SpringBoot Use cases - 2024

How to properly Handle Exception in SpringBoot Controller

Case-1: **If the employee id is null or 0, it should display bad request**.

@PutMapping(value = "/v3/emp")

**public** ResponseEntity<Employee> getEmpDetail(@RequestBody Employee emp) {

System.***out***.println("Employee-------->"+emp);

**if**(emp.getId() == **null** || emp.getId().intValue() == 0) {

**throw** **new** BadRequestException("Invalid Employee Id");

}

Employee employee = **new** Employee(123, "John Abrahm", "Permanent", **new** String[] {"Admin"});

**return** ResponseEntity.*ok*().body(employee);

}

@ControllerAdvice

**public** **class** GlobalExceptionHandler {

@ExceptionHandler(BadRequestException.**class**)

**public** ResponseEntity<AppError> handleBadRequest(RuntimeException re) {

System.***out***.println("What is the details : "+re.getMessage());

AppError appError = **new** AppError(re.getMessage(), "Provide proper valid id", "HttpStatus.BAD\_REQUEST.name()");

**return** **new** **ResponseEntity<>(appError, HttpStatus.*BAD\_REQUEST*);**

}

}

Case-2: **If the employee id is negative, it should display no such employee**

Incoming Request is like this

{

"id": -123,

"name": "John Abraham",

"status": "",

"roles":["admin","customer"]

}

@PutMapping(value = "/v4/emp")

**public** ResponseEntity<Employee> getMoreEmpInfo(@RequestBody Employee emp) {

Employee employee = **null**;

System.***out***.println("Employee-------->" + emp);

**if** (emp.getId() == **null** || emp.getId().intValue() <= 0 ) {

**throw** **new** NoSuchEmployeeException("No such employee exists with this id");

}

employee = **new** Employee(123, "John Abrahm", "Permanent", **new** String[] {"Admin"});

**return** ResponseEntity.*ok*().body(employee);

}

@ControllerAdvice

**public** **class** GlobalExceptionHandler {

@ExceptionHandler(NoSuchEmployeeException.**class**)

**public** ResponseEntity<AppError> handleNoTFound(NoSuchEmployeeException re) {

System.***out***.println("What is the details : "+re.getMessage());

AppError appError = **new** AppError(re.getMessage(), "Provide proper valid id", HttpStatus.***NOT\_FOUND***.name());

**return** **new** **ResponseEntity<>(appError, HttpStatus.*NOT\_FOUND*);**

}

}

Case-3: In case of very general exception, provide RuntimeException in ControllerAdvice class.

Incoming Request:

{

"id": 123,

"name": "John Abraham",

"status": "",

"roles":[]

}

@PutMapping(value = "/v5/emp")

**public** ResponseEntity<Employee> getMoreEmpDetails(@RequestBody Employee emp) {

Employee employee = **null**;

System.***out***.println("Employee-------->" + emp);

String role1 = emp.getRoles()[1];

System.***out***.println("Role1---->"+role1);

employee = **new** Employee(123, "John Abrahm", "Permanent", **new** String[] {"Admin"});

**return** ResponseEntity.*ok*().body(employee);

}

**To handle any type of RuntimeException**

@ControllerAdvice

**public** **class** GlobalExceptionHandler {

@ExceptionHandler(RuntimeException.**class**)

**public** ResponseEntity<AppError> handleGeneralError(RuntimeException re) {

System.***out***.println("What is the details : "+re.getMessage());

AppError appError = **new** AppError(re.getMessage(), "Highly Unexpected Exception", HttpStatus.***INTERNAL\_SERVER\_ERROR***.name());

**return** **new** ResponseEntity<>(appError, HttpStatus.***INTERNAL\_SERVER\_ERROR***);

}

}

AppError class is given below

@Data @AllArgsConstructor @NoArgsConstructor

**public** **class** AppError {

**private** String errMessage;

**private** String info;

**private** String code;

}

Other Exception classes

**public** **class** NoSuchEmployeeException **extends** RuntimeException {

**public** NoSuchEmployeeException(String message) {

**super**(message);

}

}

**public** **class** BadRequestException **extends** RuntimeException {

**private** **static** **final** **long** ***serialVersionUID*** = -1337476553418583443L;

**public** BadRequestException(String message) {

**super**(message);

}

}

**How to make Database Search using Case Insensitive using Spring JPA**

Repository class is given below

@Repository

**public** **interface** UserRepository **extends** CrudRepository<Customer, Long> {

List<Customer> findByFirstName(String firstName); // Exact Search

List<Customer> **findByFirstNameIgnoreCase(String firstName);** // Case Insensitive Search

}

Code in AutoRun class

**public** **void** searchUserByName() {

String name = "ram";

// List<Customer> customerList = userRepo.findByFirstName(name); // Provides empty result set

List<Customer> customerList = userRepo.findByFirstNameIgnoreCase(name); // Provides result

**if**(customerList.isEmpty()) {

System.***out***.println("No such customers ...");

} **else** {

System.***out***.println("List of Customers--->"+customerList);

}

}

**Is it necessary to declare @Param("name") String name in Spring JPA**

In the higher version of SpringBoot and JPA, it is not required to declare @Param("name") String name. An example is given below.

@Repository

**public** **interface** UserRepository **extends** CrudRepository<Customer, Long> {

@Query("select c from Customer c where c.firstName like %:name%")

List<Customer> searchByFirstName(@Param("name") String name);

**@Query("select c from Customer c where c.firstName like %:name%")**

**List<Customer> searchByFirstNameWithoutParam(String name);**

}

In the above case, both will work.

**What is the alternative to Like Operator in Spring JPA**

@Repository

**public** **interface** UserRepository **extends** CrudRepository<Customer, Long> {

List<Customer> **findByFirstNameContaining**(String firstName); 🡸 case **sensitive** like

List<Customer> **findByFirstNameContainingIgnoreCase**(String firstName); 🡸 case **Insensitive** like

}