PROGRAM
ON
OBJECT ORIENTED PROGRAMMING WITH JAVA

UNIT-V
SPRING FRAMEWORK

BY
DR. BIRENDRA KR. SARASWAT
GLBITM
GREATER NOIDA

Email : birendra.saraswat@glbitm.ac.in Phone: +91-9999600474

UNIT 5

CO 5	Test web and RESTful Web Services with Spring Boot using Spring Framework concepts	K_5

Text Books

1. Herbert Schildt, "Java The complete reference", McGraw Hill Education
2. Craig Walls, “Spring Boot in Action” Manning Publication

by

Dr. Uma Tomer

Spring Framework: Spring Core Basics-Spring Dependency Injection concepts, Spring Inversion of Control, AOP, Bean Scopes- Singleton, Prototype, Request, Session, Application, Web Socket, Auto wiring, Annotations, Life Cycle Call backs, Bean Configuration styles

Spring Boot: Spring Boot Build Systems, Spring Boot Code Structure, Spring Boot Runners, Logger, BUILDING RESTFUL WEB SERVICES, Rest Controller, Request Mapping, Request Body, Path Variable, Request Parameter, GET, POST, PUT, DELETE APIs, Build Web Applications

Java Editions

- Java Editions are different versions of the Java platform, each customized for specific types of applications and environments.
- These editions are built on the same core Java language but come with distinct libraries and APIs to meet different programming needs.

Java Editions

Java Edition	Purpose	Key Features & APIs	Common Uses
Java SE (Standard Edition)	General-purpose computing	Core APIs (Collections, Concurrency, Networking, I/O)	Desktop and server applications
Java EE (Enterprise Edition)	Large-scale, distributed, and enterprise-level applications	Additional APIs (Servlets, JSP, EJB, JPA, Web Services)	Web applications, enterprise systems, e-commerce platforms
Java ME (Micro Edition)	Mobile and embedded systems	Lightweight APIs (CLDC, MIDP)	Mobile apps for feature phones, embedded systems
Java FX	Rich client-side applications	APIs for GUI, media, graphics, animations	Desktop apps with modern user interfaces, multimedia apps

Java EE (Enterprise Edition)

- Java EE Key Features and APIs:
  - Servlets and JSP
  - EJB (Enterprise JavaBeans)
  - JPA (Java Persistence API)
  - JMS (Java Messaging Service)
  - Web Services APIs (JAX-RS, JAX-WS)
- Common Uses of Java Enterprise Edition:
  - Web applications
  - Business logic layers
  - Distributed enterprise systems
- Examples:
  - E-commerce platforms like Amazon,
  - Banking and insurance systems,
  - Enterprise Resource Planning (ERP) software

Spring Framework

- Spring Framework can be thougt of as a framework of frameworks because it provides support to various frameworks like EJB, Hibernate etc.

- Spring is a lightweight java framework.

- Spring Framework is based on two design principles - Dependency Injection and Aspect Oriented Programming.

- Spring Framework is a Java platform that provides comprehensive infrastructure support for developing Java applications. Spring handles the infrastructure so you can focus on your application.

- Spring enables you to build applications from “plain old Java objects” (POJOs) and to apply enterprise services non-invasively to POJOs. This capability applies to the Java SE programming model and to full and partial Java EE.

Spring Framework

- It is impossible to understand what is Spring Framework without understanding what is Dependency Injection and Inversion of Control.
- Dependency Injection also called as DI, is one of the types of Inversion of Control (IoC).

- **Inversion of Control -** this is the principle of object-oriented programming, in which objects of the program do not depend on concrete implementations of other objects, but may have knowledge about their abstractions (interfaces) for later interaction.

Dependency Injection

- **Dependency Injection** - is a composition of structural design patterns, in which for each function of the application there is one, a conditionally independent object (service) that can have the need to use other objects (dependencies) known to it by interfaces.
- Dependencies are transferred (implemented) to the service at the time of its creation.
- This is a situation where we introduce an element of one class into another.
- In practice, DI is implemented by passing parameters to the constructor or using setters.
- Libraries that implement this approach are also called IoC containers.

Aspect oriented programming(AOP)

- Aspect oriented programming - a programming paradigm that allows you to distinguish cross-through (functional) functionality in application.
- These functions, which span multiple application nodes, are called cross-cutting concerns and these cross-cutting notes are separated from the immediate business logic of the application.
- In OOP, the key unit is the class, while in AOP, the key element is the aspect.
- DI helps to separate application classes into separate modules, and AOP helps to separate cross-cutting concerns from the objects they affect.

The Spring Framework Inversion of Control (IoC)

- The Spring Framework Inversion of Control (IoC) component addresses this concern by providing a formalized means of composing disparate components into a fully working application ready for use.
- The Spring Framework codifies formalized design patterns as first-class objects that you can integrate into your own application(s).
- Numerous organizations and institutions use the Spring Framework in this manner to engineer robust, maintainable applications.

Core Container

- The Core Container consists of the Core, Beans, Context, and Expression Language modules.

- The Core and Beans modules provide the fundamental parts of the framework, including the IoC and Dependency Injection features.
- The BeanFactory is a sophisticated implementation of the factory pattern.
- It removes the need for programmatic singletons and allows you to decouple the configuration and specification of dependencies from your actual program logic.

Spring's AOP

- Spring's AOP module provides an AOP Alliance-compliant aspect-oriented programming implementation allowing you to define, for example, method-interceptors and pointcuts to cleanly decouple code that implements functionality that should be separated.
- Using source-level metadata functionality, you can also incorporate behavioral information into your code, in a manner similar to that of .NET attributes.

- The separate Aspects module provides integration with AspectJ.

- The Instrumentation module provides class instrumentation support and classloader implementations to be used in certain application servers.

Bean Scopes

- Bean Scopes refer to the lifecycle of a Bean, which means when the object of a Bean is instantiated, how long it lives, and how many objects are created for that Bean throughout its lifetime.

- Basically, it controls the instance creation of the bean, and it is managed by the Spring container.

The Spring framework

- The Spring framework provides five scopes for a bean.

- We can use three of them only in the context of a web-aware Spring ApplicationContext, and the rest of the two are available for both an IoC container and a Spring-MVC container.

The Spring framework

- **Singleton:** Only one instance will be created for a single bean definition per Spring IoC container, and the same object will be shared for each request made for that bean.

- **Prototype:** A new instance will be created for a single bean definition every time a request is made for that bean.

The Spring framework

- **Request:** A new instance will be created for a single bean definition every time an HTTP request is made for that bean. But only valid in the context of a web-aware Spring ApplicationContext.

- **Session:** Scopes a single bean definition to the lifecycle of an HTTP Session. But only valid in the context of a web-aware Spring ApplicationContext.

- **Global-Session**: Scopes a single bean definition to the lifecycle of a global HTTP Session. It is also only valid in the context of a web-aware Spring ApplicationContext.

Difference Between Singleton and Prototype Bean

Singleton	Prototype
Only one instance is created for a single bean definition per Spring IoC container.	A new instance is created for a single bean definition every time a request is made for that bean.
Same object is shared for each request made for that bean. i.e. The same object is returned each time it is injected.	For each new request a new instance is created. i.e. A new object is created each time it is injected.
By default, scope of a bean is singleton. So we don't need to declare a bean as singleton explicitly.	By default, scope is not prototype, so you have to declare the scope of a bean as prototype explicitly.
Singleton scope should be used for stateless beans.	While prototype scope is used for all beans that are stateful.

Autowiring in Spring

- It is a feature that enables the Spring framework to automatically resolve and inject dependencies between beans.

- This eliminates the need for explicit configuration of bean relationships, reducing boilerplate code and simplifying application development.

Types of Autowiring:

- **byType:**
- Spring attempts to match the dependency type with a bean of the same type in the application context.
- **byName:**
- Spring attempts to match the dependency name with a bean ID of the same name in the application context.
- **constructor:**
- Spring attempts to match the constructor arguments with corresponding beans in the application context.
- **autodetect:**
- Spring first tries constructor autowiring, and if not found, falls back to autowiring by type.

Modes	Description
No	This mode tells the framework that autowiring is not supposed to be done. It is the default mode used by Spring.
byName	It uses the name of the bean for injecting dependencies.
byType	It injects the dependency according to the type of bean.
Constructor	It injects the required dependencies by invoking the constructor.
Autodetect	The autodetect mode uses two other modes for autowiring - constructor and byType.

Bean Life Cycle

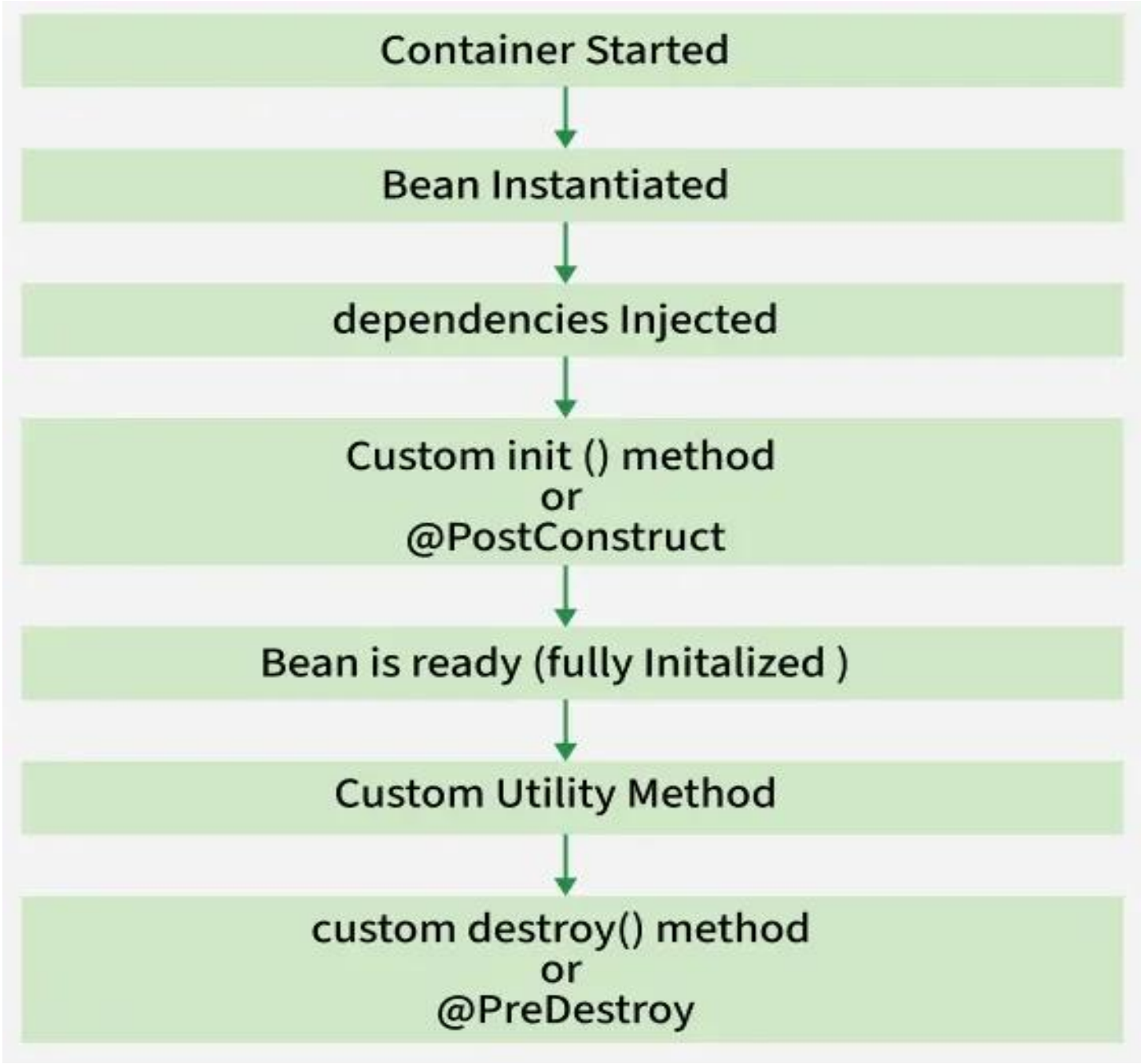
- The lifecycle of a bean in Spring refers to the sequence of events that occur from the moment a bean is instantiated until it is destroyed.

- Bean life cycle is managed by the spring container.
- When we run the program, first of all, the spring container gets started.
- After that, the container creates the instance of a bean as per the request, and then dependencies are injected.
- Finally, the bean is destroyed when the spring container is closed.
- Therefore, if we want to execute some code on the bean instantiation and just after closing the spring container, then we can write that code inside the custom init() method and the destroy() method.

Bean Life Cycle Phases

- The lifecycle of a Spring bean consists of the following phases, which are listed below

- Container Started: The Spring IoC container is initialized.
- Bean Instantiated: The container creates an instance of the bean.
- Dependencies Injected: The container injects the dependencies into the bean.
- Custom init() method: If the bean implements InitializingBean or has a custom initialization method specified via @PostConstruct or init-method.
- Bean is Ready: The bean is now fully initialized and ready to be used.
- Custom utility method: This could be any custom method you have defined in your bean.
- Custom destroy() method: If the bean implements DisposableBean or has a custom destruction method specified via @PreDestroy or destroy-method, it is called when the container is shutting down.


Container Started
Bean Instantiated
dependencies Injected
Custom init () method
or
@PostConstruct
Bean is ready (fully Initalized)
Custom Utility Method
custom destroy() method
or
@PreDestroy

WebSocket

- WebSocket provides an alternative to the limitation of efficient communication between the server and the web browser by providing bi-directional, full-duplex, real-time client/server communications.

- The server can send data to the client at any time.

- Because it runs over TCP, it also provides a low-latency, low-level communication, and reduces the overhead of each message.

- Modern web browsers implement the WebSocket protocol and provide a JavaScript API to connect to endpoints, send messages, and assign callback methods for WebSocket events (such as opened connections, received messages, and closed connections).

WebSocket

- JSR 356 : JSR 356, or the Java API for WebSocket, specifies an API that Java developers can use for integrating WebSockets within their applications, both on the server side, as well as on the Java client side.

- This Java API provides both server and client side components:

- Server: everything in the **jakarta.websocket.server** package
- Client: the content of **jakarta.websocket** package, which consists of client side APIs, and also common libraries to both server and client

Spring Container

- The Spring Container, also known as the IoC (Inversion of Control) container, is the core component of the Spring Framework.
- It manages the lifecycle and configuration of objects (called beans) within a Spring application.
- The container creates, configures, and manages the dependencies between beans, following the IoC principle where the container, not the developer, controls the object lifecycle.

What is Java Spring Boot?

- Java Spring Boot (Spring Boot) is a tool that makes developing web applications and microservices with Java Spring Framework faster and easier.

Java Spring Framework (Spring Framework)

- Java Spring Framework (Spring Framework) is a popular, open source, enterprise-level framework for creating stand-alone, production-grade applications that run on the Java virtual machine (JVM).
- Spring Boot streamlines and simplifies Spring Framework development through three core features:
  - Autoconfiguration
  - An opinionated approach to configuration
  - The ability to create stand-alone applications

Build Systems

- In Java Spring Boot, a build system is a tool that automates the process of compiling, packaging, and managing dependencies for your project. It streamlines the development workflow by handling tasks such as:
- **Compiling Source Code**: Translates Java source code into bytecode, which the Java Virtual Machine (JVM) can execute.
- **Managing Dependencies:** Downloads and manages external libraries or frameworks that your project relies on.
- **Packaging Applications:** Bundles compiled code and resources into deployable formats (e.g., JAR or WAR files).
- **Running Tests:** Executes unit and integration tests to ensure code quality.
- **Deploying Applications:** Packages the application for deployment to different environments.

Key Build Systems for Spring Boot

- **Maven:**
- A widely used build tool that uses an XML-based configuration file (pom.xml) to define project structure, dependencies, and build processes.

- **Gradle:**
- A more modern build tool that uses a Groovy or Kotlin-based Domain Specific Language (DSL) for configuration, offering greater flexibility and performance compared to Maven.

How Build Systems Work in Spring Boot

- **Project Setup**: You define your project's structure and dependencies in the build configuration file (e.g., pom.xml for Maven, build.gradle for Gradle).

- **Dependency Management**: The build system automatically downloads and manages the necessary libraries based on the defined dependencies.

- **Compilation:** It compiles the Java source code into bytecode.

- **Packaging**: It bundles the compiled code and resources into an executable JAR or WAR file.

- **Execution**: You can run the packaged application.

Benefits of Using a Build System

- **Automation:** Automates repetitive tasks, saving time and effort.
- **Dependency Management:** Simplifies the process of managing external libraries.
- **Consistency:** Ensures consistent builds across different environments.
- **Reproducibility:** Makes builds reproducible, regardless of the machine used.
- **Integration:** Integrates with other development tools and platforms.

Spring Boot - Code Structure

- There is no specific layout or code structure for Spring Boot Projects.
- However, there are some best practices followed by developers that will help us too.
- We can divide your project into layers like ***service layer, entity layer, repository layer,, etc.***
- We can also divide the project into modules.
- For example, the parent project has two child modules.
- The first module is for the data layer and the second module is for the web layer.
- We can also divide the project into features.

.

- Two approaches to structure their spring boot projects.
- Structure by Feature
- Structure by Layer

Structure 1: By feature

- In this approach, all classes pertaining to a certain feature are placed in the same package.
- The structure by feature looks is shown in below example

- ***The advantages of this structure is as follows:***

- Find a class to be modified is easy.
- By deleting a particular sub-package, all the classes related to a certain feature can be deleted.
- Testing and Refactoring is easy.
- Features can be shipped separately

```
com
 +- gfg
     +- demo
         +- MyApplication.java
         |
         +- customer
         |   +- Customer.java
         |   +- CustomerController.java
         |   +- CustomerService.java
         |   +- CustomerRepository.java
         |
         +- order
             +- Order.java
             +- OrderController.java
             +- OrderService.java
             +- OrderRepository.java
```

Structure 2: By Layer

- Another way to place the classes is by layer i.e; all controllers can be placed in controllers package and services under services package and all entities under domain or model etc.

- ***Disadvantages when compared to Structure by Feature.***

- Features or Modules cannot be shipped separately.
- Hard to locate a class pertaining to a certain feature.
- Code Refactoring on a certain feature is difficult since the feature classes located in every layer.

```
com
 +- gfg
     +- demo
         +- MyApplication.java
         |
         +- domain
         |   +- Customer.java
         |   +- Order.java
         |
         +- controllers
         |     +- OrderController.java
         |   +- CustomerController.java
         |
         +- services
         |    +- CustomerService.java
         |    +- OrderService.java
         |
         +- repositories
              +- CustomerRepository.java
              +- OrderRepository.java
```

Spring Boot Runners

- Spring Boot provides two interfaces, CommandLineRunner and ApplicationRunner, to execute code after the application context is loaded but before the application is fully running.
- These interfaces are useful for performing initialization tasks or setting up the application environment.

CommandLineRunner

- This interface has a single run method that takes a String array as an argument representing the command-line arguments passed to the application.
- It's simpler to use and suitable for basic tasks that need access to the raw command-line arguments.

ApplicationRunner

- This interface also has a single run method, but it takes an ApplicationArguments object as an argument.
- The ApplicationArguments object provides access to both raw and parsed command-line arguments, allowing you to differentiate between option and non-option arguments.
- It offers more flexibility compared to CommandLineRunner.

Execution Order

- If multiple runners are defined, ApplicationRunner implementations are executed first, followed by CommandLineRunner implementations.
- The execution order within each type is determined by the alphabetical order of their class names by default.
- To customize the execution order, you can implement the Ordered interface or use the @Order annotation.

Loggers:

- Named entities within the application that generate log messages. Each logger is associated with a level, which determines the severity of messages it will output.
- Log Levels: are : These categorize the importance of log messages. Common levels include:
- TRACE: Very detailed information, used for granular debugging.
- DEBUG: Information useful for debugging.
- INFO: General information about the application's operation.
- WARN: Indicates potential issues that do not prevent the application from functioning.
- ERROR: Indicates errors that may prevent the application from functioning correctly.
- FATAL: Indicates critical errors that may cause the application to terminate.
- OFF: Disables logging.

Restful Web Services

- Restful Web Services is a stateless client-server architecture where web services are resources and can be identified by their URIs.
- REST Client applications can use HTTP GET/POST methods to invoke Restful web services.
- REST doesn’t specify any specific protocol to use, but in almost all cases it’s used over HTTP/HTTPS.
-
- When compared to SOAP web services, these are lightweight and doesn’t follow any standard.
- We can use XML, JSON, text or any other type of data for request and response.

RESTful API

- A RESTful API in Java is an interface that allows different software applications to communicate using HTTP methods like GET, POST, PUT, and DELETE.

- It follows the Representational State Transfer (REST) architectural style, which emphasizes a client-server model where clients request data and servers respond with the requested data or perform actions.

- RESTful APIs use common web technologies, making them work smoothly across different platforms.

Key principles of RESTful APIs include

Client-Server Architecture:

The client and server operate independently, allowing for flexibility and scalability.

Stateless Communication:

Each request from a client to a server must contain all the information needed to fulfill the request. No session state is stored on the server.

Uniform Interface:

Resources are accessed through a consistent set of operations (GET, POST, PUT, DELETE) and identified by URIs.

Cacheable Data:

Responses from the server can be explicitly marked as cacheable to improve performance.

Layered System:

The application can be divided into multiple layers, enhancing modularity and scalability.

Java RESTful Web Services API

- Java API for RESTful Web Services (JAX-RS) is the Java API for creating REST web services.

- JAX-RS uses annotations to simplify the development and deployment of web services.

- JAX-RS is part of JDK, so you don’t need to include anything to use it’s annotations.

Restful Web Services Annotations

- Some of the important JAX-RS annotations are:
- @Path: used to specify the relative path of class and methods. We can get the URI of a webservice by scanning the Path annotation value.
- @GET, @PUT, @POST, @DELETE and @HEAD: used to specify the HTTP request type for a method.
- @Produces, @Consumes: used to specify the request and response types.
- @PathParam: used to bind the method parameter to path value by parsing it.

RESTful APIs

- HTTP methods are the backbone of RESTful APIs, enabling seamless communication. between clients and servers.
- The four most commonly used HTTP methods—GET, POST, PUT, and DELETE

What Are HTTP Methods in APIs?

- HTTP (Hypertext Transfer Protocol) methods are standardized actions that clients (e.g., browsers or applications) use to interact with resources on a server.

- In RESTful APIs, these methods map to CRUD (Create, Read, Update, Delete) operations, making them essential for managing data.

Key HTTP Methods:

- **GET**: Retrieve data from the server.
- **POST**: Create a new resource or submit data.
- **PUT**: Update or replace an existing resource.
- **DELETE**: Remove a resource.


thank you