Implementing Validations for RESTful Services

The validation is a common requirement in all the services. We will discuss Java Validation API to add validation in our beans file. When we get a request to create a user, we should validate its content. If it is invalid, we should return a proper response.

<**dependency**>

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

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

</**dependency**>

Let's see how to validate a request.

**Step 1:** Open the **UserResource.java** file.

**Step 2:** Add **@Valid** annotation. It is a Javax validation API. Its default classpath is spring-boot-starter-web.

**UserResource.java**

1. **package** com.javatpoint.server.main.user;
2. **import** java.net.URI;
3. **import** java.util.List;
4. **import** javax.validation.Valid;
5. **import** org.springframework.beans.factory.annotation.Autowired;
6. **import** org.springframework.http.ResponseEntity;
7. **import** org.springframework.web.bind.annotation.DeleteMapping;
8. **import** org.springframework.web.bind.annotation.GetMapping;
9. **import** org.springframework.web.bind.annotation.PathVariable;
10. **import** org.springframework.web.bind.annotation.PostMapping;
11. **import** org.springframework.web.bind.annotation.RequestBody;
12. **import** org.springframework.web.bind.annotation.RestController;
13. **import** org.springframework.web.servlet.support.ServletUriComponentsBuilder;
14. @RestController
15. **public** **class** UserResource
16. {
17. @Autowired
18. **private** UserDaoService service;
19. @GetMapping("/users")
20. **public** List<User> retriveAllUsers()
21. {
22. **return** service.findAll();
23. }
24. //retrieves a specific user detail
25. @GetMapping("/users/{id}")
26. **public** User retriveUser(@PathVariable **int** id)
27. {
28. User user= service.findOne(id);
29. **if**(user==**null**)
30. //runtime exception
31. **throw** **new** UserNotFoundException("id: "+ id);
32. **return** user;
33. }
34. //method that delete a user resource
35. @DeleteMapping("/users/{id}")
36. **public** **void** deleteUser(@PathVariable **int** id)
37. {
38. User user= service.deleteById(id);
39. **if**(user==**null**)
40. //runtime exception
41. **throw** **new** UserNotFoundException("id: "+ id);
42. }
43. //method that posts a new user detail and returns the status of the user resource
44. @PostMapping("/users")
45. **public** ResponseEntity<Object> createUser(@RequestBody  @Valid  User user)
46. {
47. User sevedUser=service.save(user);
48. URI location=ServletUriComponentsBuilder.fromCurrentRequest().path("/{id}").buildAndExpand(sevedUser.getId()).toUri();
49. **return** ResponseEntity.created(location).build();
50. }
51. }

Now we will add validations in **User** class on **name** and **date of birth**. Let suppose that name should have at least five characters and date of birth should be in past not in present.

**Step 3:** Open the **User.java** file.

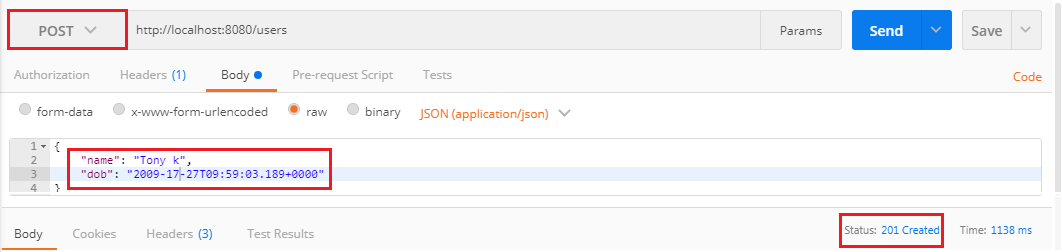
**Step 4:** Add **@Size(min=5)** annotation just above the **name** variable.

**Step 5:** Add **@Past** annotation just above the **dob** variable.

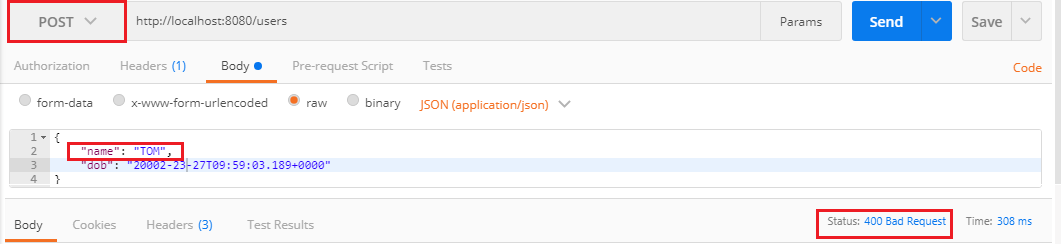
**User.java**

1. **package** com.javatpoint.server.main.user;
2. **import** java.util.Date;
3. **import** javax.validation.constraints.Past;
4. **import** javax.validation.constraints.Size;
5. **public** **class** User
6. {
7. **private** Integer id;
8. @Size(min=5)
9. **private** String name;
10. @Past
11. **private** Date dob;
12. //default constructor
13. **protected** User()
14. {
16. }
17. **public** User(Integer id, String name, Date dob)
18. {
19. **super**();
20. **this**.id = id;
21. **this**.name = name;
22. **this**.dob = dob;
23. }
24. **public** Integer getId()
25. {
26. **return** id;
27. }
28. **public** **void** setId(Integer id)
29. {
30. **this**.id = id;
31. }
32. **public** String getName()
33. {
34. **return** name;
35. }
36. **public** **void** setName(String name)
37. {
38. **this**.name = name;
39. }
40. **public** Date getDob()
41. {
42. **return** dob;
43. }
44. **public** **void** setDob(Date dob)
45. {
46. **this**.dob = dob;
47. }
48. @Override
49. **public** String toString()
50. {
51. //return "User [id=" + id + ", name=" + name + ", dob=" + dob + "]";
52. **return** String.format("User [id=%s, name=%s, dob=%s]", id, name, dob);
53. }
54. }

**Step 5:** Open the Rest client **Postman** and send a **POST** request with new user name **Tony k**. It returns **Status: 201 Created**.



Now we send another **POST** request. But the name should have less than five characters. It returns the **Status: 400 Bad Request**.



When we create a RESTful services we need to think about consumer that how does the consumer know what is wrong. To resolve this problem we will add a method handleMethodArgumentNotValid() which is defined in the **ResponseEntityExceptionHandler** class. This is the method which is fired when a bad request occurs.

1. **protected** ResponseEntity<Object> handleMethodArgumentNotValid(  MethodArgumentNotValidException ex, HttpHeaders headers, HttpStatus status, WebRequest request)
2. {
3. **return** handleExceptionInternal(ex, **null**, headers, status, request);
4. }

**Step 6:** Copy and paste the above method in **CustomizedResponseEntityExceptionHandler.java** file.

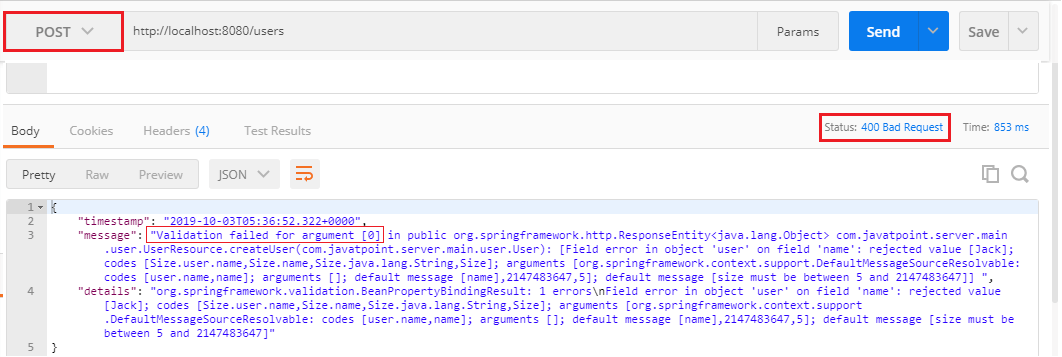
**Step 7:** Override the method by adding the annotation **@Override**.

**CustomizedResponseEntityExceptionHandler.java**

1. **package** com.javatpoint.server.main;
2. **import** java.util.Date;
3. **import** org.springframework.http.HttpHeaders;
4. **import** org.springframework.http.HttpStatus;
5. **import** org.springframework.http.ResponseEntity;
6. **import** org.springframework.web.bind.MethodArgumentNotValidException;
7. **import** org.springframework.web.bind.annotation.ControllerAdvice;
8. **import** org.springframework.web.bind.annotation.ExceptionHandler;
9. **import** org.springframework.web.bind.annotation.RestController;
10. **import** org.springframework.web.context.request.WebRequest;
11. **import** org.springframework.web.servlet.mvc.method.annotation.ResponseEntityExceptionHandler;
12. **import** com.javatpoint.server.main.exception.ExceptionResponse;
13. **import** com.javatpoint.server.main.user.UserNotFoundException;
14. //defining exception handling for all the exceptions
15. @ControllerAdvice
16. @RestController
17. **public** **class** CustomizedResponseEntityExceptionHandler **extends** ResponseEntityExceptionHandler
18. {
19. @ExceptionHandler(Exception.**class**)
20. //override method of ResponseEntityExceptionHandler class
21. **public** **final** ResponseEntity<Object> handleAllExceptions(Exception ex, WebRequest request)
22. {
23. //creating exception response structure
24. ExceptionResponse exceptionResponse= **new** ExceptionResponse(**new** Date(), ex.getMessage(), request.getDescription(**false**));
25. //returning exception structure and specific status
26. **return** **new** ResponseEntity(exceptionResponse, HttpStatus.INTERNAL\_SERVER\_ERROR);
27. }
28. @ExceptionHandler(UserNotFoundException.**class**)
29. //override method of ResponseEntityExceptionHandler class
30. **public** **final** ResponseEntity<Object> handleUserNotFoundExceptions(UserNotFoundException ex, WebRequest request)
31. {
32. //creating exception response structure
33. ExceptionResponse exceptionResponse= **new** ExceptionResponse(**new** Date(), ex.getMessage(), request.getDescription(**false**));
34. //returning exception structure and specific status
35. **return** **new** ResponseEntity(exceptionResponse, HttpStatus.NOT\_FOUND);
36. }
37. @Override
38. **protected** ResponseEntity<Object> handleMethodArgumentNotValid(MethodArgumentNotValidException ex, HttpHeaders headers, HttpStatus status, WebRequest request)
39. {
40. ExceptionResponse exceptionResponse= **new** ExceptionResponse(**new** Date(), ex.getMessage(), ex.getBindingResult().toString());
41. //returning exception structure and specific status
42. **return** **new** ResponseEntity(exceptionResponse, HttpStatus.BAD\_REQUEST);
43. }
44. }

**Step 8:** Now, we send a **POST** request through **Postman**. It returns the exception structure with message **Validation failed for argument** and other details.

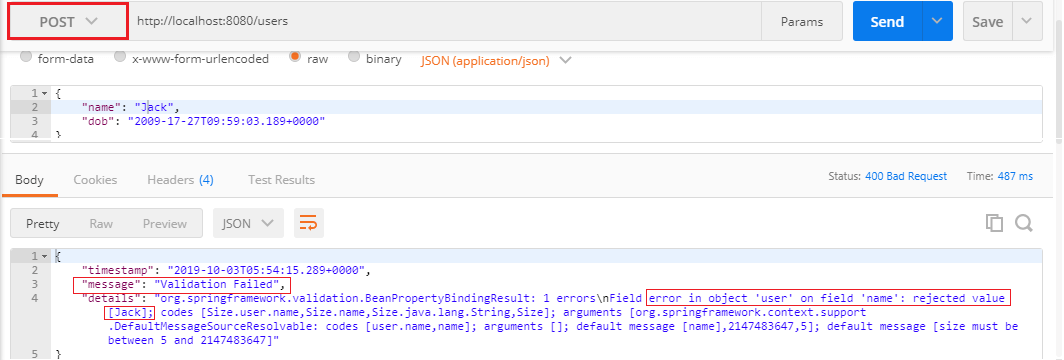
Advertisement



It is difficult to understand the message for the user. So we will now customize the message with the string **Validation Failed** instead of getting message.

1. @Override
2. **protected** ResponseEntity<Object> handleMethodArgumentNotValid(MethodArgumentNotValidException ex, HttpHeaders headers, HttpStatus status, WebRequest request)
3. {
4. ExceptionResponse exceptionResponse= **new** ExceptionResponse(**new** Date(), "Validation Failed", ex.getBindingResult().toString());
5. //returning exception structure and specific status
6. **return** **new** ResponseEntity(exceptionResponse, HttpStatus.BAD\_REQUEST);
7. }

**Step 9:** Again send a **POST** request. It returns the message which we have customized.



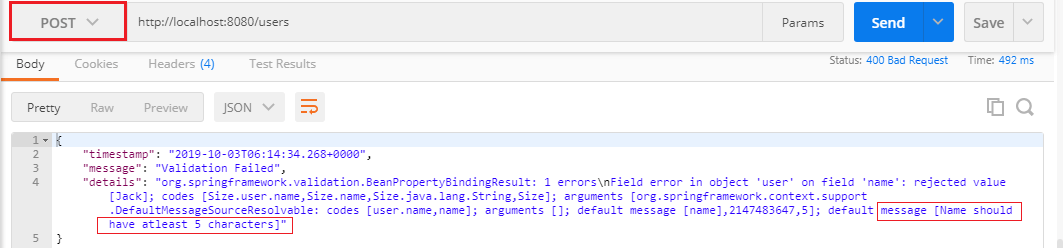
It might be useful for the consumer. Now we customize the message again and make it more error specific.

Advertisement

**Step 10:** Open **User.java** file and add an attribute **message="Name should have at least 5 characters"** in **@Size** annotation.

1. @Size(min=5, message="Name should have at least 5 characters")

**Step 11:** Again, send a **POST** request. It returns the more specific exception which we have specified.



Advertisement

We can further customize the exception by following the **BindingResult** interface. There is a wide variety of exception messages. There are following validation classes defined in validation-api-2.0.1.Final.jar.

