# Spring Boot – Validation using Hibernate Validator

Last Updated : 20 Aug, 2025

Validating user input is an important part of building secure and reliable applications. Spring Boot makes this simple by integrating with Hibernate Validator, the reference implementation of JSR 380 (Bean Validation API). Using it, developers can enforce validation rules on data models with simple annotations like @NotNull, @Size, @Email, etc.

In this article, we will explore how to:

- Use common validation annotations.
- Apply them in entity classes.
- Handle validation errors gracefully.
- Expose a REST API that validates input automatically.

## Common Hibernate Validator Annotations

Hibernate validators provide the following annotations that are very helpful for software development.

1. **@NotNull:** @NotNull ensures that a field is not null but allows empty values (e.g., an empty string or an empty collection).
2. **@NotEmpty:** @NotEmpty ensures that a field is not null and also not empty, meaning it must contain at least one element (for collections) or at least one character (for strings).
3. **@NotBlank:** @NotBlank applies only to strings and ensures they are not null, not empty and contain at least one non-whitespace character (i.e., spaces alone are not allowed).
4. **@Min:** Given Minimum value has to be satisfied
5. **@Max:** Given Maximum value has to be satisfied

6. **@Size:** Field size should be less than or greater than the specified field size
7. **@Email:** Email can be validated with this
8. **@Pattern:** Given the RegEx Pattern has to be satisfied.

## Step-by-Step Implementation

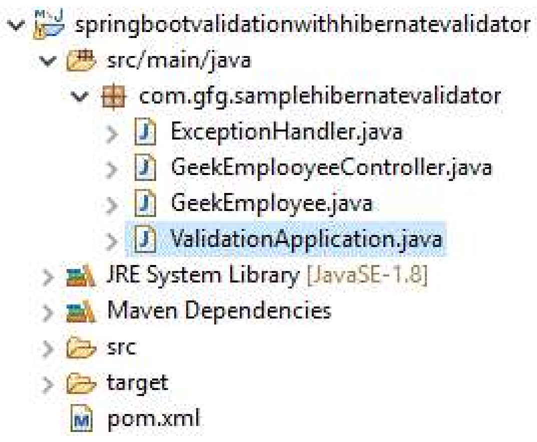
### Step 1: Create the project

Use Spring Initializr (or your IDE's wizard):

- **Project:** Maven
- **Language:** Java
- **Spring Bo**ot: 3.x
- **Dependencies:** Spring Web, Validation (spring-boot-starter-validation), Lombok (optional but handy)

*Java 17+ is recommended for Spring Boot 3.x.*

### Step 2: Project structure (minimal)



```
springbootvalidationwithhibernatevalidator
  src/main/java
    com.gfg.samplehibernatevalidator
      ExceptionHandler.java
      GeekEmplooyeeController.java
      GeekEmployee.java
      ValidationApplication.java
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  src
  target
  pom.xml
```

### Step 3: Create an Entity Class

Let's create an entity class **GeekEmployee** with validation annotations applied to its fields.

### GeekEmployee.java

```
package com.gfg.samplehibernatevalidator;

import jakarta.validation.constraints.Email;
import jakarta.validation.constraints.Min;
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.NotEmpty;
import java.util.List;

public class GeekEmployee {

    @NotBlank(message = "Employee name cannot be blank")
    private String geekEmployeeName;

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

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

    @NotEmpty(message = "Qualifications cannot be empty")
    private List<String> qualifications;

    // Getters and Setters
    public String getGeekEmployeeName() {
        return geekEmployeeName;
    }

    public void setGeekEmployeeName(String geekEmployeeName) {
        this.geekEmployeeName = geekEmployeeName;
    }

    public String getGeekEmailId() {
        return geekEmailId;
    }

    public void setGeekEmailId(String geekEmailId) {
        this.geekEmailId = geekEmailId;
    }
```

```java
    public double getSalary() {
        return salary;
    }

    public void setSalary(double salary) {
        this.salary = salary;
    }

    public List<String> getQualifications() {
        return qualifications;
    }

    public void setQualifications(List<String> qualifications) {
        this.qualifications = qualifications;
    }
}
```

## Step 4: Exception handling for Validators Errors

When validation fails, Spring Boot throws a MethodArgumentNotValidException. We can handle this exception globally using @ControllerAdvice and return a structured error response.

### GlobalExceptionHandler.java:

```java
package com.gfg.samplehibernatevalidator;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.MethodArgumentNotValidException;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;

import java.util.HashMap;
import java.util.Map;

@ControllerAdvice
public class ExceptionHandler {

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

        return new ResponseEntity<>(errors, HttpStatus.BAD_REQUEST);
    }
}
```

Let us try to save the geek employees by accepting the inputs like "geekEmployeeName", "salary", "geekEmailId" and "qualifications". We need a rest controller file to achieve the same.

## Step 5: REST Controller for Validation

Let’s create a REST controller to test the validation.

### GeekEmployeeController.java:

```
package com.gfg.samplehibernatevalidator;

import jakarta.validation.Valid;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/geek")
public class GeekEmployeeController {

    @PostMapping("/addEmployee")
    public ResponseEntity<String> addEmployee(@Valid @RequestBody
GeekEmployee employee) {
        return new ResponseEntity<>(
                "Employee details are valid and added successfully! \n" +
                "Name: " + employee.getGeekEmployeeName() + "\n" +
                "Email: " + employee.getGeekEmailId() + "\n" +
                "Salary: " + employee.getSalary() + "\n" +
                "Qualifications: " + employee.getQualifications(),
                HttpStatus.OK
        );
    }
}
```

**Note:**

- **@Valid:** This annotation triggers validation on the Employee object.
- **Response:** If validation passes, the employee object is returned with a 201 CREATED status.

## Step 6: Running the Application

Open main class and Run the Spring Boot application using the following command

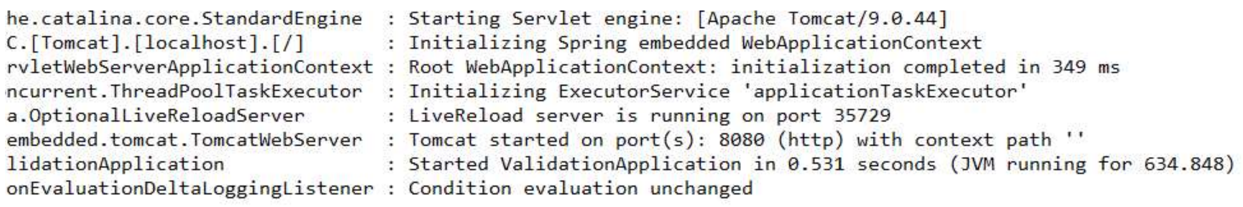
> *mvn spring-boot:run*

### Main Class: ValidationApplication.java

```java
package com.gfg.samplehibernatevalidator;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class ValidationApplication {

    public static void main(String[] args) {
        SpringApplication.run(ValidationApplication.class, args);
    }
}
```

> ***Note:** To run application using IDE goto main method in Spring Boot Application class and run the main method.*

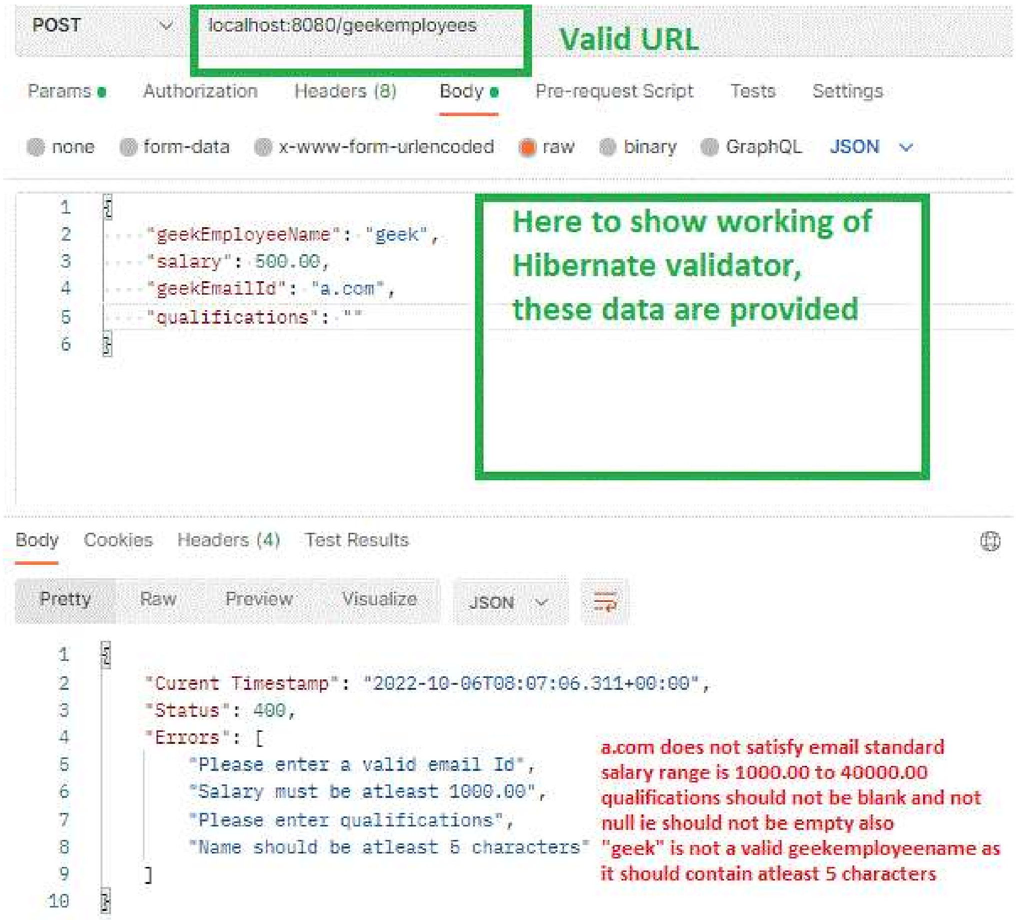
On running the application, in the console, we can see as following



```
he.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.44]
C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
rvletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 349 ms
ncurrent.ThreadPoolTaskExecutor  : Initializing ExecutorService 'applicationTaskExecutor'
a.OptionalLiveReloadServer       : LiveReload server is running on port 35729
embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
lidationApplication              : Started ValidationApplication in 0.531 seconds (JVM running for 634.848)
onEvaluationDeltaLoggingListener : Condition evaluation unchanged
```

## Step 7: Test Application (Postman or curl).

Let's check the working part by using the Postman client as we are using post-mapping


POST
Valid URL
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Here to show working of Hibernate validator, these data are provided
Body
Cookies
Headers (4)
Test Results
Pretty
Raw
Preview
Visualize
JSON
a.com does not satisfy email standard
salary range is 1000.00 to 40000.00
qualifications should not be blank and not null ie should not be empty also
"geek" is not a valid geekemployeename as it should contain atleast 5 characters

```
localhost:8080/geekemployees
```

```
{
    "geekEmployeeName": "geek",
    "salary": 500.00,
    "geekEmailId": "a.com",
    "qualifications": ""
}
```

```
{
    "Curent Timestamp": "2022-10-06T08:07:06.311+00:00",
    "Status": 400,
    "Errors": [
        "Please enter a valid email Id",
        "Salary must be atleast 1000.00",
        "Please enter qualifications",
        "Name should be atleast 5 characters"
    ]
}
```

Using the Hibernate validator, front-end validations can be checked and that helps a lot in spring boot projects simply a dependency makes these kinds of validations validate a page.

Comment More info Advertise with us