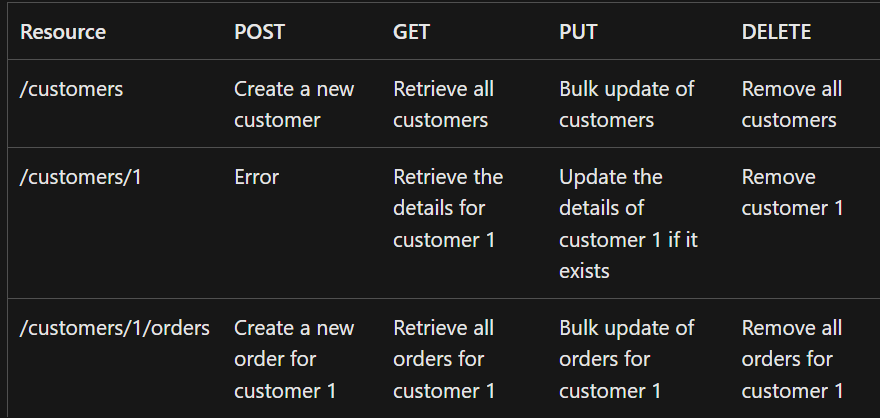
***RESTful APIs operations***

* REST APIs use a uniform interface, which helps to decouple the client and service implementations. For REST APIs, built on HTTP, the uniform interface includes using standard HTTP verbs to perform operations on resources. The most common operations are GET, POST, PUT, PATCH and DELETE.
* GET retrieves a representation of the resource at the specified URI. The body of the response message contains the details of the requested resource.
* POST creates a new resource at the specified URI. The body of the request message provides the details of the new resource. Note that POST can also be used to trigger operations that don't actually create resources.
* PUT either creates or replaces the resource at the specified URI. The body of the request message specifies the resource to be created or updated.
* PATCH performs a partial update of a resource. The request body specifies the set of changes to apply to the resource.
* DELETE removes the resource at the specified URI.

Examples of some scenarios:



* The differences between POST, PUT, and PATCH can be confusing.
* A POST request creates a resource. The server assigns a URI for the new resource, and returns that URI to the client. In the REST model, you frequently apply POST requests to collections. The new resource is added to the collection. A POST request can also be used to submit data for processing to an existing resource, without any new resource being created.
* A PUT request creates a resource or updates an existing resource. The client specifies the URI for the resource. The request body contains a complete representation of the resource. If a resource with this URI already exists, it is replaced. Otherwise a new resource is created, if the server supports doing so. PUT requests are most frequently applied to resources that are individual items, such as a specific customer, rather than collections. A server might support updates but not creation via PUT. Whether to support creation via PUT depends on whether the client can meaningfully assign a URI to a resource before it exists. If not, then use POST to create resources and PUT or PATCH to update.
* A PATCH request performs a partial update to an existing resource. The client specifies the URI for the resource. The request body specifies a set of changes to apply to the resource. This can be more efficient than using PUT, because the client only sends the changes, not the entire representation of the resource. Technically PATCH can also create a new resource (by specifying a set of updates to a "null" resource), if the server supports this.
* PUT requests must be idempotent. If a client submits the same PUT request multiple times, the results should always be the same (the same resource will be modified with the same values). POST and PATCH requests are not guaranteed to be idempotent.

J-UNIT TESTING VIDEO AND DETAILS

***Spring Boot Features***

Spring Boot is an open-source tool that makes it easier for developers to create standalone digital products and production-ready spring applications including Java applications and web services. It uses a micro framework, which makes it most useful for creating microservices for websites and mobile apps.

ADVANTAGES:

Springboot works well with several servlet containers

Bootstrapping saves memory space

Decreased boilerplate code

No XML configuration required

WAR files are not required

POM dependency management

A large community of helpful users

DISADVANTAGES:

Large deployment files that result from unused dependencies

Long amount of time that it takes to replace legacy systems with Springboot applications.

Its inability to build large, monolithic applications

Needing to rely on the Spring ecosystem instead of using other tools that many developers find advantageous.

Features:

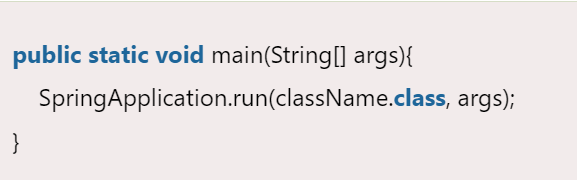
* Inversion of control (IoC)
* Dependency injection (DI)
* Relying on Spring MVC to handle requests from HTTP servers
* Spring Boot auto-configuration that doesn’t require manual configuration
* Building REST API services
* Command-line interface (CLI)
* It is an easy approach to building microservices that can run alongside each other without interfering with their functionality
* Web Development
* SpringApplication
* Application events and listeners
* Admin features
* Externalized Configuration
* Properties Files
* YAML Support
* Type-safe Configuration
* Logging
* Security

In detail:

Web Development:

It is well suited Spring module for web application development. We can easily create a self-contained HTTP server using embedded Tomcat, Jetty or Undertow. We can use the spring-boot- starter-web module to start and running application quickly.

Spring Application:

It is a class which provides the convenient way to bootstrap a spring application which can be started from main method. You can call start your application just by calling a static run() method.  


Application Events and Listeners:

Spring Boot uses events to handle variety of tasks. It allows us to create factories file that are used to add listeners. we can refer it by using ApplicationListener key.

Admin Features:

Spring Boot provides the facility to enable admin related features for the application. It is used to access and manage application remotely. We can enable it by simply using spring.application.admin.enabled property.

Externalized Configuration:

Spring Boot allows us to externalize our configuration so that we can work with the same application in different environments. Application use YAML files to externalize configuration.

Properties Files:

Spring Boot provides rich set of Application Properties. So, we can use that in properties file of our project. Properties file is used to set properties like: **server-port = 8082** and many others. It helps to organize application properties.

YAML Support:

It provides convenient way for specifying hierarchical configuration. It is a superset of JSON. The SpringApplication class automatically support YAML. It is successful alternative of properties.

Type-safe Configuration:

Strong type-safe configuration is provided to govern and validate the configuration of application. Application configuration is always a crucial task which should be type-safe. We can also use annotation provided by this library.

Logging:

Spring Boot uses Common logging for all internal logging. Logging dependencies are managed by default. We should not change logging dependencies, if there is no required customization is needed.

Security:

Spring Boot applications are spring bases web applications. So, it is secure by default with basic authentication on all HTTP endpoints. A rich set of Endpoints are available for develop a secure Spring Boot application.

POSTMAN AND SWAGGER – separate document

Microservices and Non-Banking: Sir URL