spring Boot with AWS services

TechWithTony

1. **Spring Boot CRUD Operations / HTTP Methods in Spring RESTful Services**
2. **POST. (Create)**
3. **GET (Read)**
4. **PUT. (Update)**
5. **DELETE. (Delete)**
6. **POST (Create):**
7. To Add New data into DB or any storage service
8. Most common response for the http method are 201 create , 409 conflict i.e. resource already exist .
9. This can be used to update the entity but most proper http method for update is PUT.
10. We should annotate out method in the controller as @PostMapping
11. **GET (Read):**

1. The GET method used to retrieve information from the REST API and not to modify it in any way.
2. If the resource is found on the server then it must return HTTP response code 200 (OK) – along with response body (can be XML, JSON or (possible) RAW)
3. In case resource is NOT found on server then it must return 404 (Not Found).
4. In case that GET request itself is not correctly formed then server will return 400 (Bad Request)
5. We should annotate out method in the controller as @GetMapping
6. **PUT (Update) :**
7. The **PUT** method is used to update existing resource.
8. **PUT** method is idempotent, client can send the same request multiple time, and it will have same result as sending once.
9. In case of successful update then use HTTP response code 200 (OK) or 204 (No Content). 204 (No Content) normally is use if the action has been performed but the response does not include an entity
10. If a new resource created by **PUT** request, the response should be 201 (Created)
11. Should annotate with @PutMapping
12. **DELETE. (Delete) :**
13. **DELETE** method are used to delete a resource identified by the request URI.
14. A successful deletion should return HTTP response code 200 (OK) or 204 (No Content). DELETE can be a long-running or asynchronous request, so return 202 (Accepted) if the action has been queued.
15. In case that the resource doesn't exist, return 404 (Not Found)
16. Should annotate with @DeleteMapping
17. **Spring @RequestParam vs @PathVariable Annotations**
18. @**RequestParam**

used to get the request parameters. @RequestParam automatically binds the request parameters to the arguments of your handler method. It also provides auto type conversion for some standard type like int, long, float, string, date etc

**Usage** :

***http://localhost:8080/*mypath*/*myendpoint*/?*argument1*=*techwithTony*&*argument2*=*700**

**@RequestMapping(value = "/mypath/myendpoint")**

**public String sample(**

**@RequestParam(value="argument1", required=true) String argument1,**

**@RequestParam(value=" argument2", required=false) String argument2){**

**...**

**}**

1. **@PathVariable**

*used to pass parameter along with the url, sometimes we need to pass parameters along with the url to get the data. Spring MVC provides support for customizing the URL in order to get data.*

**Usage:**

***http://localhost:8080/*mypath*/*myendpoint*/TechWithTony?* argument1*=arg1&* argument2*=arg2***

***@RequestMapping(value = "/*mypath */*myendpoint*/{*pathvar1*}")***

***public String bookmark(***

***@PathVariable(value="*pathvar1*") String* pathvar1**

***@RequestParam(value="* argument1*", required=true) String* argument1*,***

***@RequestParam(value="* argument2*", required=false) String* argument2*){***

***...***

***}***

1. **@RequestBody ,@ResponseBody Annotations**
2. **@RequestBody**:

The ***@RequestBody****annotation maps the****HttpRequest****body to a transfer or domain object, enabling automatic deserialization of the inbound****HttpRequest****body onto a Java object. By default, the type we annotate with the****@RequestBody****annotation must correspond to the JSON sent from our client-side controller.*

1. ***@ResponseBody****:*

*The****@ResponseBody****annotation tells a controller that the object returned is automatically serialized into* **JSON** *and passed back into the****HttpResponse****object.*

1. ***Setting the ContentType*** *:*

*When we use the @ResponseBody annotation, we're still able to explicitly set the content type that our method returns.*

1. ***Rest Service Validations:***

***Step 1 : Add the*** hibernate-validator ***dependency in pom file***

***Step 2: Annotate the controller class with*** @Validated

***Step 3: Create a class by extends with*** ResponseEntityExceptionHandler

***And annotate with*** @ControllerAdvice

***Commonly used Annotations for Get method RequestParams and PathVaribales***

1. @NotEmpty
2. @Size
3. @Min
4. @Max
5. **@**NotBlank

*Above same annotations we can use for* ***POST*** *and* ***PUT*** *http methods*