### **Full Series Link:** [**https://www.youtube.com/playlist?list...**](https://www.youtube.com/playlist?list=PL0zysOflRCen-GihOcm1hZfYAlwr63K_M)[**#backend\_course**](https://www.youtube.com/hashtag/backend_course)[**#spring\_boot**](https://www.youtube.com/hashtag/spring_boot)

### **What do you understand by the term ‘Spring Boot’?**

Spring Boot is a project that is built on the top of the Spring Framework. It provides an easier and faster way to set up, configure, and run both simple and web-based applications.

It is a Spring module that provides the **RAD (*Rapid Application Development*)** feature to the Spring Framework. It is used to create a stand-alone Spring-based application that you can just run because it needs minimal Spring configuration.



In short, Spring Boot is the combination of **Spring Framework** and **Embedded Servers**.

In Spring Boot, there is no requirement for XML configuration (deployment descriptor). It uses convention over configuration software design paradigm that means it decreases the effort of the developer.

### **Explain the advantages of using Spring Boot for application development.**

* Fast and easy development of Spring-based applications;
* No need for the deployment of war files;
* The ability to create standalone applications;
* Helping to directly embed Tomcat, Jetty, or Undertow into an application;
* No need for XML configuration;
* Reduced amounts of source code;
* Additional out-of-the-box functionality;
* Easy start;
* Simple setup and management;
* Large community and many training programs to facilitate the familiarization period.`

### **Differentiate between Spring and Spring Boot.**

* The Spring Framework provides multiple features like dependency injection, data binding, aspect-oriented programming (AOP), data access, and many more that help easier development of web applications whereas Spring Boot helps in easier usage of the Spring Framework by simplifying or managing various loosely coupled blocks of Spring which are tedious and have a potential of becoming messy.
* Spring boot simplifies commonly used spring dependencies and runs applications straight from a command line. It also doesn’t require an application container and it helps in monitoring several components and configures them externally.

### **What are the features of Spring Boot?**

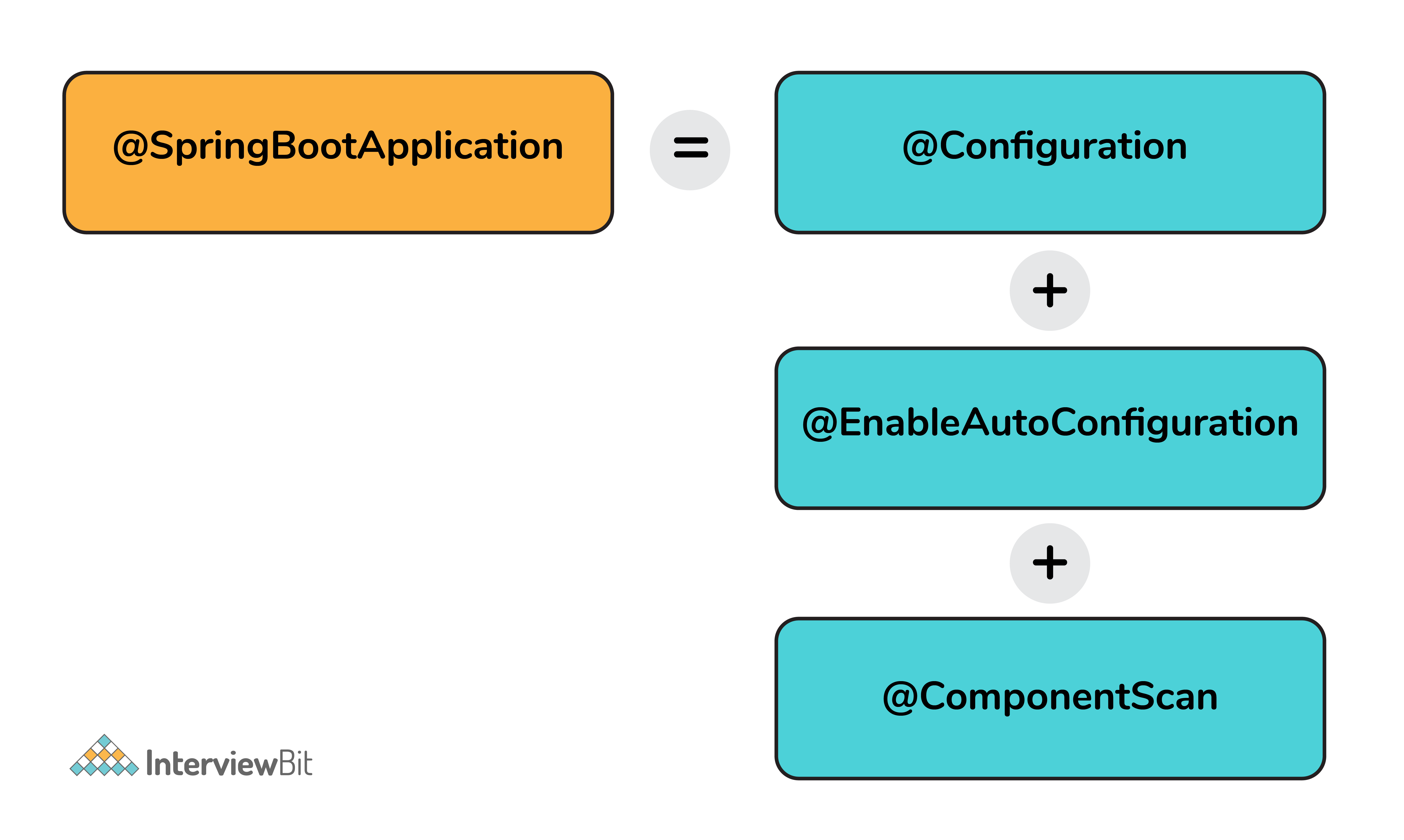
* **Spring Boot CLI** – This allows you to Groovy / Maven for writing Spring boot application and avoids boilerplate code.
* **Starter Dependency** – With the help of this feature, Spring Boot aggregates common dependencies together and eventually improves productivity and reduces the burden on
* **Spring Initializer**– This is a web application that helps a developer in creating an internal project structure. The developer does not have to manually set up the structure of the project while making use of this feature.
* **Auto-Configuration** – This helps in loading the default configurations according to the project you are working on. In this way, unnecessary WAR files can be avoided.
* **Spring Actuator** – **Spring Boot Actuator** is a sub-project of the Spring Boot Framework. It includes a number of additional features that help us to monitor and manage the Spring Boot application. It contains the actuator endpoints (the place where the resources live). We can use **HTTP** and **JMX** endpoints to manage and monitor the Spring Boot application. If we want to get production-ready features in an application, we should use the S**pring Boot actuator.**

**/\***Spring boot uses actuator to provide “Management EndPoints” which helps the developer in going through the Application Internals, Metrics etc.\*/

* **Logging and Security** – This ensures that all the applications made using Spring Boot are properly secured without any hassle.

### **What does @SpringBootApplication annotation do internally?**

### As per the Spring Boot documentation, the @SpringBootApplication annotation is one point replacement for using @Configuration, @EnableAutoConfiguration and @ComponentScan annotations alongside their default attributes.



This enables the developer to use a single annotation instead of using multiple annotations thus lessening the lines of code. However, Spring provides loosely coupled features which is why we can use these annotations as per our project needs.

**Ques: - Explain @SpringBootApplication?**

**Ans: -** Spring Boot @SpringBootApplication annotation is used to mark a configuration class that declares one or more @Bean methods and also triggers auto-configuration and component scanning.

It’s same as declaring a class with or @springbootApplication consist of the following

@Configuration,

@EnableAutoConfiguration and

@ComponentScan annotations.

**Ques: - what is @componentscan?**

**Ans: -** @ComponentScan annotation enables component scanning in Spring. Java classes that are decorated with stereotypes such as @Component, @Configuration, @Service are auto-detected by Spring. The @ComponentScan's basePackages attribute specifies which packages should be scanned for decorated beans.

**@ComponentScan enables Spring to scan for things like configurations, controllers, services, and other components we define.**

**Ques: - what is @EnableAutoConfiguration?**

Ans: - **The**@EnableAutoConfiguration **annotation enables Spring Boot to auto-configure the application context**. **Therefore, it automatically creates and registers beans based on both the included jar files in the classpath and the beans defined by us.**

**Ques: - what is @Configuration?**

**Ans: -** Spring @Configuration annotation is part of the spring core framework. Spring Configuration annotation indicates that the class has @Bean definition methods. So, Spring container can process the class and generate Spring Beans to be used in the application.

**Ques: - what is @Autowired or Autowiring?**

**Ans: -** Autowiring feature of spring framework enables you to inject the object dependency implicitly. It internally uses setter or constructor injection.

Autowiring can't be used to inject primitive and string values. It works with reference only.

## Autowiring Modes

|  |  |  |
| --- | --- | --- |
| **No.** | **Mode** | **Description** |
| 1) | no | It is the default autowiring mode. It means no autowiring bydefault. |
| 2) | byName | The byName mode injects the object dependency according to name of the bean.  In such case, property name and bean name must be same.  It internally calls setter method. |
| 3) | byType | The byType mode injects the object dependency according to type.  So, property name and bean name can be different.  It internally calls setter method. |
| 4) | constructor | The constructor mode injects the dependency by calling the constructor  of the class. It calls the constructor having large number of p arameters. |
| 5) | autodetect | It is deprecated since Spring 3. (For both setter & constructor) |

**Ques: - Explain dialect in yaml?**

**Ans: -** The dialect specifies the type of database used in hibernate so that hibernate generate appropriate type of SQL statements. For connecting any hibernate application with the database, it is required to provide the configuration of SQL dialect.

Example

|  |  |
| --- | --- |
| PostgreSQL | org.hibernate.dialect.PostgreSQLDialect |

### **What are the effects of running Spring Boot Application as “Java Application”?**

* **T**he application automatically launches the tomcat server as soon as it sees that we are running a web application.

### **What is Spring Boot dependency management system?**

* It is basically used to manage dependencies and configuration automatically without the need of specifying the version for any of that dependencies.

### **What are the possible sources of external configuration?**

* Spring Boot allows the developers to run the same application in different environments by making use of its feature of external configuration. This uses environment variables, properties files, command-line arguments, YAML files, and system properties to mention the required configuration properties for its corresponding environments. Following are the sources of external configuration:
  + **Command-line properties** – Spring Boot provides support for command-line arguments and converts these arguments to properties and then adds them to the set of environment properties.
  + **Application Properties** – By default, Spring Boot searches for the application properties file or its YAML file in the current directory of the application, classpath root, or config directory to load the properties.
  + **Profile-specific properties** – Properties are loaded from the application-{profile}.properties file or its YAML file. This file resides in the same location as that of the non-specific property files and the {profile} placeholder refers to an active profile or an environment.

### **Can we change the default port of the embedded Tomcat server in Spring boot?**

* Yes, we can change it by using the application properties file by adding a property of server.port and assigning it to any port you wish to.
* For example, if you want the port to be 8081, then you have to mention server.port=8081. Once the port number is mentioned, the application properties file will be automatically loaded by Spring Bo ot and the specified configurations will be applied to the application.

### **What is the default port of tomcat in spring boot?**

### The default port of the tomcat server-id 8080. It can be changed by adding **sever.port** properties in the **application.property** file.

### **Can we override or replace the Embedded tomcat server in Spring Boot?**

### Yes, we can replace the Embedded Tomcat server with any server by using the Starter dependency in the **pom.xml** file. Like you can use spring-boot-starter-jetty as a dependency for using a jetty server in your project.

### **Can you tell how to exclude any package without using the basePackages filter?**

We can use the  exclude  attribute while using the annotation  @SpringBootApplication  as follows:

@SpringBootApplication(exclude= {Student.class})

public class InterviewBitAppConfiguration {}

### **How to disable specific auto-configuration class?**

* You can use the exclude attribute of @EnableAutoConfiguration for this purpose as shown below:

@EnableAutoConfiguration(exclude = {InterviewBitAutoConfiguration.class})

If the class is not specified on the classpath, we can specify the fully qualified name as the value for the excludeName.

//By using "excludeName"

@EnableAutoConfiguration(excludeName={Foo.class})

* You can add into the application.properties and multiple classes can be added by keeping it comma-separated.

### **Can the default web server in the Spring Boot application be disabled?**

### Yes! application.properties is used to configure the web application type, by mentioning spring.main.web-application-type=none.

### **What are the uses of @RequestMapping and @RestController annotations in Spring Boot?**

* **@RequestMapping:**
  + This provides the routing information and informs Spring that any HTTP request matching the URL must be mapped to the respective method.
  + org.springframework.web.bind.annotation.RequestMapping has to be imported to use this annotation.
* **@RestController:**
  + This is applied to a class to mark it as a request handler thereby creating RESTful web services using Spring MVC. This annotation adds the @ResponseBody and @Controller annotation to the class.
  + org.springframework.web.bind.annotation.RestController has to be imported to use this annotation.

Check out more Interview Questions on Spring Boot [here](https://www.interviewbit.com/spring-boot-interview-questions/)

**Example**

@Controller

@RequestMapping("books")

**public** **class** BooksController

{

@RequestMapping(value = "/{name}", method = RequestMethod.GET)

**public** Employee getBooksByName()

{

**return** booksTemplate;

}

}

### **Explain @RestController annotation in Sprint boot?**

### It is a combination of @Controller and @ResponseBody, used for creating a restful controller. It converts the response to JSON or XML. It ensures that data returned by each method will be written straight into the response body instead of returning a template.

### **What is the difference between @RestController and @Controller in Spring Boot?**

### @Controller Map of the model object to view or template and make it human readable but @RestController simply returns the object and object data is directly written in HTTP response as JSON or XML.

### **What are the differences between @RequestParam and @PathVariable annotations?**

Even though both these annotations are used to extract some data from URL, there is a key difference between them.

1. The @RequestParam is used to extract **query parameters** that is anything after “?” in the URL.
2. The @PathVariable is used to extract the data present as part of the URI itself.]
3. For example, if the given URL is

[**http://localhost:8080/InterviewBit/Spring/SpringMVC/?format=json**](http://localhost:8080/InterviewBit/Spring/SpringMVC/?format=json),

then you can access the query parameter “format” using the @RequestParam annotation and **/Spring/{type}** using the @PathVariable, which will give you SpringMVC.

@RequestMapping("/Spring/{type}")

public void getQuestions(@PathVariable("type") String type,

@RequestParam(value = "format", required = false) String format){

/\* Some code \*/

}

### What is Spring Actuator? What are its advantages?

An actuator is an additional feature of Spring that helps you to monitor and manage your application when you push it to production. These actuators include auditing, health, CPU usage, HTTP hits, and metric gathering, and many more that are automatically applied to your application.

### 27. How to enable Actuator in Spring boot application?

To enable the spring actuator feature, we need to add the dependency of “spring-boot-starter-actuator” in pom.xml.

<dependency>

<groupId> org.springframework.boot</groupId>

<artifactId> spring-boot-starter-actuator </artifactId>

</dependency>

### 28. What are the actuator-provided endpoints used for monitoring the Spring boot application?

Actuators provide below pre-defined endpoints to monitor our application -

* Health
* Info
* Beans
* Mappings
* Configprops
* Httptrace
* Heapdump
* Threaddump
* Shutdown

### What is the use of Profiles in spring boot?

While developing the application we deal with multiple environments such as dev, QA, Prod, and each environment requires a different configuration. For eg., we might be using an embedded H2 database for dev but for prod, we might have proprietary Oracle or DB2. Even if DBMS is the same across the environment, the URLs will be different.

### What is the difference between RequestMapping and GetMapping?

RequestMapping can be used with GET, POST, PUT, and many other request methods using the method attribute on the annotation. Whereas getMapping is only an extension of RequestMapping which helps you to improve on clarity on request.

**Delete Operation using Custom Query**

We can also write our custom query to delete the records. @Query is annotation using that we can write our custom query to execute the operation in the **database but when we want to perform delete or update operation at the time @Modifying must be required otherwise it will generate exception call “org.hibernate.hql.internal.QueryExecutionRequestException: Not supported for DML operations“. Means that for Data Manipulation Operations we need to mark  @Modifying at the method level. Let’s see some examples so that we get more idea:**

**package** *com.javadeveloperzone.dao*;

**import** *com.javadeveloperzone.model.Employee*;

**import** *org.springframework.data.jpa.repository.Modifying*;

**import** *org.springframework.data.jpa.repository.Query*;

**import** *org.springframework.data.repository.CrudRepository*;

**import** *org.springframework.data.repository.query.Param*;

**import** *org.springframework.stereotype.Repository*;

**import** *org.springframework.transaction.annotation.Transactional*;

/\*\*

\* Created by Java Developer Zone on 03-08-2017.

\*/

@Repository

@Transactional

**public** **interface** EmployeeRepository **extends** CrudRepository<Employee,**Long**>{

@Modifying // to mark delete or update query

@Query(value = "DELETE FROM Employee e WHERE e.employeeName = :name") // it will delete all the record with specific name

**int** deleteByName(@Param("name") **String** name);

@Modifying

@Query(value = "DELETE FROM Employee WHERE employeeName = :name",nativeQuery = **true**) // if want to write nativequery then mask nativeQuery as true

**int** deleteByNameNative(@Param("name") **String** name);

}

Mappings

Like to handle the HTTP Get requests and get the data only use **@GetMapping**

To perform add/update operation, use HTTP POST/PUT request i.e. **@PostMapping** **or @PutMapping**

And to perform delete operation, use HTTP Delete request i.e. **@DeleteMapping** annotation.

Update Operation using Custom Query

**package** com.example.eatick.repository;

**import** org.springframework.data.jpa.repository.JpaRepository;

**import** org.springframework.data.jpa.repository.Modifying;

**import** org.springframework.data.jpa.repository.Query;

**import** org.springframework.data.repository.query.Param;

**import** org.springframework.stereotype.Repository;

**import** org.springframework.transaction.annotation.Transactional;

**import** com.example.eatick.entities.Beverage;

@Repository

**@Transactional**

**public** **interface** BeverageRepository **extends** JpaRepository<Beverage,Integer>

{

@Query(value="SELECT \* FROM beverage WHERE beverage\_id= :beverage\_id",nativeQuery=**true**)

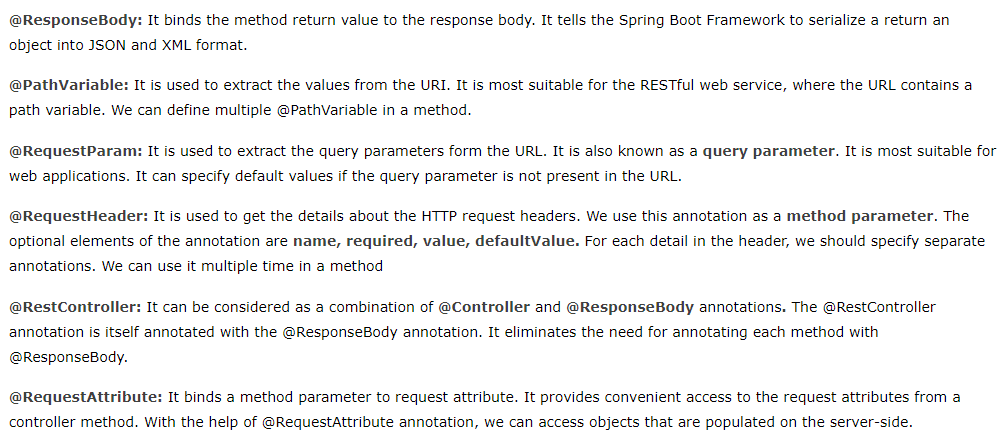
Beverage getById(@Param("beverage\_id") **int** beverageId);

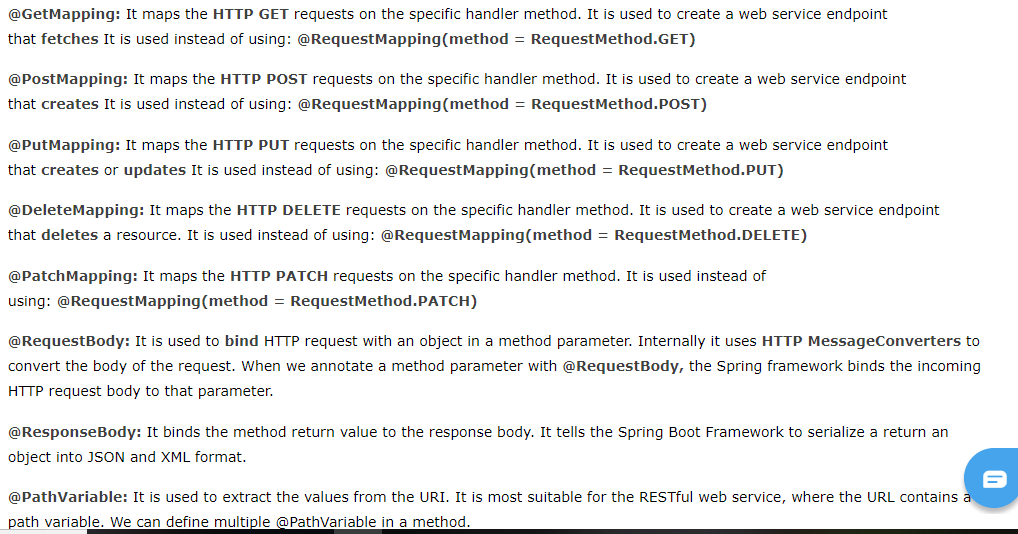
**@Modifying**

@Query(value="UPDATE beverage SET name =:name WHERE beverage\_id= :beverage\_id",nativeQuery=**true**)

**void** update(@Param("beverage\_id") **int** beverageId ,@Param("name")String name);

}

=



Graphical user interface, text

Description automatically generated