

CHAPTER 1: - INTRODUCTION

1.1 Project Description:

In this project, we will design, develop, and deploy a RESTful API using the Spring Boot framework. The goal is to create a scalable, secure, and well-structured API that can serve as a foundation for various web and mobile applications. This project will cover essential concepts of API development, including CRUD operations, data validation, security, documentation, and deployment.

Project Features:

- 1. User Management:* Implement a user management system that allows users to register, log in, and manage their profiles. Utilize Spring Security to secure endpoints and enforce access control.
- 2. Resource Management:* Create resources related to a specific domain. For example, if building a task management API, resources could be tasks, projects, or categories. Implement CRUD operations (Create, Read, Update, Delete) for these resources.
- 3. Validation:* Implement input validation to ensure data integrity. Use Spring Boot's validation framework to handle validation errors gracefully and provide informative responses.
- 4. Exception Handling:* Implement a custom exception handling mechanism to provide meaningful error messages in JSON format when errors occur. Handle common HTTP error codes such as 404 (Not Found) and 400 (Bad Request).
- 5. Data Persistence:* Use Spring Data JPA to interact with a relational database (MySQL)

Project Deliverables:

- A fully functional RESTful API built with Spring Boot.
- Source code hosted on a version control system (e.g., GitHub).
- API documentation.
- Deployment instructions.
- Test coverage reports.
- A presentation or documentation summarizing the project's design decisions and challenges faced during development.

Skills Gained:

- Proficiency in Spring Boot for API development.
- Hands-on experience with Spring Security, Spring Data JPA, and Spring Validation.
- Knowledge of API design best practices.
- Experience with token-based authentication and authorization.
- Deployment and CI/CD pipeline setup.
- Logging and monitoring skills for maintaining production-ready APIs.

This project will provide valuable experience in building RESTful APIs using Spring Boot, making it suitable for individuals or teams looking to enhance their Java and API development skills.

CHAPTER 2: - LITERATURE SURVEY

2.1 Existing System:

The existing RESTful API system for the cloud vendor has been in operation for several years and has evolved to meet customer needs. It currently provides a wide range of cloud management capabilities.

The existing RESTful API system for the cloud vendor has been in operation for a while and serves as the backbone for cloud vendor operations. It provides various services to both company and employee.

Advantages:

- Easy to maintain the system.

2.2 Proposed System:

The proposed RESTful API system is to have everything completely computerized among company's data (E.g., Cloud Vendor) so on. The software is little complex because to handle this system it may requires some technical knowledge (i.e., Add, update, delete, get) to perform operations.

Advantages of Proposed System:

- User can add n number of data through this application.
- It is convenient for the companies

2.3 Tools and Technologies:

1. IDE[Integrated Development Environment]: Eclipse
2. Server: Apache Tomcat
3. Database: MySQL Workbench
4. Language: Java
5. API Testing: Postman

2.4 System Requirements Specifications:

2.4.1 Hardware Requirements

1. PROCESSOR: Intel core i3
2. RAM: 256MB Or More
3. HARD DISC: 4GB and higher
4. INTERFACE: High speed LAN connection

2.4.2 Software Requirements

1. OPERATING SYSTEM: WINDOWS XP/2007/08 and more
2. SERVER-SIDE SOFTWARE: Tomcat server
3. MIDDLEWARE: JAVA
4. DATABASE: MySQL

CHAPTER 3: - SOFTWARE DEVELOPMENT REQUIREMENTS

Functional and non-functional requirements are two essential categories of requirements in software development that help define what a system or software application must do and how it should perform. They serve as the foundation for designing, building, and testing software systems.

3.1 Functional Requirements:

Functional requirements define the specific functionalities, features, and interactions that a software system or application must provide. These requirements describe what the system should do to meet the needs of its users and achieve its intended purpose. Functional requirements are typically expressed in terms of user stories, use cases, or specific system behaviors.

Key characteristics of functional requirements include:

- **Specificity:** They are detailed and specific, outlining the exact behavior or functionality expected from the system.
- **Measurability:** Functional requirements should be testable, allowing for verification and validation.
- **User-Centric:** They are often written from the perspective of end-users and stakeholders to ensure that the system aligns with their needs and expectations.

Examples of functional requirements for a web-based email application might include:

- Users must be able to create an email account.
- Users must be able to send and receive emails.
- The system must allow users to attach files to emails.
- Users must be able to organize emails into folders.

3.2 Non-Functional Requirements:

Non-functional requirements, often referred to as quality attributes or "Utilities," define the characteristics, qualities, and constraints that govern how a software system performs its functions rather than what it does. These requirements focus on aspects such as performance, security, scalability, usability, and reliability.

Key characteristics of non-functional requirements include:

- Qualitative Nature: Non-functional requirements are often qualitative and describe the system's behavior in terms of performance metrics or constraints.
- Cross-Cutting: They can apply to multiple parts of the system and may affect various functional features.
- Trade-offs: Non-functional requirements may sometimes conflict with one another, requiring trade-offs to achieve the desired system qualities.

Examples of non-functional requirements for the same web-based email application might include:







- Performance: The system should load emails within two seconds on average.
- Security: User data must be encrypted during transmission and storage.
- Scalability: The application should support 10,000 concurrent users without significant performance degradation.
- Usability: The user interface must be intuitive and accessible, complying with WCAG accessibility standards.
- Reliability: The system should have a 99.9% uptime, with planned maintenance windows.

CHAPTER 4: - DETAILED DESIGN

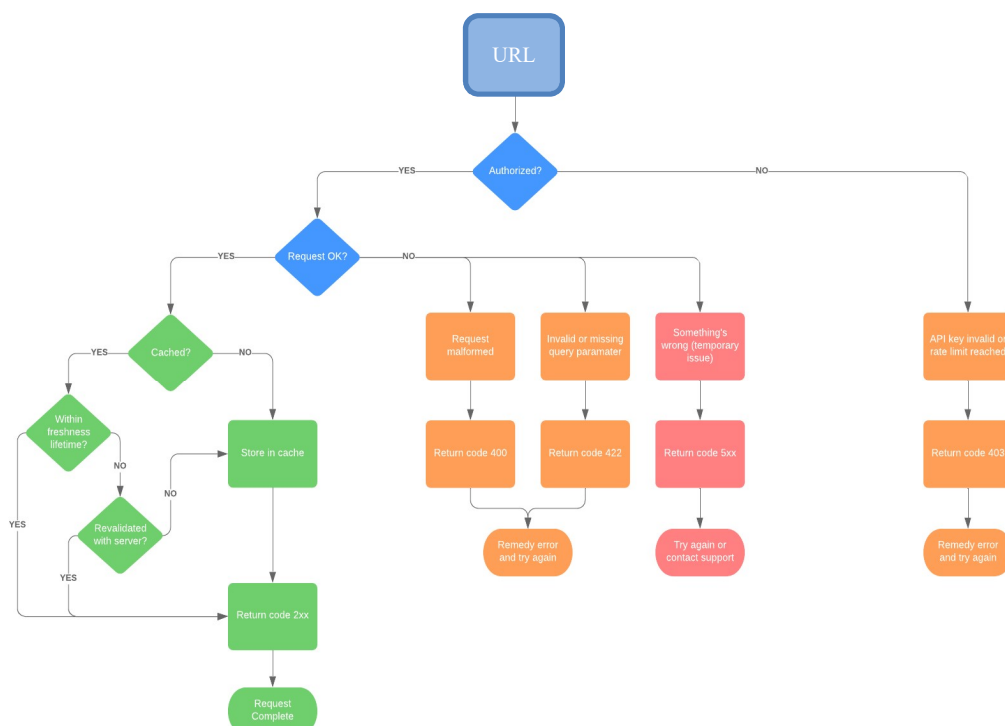
4.1 ER-Diagram (Entity-Relationship):

Symbols Used in ER Model

ER Model is used to model the logical view of the system from a data perspective which consists of these symbols:

Figures	Symbols	Represents
Rectangle		Entities in ER Model
Ellipse		Attributes in ER Model
Diamond		Relationships among Entities
Line		Attributes to Entities and Entity Sets with Other Relationship Types
Double Ellipse		Multi-Valued Attributes
Double Rectangle		Weak Entity

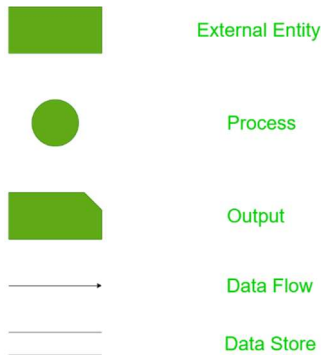
- Rectangles: Rectangles represent Entities in the ER Model.
- Ellipses: Ellipses represent Attributes in the ER Model.
- Diamond: Diamonds represent Relationships among Entities.
- Lines: It represent attributes to entities and entity sets with other relationship types.
- Double Ellipse: Double Ellipses represent Multi-Valued Attributes.
- Double Rectangle: Double Rectangle represents a Weak Entity.
- Symbols used in ER Diagram



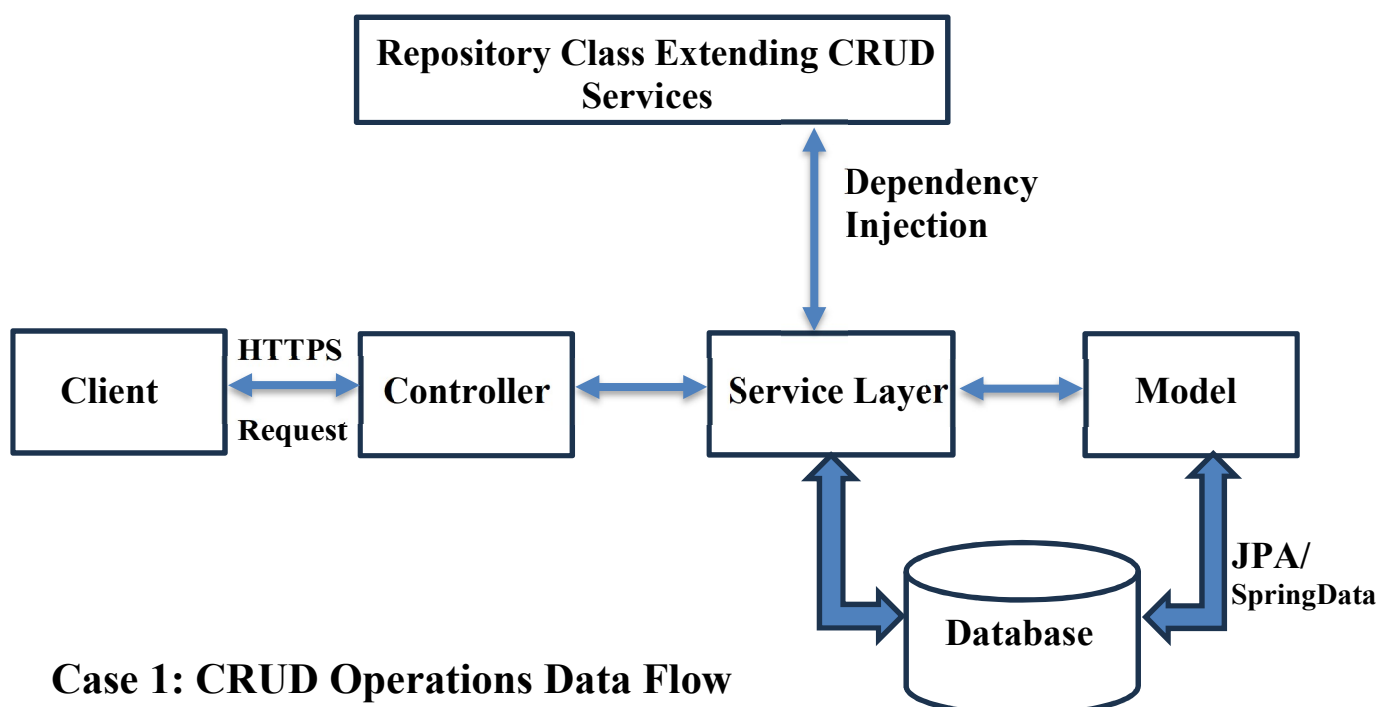
4.2DFD (Data Flow Diagram):

Components of Data Flow Diagram:

Following are the components of the data flow diagram that are used to represent source, destination, storage and flow of data.



- **Entities:** Entities include source and destination of the data. Entities are represented by rectangle with their corresponding names.
- **Process:** The tasks performed on the data is known as process. Process is represented by circle. Somewhere round edge rectangles are also used to represent process.
- **Data Storage:** Data storage includes the database of the system. It is represented by rectangle with both smaller sides missing or in other words within two parallel lines.
- **Data Flow:** The movement of data in the system is known as data flow. It is represented with the help of arrow. The tail of the arrow is source and the head of the arrow is destination.



CHAPTER 5: - IMPLEMENTATION

5.1 CODE

CloudVendor.java

```
package com.rrce.restapi.model;

import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="cloud_vendor_info")
@ApiModel(description = "This table holds cloud vendor information.")
public class CloudVendor
{
    @Id
    @ApiModelProperty(notes="This is a Cloud Vendor Id. It shall be unique.")
    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 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 getVendorPhoneNumber() {
    return vendorPhoneNumber;
}

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

CloudVedorController.java

```
package com.rrce.restapi.controller;

import io.swagger.annotations.ApiOperation;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import com.rrce.restapi.model.CloudVendor;
import com.rrce.restapi.response.ResponseHandler;
import com.rrce.restapi.service.CloudVendorService;
import java.util.List;

@RestController
@RequestMapping("/CloudVendor")
public class CloudVendorController
{
    CloudVendorService cloudVendorService;

    public CloudVendorController(CloudVendorService cloudVendorService) {
        this.cloudVendorService = cloudVendorService;
    }
}
```

```
// Read Specific Cloud Vendor Details from DB
@GetMapping("getDataById/{vendorId}")
@ApiOperation(value = "Cloud vendor id", notes = "Provide cloud vendor details",
response = ResponseEntity.class)
public                               ResponseEntity<Object>
getCloudVendorDetails(@PathVariable("vendorId") String vendorId)
{
return ResponseHandler.responseBuilder("Requested Vendor Details are given
here",
HttpStatus.OK, cloudVendorService.getCloudVendor(vendorId));
}

// Read All Cloud Vendor Details from DB
@GetMapping("/getData")
public List<CloudVendor> getAllCloudVendorDetails()
{
return cloudVendorService.getAllCloudVendors();
}

@PostMapping("/postData")
public String createCloudVendorDetails(@RequestBody CloudVendor
cloudVendor)
{
cloudVendorService.createCloudVendor(cloudVendor);
return "Cloud Vendor Created Successfully";
}

@PostMapping("/updateData")
public String updateCloudVendorDetails(@RequestBody CloudVendor
cloudVendor)
{
cloudVendorService.updateCloudVendor(cloudVendor);
return "Cloud Vendor Updated Successfully";
}

@DeleteMapping("deleteData/{vendorId}")
public String deleteCloudVendorDetails(@PathVariable("vendorId") String
vendorId)
{
cloudVendorService.deleteCloudVendor(vendorId);
return "Cloud Vendor Deleted Successfully";
}
```

```
}  
}
```

CloudVendorException.java

```
package com.rrce.restapi.exception;  
  
import org.springframework.http.HttpStatus;  
  
public class CloudVendorException {  
    private final String message;  
    private final Throwable throwable;  
    private final HttpStatus httpStatus;  
  
    public CloudVendorException(String message, Throwable throwable, HttpStatus  
    httpStatus) {  
        this.message = message;  
        this.throwable = throwable;  
        this.httpStatus = httpStatus;  
    }  
  
    public String getMessage() {  
        return message;  
    }  
  
    public Throwable getThrowable() {  
        return throwable;  
    }  
  
    public HttpStatus getHttpStatus() {  
        return httpStatus;  
    }  
}
```

CloudVendorExceptionHandler.java

```
package com.rrce.restapi.exception;

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);
    }
}
```

CloudVendorNotFoundException.java

```
package com.rrce.restapi.exception;

public class CloudVendorNotFoundException extends RuntimeException {

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

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

CloudVendorRepository.java

```
package com.rrce.restapi.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import com.rrce.restapi.model.CloudVendor;
import java.util.List;

public interface CloudVendorRepository extends JpaRepository<CloudVendor,
String>
{
    List<CloudVendor> findByVendorName(String vendorName);
}
```

Response.java

```
package com.rrce.restapi.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);
    }
}
```

RestAPIMain.java

```
package com.rrce.restapi;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import springfox.documentation.builders.PathSelectors;
import springfox.documentation.builders.RequestHandlerSelectors;
import springfox.documentation.service.ApiInfo;
import springfox.documentation.service.Contact;
import springfox.documentation.spi.DocumentationType;
import springfox.documentation.spring.web.plugins.Docket;

import java.util.Collections;

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

5.2 SNAPSHOTS

MAIN:

```
Console x
RestApiApplication [Java Application] [pid: 16136]

:: Spring Boot :: (v2.7.1)

2023-09-29 19:26:40.416 INFO 16136 --- [main] com.rnce.restapi.RestApiApplication : Starting RestApiApplication using Java 20.0.1 on Nag
2023-09-29 19:26:40.419 INFO 16136 --- [main] com.rnce.restapi.RestApiApplication : No active profile set, falling back to 1 default pro
2023-09-29 19:26:41.452 INFO 16136 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT
2023-09-29 19:26:41.538 INFO 16136 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 61 ms. F
2023-09-29 19:26:42.411 INFO 16136 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2023-09-29 19:26:42.426 INFO 16136 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2023-09-29 19:26:42.426 INFO 16136 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.64]
2023-09-29 19:26:42.594 INFO 16136 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2023-09-29 19:26:42.595 INFO 16136 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed
2023-09-29 19:26:42.824 INFO 16136 --- [main] o.hibernate.jpa.internal.util.LogHelper : HHH000204: Processing PersistenceUnitInfo [name: defa
2023-09-29 19:26:42.882 INFO 16136 --- [main] org.hibernate.Version : HHH000412: Hibernate ORM core version 5.6.9.Final
2023-09-29 19:26:43.078 INFO 16136 --- [main] o.hibernate.annotations.common.Version : HCANN000001: Hibernate Commons Annotations {5.1.2.Fi
2023-09-29 19:26:43.186 INFO 16136 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2023-09-29 19:26:43.385 INFO 16136 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2023-09-29 19:26:43.402 INFO 16136 --- [main] org.hibernate.dialect.Dialect : HHH000400: Using dialect: org.hibernate.dialect.MySQL
2023-09-29 19:26:44.011 INFO 16136 --- [main] o.h.e.t.j.p.i.JtaPlatformInitiator : HHH000490: Using JtaPlatform implementation: [org.hil
2023-09-29 19:26:44.025 INFO 16136 --- [main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence
2023-09-29 19:26:44.693 WARN 16136 --- [main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. There
2023-09-29 19:26:45.192 INFO 16136 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context p
2023-09-29 19:26:45.498 INFO 16136 --- [main] com.rnce.restapi.RestApiApplication : Started RestApiApplication in 5.694 seconds (JVM run
```

Create:

CloudVendor / createRequest

POST http://localhost:8080/postData

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
1 {
2   "vendorId": "45",
3   "vendorName": "sadiq",
4   "vendorAddress": "kumbalgoodu",
5   "vendorPhoneNumber": "9876543219"
6 }
7
```

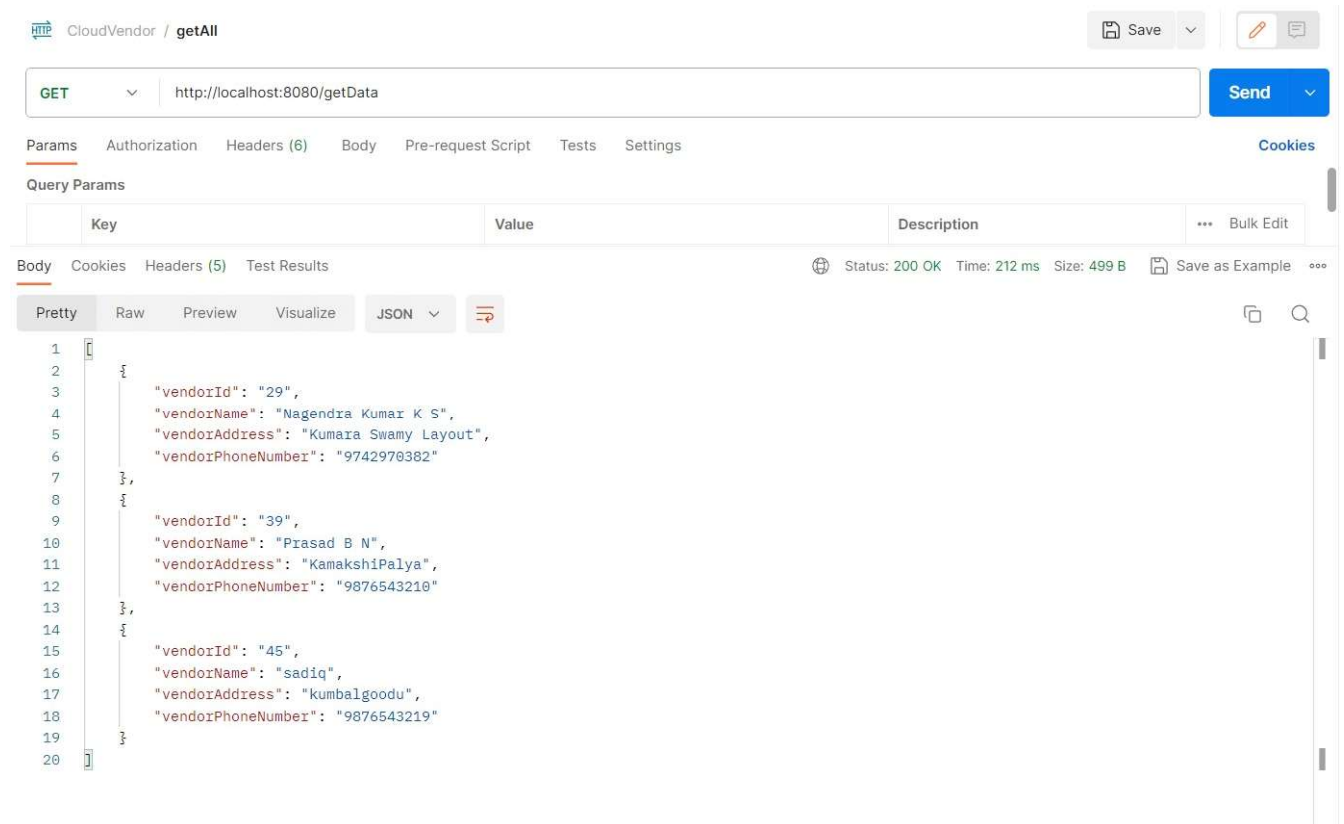
Body Cookies Headers (5) Test Results

Status: 200 OK Time: 466 ms Size: 197 B Save as Example

Pretty Raw Preview Visualize Text

```
1 Cloud Vendor Created Successfully
```


Get All :



CloudVendor / getAll

GET http://localhost:8080/getData

Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

Key	Value	Description	...	Bulk Edit
-----	-------	-------------	-----	-----------

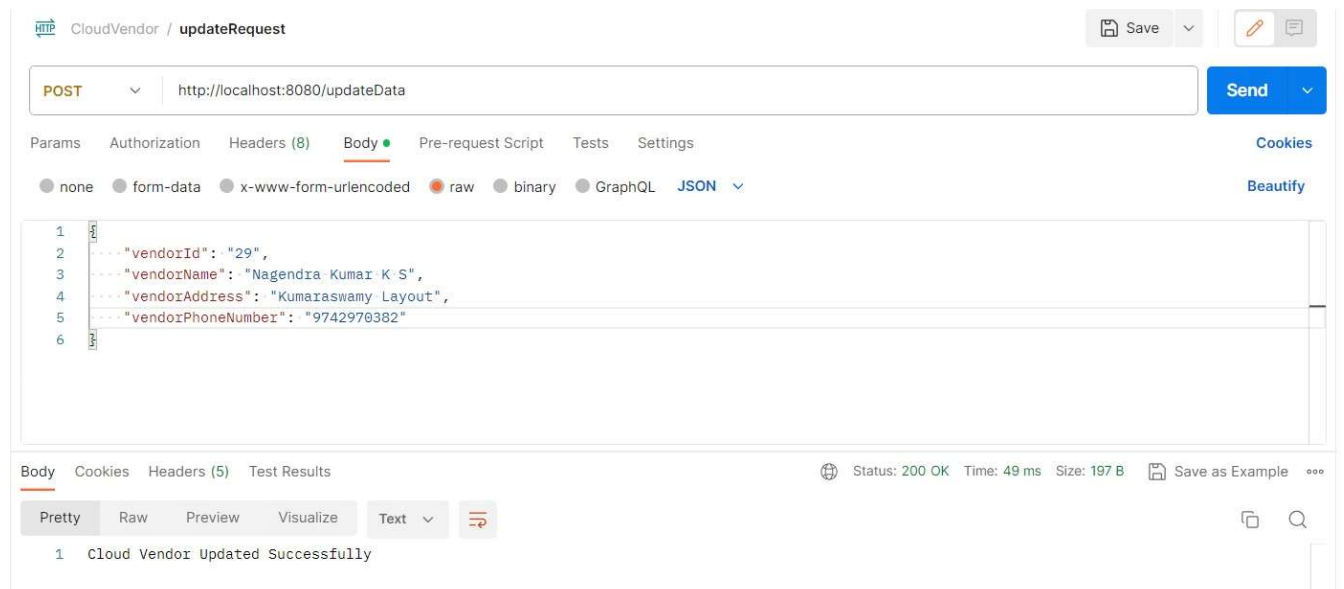
Body Cookies Headers (5) Test Results

Status: 200 OK Time: 212 ms Size: 499 B Save as Example

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "vendorId": "29",
4     "vendorName": "Nagendra Kumar K S",
5     "vendorAddress": "Kumara Swamy Layout",
6     "vendorPhoneNumber": "9742970382"
7   },
8   {
9     "vendorId": "39",
10    "vendorName": "Prasad B N",
11    "vendorAddress": "Kamakshi Palya",
12    "vendorPhoneNumber": "9876543210"
13  },
14  {
15    "vendorId": "45",
16    "vendorName": "sadiq",
17    "vendorAddress": "kumbalgoodu",
18    "vendorPhoneNumber": "9876543219"
19  }
20 ]
```

Update:



CloudVendor / updateRequest

POST http://localhost:8080/updateData

Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON

Beautifuly

```
1 {
2   "vendorId": "29",
3   "vendorName": "Nagendra Kumar K S",
4   "vendorAddress": "Kumaraswamy Layout",
5   "vendorPhoneNumber": "9742970382"
6 }
```

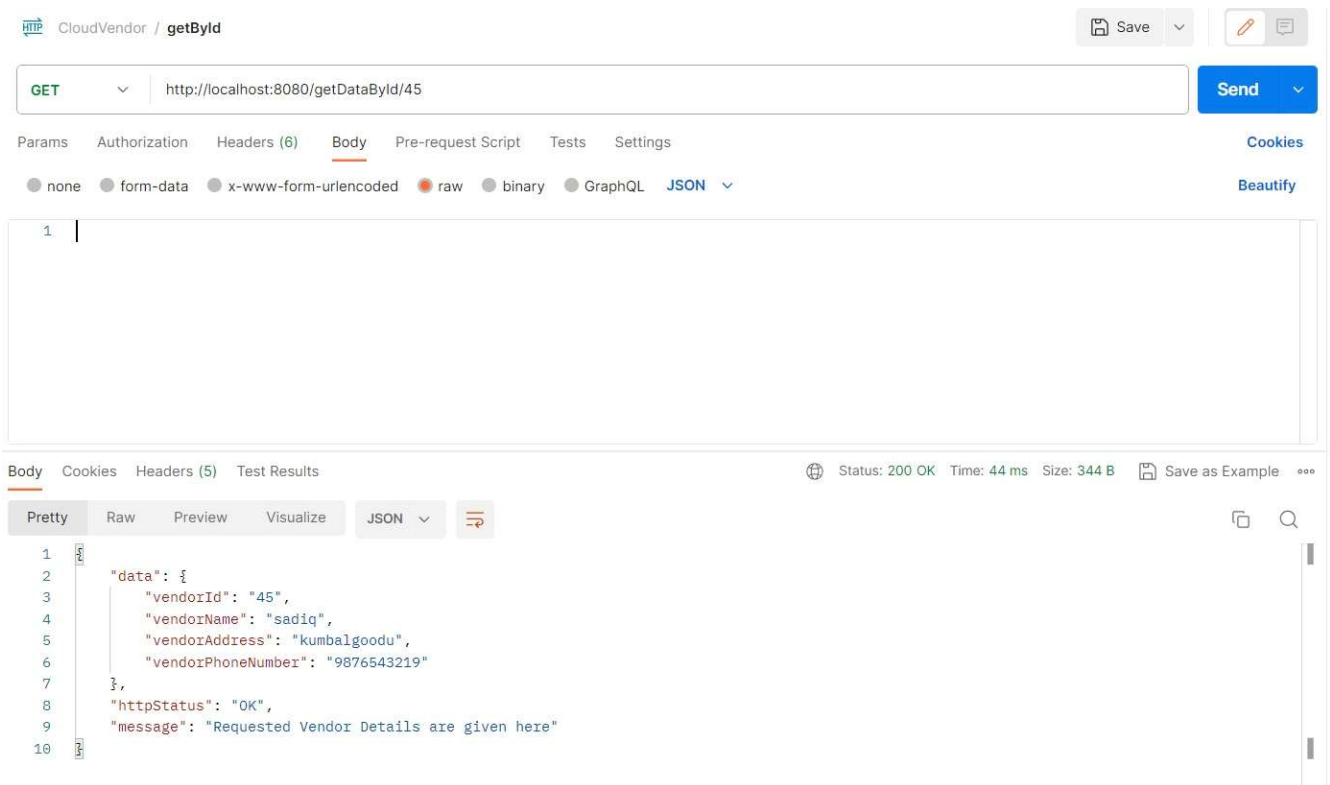
Body Cookies Headers (5) Test Results

Status: 200 OK Time: 49 ms Size: 197 B Save as Example

Pretty Raw Preview Visualize Text

```
1 Cloud Vendor Updated Successfully
```

Get By ID:



CloudVendor / **getById**

GET http://localhost:8080/getDataById/45

Params Authorization Headers (6) **Body** Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

1

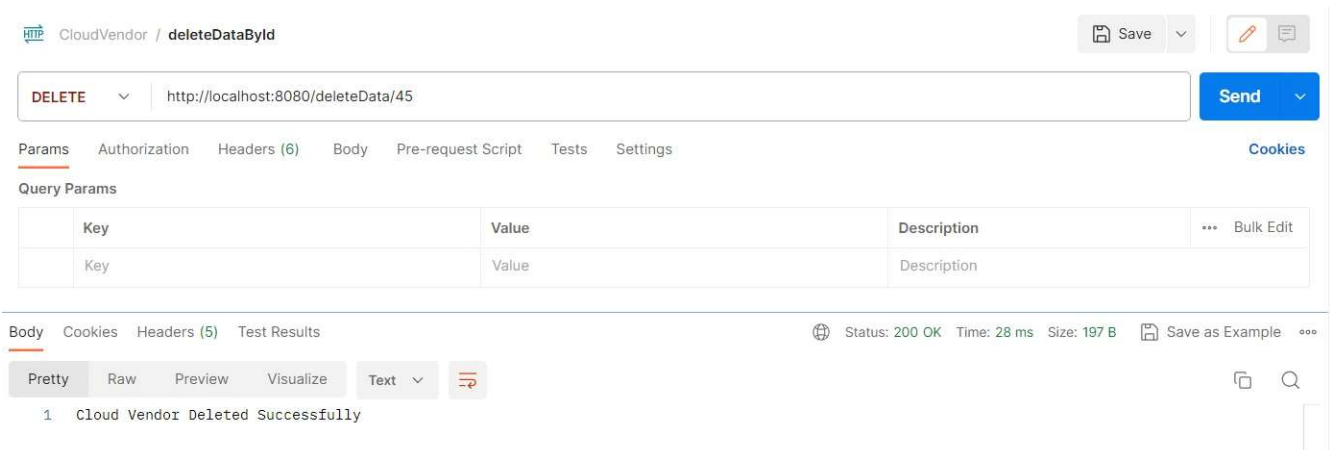
Body Cookies Headers (5) Test Results

Status: 200 OK Time: 44 ms Size: 344 B Save as Example

Pretty Raw Preview Visualize JSON

```
1 {
2   "data": {
3     "vendorId": "45",
4     "vendorName": "sadiq",
5     "vendorAddress": "kumbalgoodu",
6     "vendorPhoneNumber": "9876543219"
7   },
8   "httpStatus": "OK",
9   "message": "Requested Vendor Details are given here"
10 }
```

Delete:



CloudVendor / **deleteDataById**

DELETE http://localhost:8080/deleteData/45

Params Authorization Headers (6) **Body** Pre-request Script Tests Settings Cookies

Query Params

Key	Value	Description	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (5) Test Results

Status: 200 OK Time: 28 ms Size: 197 B Save as Example

Pretty Raw Preview Visualize Text

1 Cloud Vendor Deleted Successfully

CHAPTER 6: CONCLUSION

In the ever-evolving landscape of web development, building efficient and scalable APIs is a paramount concern. RESTful APIs have emerged as a widely accepted architectural style for designing networked applications due to their simplicity, flexibility, and statelessness. When combined with the robust and developer-friendly Spring Boot framework, RESTful APIs become even more powerful. In this conclusion, we will summarize the key points discussed throughout this exploration of RESTful APIs in the context of Spring Boot. In the ever-evolving landscape of web development, building efficient and scalable APIs is a paramount concern. RESTful APIs have emerged as a widely accepted architectural style for designing networked applications due to their simplicity, flexibility, and statelessness. When combined with the robust and developer-friendly Spring Boot framework, RESTful APIs become even more powerful. In this conclusion, we will summarize the key points discussed throughout this exploration of RESTful APIs in the context of Spring Boot.

REST, which stands for Representational State Transfer, is not just an architectural style; it is a set of principles that guide the design of APIs to promote scalability, maintainability, and ease of use. RESTful APIs leverage HTTP as their communication protocol, making them a natural choice for web applications. One of the fundamental principles of REST is the use of resources, identified by unique URIs, which can be manipulated through standard HTTP methods (GET, POST, PUT, DELETE). This approach simplifies API design and promotes a consistent and intuitive interaction model.

CHAPTER 7: BIBILOGRAPHY

7.1 Websites:

<https://www.w3schools.com/>

<https://stackoverflow.com/>

<https://www.javatpoint.com/>

7.2 Referenced Videos:

<https://www.youtube.com/watch?v=L2o485T70Do&list=PLcs1FE1CmEu121ggGwlQt47d0SqNkzSTK&index=3>

<https://www.youtube.com/watch?v=L2o485T70Do&list=PLcs1FE1CmEu121ggGwlQt47d0SqNkzSTK&index=3>