**@SpringBootApplication:**

@SpringBootApplication is an annotation in the Spring Framework for building Spring Boot applications

It is a combination of three other annotations: @Configuration, @EnableAutoConfiguration, and @ComponentScan.

* @Configuration: Indicates that a class is a configuration class, which can contain bean definitions and other Spring configuration.
* @EnableAutoConfiguration: Enables Spring Boot's auto-configuration feature, which automatically configures the Spring application based on class path settings, such as adding default implementations for various beans based on the dependencies present in the class path.
* @ComponentScan: Scans the package and its sub-packages for classes annotated with Spring stereotype annotations such as @Component, @Service, @Repository, @Controller, and others.

**@SpringBootConfiguration**

@SpringBootConfiguration is a meta-annotation in Spring Boot that is used to indicate that a class is a configuration class.

It is a combination of two other annotations: @Configuration and @Component.

The @SpringBootConfiguration annotation is used to create configuration classes in Spring Boot applications

**@EnableAutoConfiguration**

@EnableAutoConfiguration is an annotation in Spring Boot that is used to enable the auto-configuration feature.

**@Service**

* @Service is an annotation in Spring Framework that is used to indicate that a class is a service component.
* It makes it easy to register and inject instances of the service into other components.

**@Repository Annotation**

@Repository Annotation is a specialization of **@Component** annotation which is used to indicate that the class provides the mechanism for storage, retrieval, update, delete and search operation on objects.

**@Controller Annotation**

The @Controller annotation indicates that a particular class serves the role of a **controller**. Spring Controller annotation is typically used in combination with annotated handler methods based on the**@RequestMapping** annotation.

* **@Controller:**This is used to specify the controller.
* **@RequestMapping:** This is used to map to the Spring MVC controller method.
* **@ResponseBody:** Used to bind the HTTP response body with a domain object in the return type.

**@RequestParam vs @QueryParam**

* both **@RequestParam** and **@QueryParam** are used to extract data from incoming HTTP requests,
* but **@RequestParam** is used in the Spring MVC framework to extract data from query parameters or form data, It is typically used when you want to extract specific parameters from the request, rather than processing the entire request body.
* while **@QueryParam** is used in JAX-RS to extract data from the query parameters of a URI.

**@PathVariable vs @PathParam**

* **@PathVariable** is used in Spring MVC to extract dynamic parameters from a URL path,
* while **@PathParam** is used in JAX-RS (Java API for RESTful Web Services) to extract dynamic parameters from a URL path.

**HTTP Request**

An HTTP (Hypertext Transfer Protocol) request is a message sent by a client to a server to request information or to perform an action. It typically consists of a request line, headers, and an optional message body.

The request line contains the HTTP method (e.g. GET, POST, PUT, DELETE), the requested URL, and the HTTP version being used

**i) GET**

This method retrieves information from the given server using a given URI. GET request can retrieve the data. It cannot apply other effects on the data.

**ii) POST**

The POST request sends the data to the server. For example, file upload, customer information, etc. using the HTML forms.

**iii) PUT**

The PUT method is used to replace all the current representations of the target resource with the uploaded content.

**iv)** **DELETE**

The DELETE method is used to remove all the current representations of the target resource, which is given by URI.

**Difference between PUT and PATCH**

| **PUT** | **PATCH** |
| --- | --- |
| PUT is a method of modifying resource where the client sends data that updates the entire resource . | PATCH is a method of modifying resources where the client sends partial data that is to be updated without modifying the entire data. |
| HTTP PUT is said to be idempotent, So if you send retry a request multiple times, that should be equivalent to a single request modification | HTTP PATCH is basically said to be non-idempotent. So if you retry the request N times, you will end up having N resources with N different URIs created on the server. |

**Difference Between @Controller and @RestController Annotation**

| **@Controller** | **@RestController** |
| --- | --- |
| The **@Controller** annotation indicates that a particular class serves the role of a **controller**.  Spring Controller annotation is typically used in combination with annotated handler methods based on the[@RequestMapping](https://www.geeksforgeeks.org/spring-requestmapping-annotation-with-example/) annotation. | **Rest Controller** is used for making restful web services with the help of the @RestController annotation. This annotation is used at the class level and allows the class to handle the requests made by the client. |
| In @Controller, we need to use @ResponseBody on every handler method. | In @RestController, we don’t need to use @ResponseBody on every handler method. |

**@RequestBody and ResponseBody**

**@RequestBody** and **@ResponseBody** are annotations in the Spring Framework used to handle the request and response of HTTP requests made to a Spring MVC controller.

**@RequestBody** is used to indicate that the request payload should be mapped to a specific object type. It is used to extract the request data and bind it to the method argument. This annotation is typically used with the POST, PUT and PATCH requests to extract the data sent in the request body

Example:

**@PostMapping("/users")**

**public ResponseEntity<User> addUser(@RequestBody User user) {**

**// add user to database and return response**

**}**

Here, the **@RequestBody** annotation is used to map the request body to a **User** object type, which can be used to add a new user to the database.

**@ResponseBody** is used to indicate that the return value of the method should be serialized into the response body. It is used to return data to the client as part of the response body. This annotation is typically used with the GET requests to return the data requested by the client.

Example:

**@GetMapping("/users/{id}")**

**@ResponseBody**

**public User getUser(@PathVariable Long id) {**

**// get user from database and return**

**}**

Here, the **@ResponseBody** annotation is used to serialize the returned **User** object and send it as the response body to the client.