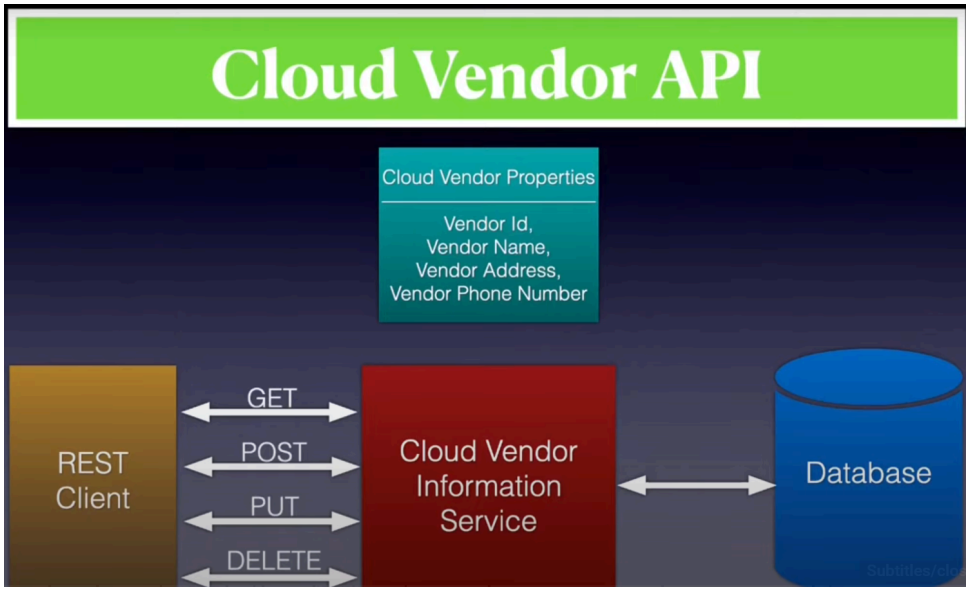
Creating Java REST API with Spring Boot , Spring Data JPA and MySQL | REST API CRUD Operations

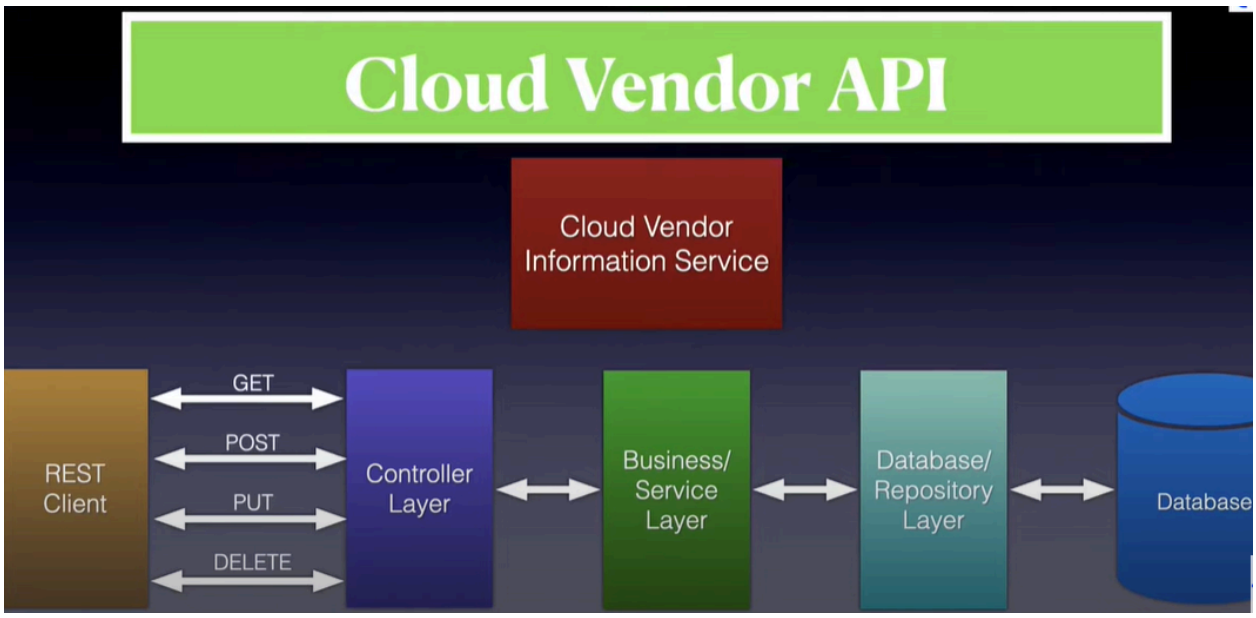
Contents:

- Spring Boot, Spring Data JPA and MySQL
- Cloud Vendor Scenario , REST API and Client
- Spring Boot REST API Project Structure
- Create Spring Boot Project using Spring Initializer with all Dependencies
- pom.xml Dependencies
- Create and Configure application.yaml / application.yml
- MySQL Workbench Schema Verification
- Maven Dependency refresh
- Add @Entitiy , @Table and @Id Annotations to Model Class
- Controller Layer Explanation
- Create Repository with JpaRepository Create Service Interface for CRUD and GetAll
- Service Implementation @Service - CRUD and GetAll methods Implementation
- Implement Controller CRUD and GetAll methods
- Starting Spring Boot Application
- Detailed testing Create , Read , Update , Delete (CRUD) and ReadAll using Postman and MySQL Workbench

Spring Data JPA(Library) focuses on using JPA to store data in a relational database.



Springboot project architecture



CODE:
RestDemoApplication.java: CLASS

```
package com.example.restdemo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class RestDemoApplication {
	public static void main(String[] args) {
		SpringApplication.run(RestDemoApplication.class, args);
	}
}
```

controller.CloudVendorController.java: CLASS
NOTE: Modify controller to interact with the Service layer

```
package com.example.restdemo.controller;
import java.util.List;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
```

```
import com.example.restdemo.model.CloudVendor;
//import com.example.restdemo.repository.CloudVendorRepository;
import com.example.restdemo.service.CloudVendorService;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;

@RestController
@RequestMapping("/cloudvendor")
public class CloudVendorController {
	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.getAllCloudVendor();
	}
	@PostMapping
	public String createCloudVendorDetails(@RequestBody CloudVendor cloudVendor) {
		cloudVendorService.createCloudVendor(cloudVendor);
		return "Created Successfully";
	}
	@PutMapping
	public String updateCloudVendorDetails(@RequestBody CloudVendor cloudVendor) {
		cloudVendorService.updateCloudVendor(cloudVendor);
		return "Updated Successfully";
	}
	@DeleteMapping("{vendorId}")
	public String deleteCloudVendorDetails(@PathVariable("vendorId") String vendorId) {
		cloudVendorService.deleteCloudVendor(vendorId);
		return "Deleted Successfully";		}}
```

model.CloudVendor.java: CLASS
NOTE: Include entity,Table and ID

```
package com.example.restdemo.model;

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

@Entity
@Table(name="cloud_vendor_info")
public class CloudVendor {

	@Id
	private String vendorId;
	private String vendorName;
	private String vendorAddress;
	private String vendorPhone;

	public CloudVendor() {

	}

	public CloudVendor(String vendorId, String vendorName, String vendorAddress, String
vendorPhone) {
		this.vendorId = vendorId;
		this.vendorName = vendorName;
		this.vendorAddress = vendorAddress;
		this.vendorPhone = vendorPhone;
	}
	public String getVendorId() {
		return vendorId;
	}
	public void setVendorId(String vendorId) {
		this.vendorId = vendorId;
	}
	public String getVendorName() {
		return vendorName;
	}
```

```
	public void setVendorName(String vendorName) {
		this.vendorName = vendorName;	}
	public String getVendorAddress() {
		return vendorAddress;
	}
	public void setVendorAddress(String vendorAddress) {
		this.vendorAddress = vendorAddress;
	}
	public String getVendorPhone() {
		return vendorPhone;
	}
	public void setVendorPhone(String vendorPhone) {
		this.vendorPhone = vendorPhone;
	}
}
```

repository.CloudVendorRepository.java: INTERFACE
NOTE: Create this interface to extend with JPA repository

```
package com.example.restdemo.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import com.example.restdemo.model.CloudVendor;

public interface CloudVendorRepository extends JpaRepository<CloudVendor,String> {

}
```

service.CloudVendorService.java : INTERFACE
NOTE: Create methods to perform CRUD operations

```
package com.example.restdemo.service;
import java.util.List;
import com.example.restdemo.model.CloudVendor;
public interface CloudVendorService {
	public String createCloudVendor(CloudVendor cloudVendor);
	public String updateCloudVendor(CloudVendor cloudVendor);
	public String deleteCloudVendor(String cloudvendorId);
	public CloudVendor getCloudVendor(String cloudvendorId);
	public List<CloudVendor> getAllCloudVendor();  }
```

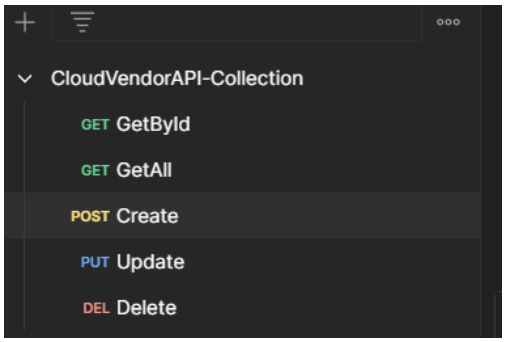
service.impl.CloudVendorSeriveImpl.java- CLASS
NOTE: Implementation of the Service interface

```
package com.example.restdemo.service.impl;

import java.util.List;
import org.springframework.stereotype.Service;
import com.example.restdemo.model.CloudVendor;
import com.example.restdemo.repository.CloudVendorRepository;
import com.example.restdemo.service.CloudVendorService;

@Service
public class CloudVendorSeriveImpl implements CloudVendorService {

	CloudVendorRepository cloudVendorRepository;

	public CloudVendorSeriveImpl(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";
	}

	@Override
	public String deleteCloudVendor(String cloudVendorId) {
		cloudVendorRepository.deleteById(cloudVendorId);
		return "Success";
	}
```

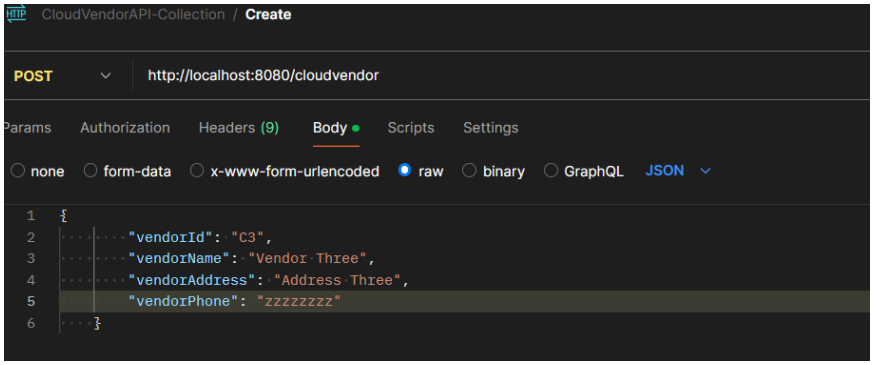
```
        @Override
        public CloudVendor getCloudVendor(String cloudVendorId) {
                return cloudVendorRepository.findById(cloudVendorId).get();        }
        @Override
        public List<CloudVendor> getAllCloudVendor() {
                return cloudVendorRepository.findAll();
        }
}
```

RUN the springboot application, open mySql workbench
If schema not available error occurs - create schema manually

Create requests-



Create 3 IDs using POST



GetById|GetAll-

CloudVendorAPI-Collection / **GetById**

GET http://localhost:8080/cloudvendor/C1

Params Authorization Headers (7) Body Scripts Settings

Query Params

Key	Value
Key	Value

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON


```
{
    "vendorId": "C1",
    "vendorName": "Vendor one",
    "vendorAddress": "Address One",
    "vendorPhone": "xxxxx"
}
```


-

CloudVendorAPI-Collection / **GetAll**

GET http://localhost:8080/cloudvendor

Params Authorization Headers (7) Body Scripts Settings

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON


```
[
    {
        "vendorId": "C1",
        "vendorName": "Vendor one",
        "vendorAddress": "Address One",
        "vendorPhone": "xxxxx"
    },
    {
        "vendorId": "C2",
        "vendorName": "Vendor Twoe",
        "vendorAddress": "Address Two",
        "vendorPhone": "yyyyy"
    },
    {
        "vendorId": "C3",
        "vendorName": "Vendor Three",
        "vendorAddress": "Address Three",
        "vendorPhone": "zzzzzzzz"
    }
]
```


Fields created in DB

cloud_vendor_info

Limit to 1000 rows

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

Result Grid Filter Rows: Edit:

vendor_address	vendor_id	vendor_name	vendor_phone
Address One	C1	Vendor one	xxxxx
Address Two	C2	Vendor Twoe	yyyyy
Address Three	C3	Vendor Three	zzzzzzzz
NULL	NULL	NULL	NULL

UPDATE:

CloudVendorAPI-Collection / **Update**

PUT http://localhost:8080/cloudvendor

Params Authorization Headers (9) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary Graph


```
{
    "vendorId": "C3",
    "vendorName": "Vendor Three UPDATED",
    "vendorAddress": "LONDON",
    "vendorPhone": "zzzzzzzz"
}
```


Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize Text

```
Updated Successfully
```

Changes reflected in DB:

cloud_vendor_info | cloud_vendor_info

Limit to 1000 rows

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

Result Grid | Filter Rows: | Edit:

vendor_address	vendor_id	vendor_name	vendor_phone
Address One	C1	Vendor one	xxxxx
Address Two	C2	Vendor Twoe	yyyyy
LONDON	C3	Vendor Three UPDATED	zzzzzzzz
NULL	NULL	NULL	NULL

DELETE:

CloudVendorAPI-Collection / **Delete**

DELETE http://localhost:8080/cloudvendor/C1

Params Authorization Headers (7) Body Scripts Settings

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize Text

```
1 Deleted Successfully
```

GetAll:

CloudVendorAPI-Collection / **GetAll**

GET http://localhost:8080/cloudvendor

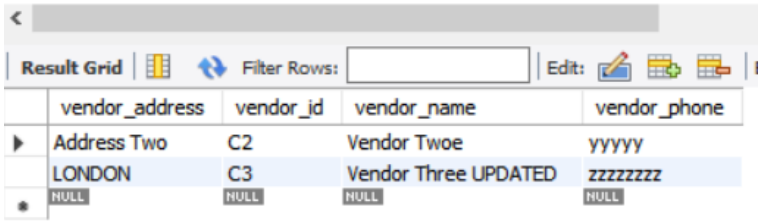
Params Authorization Headers (7) Body Scripts Setting

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON


```
1  [
2      {
3          "vendorId": "C2",
4          "vendorName": "Vendor Twoe",
5          "vendorAddress": "Address Two",
6          "vendorPhone": "yyyyy"
7      },
8      {
9          "vendorId": "C3",
10         "vendorName": "Vendor Three UPDATED",
11         "vendorAddress": "LONDON",
12         "vendorPhone": "zzzzzzzz"
13     }
14 ]
```

Deleted from DB-



vendor_address	vendor_id	vendor_name	vendor_phone
Address Two	C2	Vendor Twoe	yyyyy
LONDON	C3	Vendor Three UPDATED	zzzzzzzz
NULL	NULL	NULL	NULL