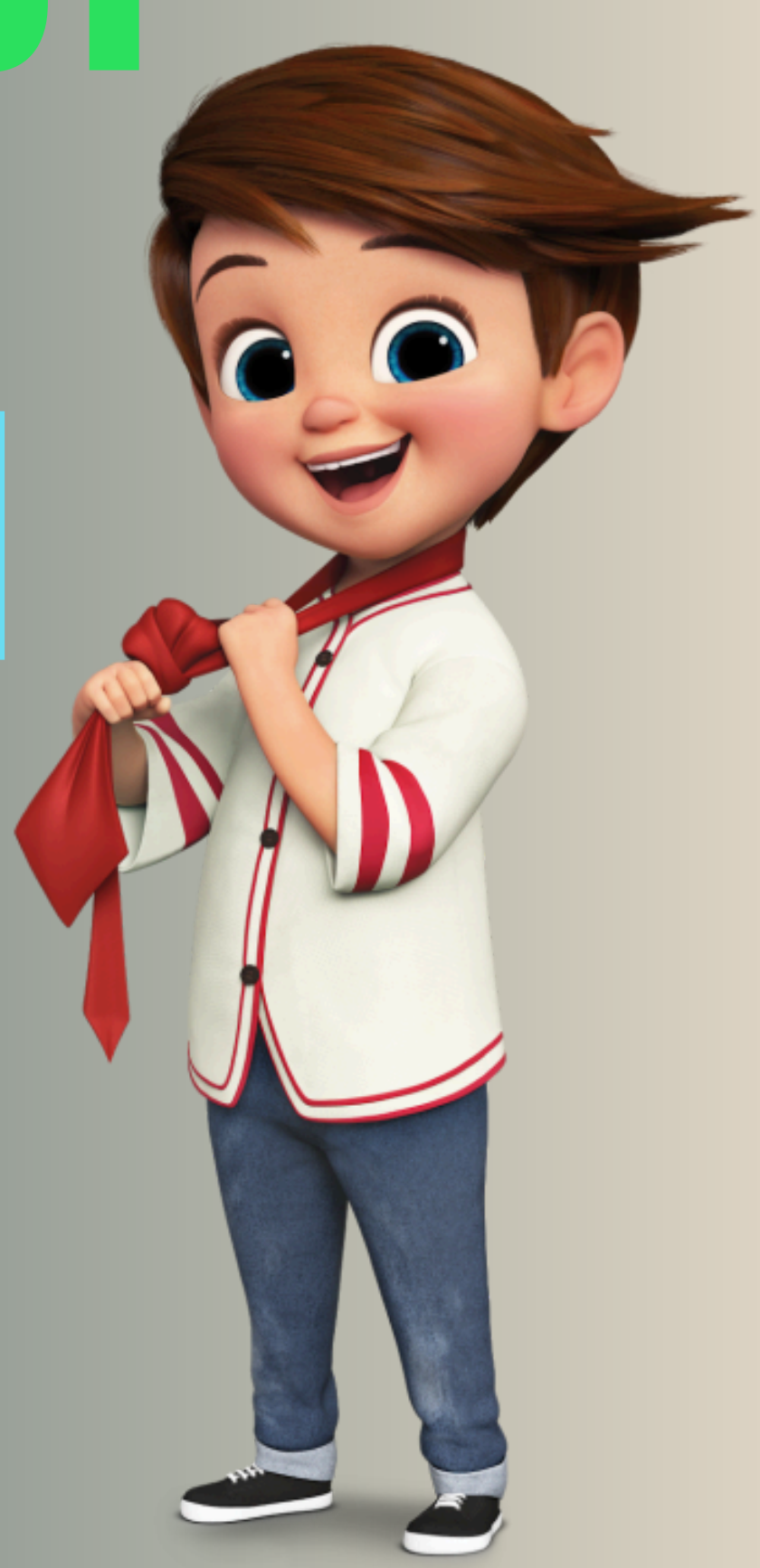
SPRING BOOT REQUEST VALIDATION GUIDE

Rushikesh Thakare



BASIC

1 VALIDATION SETUP

- Add Validation Dependency

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
```

- Annotate Your Model

```
public class UserRequest {

    @NotBlank(message = "Username is mandatory")
    @Size(min = 3, max = 20, message = "Username must be 3-20
characters")
    private String username;

    @Email(message = "Email should be valid")
    private String email;

    @Min(value = 18, message = "Age must be at least 18")
    @Max(value = 100, message = "Age must be less than 100")
    private int age;

    @Pattern(regexp = "^(?=.*[A-Za-z])(?=.*\\d)[A-Za-z\\d]{8,}$",
            message = "Password must contain at least 8 characters
with letters and numbers")
    private String password;

    // Getters and setters
}
```

CONTROLLER

2 VALIDATION

- Basic Validation

```
@PostMapping("/users")
public ResponseEntity<String> createUser(@Valid @RequestBody
UserRequest userRequest) {
    // Business logic here
    return ResponseEntity.ok("User created successfully");
}
```

2

CONTROLLER VALIDATION

- Handling Validation Errors

```
@RestControllerAdvice
public class GlobalExceptionHandler {

    @ResponseStatus(HttpStatus.BAD_REQUEST)
    @ExceptionHandler(MethodArgumentNotValidException.class)
    public Map<String, String>
handleValidationExceptions(MethodArgumentNotValidException ex)
{
        Map<String, String> errors = new HashMap<>();
        ex.getBindingResult().getAllErrors().forEach(error -> {
            String fieldName = ((FieldError) error).getField();
            String errorMessage = error.getDefaultMessage();
            errors.put(fieldName, errorMessage);
        });
        return errors;
    }
}
```

3

COMMON VALIDATION ANNOTATIONS

- Handling Validation Errors

Annotation	Purpose	Example
@NotNull	Rejects null values	@NotNull private String name;
@NotBlank	Rejects empty strings	@NotBlank private String username;
@Size	Validates string/collection size	@Size(min=2, max=30) String name;
@Email	Validates email format	@Email private String email;
@Min/@Max	Number range validation	@Min(18) int age;
@Pattern	Regex validation	@Pattern(regexp="[A-Za-z]+")
@Future/@Past	Date validation	@Future LocalDate eventDate;

4

ADVANCED VALIDATION

1. Custom validator

```
@Documented
@Constraint(validatedBy = PhoneNumberValidator.class)
@Target({ElementType.FIELD})
@Retention(RetentionPolicy.RUNTIME)
public @interface ValidPhoneNumber {
    String message() default "Invalid phone number";
    Class<?>[] groups() default {};
    Class<? extends Payload>[] payload() default {};
}
```

```
public class PhoneNumberValidator implements
ConstraintValidator<ValidPhoneNumber, String> {
    @Override
    public boolean isValid(String phone, ConstraintValidatorContext
context) {
        return phone != null && phone.matches("^\\+?[0-9]{10,15}$");
    }
}
```

- Usage:

```
@ValidPhoneNumber
private String phone;
```

2. Group Validation:

```
public interface BasicValidation {}
public interface AdvancedValidation {}

public class UserRequest {
    @NotBlank(groups = BasicValidation.class)
    private String username;

    @Email(groups = AdvancedValidation.class)
    private String email;
}
```

```
@PostMapping("/users")
public ResponseEntity<?> createUser(
    @Validated(BasicValidation.class) @RequestBody UserRequest
userRequest) {
    // Only basic validation will be performed
    return ResponseEntity.ok("User created");
}
```

THANKS

Rushikesh Thakare