<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-validation</artifactId>

</dependency>

**import** jakarta.validation.constraints.NotNull;

@NotNull(message = "First name cannot be null")

@Size(min = 3, message = "First name must contain at least 3 characters")

private String firstName;

@NotNull(message = "Email cannot be null")

@Email(message = "Email should be valid")

@Column(unique = true)

private String email;

@Size(min = 6, max = 20, message = "Password must be between 6 and 20 characters")

@Pattern(regexp = "^[a-zA-Z0-9]\*$", message = "Password must be alphanumeric")

private String password;

@NotNull(message = "Mobile number cannot be null")

@NotEmpty(message = "Mobile number cannot be empty")

@Pattern(regexp = "^[0-9]{10}$", message = "Mobile number must be a 10-digit number")

@Column(unique = true)

private String mobileNumber;

@NotNull(message = "Salary cannot be null")

@Min(value = 10000, message = "Salary must be at least 10,000")

@Max(value = 50000, message = "Salary must be less than or equal to 50,000")

private Integer salary;

**1. @NotNull**

Ensures that the field is not null.

@NotNull(message = "Name cannot be null")

private String name;

**2. @NotEmpty**

Ensures that the field is not null and not empty (works for String, Collection, Map, and arrays).

@NotEmpty(message = "Username cannot be empty")

private String username;

“ “

**3. @NotBlank**

Ensures that the field is not null, not empty, and contains at least one non-whitespace character (works for String only).

@NotBlank(message = "Email cannot be blank")

private String email;

**4. @Size**

Ensures that the field's length is between a specified minimum and maximum value (works for String, Collection, Map, and arrays).

@Size(min = 5, max = 100, message = "Password must be between 5 and 100 characters")

private String password;

**5. @Min and @Max**

Validates that a numeric field has a value greater than or equal to min and less than or equal to max.

@Min(value = 18, message = "Age must be at least 18")

@Max(value = 65, message = "Age must be less than or equal to 65")

private Integer age;

**6. @Email**

Validates that the field contains a valid email address.

@Email(message = "Please provide a valid email address")

private String email;

**7. @Pattern**

Validates that the field matches a specified regular expression (regex).

@Pattern(regexp = "^[A-Za-z0-9]{5,10}$", message = "Username must be alphanumeric and between 5 to 10 characters long")

private String username;

**8. @Positive e.g age and @PositiveOrZero e.g job experience in years**

Ensures that the field is a positive number (@Positive) or a positive number or zero (@PositiveOrZero).

@Positive(message = "Amount must be positive")

private BigDecimal amount;

@PositiveOrZero(message = "Quantity cannot be negative")

private Integer quantity;

**9. @Negative and @NegativeOrZero**

Ensures that the field is a negative number (@Negative) or a negative number or zero (@NegativeOrZero).

@Negative(message = "Balance must be negative")

private BigDecimal balance;

@NegativeOrZero(message = "Discount cannot be positive")

private Integer discount;

**10. @Future and @Past**

Validates that a date is in the future (@Future) or in the past (@Past).

@Future(message = "Date must be in the future")

private LocalDate eventDate;

@Past(message = "Birthdate must be in the past")

private LocalDate birthDate;

**11. @FutureOrPresent and @PastOrPresent**

Validates that a date is either in the future or present (@FutureOrPresent), or in the past or present (@PastOrPresent).

@FutureOrPresent(message = "Order date cannot be in the past")

private LocalDate orderDate;

@PastOrPresent(message = "Creation date cannot be in the future")

private LocalDate creationDate;

**12. @Valid**

This annotation is used to trigger validation of nested objects or collections. It is often used in combination with @RequestBody in Spring Boot or with entities that contain other entities.

public class Person {

@Valid

private Address address;

// Getters and setters

}

* **Note**: You will need to annotate the nested Address class with validation annotations too if you want it to be validated.

**13. @AssertTrue and @AssertFalse**

Validates that a boolean field is true (@AssertTrue) or false (@AssertFalse).

@AssertTrue(message = "Terms must be accepted")

private Boolean termsAccepted;

@AssertFalse(message = "Terms must not be rejected")

private Boolean termsRejected;

**14. @DecimalMin and @DecimalMax**

These are used to validate decimal (floating-point) values with a minimum or maximum value.

@DecimalMin(value = "10.00", message = "Amount must be greater than or equal to 10.00")

private BigDecimal amount;

@DecimalMax(value = "1000.00", message = "Amount must be less than or equal to 1000.00")

private BigDecimal totalAmount;

**15. @Digits**

Validates that the field contains a valid number with a specified number of integer and fraction digits.

@Digits(integer = 5, fraction = 2, message = "Amount must be a valid number with up to 5 integer digits and 2 decimal places")

private BigDecimal amount;

**16. @NotBlank for Arrays or Collections**

You can also validate non-emptiness for arrays or collections.

@NotBlank(message = "Name cannot be empty")

private List<String> names;

**Example of Using Multiple Validations:**

Here’s how you might use multiple annotations on a single field:

import javax.validation.constraints.\*;

public class User {

@NotNull(message = "Username cannot be null")

@Size(min = 3, max = 20, message = "Username must be between 3 and 20 characters")

private String username;

@Email(message = "Please provide a valid email address")

private String email;

@Min(value = 18, message = "Age must be at least 18")

@Max(value = 100, message = "Age must be less than or equal to 100")

private Integer age;

@Pattern(regexp = "^[A-Za-z0-9]{8,16}$", message = "Password must be alphanumeric and between 8 to 16 characters long")

private String password;

// Getters and Setters

}