**Mapping on Spring Boot**

Mapping in Spring Boot refers to how HTTP requests are mapped to handler methods in your controller classes. It is a crucial part of building RESTful web services in Spring Boot. Here’s an overview of key concepts and annotations used in Spring Boot mapping:

**1. Request Mapping Annotations**

**- @RequestMapping**: A versatile annotation used to map HTTP requests to specific handler methods or classes. It can be applied at the class level to define a base path for all methods in the controller and at the method level to handle specific requests.

@RestController

@RequestMapping("/employees")

public class EmployeeController {

@RequestMapping(method = RequestMethod.GET)

public List<Employee> getAllEmployees() {

// logic to get all employees

}

}

**- @GetMapping**: A shortcut for `@RequestMapping(method = RequestMethod.GET)`. It’s used to handle GET requests.

@GetMapping("/{id}")

public Employee getEmployeeById(@PathVariable Long id) {

// logic to get employee by id

}

**- @PostMapping**: A shortcut for `@RequestMapping(method = RequestMethod.POST)`. It’s used to handle POST requests.

@PostMapping

public Employee createEmployee(@RequestBody Employee employee) {

// logic to create an employee

}

**- @PutMapping:** A shortcut for `@RequestMapping(method = RequestMethod.PUT)`. It’s used to handle PUT requests.

@PutMapping("/{id}")

public Employee updateEmployee(@PathVariable Long id, @RequestBody Employee employee) {

// logic to update an employee

}

**- @DeleteMapping**: A shortcut for `@RequestMapping(method = RequestMethod.DELETE)`. It’s used to handle DELETE requests.

@DeleteMapping("/{id}")

public void deleteEmployee(@PathVariable Long id) {

// logic to delete an employee

}

**2. Mapping Path Variables**

**- @PathVariable:** Binds a method parameter to a URI template variable.

@GetMapping("/{id}")

public Employee getEmployeeById(@PathVariable("id") Long id) {

// logic to get employee by id

}

**3. Mapping Query Parameters**

**- @RequestParam**: Binds a method parameter to a web request parameter.

@GetMapping("/search")

public List<Employee> searchEmployees(@RequestParam("name") String name) {

// logic to search employees by name

}

**4. Mapping Request Bodies**

**- @RequestBody**: Maps the HTTP request body to a method parameter.

@PostMapping

public Employee createEmployee(@RequestBody Employee employee) {

// logic to create an employee

}

**5. Mapping to Response Bodies**

**- @ResponseBody**: Indicates that the return value of a method should be written directly to the HTTP response body.

**- @RestController**: A convenience annotation that combines `@Controller` and `@ResponseBody`. It’s used for creating RESTful web services.

**6. Mapping with Headers**

- You can specify conditions based on headers using `headers` attribute in `@RequestMapping` or its shortcuts.

@GetMapping(value = "/header", headers = "key=value")

public String getByHeader() {

// logic based on header value

}

**7. Mapping with Produces and Consumes**

**- produces**: Specifies the media types that the method can produce (like JSON, XML).

**- consumes**: Specifies the media types that the method can consume.

@PostMapping(value = "/json", consumes = "application/json", produces = "application/json")

public Employee createEmployeeJson(@RequestBody Employee employee) {

// logic to create an employee with JSON request and response

}

**8. Mapping with Exception Handling**

**- @ExceptionHandler:** Maps specific exceptions to handler methods within a controller.

@ExceptionHandler(EmployeeNotFoundException.class)

public ResponseEntity<String> handleNotFound(EmployeeNotFoundException ex) {

return new ResponseEntity<>(ex.getMessage(), HttpStatus.NOT\_FOUND);

}

**9. Mapping Class-Level Endpoints**

- Applying `@RequestMapping` at the class level to set a base URL for all endpoints in the controller.

@RestController

@RequestMapping("/api/employees")

public class EmployeeController {

// All methods will have the base path /api/employees

}

**10. Mapping with Custom Annotations**

- You can create custom annotations by combining multiple Spring annotations to simplify mapping.

By understanding and using these mapping techniques in Spring Boot, we can efficiently build and manage RESTful APIs, handling everything from simple GET requests to complex operations involving multiple parameters and request types.