M.Kowsthabi

20A21A1240

Swarnandhra College of Engineering and Technology

# Spring RESTful API Web Service Abstract:

Spring REST API (Application Programming Interface) is a powerful framework that facilitates the development of scalable and robust web services using the Spring framework. This abstraction enables the creation of RESTful (Representational State Transfer) APIs, allowing seamless communication between different software systems over the HTTP protocol.

The Spring framework provides a comprehensive set of tools and features that simplify the process of designing, implementing, and maintaining RESTful APIs. Leveraging the principles of modularity, flexibility, and extensibility, developers can create RESTful services that cater to the evolving needs of modern web and mobile applications.

# Key features of Spring REST API include:

1)Annotation-driven development

2)Request Handling

3)Data Serialization and Deserialization

4)Security

5)Testing and Documentation

6)Integration with Other Spring Projects

Overall , the Spring REST API empowers developers to build scalable and maintainable web services by providing a flexible and feature-rich framework. Its adherence to industry best practices and standards ensures the creation of RESTful APIs that meet the demands of modern software development.

Steps for implementing Basic Spring REST API Service : Initializing the Spring Boot Application : ¬

The first thing you should do is familiarize yourself with the basics of Spring and set up a Spring Boot application.

Open Google chrome or any other browser and type start.spring.io and browse then the following page will be displayed.

Select the project as Maven and language as java and go with the springboot version 3.2.0 then name your project metadata as you like and do the packaging with jar(Java ARcheive) file and with the java version 17.

Then you need to add some dependencies that are needed for our project.

Here I added only the Spring Web dependency which is used for building web,including RESTful application using Spring MVC and it uses Apache Tomcat as the default embedded container. ¬

Then the spring initializer page will be displayed as follows.

As we can see the there is a Generate button in the above picture. Click on that Generate button then a Restdeom-Showoff3 zip file will be downloaded automatically. ¬

Then extract that Restdemo-Showoff3 zip file

After Extracting the zip file a file folder will be generated and in that folder there are some files like pom.xml , src , gitignore file ,mvnw files etc.Then go the folder path and remove the particular path present in it

Remove that particular path and type cmd(command prompt)

After clicking enter you will able to see a command prompt window with that respective path

Then type code . command then you will be able to see the vscode(Visual Studio Code) window with RESTDEMO\_SHOWOFF3 folder creation with that particular files.

After that I created a Model folder and Controller folder inside the java/com/example/RestdemoShowoff3 and inside that Model folder

I created CloudVendor.java and inside that Controller folder I created CloudVendorAPI.java file. ¬

This combination of files forms a basic Spring Boot application with a RESTful API for retrieving cloud vendor details based on a provided VendorId. The CloudVendor class defines the structure of a cloud vendor, and the CloudVendorAPI class serves as the controller handling HTTP requests related to cloud vendors

Model class: In Spring MVC, the model works a container that contains the data of the application. Here, a data can be in any form such as objects, strings, information from the database, etc. Controller class: In Spring Boot, the controller class is responsible for processing incoming REST API requests, preparing a model, and returning the view to be rendered as a response. The controller classes in Spring are annotated either by the @Controller or the @RestController annotation. ¬

In CloudVendor.java I declared four variables like vendorId , vendorName,vendorAddress and vendorPhoneNumber and all with string data types. And also I initialized zero-argument constructor and parameterized constructor,getter and setter methods and also the toString() method for all Vendor details. ¬ The code for the CloudVendor.java is as follows.

Annotations used in CloudVendorAPI.java :

In Spring Boot, are @RestController, @RequestMapping, and @GetMapping annotations commonly used for building RESTful APIs.

Here's an explanation of each:

@RestController: ¬

The @RestController annotation is a specialized version of the @Controller annotation. It is used to indicate that the class defines a RESTful API endpoint.

¬ When you annotate a class with @RestController, it implies that every method inside the class is treated as a controller method and returns the response in a format suitable for RESTful services (typically JSON). @RequestMapping: ¬

The @RequestMapping annotation is used to map web requests to specific methods in a controller class. It can be applied at the class level and/or method level. ¬ It allows you to define the base URI for all the methods in the class and then further refine the URI for each method. ¬

It can specify the HTTP method (GET, POST, PUT, DELETE) and other request parameters

. @GetMapping: ¬ The @GetMapping annotation is a specialized version of @RequestMapping focused on HTTP GET requests. It is a shortcut for @RequestMapping(method = RequestMethod.GET). ¬ It simplifies the code by making it more concise when you're dealing specifically with GET requests

The following is the code for CloudVendorAPI.java. ¬ Here I used only @RestController,@RequestMapping,and @GetMapping annotations and by using the vendorId I will retrieve the details of vendors

¬ This code appears to be a Java class representing a RESTful API for managing cloud vendor details using the Spring Framework's @RestController and @RequestMapping annotations.

¬ Overall ,this code defines a RESTful API for managing cloud vendor details with a single endpoint that retrieves vendor details based on the provided vendorId. However, the implementation is currently using hardcoded values instead of utilizing the vendorId parameter to fetch specific details dynamically from a data source. The actual implementation of retrieving details based on vendorId should be added for the code to be functional

Then after run the application. As default the application will run on port 8080 in Apache Tomcat server.

Then for displaying the output(Vendor Details) we can use any browser. And type localhost:8080/vendorId(Here I gave C1 as vendorId) ¬ Then you can see the vendor details output as follows.

There are many tools that you can use to test your REST API. Postman is a popular REST API testing tool, and you can use it to test the simple API you have built. After running the Spring application, you can launch Postman and experiment with the url with specific qualities. ¬

Here to retrieve the vendor details I have passed the vendorId.So the respective url will be

<http://localhost:8080/vendordetails/C1>

Simple CRUD Application in Spring Boot Abstract : The "SimpleCrudApp" is a lightweight CRUD (Create, Read, Update, Delete) application developed using the Spring Boot framework. The application showcases fundamental principles of RESTful API development, focusing on managing Vendor entities without the use of a traditional database. Instead, an in-memory data structure is employed to store and manipulate Vendor objects. Controller class :

¬ The main controller class that handles HTTP requests related to vendor entities. ¬ Defines endpoints for creating, retrieving, updating, and deleting Vendor records. ¬

Utilizes Spring annotations such as

@RestController,

@PostMapping,

@GetMapping,

@PutMapping, and

@DeleteMapping to facilitate the mapping of HTTP methods to CRUD operations.

In-Memory Data Storage : ¬

Utilizes a simple in-memory list to store Vendor objects during the application's runtime. ¬

Serves as a substitute for a traditional database, enabling rapid development and prototyping without external dependencies.

Testing : ¬

The application can be tested using tools like Postman, cURL, or any HTTP client to interact with the RESTful API. ¬

Here I am using Postman tool to test the RESTful API. ¬

Here firstly I created a Model folder and Controller folder inside the java/com/example/RestdemoShowoff3 and inside that Model folder I created CloudVendoe.java and inside that Controller folder I created a CloudVendorAPI.java file.

CloudVendor.java :

¬ This Java code defines a class named CloudVendor within the package com.example.RestdemoShowoff3.Model. ¬

The class has four private attributes: vendorId, vendorName, vendorAddress, and vendorPhoneNumber, representing different aspects of a cloud vendor. ¬

the CloudVendor class represents a model for a cloud vendor with attributes such as ID, name, address, and phone number. It provides methods to access and modify these attributes, and the toString method generates a string representation of the object for easy debugging and logging

CloudVendorAPI.java : ¬

This java code represents a Spring Boot RESTful controller for managing cloud vendor details. ¬ It uses four annotations for basic crud application.

They are @GetMapping ,

@PostMapping ,

@PutMapping ,

@DeleteMapping

@RestController: ¬

This annotation marks the class as a Spring MVC controller that handles HTTP requests and automatically serializes the response into JSON. It's a specialized version of @Controller with the @ResponseBody annotation applied to all methods. @RequestMapping: ¬

This annotation at the class level maps the base URI "/vendordetails" to this controller. It establishes the base path for all endpoint mappings defined within this controller. @GetMapping: ¬

This annotation maps the HTTP GET request to the specified URI pattern, in this case, "/vendordetails/{vendorId}". It is used to retrieve details of a specific cloud vendor based on the provided vendorId path variable.

@PostMapping: ¬

This annotation maps the HTTP POST request to the "/vendordetails" URI. It is used for adding new cloud vendor details by consuming the request body, which is expected to contain JSON representing a CloudVendor object.

@PutMapping: ¬

This annotation maps the HTTP PUT request to the "/vendordetails" URI. It is used for updating cloud vendor details by consuming the request body, which is expected to contain JSON representing a modified CloudVendor object.

@DeleteMapping: ¬

This annotation maps the HTTP DELETE request to the "/vendordetails/{vendorId}"

URI. It is used to delete the cloud vendor details associated with the provided vendorId path variable. Methods or CRUD operations of CloudVendorAPI.java @GetMapping: ¬

An HTTP GET method that retrieves details of a specific cloud vendor based on the vendorId path variable. In the example, it returns a hardcoded CloudVendor object

@PostMapping:

¬ An HTTP POST method that adds new cloud vendor details by consuming the request body containing JSON representing a CloudVendor object. The method sets the provided cloudVendor in the class variable and returns a success message.