**Notes: HTTP Methods in Spring REST Controller**

**🔹 Class Overview**

@RestController

@RequestMapping("/customer")

* @RestController → Marks this class as a REST endpoint.
  + Combines @Controller + @ResponseBody.
  + So every method returns data directly as an HTTP response (no view page).
* @RequestMapping("/customer") → Common base path for all methods.

**🔹 1. GET – Read Data**

@GetMapping("/report")

public ResponseEntity<String> showCustomerReport() {

return new ResponseEntity<>("FROM GET - showCustomerReport()", HttpStatus.OK);

}

✅ **Use Case:**

* When client wants to **read/fetch** data (no modification).
* Example: /customer/report

✅ **HTTP Status:**

* HttpStatus.OK (200) → Request successful, data returned.

✅ **Real-time Example:**

* Get customer details by ID or get all customers list.

**🔹 2. POST – Create New Data**

@PostMapping("/report")

public ResponseEntity<String> registerCustomer() {

return new ResponseEntity<>("FROM POST - registerCustomer()", HttpStatus.CREATED);

}

✅ **Use Case:**

* When client sends data to **create a new record** in DB.
* Example: /customer/report with JSON body (like customer info).

✅ **HTTP Status:**

* HttpStatus.CREATED (201) → Resource successfully created.

✅ **Real-time Example:**

* Register a new customer.
* Submit a new order or payment.

**🔹 3. PUT – Full Update**

@PutMapping("/modify")

public ResponseEntity<String> updateCustomer() {

return new ResponseEntity<>("FROM PUT - updateCustomer()", HttpStatus.OK);

}

✅ **Use Case:**

* To **replace the existing record completely** with new data.
* All fields of resource are updated.

✅ **HTTP Status:**

* HttpStatus.OK (200) → Update successful.

✅ **Real-time Example:**

* Update entire customer details (name, address, phone, etc.).

**🔹 4. PATCH – Partial Update**

@PatchMapping("/pmodify")

public ResponseEntity<String> updateCustomerByNo() {

return new ResponseEntity<>("FROM PATCH - updateCustomerByNo()", HttpStatus.OK);

}

✅ **Use Case:**

* When you want to **update only few fields** instead of the full object.

✅ **HTTP Status:**

* HttpStatus.OK (200)

✅ **Real-time Example:**

* Update only customer’s phone number or email.

**🔹 5. DELETE – Remove Data**

@DeleteMapping("/remove")

public ResponseEntity<String> deleteCustomer() {

return new ResponseEntity<>("FROM DELETE - deleteCustomer()", HttpStatus.NO\_CONTENT);

}

✅ **Use Case:**

* To **delete** a resource from the server.

✅ **HTTP Status:**

* HttpStatus.NO\_CONTENT (204) → Successfully deleted, no body returned.

✅ **Real-time Example:**

* Delete a customer record or cancel a booking.

**🔹 6. HEAD – Headers Only**

@RequestMapping(value = "/head", method = RequestMethod.HEAD)

public ResponseEntity<Void> headCustomer() {

return new ResponseEntity<>(HttpStatus.OK);

}

✅ **Use Case:**

* To **check if a resource exists** without downloading the body.
* Returns only **headers**, no body content.

✅ **HTTP Status:**

* HttpStatus.OK (200)

✅ **Real-time Example:**

* API health checks or validation before fetching large data.

**🔹 7. OPTIONS – Allowed Methods**

@RequestMapping(value = "/options", method = RequestMethod.OPTIONS)

public ResponseEntity<Void> optionsCustomer() {

return ResponseEntity.ok().build();

}

✅ **Use Case:**

* To know **which HTTP methods are allowed** for a specific endpoint.

✅ **HTTP Status:**

* HttpStatus.OK (200) → “Allow” header auto-added by Spring.

✅ **Real-time Example:**

* Used in browsers during **CORS preflight requests**.

**🔹 8. TRACE – Request Debugging**

@RequestMapping(value = "/trace", method = RequestMethod.TRACE)

public ResponseEntity<String> traceCustomer() {

return new ResponseEntity<>("FROM TRACE - traceCustomer()", HttpStatus.OK);

}

✅ **Use Case:**

* Used for **debugging** – the server echoes the request it received.
* Rarely used in production (can be disabled for security).

✅ **HTTP Status:**

* HttpStatus.OK (200)

**🧩 About ResponseEntity**

ResponseEntity<T> → A wrapper to build complete HTTP responses.  
It allows you to set:

* Response **Body** (e.g., message or data)
* **Status Code** (OK, CREATED, etc.)
* **Headers** (optional)

Example:

return new ResponseEntity<>("Success", HttpStatus.OK);

**✅ Summary Table**

| **HTTP Method** | **Purpose** | **Typical Status** | **Example Action** |
| --- | --- | --- | --- |
| GET | Read data | 200 OK | Get customer info |
| POST | Create new | 201 Created | Register new customer |
| PUT | Full update | 200 OK | Update all fields |
| PATCH | Partial update | 200 OK | Update one field |
| DELETE | Delete resource | 204 No Content | Remove record |
| HEAD | Headers only | 200 OK | Check resource availability |
| OPTIONS | Supported methods | 200 OK | CORS preflight check |
| TRACE | Debug request | 200 OK | Request echo (for testing) |

**📘 Notes: When to Bind Method to Http GET**

**✅ When to Use GET**

* **Use GET when client wants to *read/fetch* data (not modify it).**
* Data flows **from client → server** only for identification/filtering, but the server does not change any data.

**Examples:**

1. Get **Product data** by id.  
   GET /products?id=101
2. Get **Book information** by isbn.  
   GET /books?isbn=12345
3. Get **Employee details** by id.  
   GET /employees?id=501

**✅ Passing Data in GET Requests**

To pass data with a GET request → we use **Query Parameters**.

**Example URL:**

http://localhost:9999/welcome/msg?id=10&name=sachin

Here:

* id = 10
* name = sachin
* Parameters separated by &

Eg: GetMethod

**✅ Important Notes**

1. **Data type conversion**
   * All query parameters arrive as **String** in HTTP.
   * Spring’s **DispatcherServlet** + **Data Binder** automatically convert them into target types (Integer, Double, etc.) before your method executes.
2. **Optional Parameters**
   * Use required = false in @RequestParam if a query param is **not mandatory**.
3. **Multiple Parameters**
   * You can send multiple parameters joined by &.  
     Example:
4. /welcome/msg?id=10&name=sachin&city=mumbai

**✅ Summary**

* **Bind method to GET** → when the client only wants to **fetch/read data**.
* **Use @RequestParam** → to extract query parameters from URL.
* Data always arrives as String, converted automatically by Spring.
* Can make parameters **mandatory or optional** using required=true/false.
* Multiple parameters supported using &.

Eg: GetMethod

**✅ Important Notes**

1. **Data type conversion**
   * All query parameters arrive as **String** in HTTP.
   * Spring’s **DispatcherServlet** + **Data Binder** automatically convert them into target types (Integer, Double, etc.) before your method executes.
2. **Optional Parameters**
   * Use required = false in @RequestParam if a query param is **not mandatory**.
3. **Multiple Parameters**
   * You can send multiple parameters joined by &.  
     Example:
4. /welcome/msg?id=10&name=sachin&city=mumbai

**✅ Summary**

* **Bind method to GET** → when the client only wants to **fetch/read data**.
* **Use @RequestParam** → to extract query parameters from URL.
* Data always arrives as String, converted automatically by Spring.
* Can make parameters **mandatory or optional** using required=true/false.
* Multiple parameters supported using &.

**Path Parameters in Spring REST**

**🔹 What are Path Parameters?**

* Path Parameters (also called **URI parameters**) are **part of the URL path itself**.
* They are used to **send data from the client to the server** as a part of the resource path.
* They usually represent **unique identifiers** (like ID, name, code, etc.) for resources.

**🧭 Example URL**

www.ineuron.in/course/{trainer}/{courseName}

✅ Example filled URL:

www.ineuron.in/course/nitin/springboot

Here:

* {trainer} → nitin
* {courseName} → springboot

**🔹 Where can Path Parameters be used?**

They can appear **anywhere** in the URL path — not just at the end.

✅ Examples:

| **Example URL** | **Description** |
| --- | --- |
| /course/{id} | Parameter at end |
| /course/{id}/details | Parameter in middle |
| /trainer/{name}/course/{courseName} | Multiple parameters |

**🔹 How to Capture Path Parameters in Spring**

Use the @PathVariable annotation in the controller method.

**Example:**

@RestController

@RequestMapping("/welcome")

public class WelcomeController {

// URL: http://localhost:9999/welcome/msg/10/sachin

@GetMapping("/msg/{id}/{name}")

public ResponseEntity<String> greetMessage(

@PathVariable Integer id,

@PathVariable String name) {

String msg = "Hello " + name + ", your ID is " + id;

return new ResponseEntity<>(msg, HttpStatus.OK);

}

}

🧠 **What Happens Internally:**

* The values from the URL (10 and sachin) are extracted by Spring.
* They are automatically converted to the correct data type (Integer, String, etc.).
* Spring calls the method with these extracted values.

**🔹 Variations**

1️⃣ **Single Path Parameter**

@GetMapping("/msg/{name}")

public ResponseEntity<String> greet(@PathVariable String name) {

return ResponseEntity.ok("Hello " + name);

}

Eg: PathParamSingleParameter

2️⃣ **Multiple Path Parameters**

@GetMapping("/msg/{id}/{name}")

public ResponseEntity<String> greet(@PathVariable Integer id, @PathVariable String name) {

return ResponseEntity.ok("Hi " + name + ", ID: " + id);

}

Eg: MultiplePathParameters

3️⃣ **Path Parameter in Middle**

@GetMapping("/msg/{id}/course/{course}")

public ResponseEntity<String> show(@PathVariable Integer id, @PathVariable String course) {

return ResponseEntity.ok("ID: " + id + ", Course: " + course);

}

Eg: PathParamInTheMiddleOfUrl

**🔹 Difference between @RequestParam and @PathVariable**

| **Feature** | **@RequestParam** | **@PathVariable** |
| --- | --- | --- |
| Data Position | Sent as **query parameter** (after ?) | Sent as part of the **URL path** |
| Example URL | /student?id=10&name=sachin | /student/10/sachin |
| Syntax | @RequestParam("id") | @PathVariable("id") |
| Optional Parameter | Can be made optional using required=false | Should match exactly with URL pattern |
| Used For | Filtering, searching, optional data | Identifying specific resource (ID, name, etc.) |
| HTTP Verb | Mostly used with GET | Used with GET, PUT, DELETE |
| Source Framework | From **Spring Framework** | From **JAX-RS (Java API for RESTful Services)** but supported by Spring |

**🔹 Real-Time Use Cases**

| **Operation** | **Typical HTTP** | **Parameter Type** | **Example** |
| --- | --- | --- | --- |
| Get customer by ID | GET | Path | /customer/101 |
| Delete employee by ID | DELETE | Path | /employee/12 |
| Get product by ID | GET | Path | /product/4002 |
| Search employee by name | GET | Query | /employee/search?name=sachin |
| Filter by department and salary | GET | Query | /employee/filter?dept=IT&minSalary=50000 |

**🔹 Extra Note:**

* @PathVariable is inspired by **JAX-RS** (Java API for RESTful Web Services) where it was originally called @PathParam.
* In Spring, @PathVariable is the **Spring equivalent** and is preferred.

**✅ Summary**

| **Concept** | **Description** | **Example** |
| --- | --- | --- |
| Path Parameter | Data sent inside URL path | /students/101 |
| Query Parameter | Data sent after “?” | /students?id=101 |
| Annotation | @PathVariable |  |
| Common Use | Identify resources | /user/{id} |
| Optional Support | Not directly optional | Must match URL pattern |

## 🔹 Difference between **Query Parameter** and **Path Parameter**

| **Feature** | **Query Parameter** | **Path Parameter** |
| --- | --- | --- |
| **Definition** | Sends data from client to server as **key-value pairs** in the **URL**. | Sends data from client to server as **part of the URL path**. |
| **Format** | Appears **after “?”**, multiple params separated by “&”. | Appears as a **variable inside the URL path** — written in curly braces {}. |
| **Example URL** | /course/info?name=navin&courseId=10 | /course/info/10/navin |
| **Annotation Used** | @RequestParam | @PathVariable |
| **Position** | Always at the **end** of the URL. | Can appear **anywhere** in the URL path. |
| **Data Representation** | key=value format. | Just value (no key). |
| **Optional?** | Yes (can set required=false). | No (always required). |
| **Used for?** | Filtering, searching, sorting, pagination, etc. | Identifying specific resources (like /student/101, /order/55). |

## 🔹 Example 1 – Using **Query Parameter**

@RestController

@RequestMapping("/course")

public class CourseController {

// URL: http://localhost:9999/course/info?name=Navin&courseId=101

@GetMapping("/info")

public ResponseEntity<String> getCourseInfo(

@RequestParam String name,

@RequestParam Integer courseId) {

return ResponseEntity.ok("Course Info - Name: " + name + ", ID: " + courseId);

}

}

✅ **Output:**  
Course Info - Name: Navin, ID: 101

## 🔹 Example 2 – Using **Path Parameter**

@RestController

@RequestMapping("/course")

public class CourseController {

// URL: http://localhost:9999/course/info/101/Navin

@GetMapping("/info/{id}/{name}")

public ResponseEntity<String> getCourseDetails(

@PathVariable Integer id,

@PathVariable String name) {

return ResponseEntity.ok("Course ID: " + id + ", Name: " + name);

}

}

✅ **Output:**  
Course ID: 101, Name: Navin

## ⚠️ Common Mistake Examples

### ❌ Case 1:

@GetMapping("/info/{id}/JRTP/{name}")

// URL: /course/info/5/JRTP/navinreddy/telusko/microservices

👉 **Error:** 404 Not Found  
Because your URL has **extra segments** that are not defined in the mapping.

### ❌ Case 2:

@GetMapping("/info/{id}/JRTP/{name}")

public ResponseEntity<String> getCourseDetails(

@PathVariable Integer id,

@PathVariable String name) { ... }

// URL: /course/info/navinReddy/JRTP/5

👉 **Error:** MethodArgumentTypeMismatchException  
Because Spring tried to assign "navinReddy" (a string) to Integer id.

### ✅ Case 3 (Corrected Type):

@GetMapping("/info/{id}/JRTP/{name}")

public ResponseEntity<String> getCourseDetails(

@PathVariable String id,

@PathVariable String name) { ... }

// URL: /course/info/navinReddy/JRTP/5

✅ Works fine — both are treated as strings.

## 💡 Tip for Real Projects:

| **Situation** | **Recommended Parameter** |
| --- | --- |
| Fetch a **specific resource** (like /student/101) | **PathVariable** |
| Send **filtering or optional** information (like ?sort=asc&page=2) | **RequestParam** |