Java Spring Boot: Validation & Exception Handling

Recap & Today's Goal

- **Recap**: Last time we used `RestClient` and external API proxies.
- **Stability**: Moving away from "unsafe" APIs that crash on bad data or missing records.
- **Hardening**: Today is about stopping bad data at the gate and returning standardized errors.

Data Flow & Records

The Modern DTO

- **Input (Request DTO)**: JSON is mapped to Java Records via Jackson using reflection and the canonical constructor.
- **Separation**: Strict decoupling between the public API (DTOs) and the database (Entities).
- **The Record Win**: Zero boilerplate and built-in immutability ensure data cannot be changed after validation.

Entity to DTO Mapping

Clean Architecture

- **Custom Constructors**: Use additional constructors in Response Records to map directly from an Entity.
- **Thin Controllers**: No manual field-by-field mapping in the Controller layer.
- **Safety**: Records discard internal entity state, exposing only what the client needs to see.

Input Validation

Bean Validation Logic

- **Gatekeeper**: Validation annotations live in the DTO to define rules before data hits the Service.
- **Constraints**: Use `@NotBlank`, `@Size`, and `@Email` for standard string and format checks.
- **The Trigger**: You must add the `@Valid` annotation to Controller method parameters to activate these checks.

Global Exception Handling

Standardized Responses

- **ProblemDetail**: Implements RFC 7807 to return a consistent JSON structure (title, status, detail).
- **Global Catch**: `@RestControllerAdvice` acts as a safety net that intercepts exceptions from any Controller.
- **Professional APIs**: Prevents internal stack traces from leaking and provides a predictable format for Frontend developers.

Service Layer Logic

Business Rule Enforcement

- **Format vs. Logic**: Controllers check format; Services check "Business Rules" like duplicate emails or missing parents.
- **Defensive Coding**: Check repository existence (e.g., `existsByEmail`) before performing save actions.
- **Hand-off**: Throw custom exceptions in the Service; let the Global Handler convert them to HTTP statuses (404, 409).

Hands-on & Challenges

Core Exercises

- **Task 1**: Harden DTOs with validation constraints.
- **Task 2**: Implement the `GlobalExceptionHandler` using `ProblemDetail`.
- **Task 3**: Add existence checks and custom exceptions to your Services.

Extra Exercises