**📘 Global Exception Handling in Spring Boot**

**🧾 Overview**

This document explains the **centralized exception handling mechanism** implemented in our Spring Boot application. It ensures that all runtime errors are handled uniformly, logged properly, and returned to the client in a consistent structure.

**📌 Objectives**

* Centralize all exception responses.
* Ensure proper logging with traceability.
* Return user-friendly and developer-informative error responses.
* Avoid boilerplate try-catch in every controller.
* Support custom business exceptions (like record not found, duplicate, etc.).
* **🔧 Key Components**

| **Component** | **Description** |
| --- | --- |
| GlobalExceptionHandler | A @ControllerAdvice class that catches and processes exceptions globally. |
| ApiCustomException | Custom exception for application-specific errors. |
| ErrorMessage | POJO representing the error response. |
| ValidationDetails | Encapsulates field-specific error details. |
| ErrorConstants | Central store of error code keys. |
| ErrorHandlingUtil | Utility class to fetch messages and build validations. |

**📁 Exception Flow Diagram**

[Request] → [Controller] → [Exception Thrown] → [GlobalExceptionHandler] → [ErrorMessage Response]

🚨 Exception Types Covered

| **Exception Type** | **HTTP Status** | **Handler Method** | **Use Case Example** |
| --- | --- | --- | --- |
| RuntimeException | 500 Internal Error | handleAnyException | Unexpected backend failure |
| HttpRequestMethodNotSupportedException | 405 Not Allowed | handleMethodTypeException | GET instead of POST |
| HttpMediaTypeNotSupportedException | 415 Unsupported | handleMediaTypeException | Missing/invalid content-type |
| MethodArgumentNotValidException | 400 Bad Request | handleMethodArgumentNotValidException | Invalid input validation |
| ApiCustomException | Custom (e.g. 404) | handleApiCustomException | Record not found, duplicate, etc. |

🔧 Sample Error Response Format

{

"refId": "2ac2e320-abc1-4e2a-9a2e-9ab1c721c86a",

"code": "ERR\_404",

"message": "No record found for the given ID.",

"status": 404,

"severity": "ERROR",

"errorDetails": [

{

"field": "recordId",

"message": "Record not found for ID: 99"

}

]

}

🔍 Example Controller Use Cases

🔴 Generic 500 Error

@GetMapping("/generic")

public String throwGenericException() {

throw new RuntimeException("Generic internal error occurred.");

}

Add more methods

🧱 Centralized Handler Class

@ControllerAdvice

@Slf4j

public class GlobalExceptionHandler {

@ExceptionHandler(ApiCustomException.class)

public ResponseEntity<ErrorMessage> handleApiCustomException(ApiCustomException exception) {

List<ValidationDetails> reasons = exception.validations().isEmpty() ? null : exception.validations();

return buildErrorResponse(exception.httpStatus(), exception.code(), "Some message", reasons);

}

@ExceptionHandler(Exception.class)

public ResponseEntity<ErrorMessage> handleAnyException(Exception ex) {

return buildErrorResponse(HttpStatus.INTERNAL\_SERVER\_ERROR, "ERR\_500", "Unexpected error", null);

}

// ... other handlers

}

**Fields Explained**

| **Field** | **Description** |
| --- | --- |
| **refId** | **UUID for correlation (also used as traceId)** |
| **code** | **Application-specific error code** |
| **message** | **User-readable error message** |
| **severity** | **Hardcoded as "ERROR"** |
| **status** | **Corresponding HTTP status code** |
| **errorDetails** | **Optional list of validation failure details** |

**✅ Benefits**

* 📦 Cleaner controller code.
* 🧠 Easy debugging with structured logs.
* 🔄 Consistent error response for all clients (mobile/web).
* 📈 Extensible for new custom exceptions.

🚨 Centralized Exception Handling & Logging Framework

**Overview**

This document outlines the implementation of a centralized exception handling and logging strategy using **Spring Boot**, **SLF4J**, and **Logback**. The goal is to ensure all errors and runtime exceptions are uniformly captured, logged with traceability, and returned as structured responses to the client.

**📌 Goals**

* Consistent and human-readable logs with color-coded formatting
* Unique traceability for every request using traceId
* Structured and centralized error responses using ErrorMessage
* Dynamic project name injection into log pattern
* Support for custom business exceptions (ApiCustomException)
* Easily extensible for microservice environments

**🔧 Logging Setup**

**1. SLF4J + Logback Configuration**

We use **SLF4J** as the logging façade with **Logback** as the implementation.

encoder.setPattern(

"%cyan([%d{yyyy-MM-dd HH:mm:ss.SSS}]) " +

"%highlight(%-5level) " +

"[traceId=%X{traceId}] " +

"%blue([" + projectName + "]) " +

"%cyan(%file -> %class.%M) " +

":%line " +

"%yellow(errorCode=%X{errorCode}) " +

"%msg%n"

);

**🖍️ Log Color Scheme**

| **Log Level** | **Color** |
| --- | --- |
| ERROR | 🔴 Red |
| WARN | 🟠 Yellow |
| INFO | 🟢 Green |
| DEBUG | 🔵 Blue |
| TRACE | ⚪ Grey |

**🔁 Dynamic Project Name**

Project name is dynamically derived from the **current working directory name**, avoiding hardcoding.

String projectName = new File(System.getProperty("user.dir")).getName();

**🧵 Trace ID Generation**

A unique traceId is generated for each request and added to **MDC (Mapped Diagnostic Context)** using a Filter.

@Component

public class TraceIdFilter implements Filter {

public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain) {

try {

MDC.put("traceId", UUID.randomUUID().toString());

chain.doFilter(req, res);

} finally {

MDC.clear();

}

}

}

This traceId is included in both logs and error responses.

**📉 Benefits of This Setup**

✅ Unified logging across modules  
✅ Easy debugging with traceId  
✅ Highly structured and user-friendly error responses  
✅ Extensible for microservices or cloud-native apps  
✅ Environment-agnostic setup

┌─────────────┐

│ Http Request│

└──────┬──────┘

↓

┌────────────────────┐

│ TraceIdFilter │ ← Generates unique traceId

└────────────────────┘

↓

┌────────────────────┐

│ Controller Layer │

└────────────────────┘

↓

┌──────────────────────────┐

│ GlobalExceptionHandler │

└──────────────────────────┘

↓

┌──────────────────────────┐

│ ErrorMessage + Response │

└──────────────────────────┘

↓

┌──────────────────────────┐

│ MDC + SLF4J + Logback │ ← Color-coded, structured logs

└──────────────────────────┘

📚 Example Log Entry

[2025-07-23 19:00:03.914] INFO [traceId=3d9f...3c8b] [MyAwesomeApp] GlobalExceptionHandler -> handleAnyException :163 errorCode=ERROR\_MAPPING\_NOT\_FOUND Requested endpoint does not exist