Validating API Requests

In this section, we explored how to validate API requests in Spring Boot, ensuring that incoming data is correct, secure, and follows business rules. We covered everything from basic field validation to handling complex business logic.

Jakarta Validation

String Validation

- `@NotBlank` – Ensures a string is not empty and contains at least one non-whitespace character.
- `@NotEmpty` – Ensures a string is not empty ("") but allows whitespace.
- `@Size` – Enforces character length constraints.
- `@Pattern` – Ensures the value matches a given regex pattern (e.g., phone numbers, custom formats).
- `@Email` – Validates email format.

Number Validation

- `@Positive` – Ensures the value is greater than `0`.
- `@PositiveOrZero` – Ensures the value is 0 or greater.
- `@Negative` – Ensures the value is less than `0`.
- `@NegativeOrZero` – Ensures the value is 0 or less.
- `@Min(value)` – Ensures the number is at least value.
- `@Max(value)` – Ensures the number is at most value.

Date/Time Validation

- `@Past` – Ensures the date is in the past.
- `@PastOrPresent` – Ensures the date is in the past or today.
- `@Future` – Ensures the date is in the future.
- `@FutureOrPresent` – Ensures the date is in the future or today.

General Validation

- `@NotNull` – Ensures the value is not null.

Handling Validation Errors

- When a request contains invalid data, Spring throws `MethodArgumentNotValidException`.
- We can catch this in the controller to return structured error messages.

```java
@ExceptionHandler(MethodArgumentNotValidException.class)
public ResponseEntity<Map<String, String>> handleValidationErrors(
    MethodArgumentNotValidException exception
) {
    var errors = new HashMap<String, String>();
    exception.getBindingResult().getFieldErrors().forEach(error ->
        errors.put(error.getField(), error.getDefaultMessage()));

    return ResponseEntity.badRequest().body(errors);
}
```

Global Error Handling

- Instead of handling errors in each controller, we can move validation error handling to a global exception handler using `@ControllerAdvice`.

```java
@ControllerAdvice
public class GlobalExceptionHandler {
    @ExceptionHandler(MethodArgumentNotValidException.class)
    public ResponseEntity<Map<String, String>> handleValidationErrors(
        MethodArgumentNotValidException exception
    ) {
        // validation logic
    }
}
```

Implementing Custom Validation

- Built-in annotations don't always cover every scenario. We can create custom validation annotations when necessary.

```java
@Target(ElementType.FIELD)
@Retention(RetentionPolicy.RUNTIME)
@Constraint(validatedBy = LowercaseValidator.class)
public @interface Lowercase {
    String message() default "must be lowercase";
    Class<?>[] groups() default {};
    Class<? extends Payload>[] payload() default {};
}
```

Validating Business Rules

- Some validation requires database queries (e.g., checking if an email is already taken).
- We don’t use annotations for business rules because they get triggered before checking basic input constraints.
- Instead, we first validate input format using annotations, then check business rules in the controller.

```
if (userRepository.existsByEmail(request.getEmail())) {
    return ResponseEntity.badRequest().body(
            Map.of("email", "Email is already registered.")
    );
}
```