perplexity

17. How do you handle exception handling in your application? Tell me the best practices to handle exceptions for your APIs. explain to me e2e flow for exception handling in your application.

This is a classic "architectural maturity" question. The interviewer wants to know if you just use `try-catch` everywhere or if you have a **centralized, consistent strategy**.

Here is the answer tailored for a Senior/Lead Developer role (Spring Boot context).

1. High-Level Strategy (The "Best Practices")

"In my application, I follow a **Global Centralized Exception Handling** strategy using Spring Boot's `@ControllerAdvice`.
My best practices are:

1. **Never return raw Stack Traces** to the client (Security risk).
2. **Standardize Error Responses:** Every error (400, 404, 500) returns the exact same JSON structure (`errorCode`, `errorMessage`, `timestamp`).
3. **Categorize Exceptions:** I separate **Business Exceptions** (e.g., `InsufficientFundsException`) from **Technical Exceptions** (e.g., `DatabaseConnectionTimeout`).
4. **Log everything:** All exceptions are logged with a correlation ID/Trace ID so we can debug them in Splunk/ELK."

2. End-to-End Flow (The "Story")

"Let's take a 'Create Order' API flow to explain how an exception bubbles up."

Step 1: The Trigger (Service Layer)
"Imagine the `OrderService` tries to save an order but finds the product is out of stock.
Instead of returning `null` or `-1`, I throw a custom runtime exception:
`throw new ProductOutOfStockException("Product-123", "Inventory empty");`"

Step 2: The Bubble Up (Controller Layer)
"The `OrderController` does **not** have a try-catch block. We let the exception propagate up. This keeps the controller code clean and focused only on success paths."

Step 3: The Catch (Global Handler)
"I have a class annotated with `@RestControllerAdvice`. It acts as an interceptor for all exceptions

thrown anywhere in the application.
It has a method annotated with `@ExceptionHandler(ProductOutOfStockException.class)`."

Step 4: The Transformation
"Inside that handler method, I:

1. **Log the error:** `log.error("Stock failure: {}", ex.getMessage())`
2. **Map to Response:** Create a standardised `ErrorResponse` DTO.
   - `status`: 422 Unprocessable Entity
   - `code`: "INVENTORY_ERR_001"
   - `message`: "Product-123 is out of stock"
3. **Return:** A `ResponseEntity` wrapping this DTO."

3. The Code Structure (Implementation)

A. Standard Error Response DTO

```
public record ErrorResponse(String errorCode, String message, LocalDateTime timestamp, St
```

B. Global Exception Handler

```
@RestControllerAdvice
public class GlobalExceptionHandler {

    // 1. Handle Custom Business Exception
    @ExceptionHandler(ProductOutOfStockException.class)
    public ResponseEntity<ErrorResponse> handleStockError(ProductOutOfStockException ex)
        ErrorResponse error = new ErrorResponse(
            "STOCK_001",
            ex.getMessage(),
            LocalDateTime.now(),
            MDC.get("traceId") // Correlation ID for debugging
        );
        return new ResponseEntity<>(error, HttpStatus.UNPROCESSABLE_ENTITY);
    }

    // 2. Handle Generic/Unknown Exceptions (Safety Net)
    @ExceptionHandler(Exception.class)
    public ResponseEntity<ErrorResponse> handleGenericError(Exception ex) {
        log.error("Unexpected error occurred", ex); // Log stack trace only here
        ErrorResponse error = new ErrorResponse(
            "GENERIC_500",
            "Something went wrong, please try again",
            LocalDateTime.now(),
            MDC.get("traceId")
        );
        return new ResponseEntity<>(error, HttpStatus.INTERNAL_SERVER_ERROR);
```

```
    }
}
```

This approach ensures the frontend always gets a consistent error format, and the backend logs are clean and traceable.

⁂

1. https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/images/81815274/429036f4-3bc3-4a7a-8c6f-90246e73b9b9/image.jpg
2. https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/images/81815274/24f18c0a-b56e-4862-a619-e2959036a5c2/image.jpg
3. https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/images/81815274/97e9ed86-ed11-4035-a18b-1eb05c1bab4e/image.jpg
4. https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/images/81815274/87f62423-96f2-4071-9802-8f6699e0ecd8/image.jpg