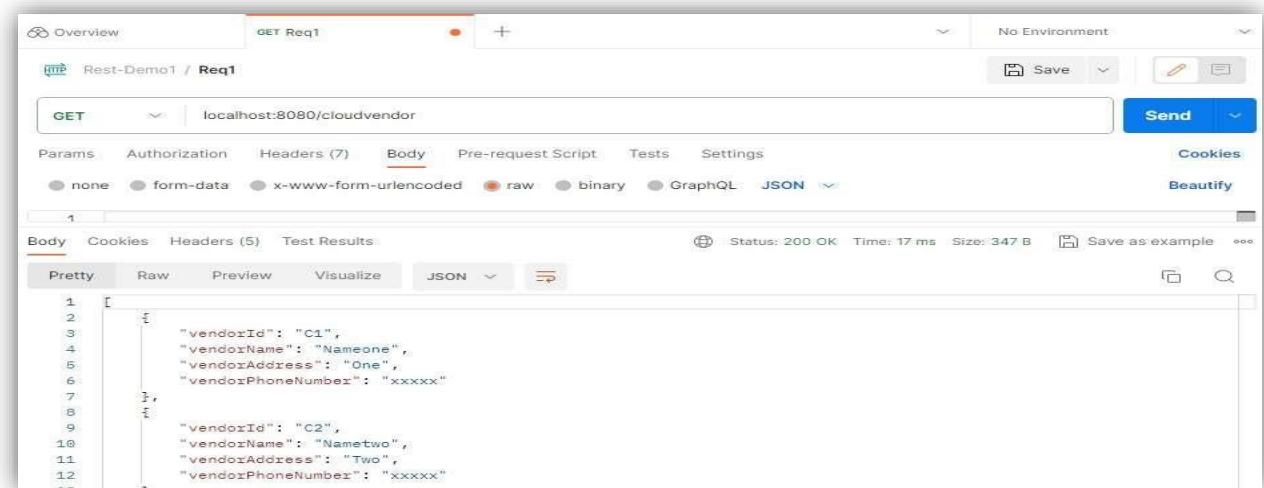
➤ Now accessing C2 using GetById using Postman



A screenshot of the Postman application. The request URL is "localhost:8080/cloudvendor/C2". The response body is displayed in JSON format:

```
1  {
2   "vendorId": "C2",
3   "vendorName": "Nametwo",
4   "vendorAddress": "Two",
5   "vendorPhoneNumber": "xxxxx"
6 }
```

➤ Again using GetAll method in Postman



A screenshot of the Postman application. The request URL is "localhost:8080/cloudvendor". The response body is displayed in JSON format, showing two rows of data:

```
1 [
2   {
3     "vendorId": "C1",
4     "vendorName": "Nameone",
5     "vendorAddress": "One",
6     "vendorPhoneNumber": "xxxxx"
7   },
8   {
9     "vendorId": "C2",
10    "vendorName": "Nametwo",
11    "vendorAddress": "Two",
12    "vendorPhoneNumber": "xxxxx"
13 }
14 ]
```

Updating Vendor Address of C2

The screenshot shows a POSTMAN interface with the following details:

- Method:** PUT
- URL:** localhost:8080/cloudvendor
- Body (JSON):**

```
1  {
2   "vendorId": "C2",
3   "vendorName": "Nametwo",
4   "vendorAddress": "UpdatedTwo",
5   "vendorPhoneNumber": "xxxxxx"
6 }
```

- Test Result:** Cloud vendor updated successfully

Mysql workbench

The screenshot shows the MySQL Workbench interface with the following details:

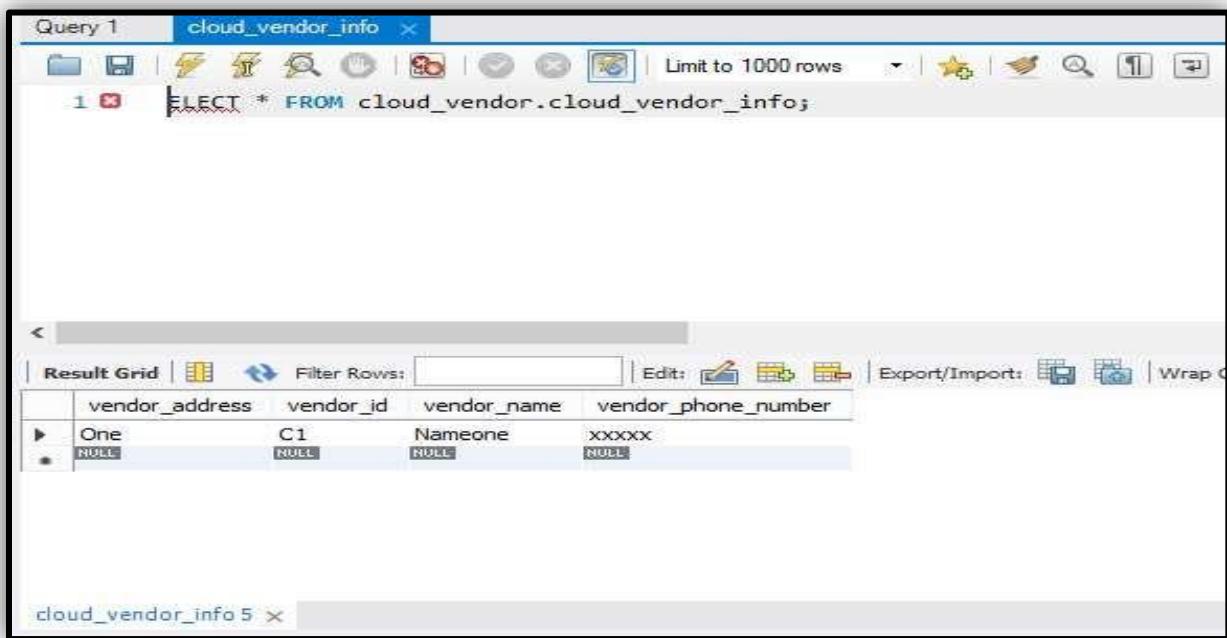
- Query:** SELECT * FROM cloud_vendor.cloud_vendor_info;
- Result Grid:**

vendor_address	vendor_id	vendor_name	vendor_phone_number
One	C1	Nameone	xxxxxx
UpdatedTwo	C2	Nametwo	xxxxxx
NULL	NULL	NULL	NULL

The screenshot shows a POSTMAN interface with the following details:

- Method:** DELETE
- URL:** localhost:8080/cloudvendor/C2
- Body (Text):** 1
- Test Result:** Cloud vendor deleted successfully
- Status:** Status: 200 OK Time: 227 ms Size: 197 B

Mysql Workbench



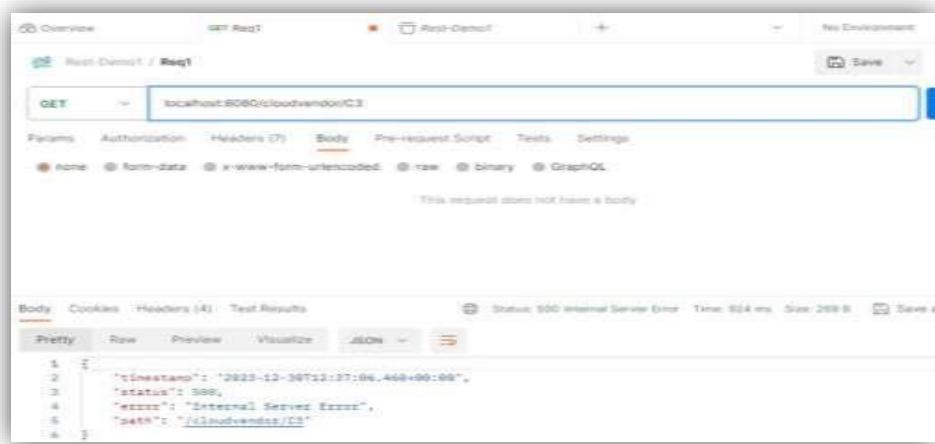
A screenshot of the MySQL Workbench interface. The main window title is "Query 1" and the tab name is "cloud_vendor_info". The query entered is "SELECT * FROM cloud_vendor.cloud_vendor_info;". Below the query, the results are displayed in a "Result Grid" table:

	vendor_address	vendor_id	vendor_name	vendor_phone_number
▶	One	C1	Nameone	XXXXXX
●	NULL	NULL	NULL	NULL

In order to learn how to properly handle exceptions in a Spring Boot REST API project, we will focus 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



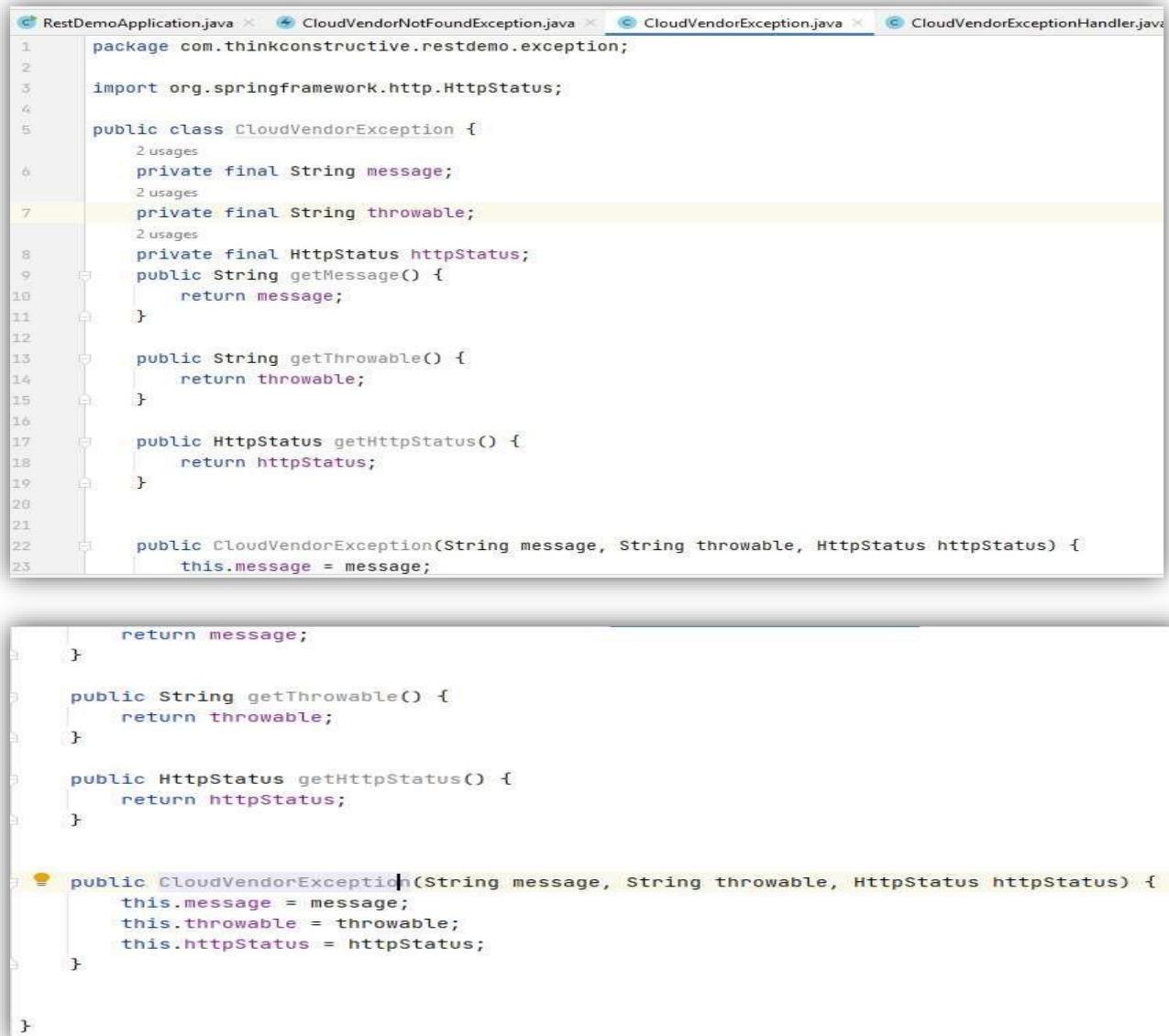
```
application.properties ×  RestDemoApplication.java ×  CloudVendorNotFoundException.java ×  CloudVendorException.java ×  CloudVendorExceptionHandler.java
package com.thinkconstructive.restdemo.exception;

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 {
6     2 usages
7     private final String message;
8     2 usages
9     private final String throwable;
10    2 usages
11    private final HttpStatus httpStatus;
12    public String getMessage() {
13        return message;
14    }
15
16    public String getThrowable() {
17        return throwable;
18    }
19
20    public HttpStatus getHttpStatus() {
21        return httpStatus;
22    }
23
24    public CloudVendorException(String message, String throwable, HttpStatus httpStatus) {
25        this.message = message;
26
27        return message;
28    }
29
30    public String getThrowable() {
31        return throwable;
32    }
33
34    public HttpStatus getHttpStatus() {
35        return httpStatus;
36    }
37
38    public CloudVendorException(String message, String throwable, HttpStatus httpStatus) {
39        this.message = message;
40        this.throwable = throwable;
41        this.httpStatus = httpStatus;
42    }
43
44 }
```

CloudVendorExceptionHandler.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

```
CloudVendor.java × CloudVendorService.java × CloudVendorServiceImpl.java × CloudVendorController.java × CloudVendor
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

The screenshot shows a POSTMAN request to `localhost:8080/cloudvendor/C3`. The response status is `404 Not Found`, and the response body is a JSON object: `{"message": "Requested Cloud Vendor doesnot exists", "throwable": "null", "httpStatus": "NOT_FOUND"}`.

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 **ResponseType** class.

Populating table

The image displays five separate Postman request windows, each showing a successful POST request to the endpoint `localhost:8080/cloudvendor`. Each request includes a JSON body with four fields: `vendorId`, `vendorAddress`, `vendorName`, and `vendorPhoneNumber`. The vendor IDs are sequentially labeled C1 through C5. The vendor names are Nameone, Namenone, Namethree, Namefour, and Namefive respectively. The vendor addresses are One, Two, Three, Four, and Five. The vendor phone numbers are all set to 'xxxxx'. Each response body contains the message "Cloud vendor created successfully".

```
POST /cloudvendor
{
  "vendorId": "C1",
  "vendorAddress": "One",
  "vendorName": "Nameone",
  "vendorPhoneNumber": "xxxxx"
}

1. Cloud vendor created successfully
```

```
POST /cloudvendor
{
  "vendorId": "C2",
  "vendorAddress": "Two",
  "vendorName": "Namenone",
  "vendorPhoneNumber": "xxxxx"
}

1. Cloud vendor created successfully
```

```
POST /cloudvendor
{
  "vendorId": "C3",
  "vendorAddress": "Three",
  "vendorName": "Namethree",
  "vendorPhoneNumber": "xxxxx"
}

1. Cloud vendor created successfully
```

```
POST /cloudvendor
{
  "vendorId": "C4",
  "vendorAddress": "Four",
  "vendorName": "Namefour",
  "vendorPhoneNumber": "xxxxx"
}

1. Cloud vendor created successfully
```

```
POST /cloudvendor
{
  "vendorId": "C5",
  "vendorAddress": "Five",
  "vendorName": "Namefive",
  "vendorPhoneNumber": "xxxxx"
}

1. Cloud vendor created successfully
```

Mysql workbench

The screenshot shows the MySQL Workbench interface with a query editor window. The query is:

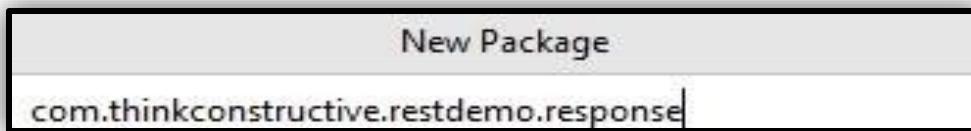
```
SELECT * FROM cloud_vendor.cloud_vendor_info;
```

The result grid displays the following data:

	vendor_address	vendor_id	vendor_name	vendor_phone_number
▶	One	C1	Nameone	xxxxxx
	Two	C2	Nametwo	xxxxxx
	Three	C3	Namethree	xxxxxx
	Four	C4	Namefour	xxxxxx
	Five	C5	Namefive	xxxxxx
	NULL	NULL	NULL	NULL

⇒ 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

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

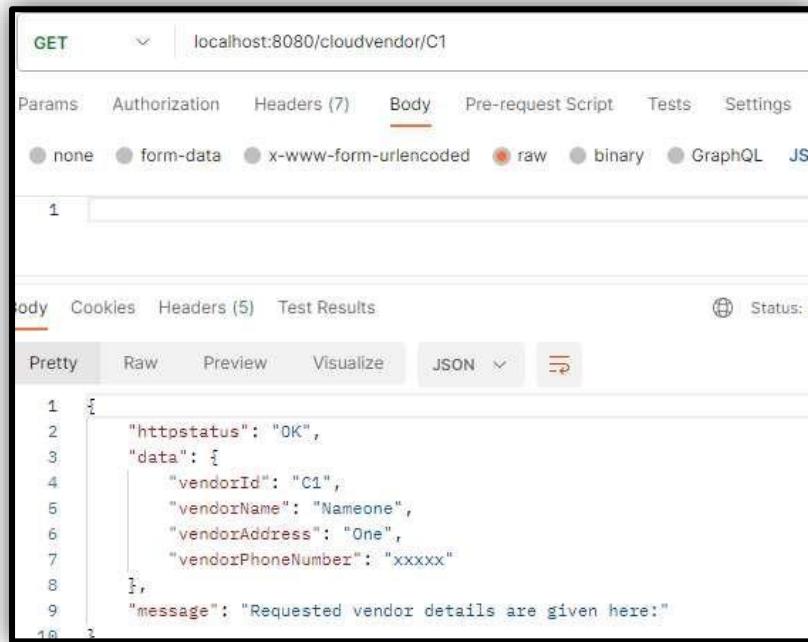
import java.util.List;

@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.OK,
        cloudVendorService.getCloudVendor(vendorId));
    }

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

GetById method using Postman



The screenshot shows a Postman request for a GET operation on the URL `localhost:8080/cloudvendor/C1`. The request parameters are set to `none`. The response body is displayed in JSON format:

```
1 {
  "httpstatus": "OK",
  "data": {
    "vendorId": "C1",
    "vendorName": "Nameone",
    "vendorAddress": "One",
    "vendorPhoneNumber": "xxxxx"
  },
  "message": "Requested vendor details are given here:"
}
```

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**

H2 Database Engine

Welcome to H2, the Java SQL database. The main features of H2 are:

- Very fast, open source, JDBC API
- Embedded and server modes; in-memory databases
- Browser based Console application
- Small footprint: around 2.5 MB jar file size

Download

Version 2.2.224 (2023-09-17)

- [!\[\]\(9b909335e35b297149cec10d9306de3f_img.jpg\) Windows Installer \(6.7 MB\)](#)
- [!\[\]\(08f44474418bff2496bb7d2327d97557_img.jpg\) All Platforms \(zip, 9.5 MB\)](#)
- [All Downloads](#)

Support

[Stack Overflow \(tag H2\)](#)

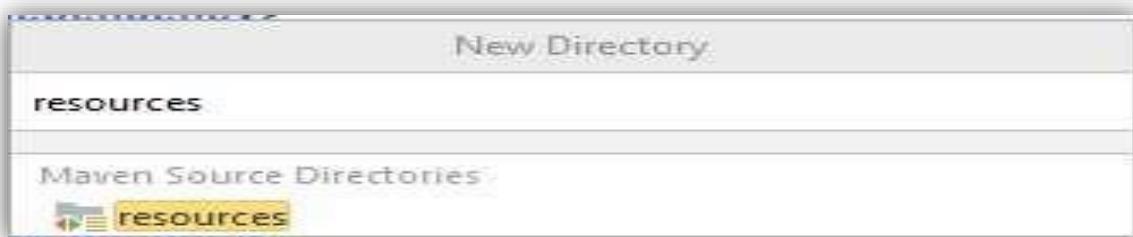
[Google Group](#)

For non-technical issues, use:
dbsupport at h2database.com

Configuring pom.xml

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

Making resources directory in test



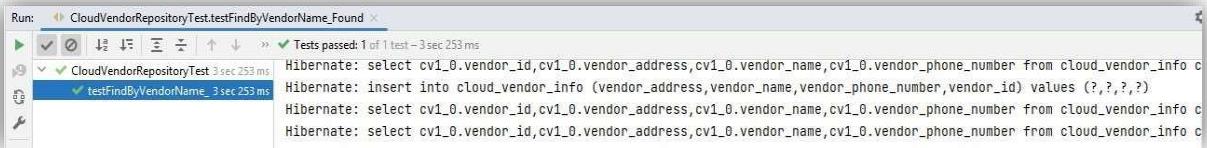
➤ Creating Application.yaml file in resources

Application.yaml

```
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
```

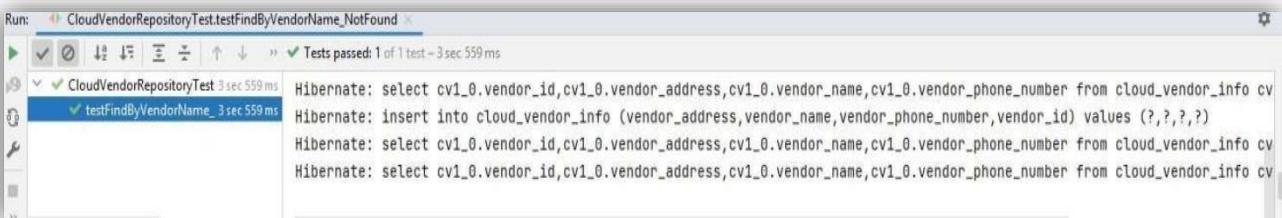
Testing Success case

```
//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());
}
```

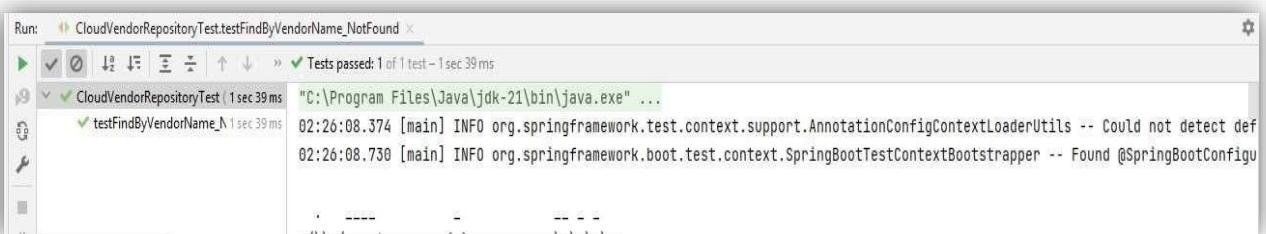


Testing Fail Test Case

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



Run CloudVendorRepositoryTest



CloudVendorServiceImplTest

```
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;

    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());
}
```

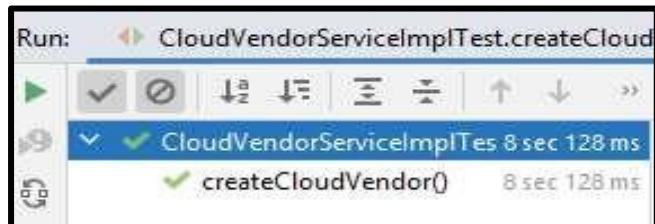
```
@Test
void testGetByVendorName() {
    mock(CloudVendor.class);
    mock(CloudVendorRepository.class);
    when(cloudVendorRepository.findByName("Amazon")).thenReturn(
        new ArrayList<CloudVendor>(Collections.singleton(cloudVendor))
    );
    assertThat(cloudVendorService.getByVendorName("Amazon").get(0).getVendorId())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

@Test
void test GetAllCloudVendors() {
    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());
}
```

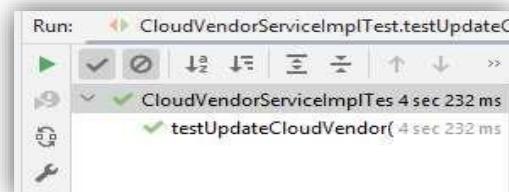
Running CreateCloudVendor Test

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

TestUpdateCloudVendor

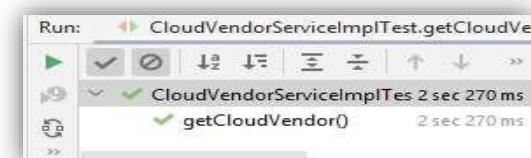


```
@Test  
void testUpdateCloudVendor() {  
    mock(CloudVendor.class);  
    mock(CloudVendorRepository.class);  
    when(cloudVendorRepository.save(cloudVendor)).thenReturn(cloudVendor);  
    assertThat(cloudVendorService.updateCloudVendor(cloudVendor)).isEqualTo("Success");  
}
```



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());  
}
```



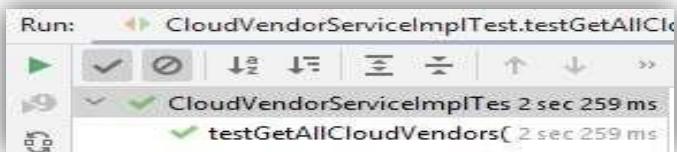
➤ **testGetByVendorName**

```
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());  
}
```



test GetAllCloudVendors

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



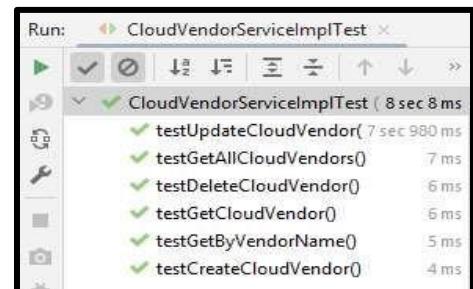
testDeleteCloudVendor

```
@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(status().isOk());
}

@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);

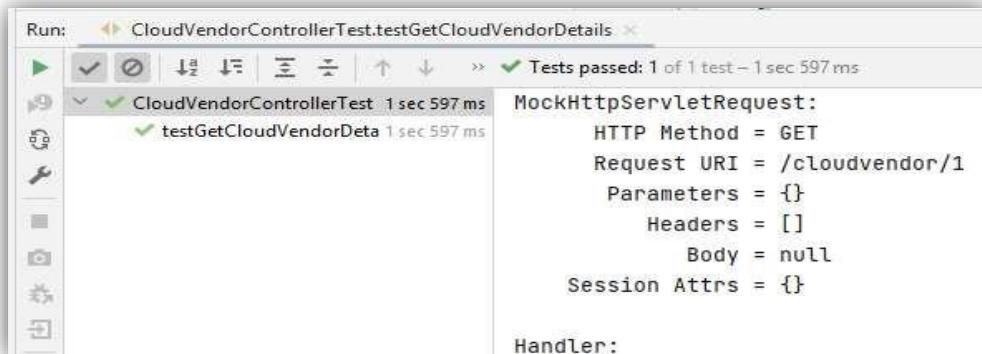
    when(cloudVendorService.updateCloudVendor(cloudVendorOne))
        .thenReturn( value: "Cloud Vendor Updated Successfully");
    this.mockMvc.perform(put( urlTemplate: "/cloudvendor")
        .contentType(MediaType.APPLICATION_JSON)
        .content(requestJson))
        .andDo(print()).andExpect(status().isOk());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

@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());
    }

```



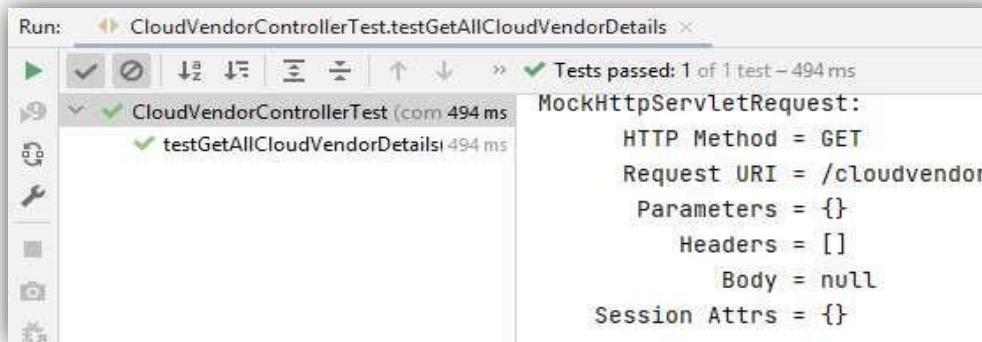
➤ **testGetCloudVendorDetails**

test GetAllCloudVendorDetails

```

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

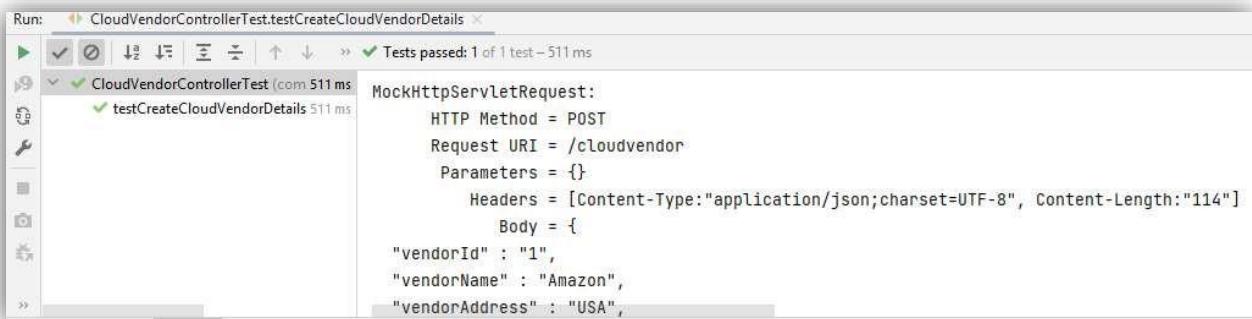
```



testCreateCloudVendorDetails

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

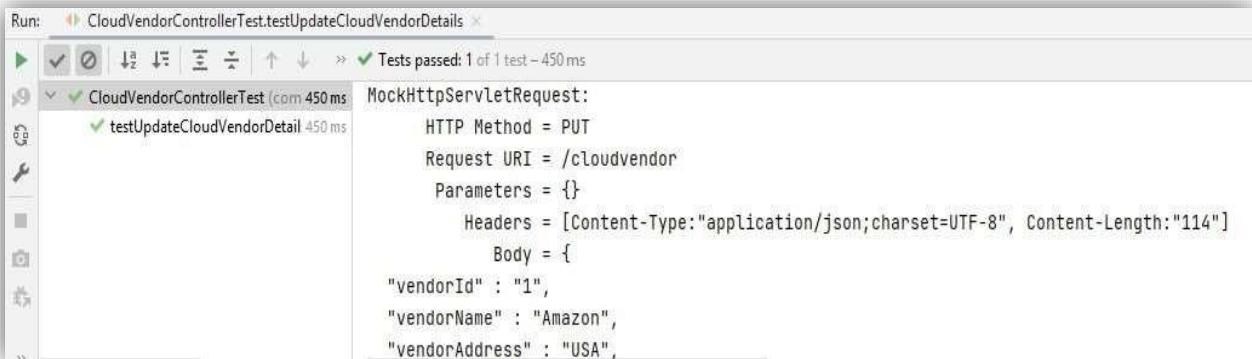
    when(cloudVendorService.createCloudVendor(cloudVendorOne)).thenReturn("Success");
    this.mockMvc.perform(post(urlTemplate: "/cloudvendor")
        .contentType(MediaType.APPLICATION_JSON)
        .content(requestJson))
        .andDo(print()).andExpect(status().isOk());
}
```



➤ testUpdateCloudVendorDetail

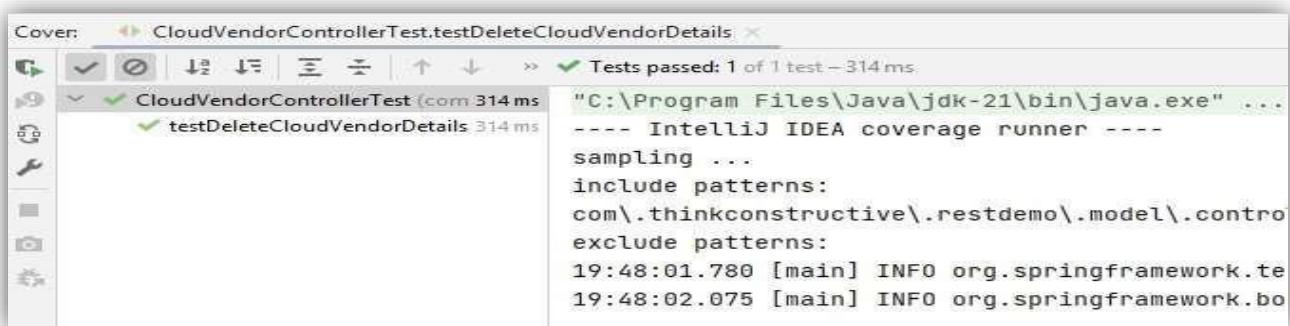
```
void testUpdateCloudVendorDetails() throws Exception {
    ObjectMapper mapper = new ObjectMapper();
    mapper.configure(SerializationFeature.WRAP_ROOT_VALUE, false);
    ObjectWriter ow = mapper.writer().withDefaultPrettyPrinter();
    String requestJson=ow.writeValueAsString(cloudVendorOne);

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

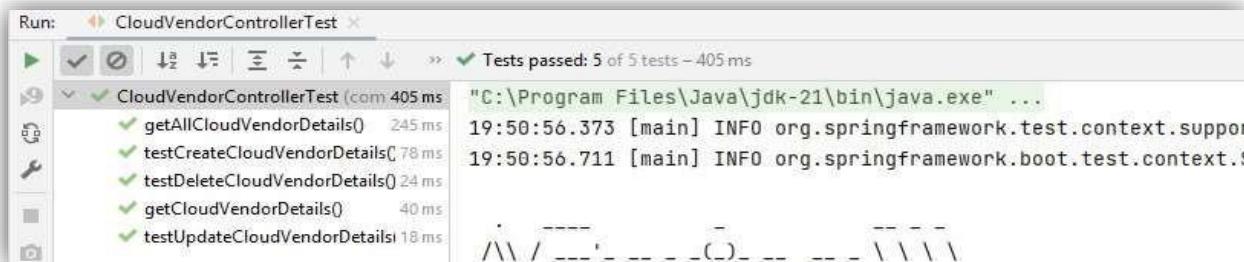


➤ **testDeleteCloudVendorDetails**

```
@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) *** <><><><><><><>