**API Flow**

**Table of Contents**

[**1.** **Read file .yaml** 2](#_Toc47272733)

[**2.** **Spring Boot Profile** 5](#_Toc47272734)

[**3.** **Get request parameters** 6](#_Toc47272735)

[**4.** **Validate request parameters** 17](#_Toc47272736)

[**5.** **Normal API processing Flow** 42](#_Toc47272737)

[**6.** **Exception Handler** 46](#_Toc47272738)

[**7.** **Filter** 53](#_Toc47272739)

**Actions form normal API**

|  |  |
| --- | --- |
| 1. Read file yaml (.yml) | parse vào object  parse vào list string  parse vào list object  parse trưc tiếp vào local parameter |
| 1. Spring Boot profile |  |
| 1. Get request | nhận bằng object:  - query string  - path variable  - body  - header |
|  | nhận trực tiếp:  - query string  - path variable  - header |
|  | nhận list data  - List String  - List Object |
|  | nhận boolean |
| 1. Validation input parameter | validate trên request object:  - String: null, blank, length, size, format  - Number: min, max, parse error  - List: empty  - Date |
|  | validate trên argment trực tiếp từ method controller |
| 1. Factory -> service -> repository -> response |  |
| 1. Spring boot Filter |  |
| 1. Flow exception -> error response |  |
| 1. Access database |  |
| 1. Access data from api (RestTemplate) |  |
| 1. Unit test |  |

# 

# **Read file .yaml**

* 1. **Parse vào Object**
* application.yml

|  |
| --- |
| api:  list:  host: localhost  port: 8080  path: /list  detail:  host: localhost  port: 8080  path: /detail |

* DetailApiConfig.java

|  |
| --- |
| @Configuration  @ConfigurationProperties(prefix = "api.detail")  @Data  public class DetailApiConfig {  private String host;  private String port;  private String path;  } |

* ListApiConfig.java

|  |
| --- |
| @Configuration  @ConfigurationProperties(prefix = "api.list")  @Data  public class ListApiConfig {  private String host;  private String port;  private String path;  } |

**Lưu ý:** nếu không định nghĩa “*prefix*” bên trong*@ConfigurationProperties* thì Spring sẽ đọc từ root của file yml

* 1. **Parse vào List String**
* application.yml

|  |
| --- |
| product:  categories:  - car  - motorcycle  - bicycle |

* ProductConfig.java

|  |
| --- |
| @Configuration  @ConfigurationProperties(prefix = "product")  @Data  public class ProductConfig {  private List<String> categories;  } |

* 1. **Parse vào List Object**
* application.yml

|  |
| --- |
| class:  students:  - student:  name: Student A  gender: male  - student:  name: Student B  gender: male  - student:  name: Student C  gender: female |

* StudentConfig.java

|  |
| --- |
| @Configuration  @ConfigurationProperties(prefix = "class")  @Data  public class StudentConfig {  private List<Student> students;  @Data  public static class Student {  private String name;  private String gender;  }  } |

**Lưu ý:** sử dụng inner static class

* 1. **Parse trực tiếp vào local parameter**
* application.yml

|  |
| --- |
| api:  list:  host: localhost  port: 8080  path: /list  detail:  host: localhost  port: 8080  path: /detail  timeout: 3000 |

Trường hợp chỉ cần lấy giá trị “*timeout*” -> có thể parse trực tiếp vào local parameter

* RestTemplateConfig.java

|  |
| --- |
| @Configuration  public class RestTemplateConfig {  @Value("${api.timeout}")  private int timeout;  @Bean  public RestTemplate restTemplate(RestTemplateBuilder builder) {  return builder  .setConnectTimeout(Duration.ofMillis(timeout))  .setReadTimeout(Duration.ofMillis(timeout))  .build();  }  } |

**Lưu ý:** nếu sử dụng cách này, thì ở Unit test code sẽ không Mock giá trị được, khi đó sẽ define file application.yml ở package “*test*”

# **Spring Boot Profile**

Trên thực tế, ứng dụng có thể sẽ sử dụng các config khác nhau tùy vào môi trường được deploy. VD: dev, staging, production, …

Do đó cơ chế **Spring Boot Profile** sẽ chỉ định application đọc config tương ứng với từng loại môi trường deploy

* application.yml: mặc định application sẽ truy cập vào đây để đọc config (trong trường hợp không chỉ định profile)

|  |
| --- |
| api:  list:  host: localhost  port: 8080  path: /list  detail:  host: localhost  port: 8080  path: /detail |

* application-dev.yml

|  |
| --- |
| api:  list:  host: http://com.example.dev  port: 80  path: /list  detail:  host: http://com.example.dev  port: 80  path: /detail |

* application-prod.yml

|  |
| --- |
| api:  list:  host: https://com.example.production  port: 443  path: /list  detail:  host: https://com.example.production  port: 443  path: /detail |

* application.properties

|  |
| --- |
| spring.profiles.active=dev |

**Lưu ý:**

**-** Mỗi file .yml sẽ chứa config tương ứng với từng môi trường. Đảm bảo file name theo cú pháp: **application-xxx.yml** -> xxx có thể thay đổi tùy ý

**-** Ở file application.properties sẽ chỉ định file config .yml nào sẽ được sử dụng. Phân biệt bằng giá trị xxx t=trong format ở trên

**-** Theo ví dụ trên, thì application sẽ đọc config từ ***application-dev.yml***

# **Get request parameters**

* 1. **Parse parameters to object**

Tiêu chí để parse parameter vào object là: name của các item trong object phải khớp với các request parameters

Có thể chỉ định tên item tùy ý bằng cách dùng annotation: ***@JsonProperty.***

*VD:*

|  |
| --- |
| @JsonProperty("student\_id")  private int id; |

* + 1. ***Query string (Paramters on Url)***

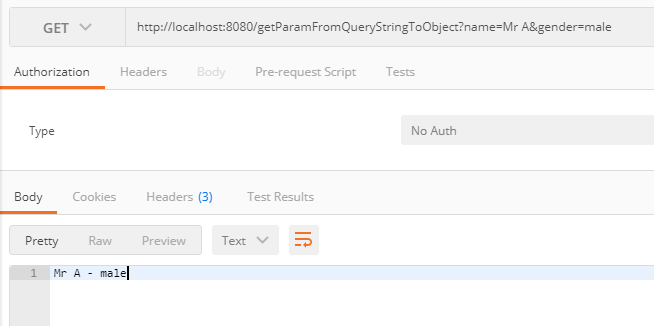
StudentModelAttribute.java

|  |
| --- |
| @Data  public class StudentModelAttribute {  private String name;  private String gender;  } |

StudentController.java

|  |
| --- |
| @GetMapping("/getParamFromQueryStringToObject")  public String getParamFromQueryStringToObject(@ModelAttribute StudentModelAttribute studentModelAttribute) {  return studentModelAttribute.getName() + " - " studentModelAttribute.getGender();  } |

**Lưu ý: @ModelAttribute** có thể có hoặc không -> thì vẫn có thể parse query strings vào object



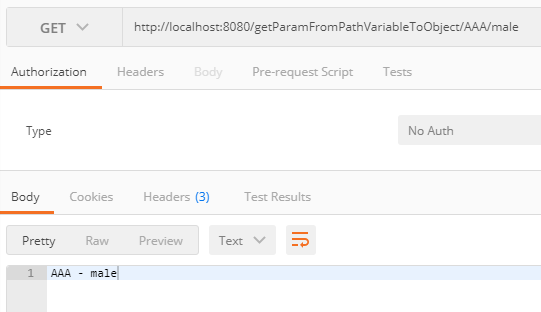
* + 1. ***Path Variables (Paramters on Uri)***

StudentRequest.java

|  |
| --- |
| @Data  public class StudentRequest {  private String name;  private String gender;  } |

StudentController.java

|  |
| --- |
| @GetMapping("/getParamFromPathVariableToObject/{name}/{gender}")  public String getParamFromPathVariableToObject(StudentRequest studentRequest) {  return studentRequest.getName() + " - " + studentRequest.getGender();  } |



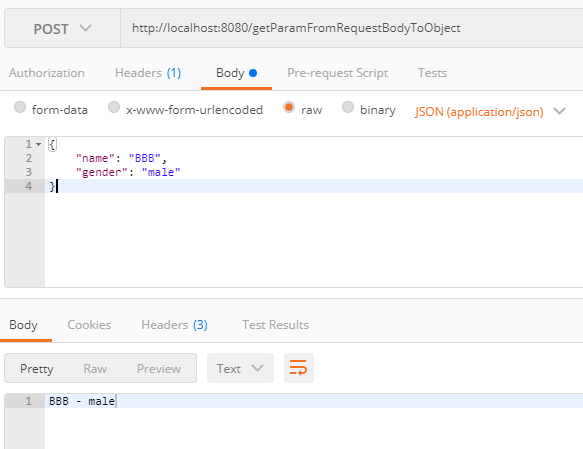
* + 1. ***Request Body***

StudentRequestBody.java

|  |
| --- |
| @Data  public class StudentRequestBody {  private String name;  private String gender;  } |

StudentController.java

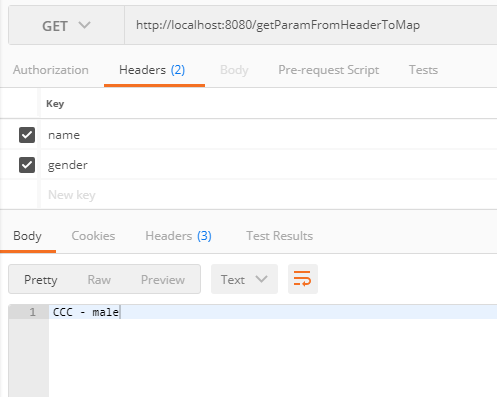
|  |
| --- |
| @PostMapping("/getParamFromRequestBodyToObject")  public String getParamFromRequestBodyToObject(@RequestBody StudentRequestBody studentRequestBody) {  return studentRequestBody.getName() + " - " + studentRequestBody.getGender();  } |



* + 1. ***Request Header***

StudentController.java

|  |
| --- |
| @GetMapping("/getParamFromHeaderToMap")  public String getParamFromHeaderToMap(@RequestHeader Map<String, String> headers) {  return headers.get("name") + " - " + headers.get("gender");  } |



* 1. **Parse parameters trực tiếp vào controller argument**

Tiêu chí để parse parameter vào Controller argument là: Controller argument name phải khớp với các request parameters

Có thể chỉ định tên Controller argument tùy ý bằng cách define tên json name trướcController argument

*VD:*

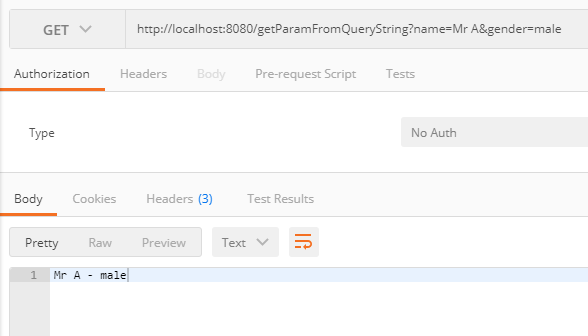
|  |
| --- |
| @RequestParam("student\_id") int id |

* + 1. ***Query string (Paramters on Url)***

StudentController.java

|  |
| --- |
| @GetMapping("/getParamFromQueryString")  public String getParamFromQueryString(@RequestParam String name,  @RequestParam(required = false) String gender) {  return name + " - " + gender;  } |

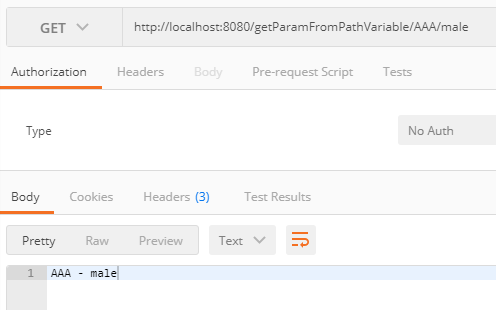
**Lưu ý:** (required = false) -> có ý nghĩa là parameter “gender” có thể truyền vào hoặc không truyền -> nếu không truyền vào sẽ nhận giá trị Null



* + 1. ***Path Variables (Paramters on Uri)***

StudentController.java

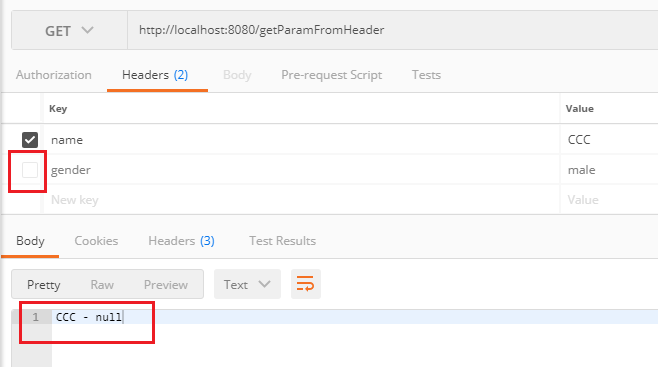
|  |
| --- |
| @GetMapping("/getParamFromPathVariable/{name}/{gender}")  public String getParamFromPathVariable(@PathVariable String name,  @PathVariable String gender) {  return name + " - " + gender;  } |

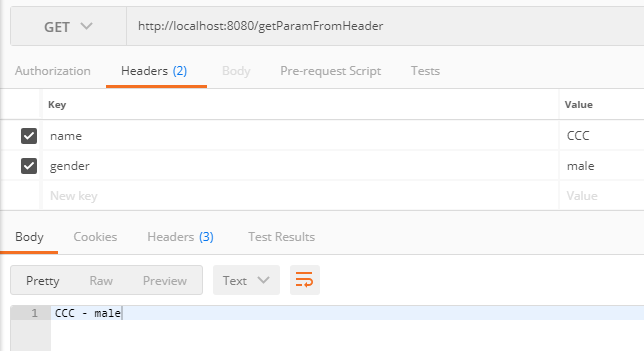


* + 1. ***Request Header***

StudentController.java

|  |
| --- |
| @GetMapping("/getParamFromHeader")  public String getParamFromHeader(@RequestHeader String name,  @RequestHeader (required = false) String gender) {  return name + " - " + gender;  } |





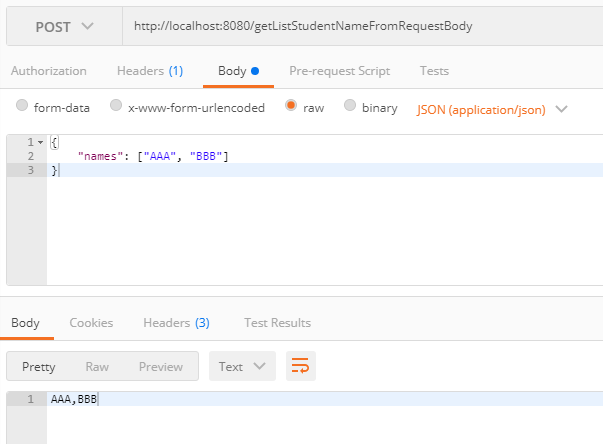
* 1. **Get List data from Request**
     1. ***List String***

StudentNamesRequestBody.java

|  |
| --- |
| @Data  public class StudentNamesRequestBody {  private List<String> names;  } |

StudentController.java

|  |
| --- |
| @PostMapping("/getListStudentNameFromRequestBody")  public String getListStudentNameFromRequestBody(  @RequestBody StudentNamesRequestBody studentNamesRequestBody) {  return String.join(", ", studentNamesRequestBody.getNames());  } |



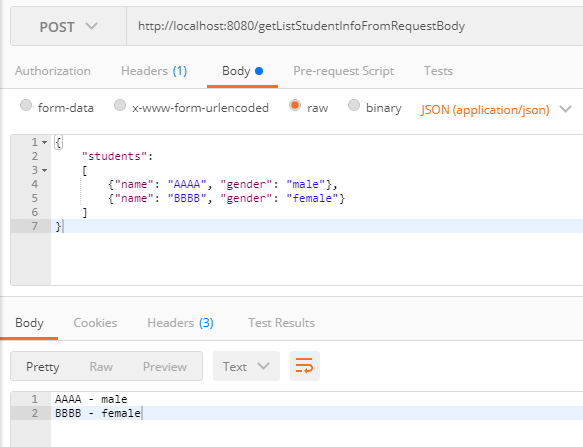
* + 1. ***List Object***

StudentsRequestBody.java

|  |
| --- |
| @Data  public class StudentsRequestBody {  private List<StudentInfo> students;  @Data  public static class StudentInfo {  private String name;  private String gender;  @Override  public String toString() {  return name + " - " + gender;  }  }  } |

StudentController.java

|  |
| --- |
| @PostMapping("/getListStudentInfoFromRequestBody")  public String getListStudentInfoFromRequestBody(  @RequestBody StudentsRequestBody studentsRequestBody) {  return studentsRequestBody.getStudents()  .stream()  .map(Object::toString)  .collect(Collectors.joining("\r\n"));  } |



* 1. **Get Boolean from Request**

StudentModelAttribute.java

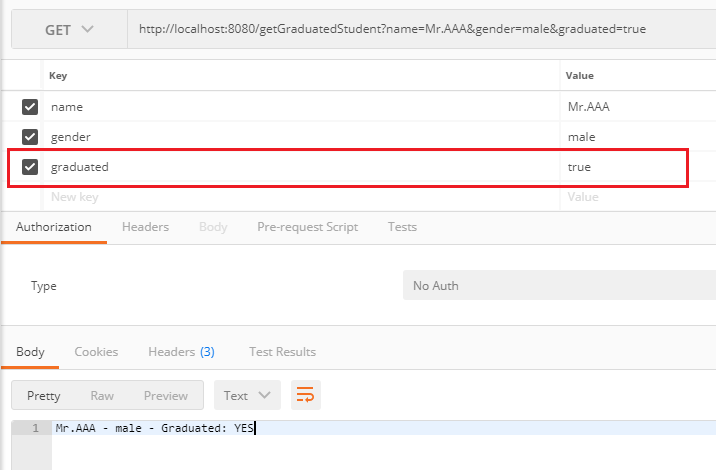
|  |
| --- |
| @Data  public class StudentModelAttribute {  private String name;  private String gender;  private boolean graduated; // true | false  } |

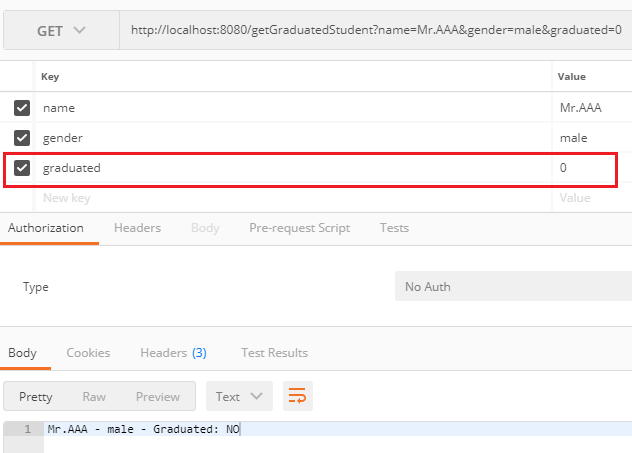
StudentController.java

|  |
| --- |
| @GetMapping("/getGraduatedStudent")  public String getGraduatedStudent(StudentModelAttribute request) {  String graduated = request.isGraduated() ? "Graduated: YES" : "Graduated: NO";  return request.getName() + " - " + request.getGender() + " - " + graduated;  } |

**Luu ý**: Đối với kiểu dữ liệu là **Boolean** thì có 2 cách truyền giá trị vào **request parameter**

* “true” or “false”
* “1” or “0”





Nếu input giá trị không phải : true, false, 0, 1 -> thì hệ thống sẽ xãy ra lỗi **binding Error boolean**

* Sẽ xử lý ở section tiếp theo

# **Validate request parameters**

Một vài cách check validate parameter đầu vào:

-> Check trực tiếp bằng java code trong Controller

-> Check bằng java script (cách này rất hạn chế dùng để tránh lỗ hổng về Security)

-> Check bằng cách dùng @Annotation -> Dưới đây là ví dụ về các annotation thông dụng

**Note:**  có 2 cách validate bằng @Annotation:

- Validate trên request object

- Validate trực tiếp trên argument ở method controller

* 1. **Các step để thực hiện check validate trên request object:**
* Ở object request: Define @annotation tương ứng với các item cần check.
* Ở controller:

+ thêm vào @Valid ở đầu method argument

+ thêm BindingResult để nhận kết quả check (có bao gồm error message)

* + 1. **Validation for String**
* ***Check String Null***

|  |  |
| --- | --- |
| @NotNull | Dùng default message. Lưu ý default message không đính kèm field name, do đó để có 1 error message đầy đủ phải thực hiện get filed name và add thêm vào message. Do đó dùng custom message được khuyên dùng |
| @NotNull(message = "fullName must not be null") | Custom message |

**Lưu ý:** Trường hợp String empty vẫn hợp lệ

EmployeeSearchRequest.java

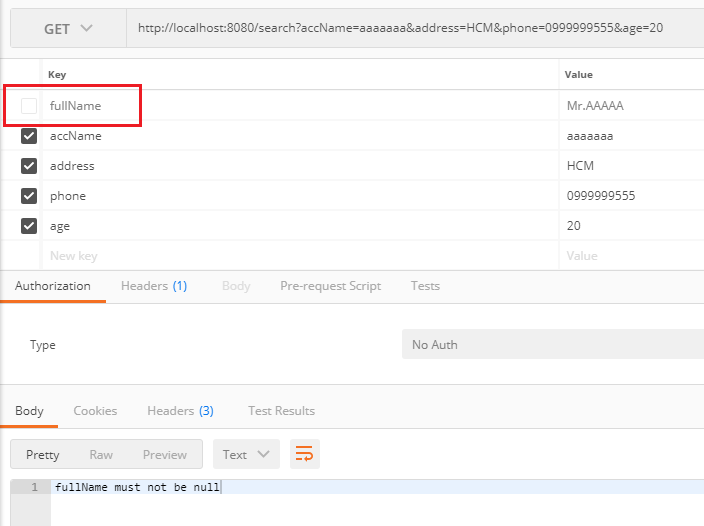
|  |
| --- |
| @Data  public class EmployeeSearchRequest {  @NotNull  private String fullName;  private String accName;  private String address;  private String phone;  } |

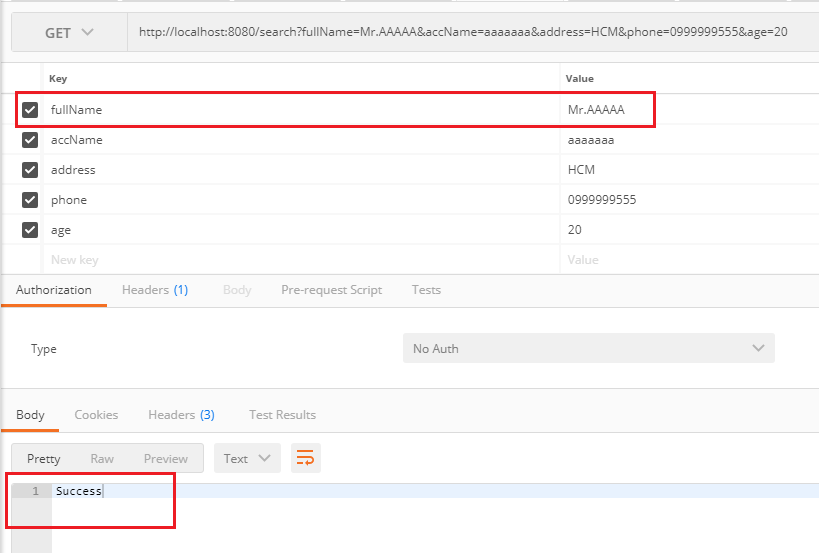
Theo đoạn code trên chỉ validate cho **“fullName” NotNull**

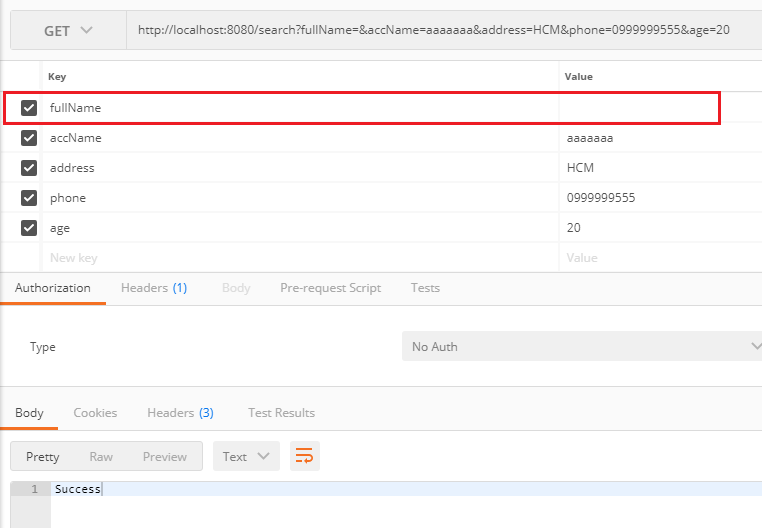
EmployeeController.java

|  |
| --- |
| @RestController  public class EmployeeController {  @GetMapping("/search")  public String add(@Valid @ModelAttribute  final EmployeeSearchRequest employeeSearchRequest,  final BindingResult bindingResult) {  if(bindingResult.hasErrors()) {  return getErrorMessage(bindingResult);  }  return "Success";  }  private String getErrorMessage(final BindingResult bindingResult) {  List<String> errMessages = new ArrayList<>();  bindingResult.getFieldErrors().forEach(  f -> errMessages.add(f.getField() + " " + f.getDefaultMessage()));  return String.join("\r\n", errMessages);  }  } |

Dùng default message thì phải get field name “**f.getField()”**







Ví dụ dùng Custom message

EmployeeSearchRequest.java

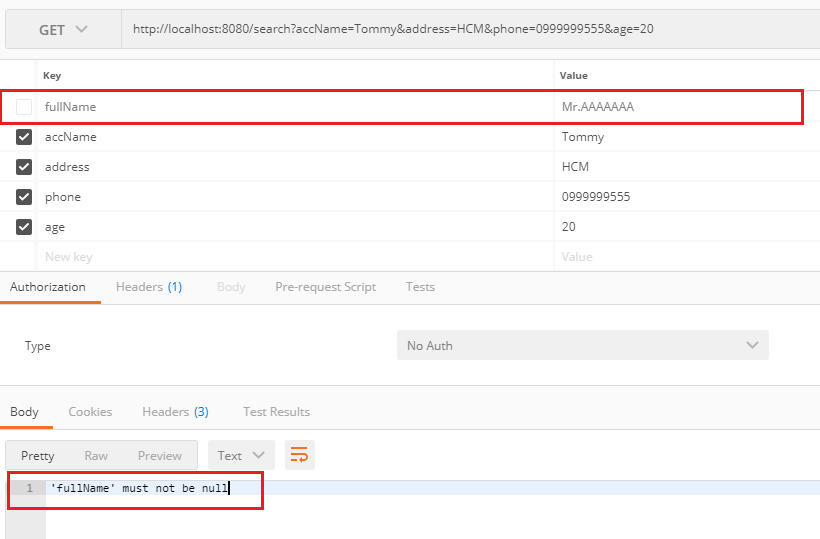
|  |
| --- |
| @Data  public class EmployeeSearchRequest {  @NotNull(message = "'fullName' must not be null")  private String fullName;  private String accName;  private String address;  private String phone;  } |

Theo đoạn code trên chỉ validate cho **“fullName” NotNull** với message ['fullName' must not be null]

EmployeeController.java

|  |
| --- |
| @RestController  public class EmployeeController {  @GetMapping("/search")  public String add(@Valid @ModelAttribute  final EmployeeSearchRequest employeeSearchRequest,  final BindingResult bindingResult) {  if(bindingResult.hasErrors()) {  return getErrorMessage(bindingResult);  }  return "Success";  }  private String getErrorMessage(final BindingResult bindingResult) {  List<String> errMessages = new ArrayList<>();  bindingResult.getFieldErrors().forEach(f -> errMessages.add(f.getDefaultMessage()));  return String.join("\r\n", errMessages);  }  } |

Khi đó không cần get field name: “**f.getField()”**

****

* ***Check String Blank***

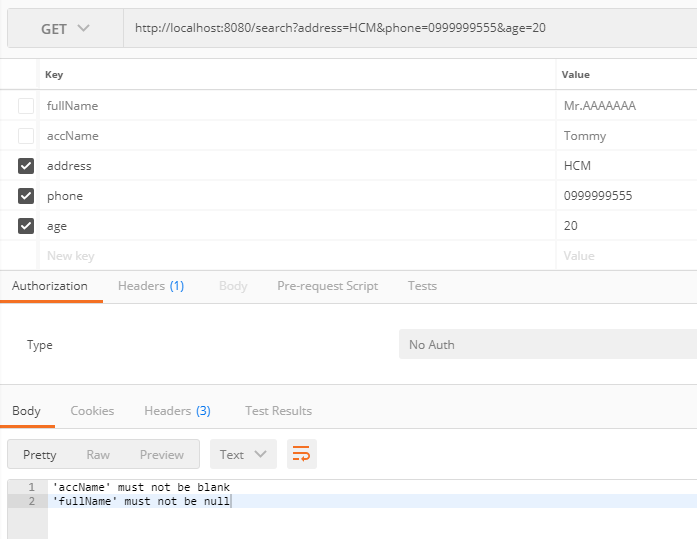
|  |  |
| --- | --- |
| @NotBlank | Default message |
| @ NotBlank (message = "accName must not be blank") | Custom message |

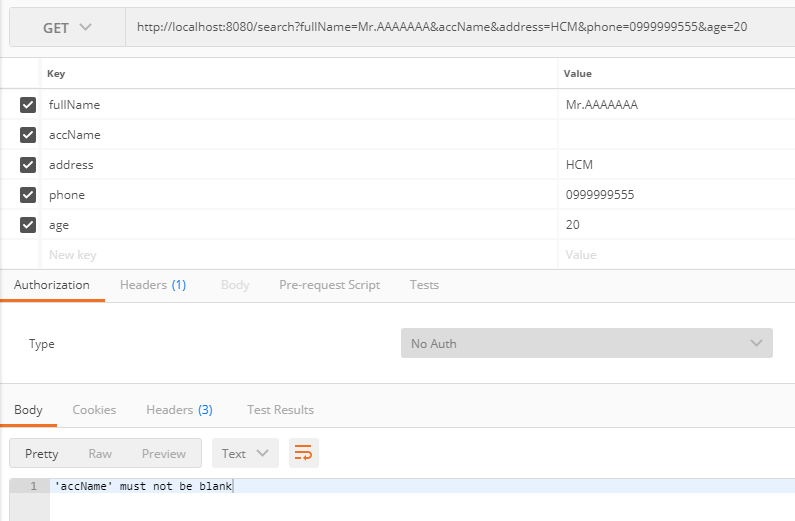
**Lưu ý:** Chỉtrường hợp String có ít nhất 1 ký tự thì mới hợp lệ. String null và empty không hợp lệ

EmployeeSearchRequest.java

|  |
| --- |
| @Data  public class EmployeeSearchRequest {  @NotNull(message = "'fullName' must not be null")  private String fullName;  @NotBlank(message = "'accName' must not be blank")  private String accName;  private String address;  private String phone;  } |

EmployeeController.java -> như trên





* ***Check String Length***

|  |  |  |
| --- | --- | --- |
| @Length(min = 5) | Default message: [length must be between 5 and 2147483647] | Nếu không định nghĩa max thì ứng dụng sẽ dùng giá trị lớn nhất là 2147483647 (max integer) |
| @Length(max = 10) | Default message: [length must be between 0 and 10] | Nếu không định nghĩa min thì ứng dụng sẽ dùng giá trị nhỏ nhất là zero |
| @Length(min = 5, max = 10) | Default message: [length must be between 5 and 10] |  |

**Lưu ý:**

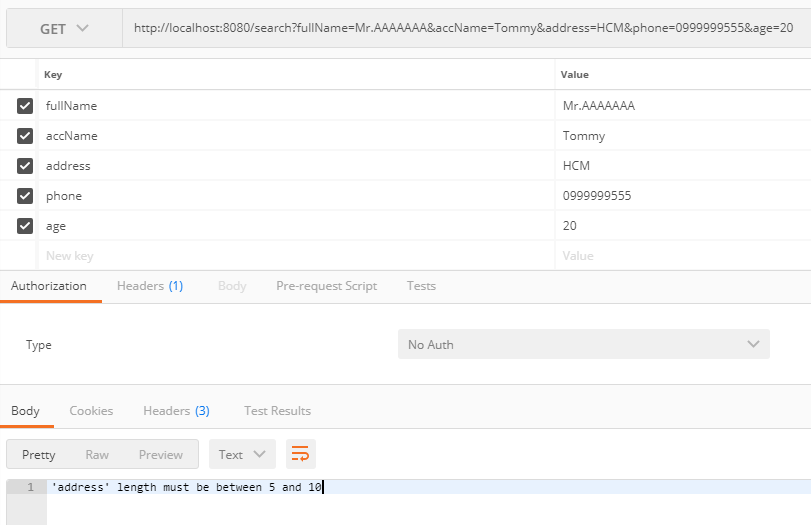
+ Nếu định nghĩa cùng 1 loại annotation trên 1 item thì sẽ chỉ check annotation sau cùng, các annotation cùng loại trước đó vô nghĩa. VD @Length, nếu chia min và max ra thành 2 dòng @Length thì chỉ nhận check annotation sau cùng

+ Để check Range cho parameter, ngoài @Length thì có thể dùng @Size, cách sử dụng hoàn toàn giống nhau (Ở ví dụ trên có thể thay thế @Length thành @Size). Vậy tùy theo quan điểm và quy định chung của dự án mà chọn annotation hợp lý. Theo chuẩn chung thì @Length dùng để check độ dài của chuỗi ký tự, @Size dùng để check độ dài Array

+ Có thể dùng **custom message** như ví dụ trên bằng cách thêm argument “message” vào annotation

EmployeeSearchRequest.java

|  |
| --- |
| @Data  public class EmployeeSearchRequest {  @NotNull(message = "'fullName' must not be null")  private String fullName;  @NotBlank(message = "'accName' must not be blank")  private String accName;    //@Length(min = 5)  //@Length(max = 10)  @Length(min = 5, max = 10, message = "'address' length must be between 5 and 10")  private String address;  private String phone;  } |



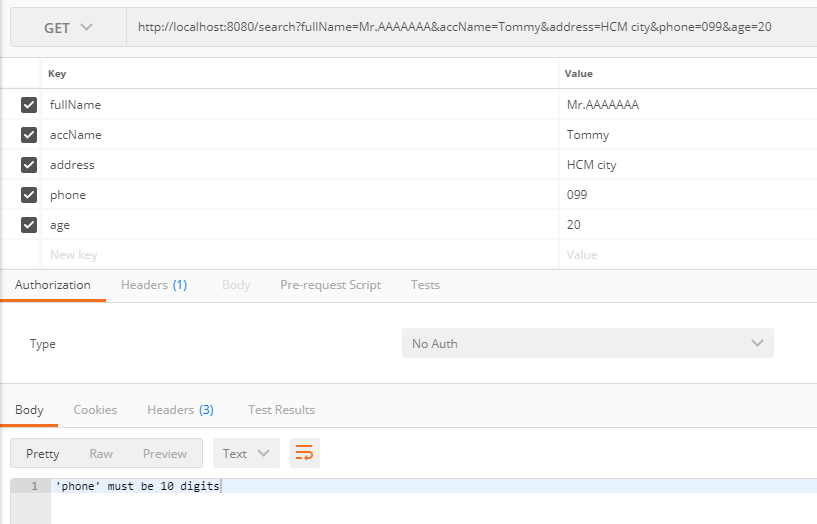
* ***Check String Format***

|  |  |
| --- | --- |
| @Pattern(regexp="(^$|[0-9]{10})") | Default message: [must match "(^$|[0-9]{10})"] |
| @Pattern(regexp="(^$|[0-9]{10})", message = "'phone' must be 10 digits") | Custom message |

**Lưu ý:** Java có hỗ trợ annotation cho Email format chuẩn: @Email

EmployeeSearchRequest.java

|  |
| --- |
| @Data  public class EmployeeSearchRequest {  @NotNull(message = "'fullName' must not be null")  private String fullName;  @NotBlank(message = "'accName' must not be blank")  private String accName;  //@Length(min = 5)  //@Length(max = 10)  @Length(min = 5, max = 10, message = "'address' length must be between 5 and 10")  private String address;  @Pattern(regexp="(^$|[0-9]{10})", message = "'phone' must be 10 digits")  private String phone;  } |



* + 1. **Validation for Number**
* ***Binding Number***

Đối với parameter là number, nếu giá trị truyền vào không phải là digit, thì sẽ xãy ra **NumberFormatException.** Và exception này đã được hệ thống try-cache và xử lý trong quá trình parser request parameter, nên không thể cache **NumberFormatException** trong quá trình này

Khi đó BindingResult sẽ sử dụng error message của **NumberFormatException**

* Đây là 1 nguyên nhân gây ra lỗ hổng security, giúp hacker dễ dàng xâm nhập vào hệ thống (bằng cách điều tra các error message default của hệ thống)

CustomerSearchRequest.java

|  |
| --- |
| @Data  public class CustomerSearchRequest {  private int age;  private int cusType; // only: 0 | 1  } |

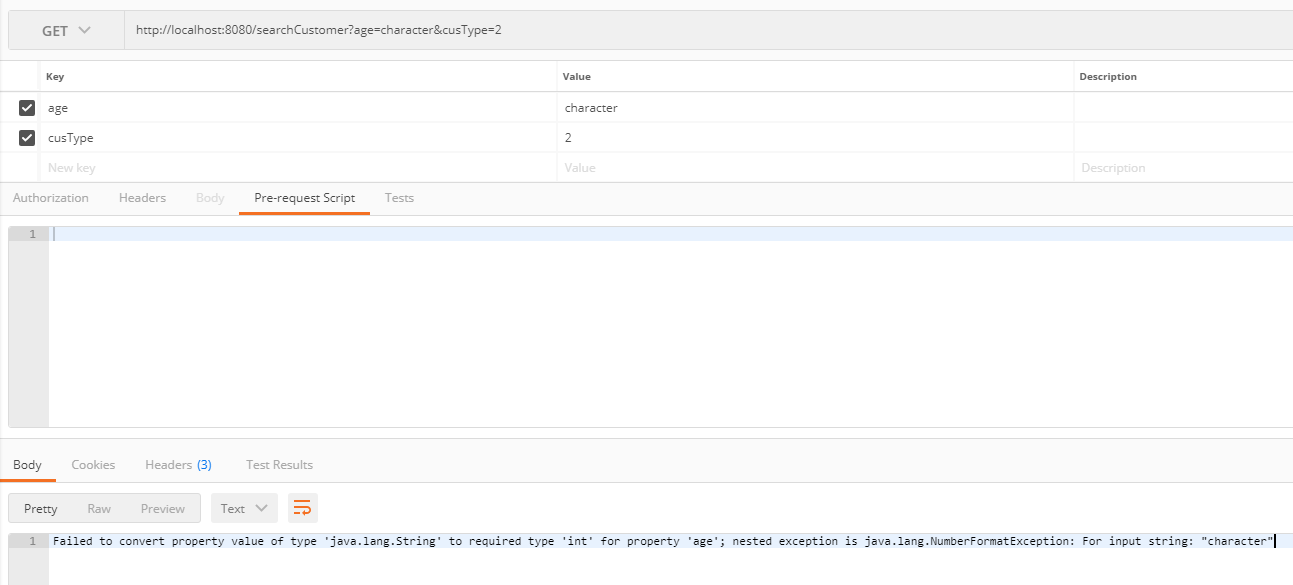
Ở object Request không define bất kì nội dung check validation nào hết

EmployeeController.java

|  |
| --- |
| @GetMapping("/searchCustomer")  public String searchCustomer(@Valid final CustomerSearchRequest request,  final BindingResult bindingResult) {  if(bindingResult.hasErrors()) {  return getErrorMessage(bindingResult);  }  return "Success";  } |

Nhưng BindingResult vẫn xử lý khi xảy ra NumberFormatException

Khi đó BindingResult sẽ sử dụng error message của NumberFormatException



* Ví dụ dưới đây cho thấy không thể cache NumberFormatException trong quá trình parse request parameter

EmployeeController.java

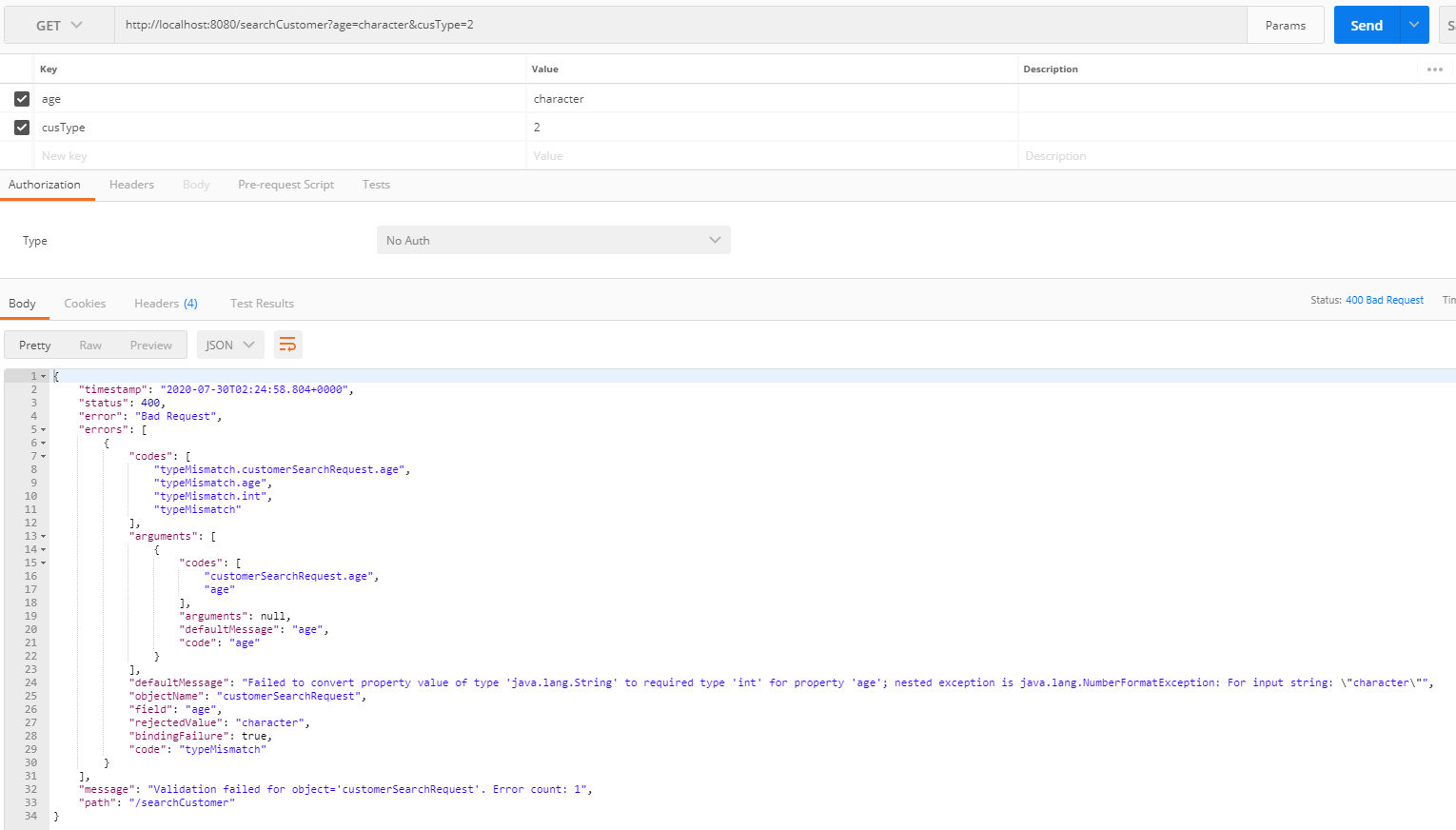
|  |
| --- |
| @GetMapping("/searchCustomer")  public String searchCustomer(@Valid final CustomerSearchRequest request,  final BindingResult bindingResult) {  try {  if(bindingResult.hasErrors()) {  return getErrorMessage(bindingResult);  }  return "Success";  } catch (Exception ex) {  return "Exception";  }  } |

Dù có try-cache thì cũng không thể cache được vì exception này đã được xử lý trong quá trình parse request parameter

* Ví dụ dưới đây cho thấy không thể cache NumberFormatException dù cho không sử dụng BindingResult

EmployeeController.java

|  |
| --- |
| @GetMapping("/searchCustomer")  public String searchCustomer(final CustomerSearchRequest request) {  try {  return "Success";  } catch (Exception e) {  return "Exception";  }  } |



Khi không dùng BindingResult thì hệ thống sẽ return default Error response chứa thông tin của NumberFormatException

* **Cách xử lý:**
* C1: Check format đầu vào
* NotBlank
  + Chỉ chấp nhận digit
  + Giới hạn Length (tùy vào kiểu dữ liệu thì sẽ có max value khác nhau)
* C2: Define lại error message: bằng cách filter BindingResult để lấy ra các error về parsing number để update lại nội dung message. Theo ví dụ trên thì mình cần phải xử lý lại method getErrorMessage() để update lại các message về default NumberFormatException:
* Dùng FieldError.bindingFailure để check error đó có phải là binding number error:

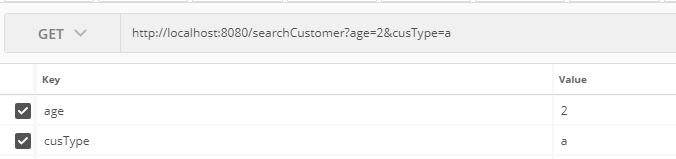
|  |  |  |
| --- | --- | --- |
| FieldError.bindingFailure | boolean | True: lỗi trong quá trình binding number  False: không phải lỗi binding number |

Dưới đây là vì dụ phân tích 2 trường hợp FieldError. bindingFailure

CustomerSearchRequest.java

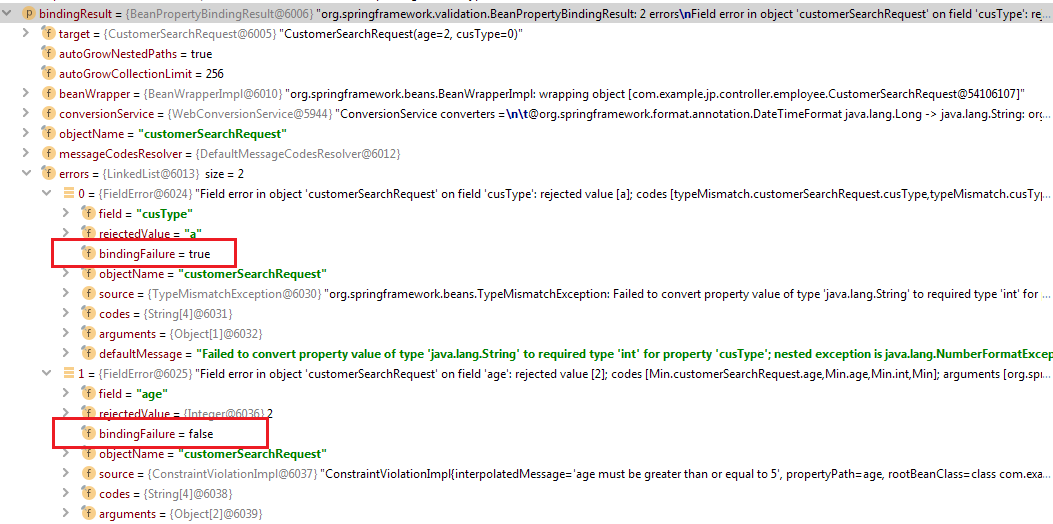
|  |
| --- |
| @Data  public class CustomerSearchRequest {  @Min(value = 5, message = "age must be greater than or equal to 5")  private int age;  private int cusType; // only: 0 | 1  } |

Theo đoạn code trên: “age” phải có gí trị nhỏ nhất là 5

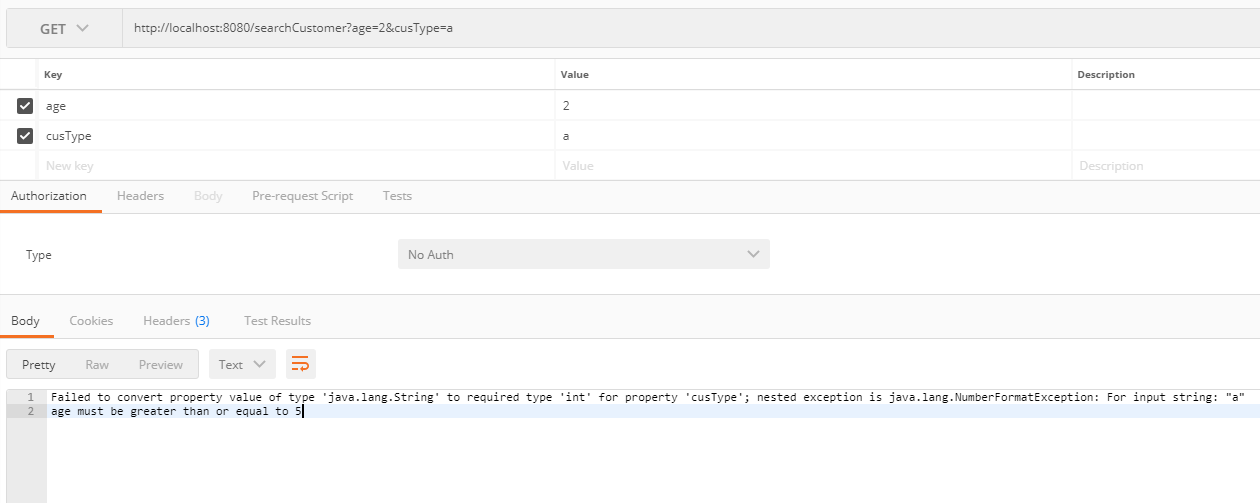


Theo request input như trên thì sẽ nhận được 2 FieldError:

* “age”: Error do giá trị input < 5 -> khi đó FieldError. bindingFailure = false
* “cusType”: Error do binding number -> khi đó FieldError. bindingFailure = true

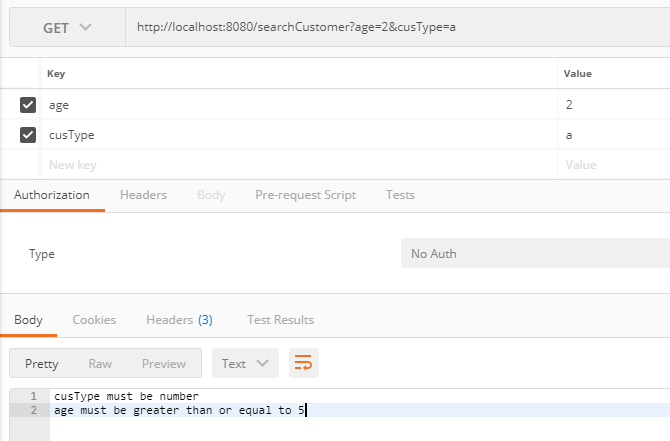


Khi đó kết quả response



Update method *getErrorMessage() ->* để thay thế message binding number thành custom message

|  |
| --- |
| private String getErrorMessage(final BindingResult bindingResult) {  List<String> errMessages = new ArrayList<>();  List<FieldError> fieldErrors = bindingResult.getFieldErrors();  fieldErrors.forEach(error -> errMessages.add(**getMessage(error)**));  return String.join("\r\n", errMessages);  }    private String **getMessage**(final FieldError error) {  return !error.isBindingFailure() ? error.getDefaultMessage() : error.getField() + " " + "must be number";  } |

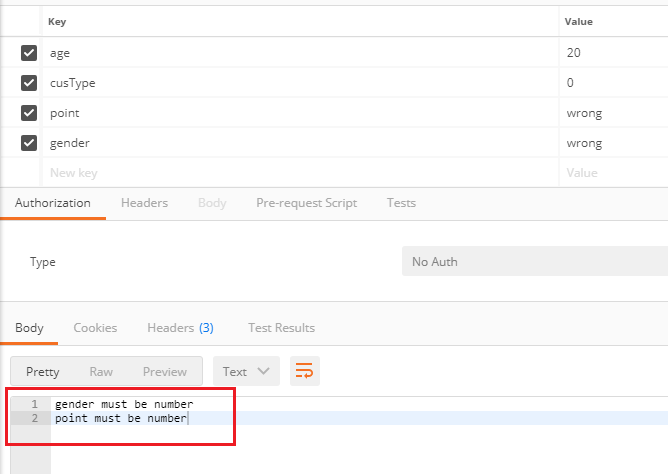


* ***Binding Boolean***

Đối với parameter là boolean, nếu giá trị truyền vào không phải là : true, false, 0 , 1 -> thì sẽ xãy ra lỗi binding tương tự như number

* **Cách xử lý: tương tự như xử lý biniding number**
* C1: Check format đầu vào
* NotBlank
  + Chỉ chấp nhận 1 trong 4 giá trị: true, false, 0 , 1
* C2: Define lại error message: bằng cách filter BindingResult để lấy ra các error về binding để update lại nội dung message

Vẫn giữ nguyên xử lý code của binding number và test kết quả thực tế



Rõ ràng Error message về ‘gender’ chưa chính xác

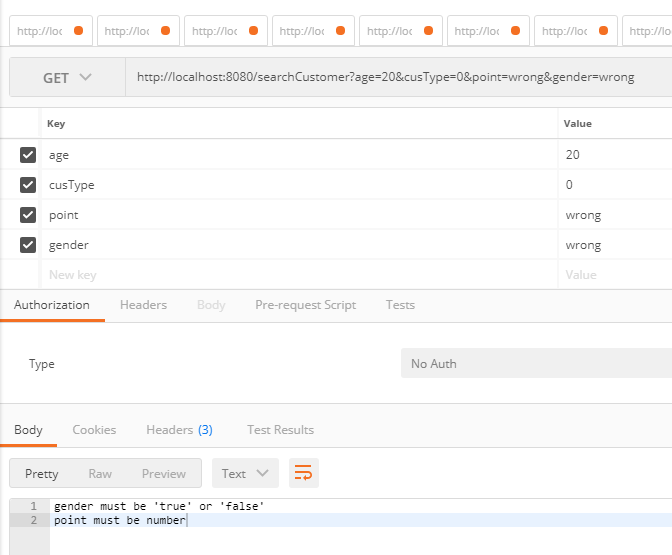
-> Ví dụ 1 cách để giải quyết trường hợp này: Cần tạo 1 **HashMap** để chứa các custom message tương ứng với từng field

- key: field name

- value: error message

EmployeeController.java

|  |
| --- |
| private String getErrorMessage(final BindingResult bindingResult) {  List<String> errMessages = new ArrayList<>();  List<FieldError> fieldErrors = bindingResult.getFieldErrors();  fieldErrors.forEach(error -> errMessages.add(getMessage(error)));  return String.join("\r\n", errMessages);  }  private HashMap<String, String> errFieldMap = new HashMap<String, String>(){{  put("age", "age must be number");  put("cusType", "cusType must be number");  put("point", "point must be number");  put("gender", "gender must be 'true' or 'false'");  }};  private String getMessage(final FieldError error) {  return !error.isBindingFailure() ? error.getDefaultMessage() : errFieldMap.get(error.getField());  } |

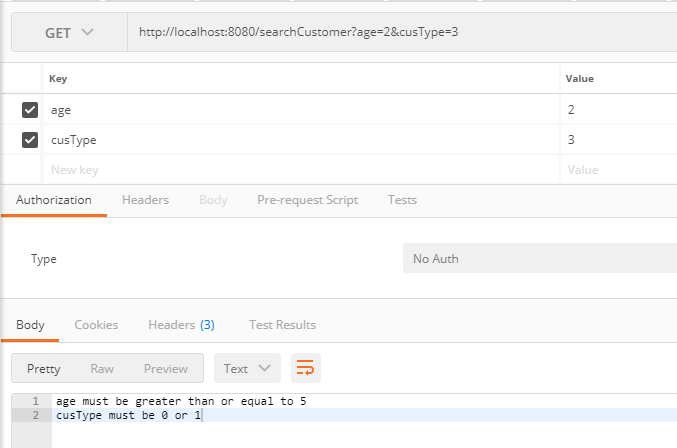


* ***Check Min, Max***

|  |  |
| --- | --- |
| @Min(5) | Default message: [must be greater than or equal to 5] |
| @Min(value = 5, message = "age must be greater than or equal to 5") | Custom message |
| @Max(10) | Default message: [must be less than or equal to 10] |
| @Max(value = 1, message = "cusType must be 0 or 1") | Custom message |

CustomerSearchRequest.java

|  |
| --- |
| @Data  public class CustomerSearchRequest {  @Min(value = 5, message = "age must be greater than or equal to 5")  private int age;  @Min(value = 0, message = "cusType must be 0 or 1")  @Max(value = 1, message = "cusType must be 0 or 1")  private int cusType; // only: 0 | 1  } |



* ***Check Range***

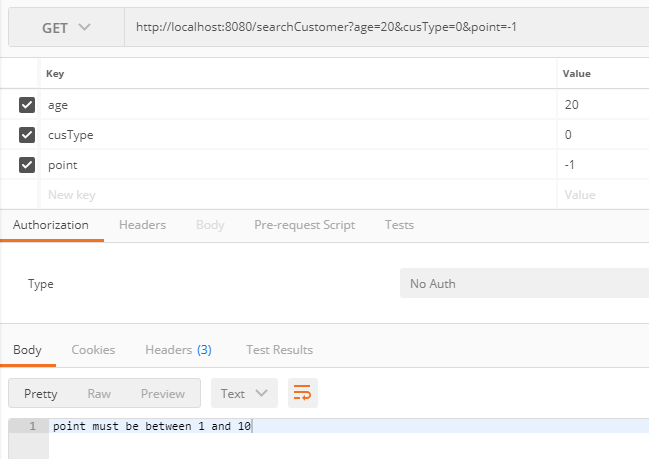
Để check min & max cho 1 number có thể dùng **@Range**

|  |  |
| --- | --- |
| **Arguments** | **Default value** |
| min | 0 |
| max | 9,223,372,036,854,775,807(max Long) |
| message | “must be between 0 and 9223372036854775807” |

VD: @Range(min = 1, max = 10, message = "point must be between 1 and 10")

CustomerSearchRequest.java

|  |
| --- |
| @Data  public class CustomerSearchRequest {  @Min(value = 5, message = "age must be greater than or equal to 5")  private int age;  @Min(value = 0, message = "cusType must be 0 or 1")  @Max(value = 1, message = "cusType must be 0 or 1")  private int cusType; // only: 0 | 1  @Range(min = 1, max = 10, message = "point must be between 1 and 10")  private int point; // form 1 -> 10  } |



* + 1. **Validation for List**

**Annotation @Valid** **áp dụng chuẩn JSR-303** annotation, và JSR-303 được dùng để validation cho JavaBeans.

**Java.util.List** không phải là JavaBean -> do đó, nó không thể được validated trực tiếp bằng cách sử dụng JSR-303 validator

Tham khảo thông tin tại **Section [3.1.3. Graph validation]** – tài liệu **JSR-303 Bean Validation** [https://beanvalidation.org/1.0/spec/#constraintdeclarationvalidationprocess-requirements](https://beanvalidation.org/1.0/spec/%23constraintdeclarationvalidationprocess-requirements)

Do đó để check validation cho java List object thì cần phải đóng gói request body vào object

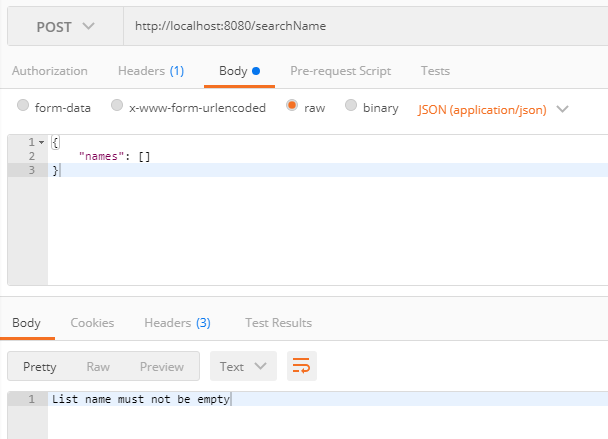
* ***Check List empty***

EmployeeNamesRequest.java

|  |
| --- |
| @Data  public class EmployeeNamesRequest {  @NotEmpty(message = "List name must not be empty")  private List<String> names;  } |

EmployeeController.java

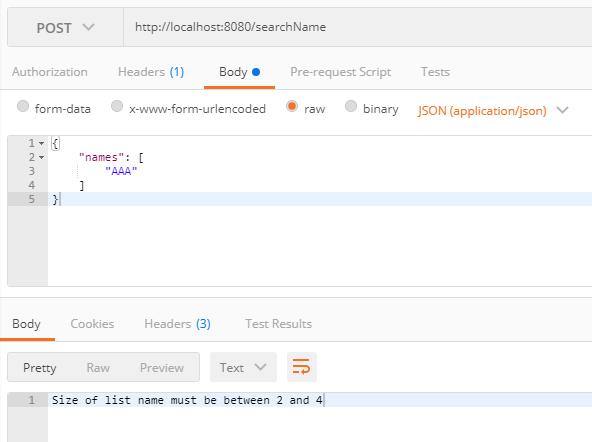
|  |
| --- |
| @PostMapping("/searchName")  public String searchName(@Valid @RequestBody EmployeeNamesRequest requests,  final BindingResult bindingResult) {  if(bindingResult.hasErrors()) {  return getErrorMessage(bindingResult);  }  return "Success ";  } |



* ***Check List Size***

EmployeeNamesRequest.java

|  |
| --- |
| @Data  public class EmployeeNamesRequest {  @Size(min = 2, max = 4, message = "Size of list name must be between 2 and 4")  private List<String> names;  } |

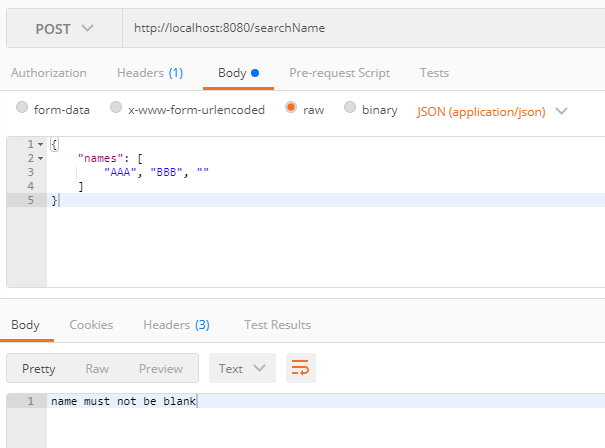


* ***Check List item***

VD: check từng item “name” trong list không được blank

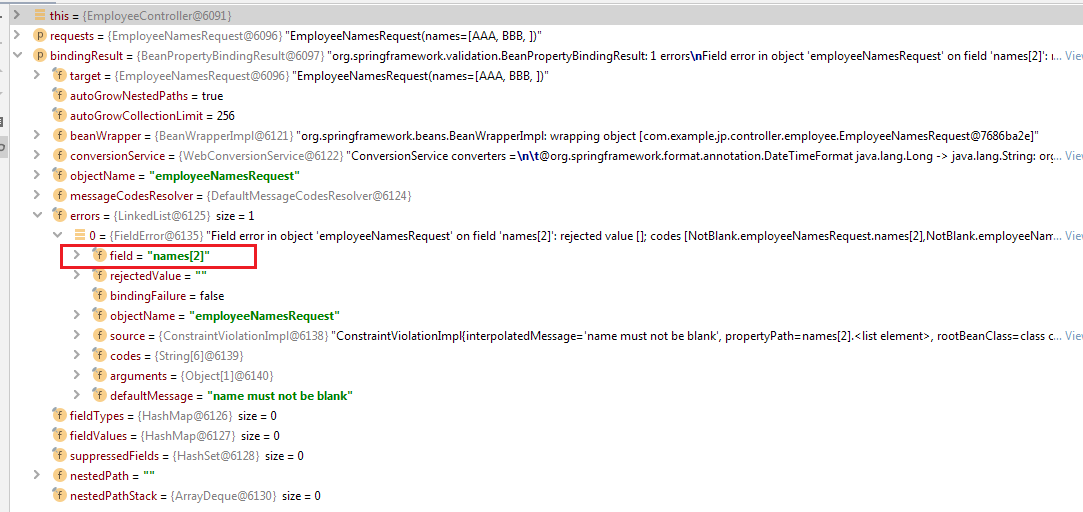
EmployeeNamesRequest.java

|  |
| --- |
| @Data  public class EmployeeNamesRequest {  @Size(min = 2, max = 4, message = "Size of list name must be between 2 and 4")  private List<@NotBlank(message = "name must not be blank") String> names;  } |



**Lưu ý:** Có thể lấy chính xác index của item không hợp lệ, bằng cách xử lý BindingResult.

Dưới đây là nội dung debug của BindingResult



* + 1. **Validation for Date format**

Giá trị được gửi từ request luôn được định dạng là String

Do đó để parse 1 giá trị request vào Date parameter thì phải định nghĩa **DateTimeFormat** cho parameter đó.

EmployeeSearchRequest.java

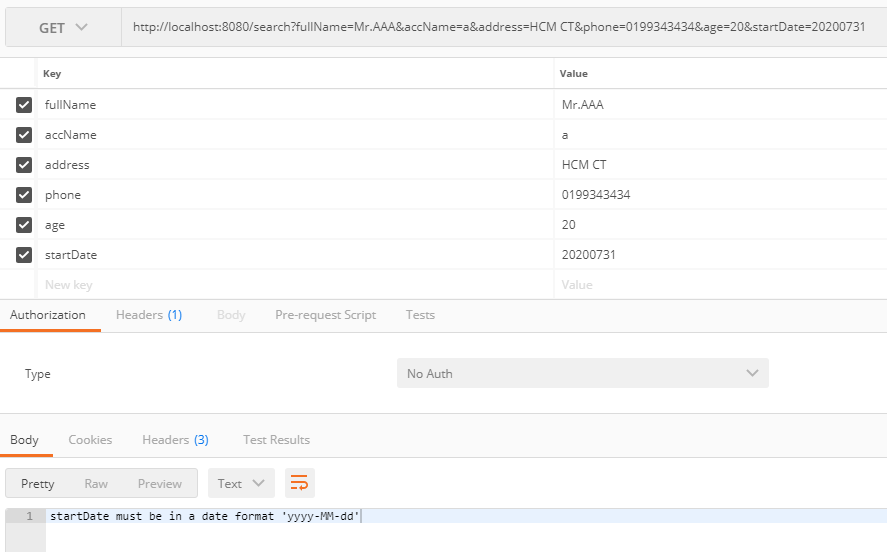
|  |
| --- |
| @Data  public class EmployeeSearchRequest {  @NotNull(message = "'fullName' must not be null")  private String fullName;  @NotBlank(message = "'accName' must not be blank")  private String accName;  @Length(min = 5, max = 10, message = "'address' length must be between 5 and 10")  //@Length(min = 5)  //@Length(max = 10)  private String address;  @Pattern(regexp="(^$|[0-9]{10})", message = "'phone' must be 10 digits")  private String phone;  //@DateTimeFormat(pattern="yyyy-MM-dd")  private Date startDate;  } |

Khi đó, nếu truyền vào giá trị không đúng format “yyyy-MM-dd” thì sẽ xảy ra **IllegalArgumentException** trong quá trình Binding parameter – BindingFailure = true

* Cách xử lý cũng tương tự như: Number, boolean

EmployeeController.java

|  |
| --- |
| private HashMap<String, String> errFieldMap = new HashMap<String, String>(){{  put("age", "age must be number");  put("cusType", "cusType must be number");  put("point", "point must be number");  put("gender", "gender must be 'true' or 'false'");  put("startDate", "startDate must be in a date format 'yyyy-MM-dd'");  }};  private String getMessage(final FieldError error) {  return !error.isBindingFailure() ? error.getDefaultMessage() : errFieldMap.get(error.getField());  } |



**Lưu ý:** Nếu không define Annotation @DateTimeFormat thì hệ thống luôn throw exception **IllegalArgumentException** dù có input bất kì giá trị nào (**trừ trường hợp NULL**)

* 1. **Check validate trên Controller argument**

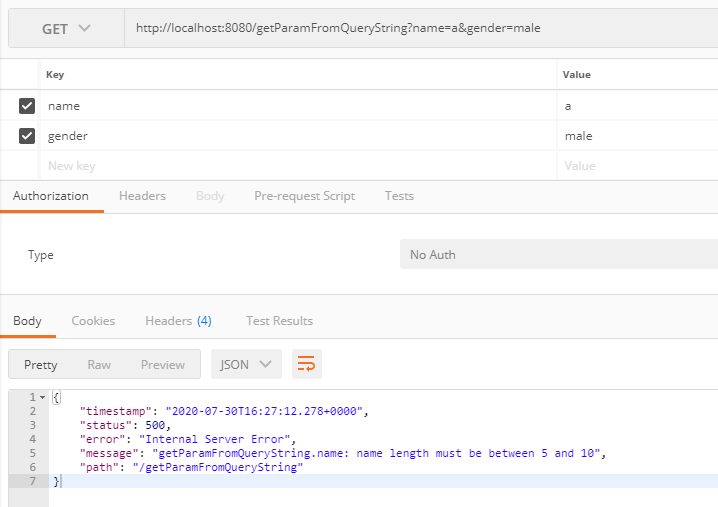
Khi validate trên controller argument thì không thể dùng **BindingResult**. Vì vậy cần phải define Exception handle để nhận exception khi validation fail

StudentController.java

|  |
| --- |
| @GetMapping("/getParamFromQueryString")  public String getParamFromQueryString(  @Length(min = 5, max = 10, message = "name length must be between 5 and 10") @RequestParam final String name,  @RequestParam(required = false) final String gender) {  return name + " - " + gender;  } |

Lưu ý: phải thêm Annotation **@Validated** ở đầu Class Controller để bật mode check validate cho parameter

Kết quả run



Spring boot tự return ra Error Response như hình trên

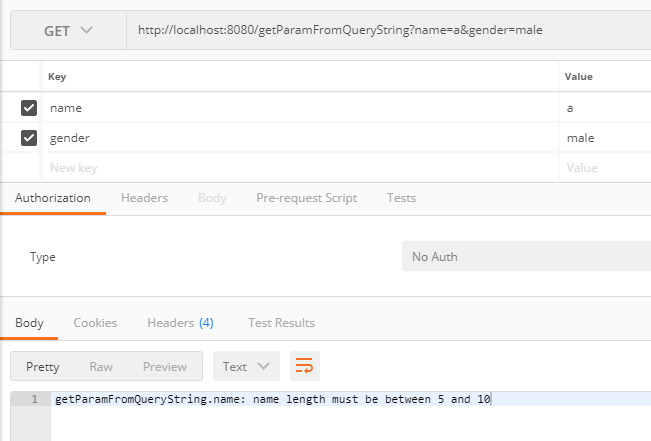
Vậy cần phải xử lý **Exception handler** để nhận Exception khi validation fail: Constraintviolationexception

ApiExceptionHandler.java

|  |
| --- |
| @RestControllerAdvice  public class ApiExceptionHandler extends ResponseEntityExceptionHandler {  @ExceptionHandler(ConstraintViolationException.class)  @ResponseStatus(value = HttpStatus.BAD\_REQUEST)  public String todoException(ConstraintViolationException ex) {  return ex.getMessage();  }  } |

@RestControllerAdvice là phương pháp quản lý , nhận tất cả các exception được Spring boot Throw ra

* Sẽ giải thích rõ ở nội dung: Quản lý Exception



# **Normal API processing Flow**

* 1. **Method GET**

API Search products với nội dung validation parameter như sau:

* “name”: Not Blank
* “price”: min = 100, max = 2000000000
* “secondhand”: True: xe cũ / False: xe mới
* “registryDate”: Đối với xe mới thì chưa có ngày đăng ký, xe cũ thì phải đi kèm ngày đăng ký
  + 1. **Validation Request**

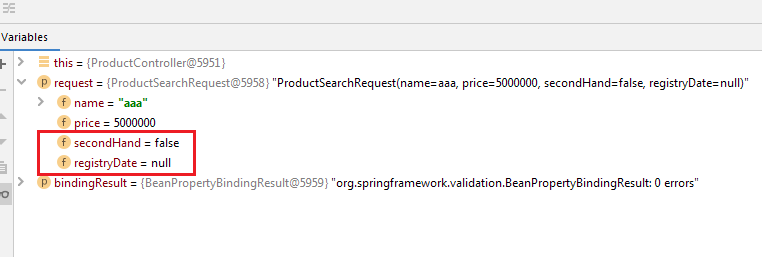
Đối với việc Validation cho registryDate thì có 2 cách:

1. Update lại method Setter của registryDate: nếu là xe mới thì không set giá trị cho “registryDate”

2. Dùng Domain check ProductRequest trước khi gửi xuống repository

**Cách 1**: ProductSearchRequest.java

|  |
| --- |
| @Data  public class ProductSearchRequest {  @NotBlank(message = "name must not be blank")  private String name;  @Range(min = 100, max = 2000000000, message = "price must be greater than or equal to 100 USD")  private int price;  private boolean secondHand;  @DateTimeFormat(pattern="yyyy-MM-dd")  private Date registryDate;  public void setRegistryDate(Date registryDate) {  if(isSecondHand()) {  this.registryDate = registryDate;  }  }  } |



**Cách 2**: ProductSearchRequest.java

|  |
| --- |
| @Data  public class ProductSearchRequest {  @NotBlank(message = "name must not be blank")  private String name;  @Range(min = 100, max = 2000000000, message = "price must be greater than or equal to 100 USD")  private int price;  private boolean secondHand;  @DateTimeFormat(pattern="yyyy-MM-dd")  private Date registryDate;  } |

Khi đó dựa vào ProductSearchRequest để tạo ra Domain object Product và xử lý validate dựa theo Requirement. Trường hợp validate fail thì throw IllegalArgumentException.

Ở ApiExceptionHandler.java sẽ handle exception này

Product.java

|  |
| --- |
| public Product(final String name, final int price, final boolean secondHand, final Date registryDate) {  this.name = name;  this.price = price;  this.secondHand = secondHand;  // Check condition before setting value for registryDate -> Check fail throw IllegalArgumentException  if((!secondHand && !ObjectUtils.isEmpty(registryDate)) ||  (secondHand && ObjectUtils.isEmpty(registryDate))) {  throw new IllegalArgumentException("Invalid registryDate");  }  this.registryDate = registryDate;  } |

* + 1. **Response**

Thông thường 1 ứng dụng chỉ cần validate dữ liệu đầu vào

Nhưng đối với 1 ứng dụng an toàn thì phải có đầy đủ Validation cho input/output. Vậy validate output như thế nào?

Đối với 1 API (Method Get) thì data output chính là thông tin Response -> cần phải check Domain trước khi khởi tạo Product.java

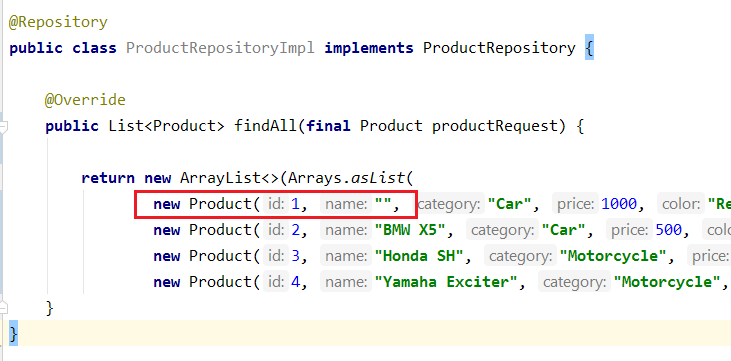
Product.java -> edit lại All argument constructor

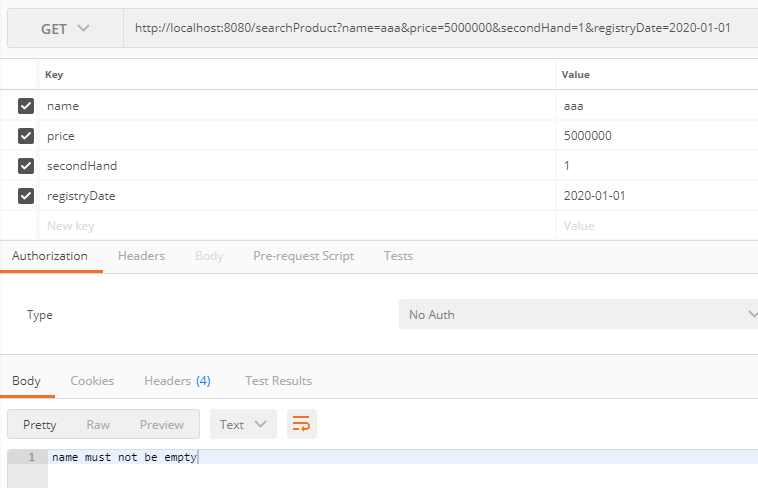
|  |
| --- |
| public Product(int id, String name, String category, int price, String color, boolean secondHand, int year, Date registryDate) {  // Validate output base on Database constrain. Ex: check require not null  Preconditions.checkArgument(!StringUtils.isEmpty(name), "name must not be empty");  Preconditions.checkArgument(price >= 100, "price must be greater than or equal 100");  // Others checking ...  this.id = id;  this.name = name;  this.category = category;  this.price = price;  this.color = color;  this.secondHand = secondHand;  this.year = year;  this.registryDate = registryDate;  } |

**Preconditions.checkArgument(Boolean, message)**

* False -> throw IllegalArgumentException với message truyền vào
* Import dependency:

|  |
| --- |
| compile group: 'com.google.guava', name: 'guava', version: '29.0-jre' |





* 1. **Method POST**

API Add new Product

* + 1. **Validation Request Body (Validate input)**
    2. **Validation Domain Product (Validate output)**

Output của API method Post là Object trước khi insert/update xuống DB

Do đó Validate Output được thưc hiện ở step Factory từ object RequestBody -> domain Product.java

# **Exception Handler**

* 1. **RestCpntrollerAdvice**
* Spring boot có hỗ trợ handler tất cả các Exception trong cùng 1 class với annotation @RestControllerAdvice
* Khi đó không cần phải try/cache ở nhiều nơi. Mà chỉ cần xác định rõ các exception có khả năng xảy ra trong Flow xử lý của mình.
* Ví dụ vài Exception cho 1 flow xử cơ bản

|  |  |
| --- | --- |
| Access Http | HttpRequestMethodNotSupportedException |
| Get request & Validation input | IllegalArgumentException  InvalidParameterException  ConstraintViolationException  NumberFormatException |
| Validation Output | IllegalArgumentException |
| Connection to DB | ResourceAccessException |
| Database Error | DataAccessException |

Ngoài ra có thể định nghĩa Custom Exception. VD DataNotFoundException.java

|  |
| --- |
| public class DataNotFoundException extends Exception {  public DataNotFoundException(String message) {  super(message);  }  } |

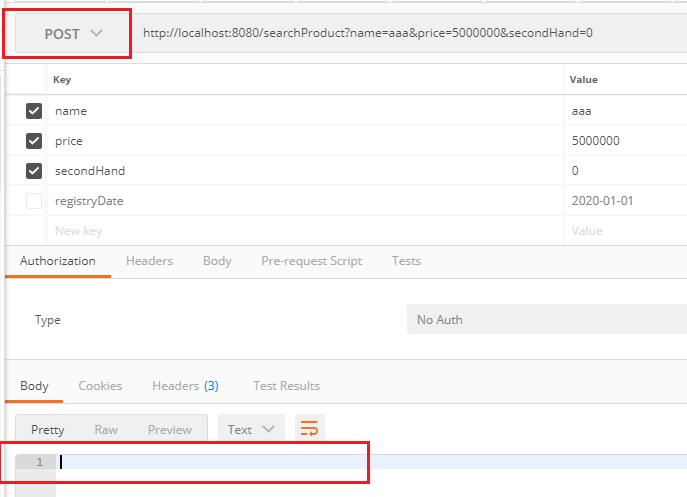
VD: ApiExceptionHandler.java

|  |
| --- |
| @RestControllerAdvice  public class ApiExceptionHandler extends ResponseEntityExceptionHandler {  @ExceptionHandler(Exception.class)  @ResponseStatus(value = HttpStatus.INTERNAL\_SERVER\_ERROR)  public ErrorResponse handleException(Exception ex) {  return new ErrorResponse(new ErrorResponse.Error(HttpStatus.INTERNAL\_SERVER\_ERROR.value(), ex.getMessage()));  }  } |

* 1. **HttpRequestMethodNotSupportedException**

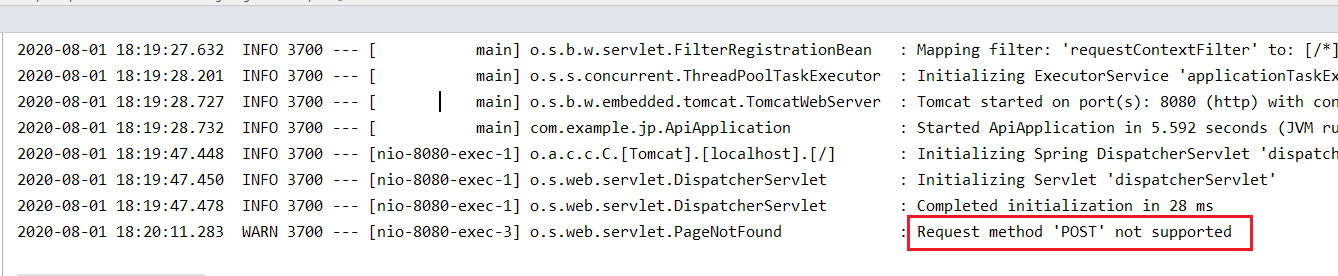
Đây là exception xử lý việc access vào API với method không đúng

Hãy xem khi access sai method thì hệ thống sẽ xử lý như thế nào nếu không handle @HttpRequestMethodNotSupportedException



Theo kết quả run như trên thì hệ thống không có response vì Spring Boot không xác được được Controller để xử lý cho request đó

Khi đó Spring Boot throw HttpRequestMethodNotSupportedException



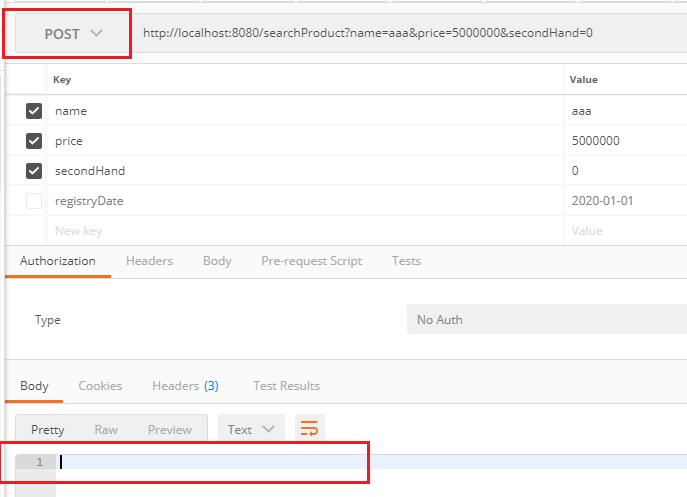
Khi đó hệ thống sẽ ghi log thông báo lỗi về Request Method

* Cần phải handler HttpRequestMethodNotSupportedException để return Error response hợp lý

Khi đó chỉ cần thêm 1 method handle HttpRequestMethodNotSupportedException trong @RestControllerAdvice

ApiExceptionHandler.java

|  |
| --- |
| **@**RestControllerAdvice  public class ApiExceptionHandler extends ResponseEntityExceptionHandler {  @ExceptionHandler(HttpRequestMethodNotSupportedException.class)  @ResponseStatus(HttpStatus.NOT\_FOUND)  public ErrorResponse handleHttpRequestMethodNotSupportedException(HttpRequestMethodNotSupportedException ex) {  return new ErrorResponse(new ErrorResponse.Error(HttpStatus.NOT\_FOUND.value(), ex.getMessage()));  }  } |



Và kết quả sau khi run: HttpRequestMethodNotSupportedException vẫn **chưa được handle**

**Nguyên nhân**:

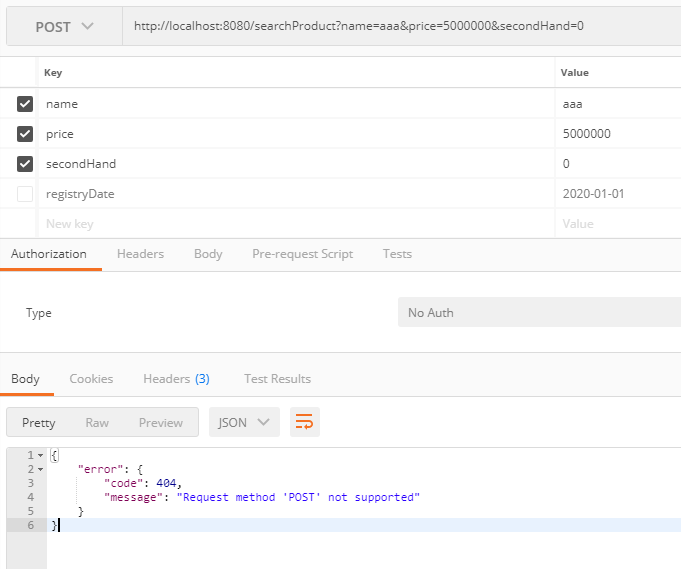
* Vì ApiExceptionHandler extends ResponseEntityExceptionHandler
* ResponseEntityExceptionHandler -> là cơ chế handler các exception xảy ra trong quá trình xử lý response (quá trình này bắt đầu từ Controller method)
* Khi access với method sai thì rõ ràng xử lý chưa vào controller. Do đó nó chưa đi vào vùng hoạt động của ResponseEntityExceptionHandler

**Cách giải quyết:**

* Chỉ cần RestControllerAdvice **KHÔNG** extends ResponseEntityExceptionHandler. Khi đó RestControllerAdvice không bị giới hạn phạm vi hoạt động

ApiEntryPointAccessHandler.java

|  |
| --- |
| @RestControllerAdvice  public class ApiEntryPointAccessHandler {  @ExceptionHandler(HttpRequestMethodNotSupportedException.class)  @ResponseStatus(HttpStatus.NOT\_FOUND)  public ErrorResponse handleHttpRequestMethodNotSupportedException(HttpRequestMethodNotSupportedException ex) {  return new ErrorResponse(new ErrorResponse.Error(HttpStatus.NOT\_FOUND.value(), ex.getMessage()));  }  } |



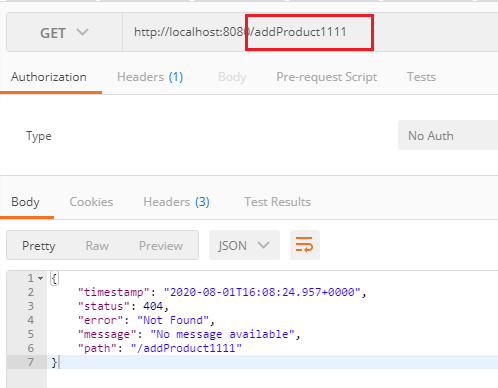
* 1. **Handle Error khi access API sai path (WebServerFactoryCustomizer & ErrorController)**
* **Flow xử lý 1 Exception:**
* Khi có 1 exception được throw, hệ thống sẽ xử lý exception đó tùy vào nội dung handler đã được define trước đó. VD:

+ try/cache: xử lý exception trực tiếp ngay vị trí xảy ra

+ RestControllerAdvice | ControllerAdvice: Nơi xử lý tập trung của các exception được throw

* Nếu trường hợp exception đó chưa được handle trong code thì spring boot sẽ return ra response mặc định ứng với HttpStatus error code (mỗi HttpStatus error có 1 response khác nhau).

VD 1 response mặc định của spring boot về HttpStatus NOT\_FOUND



* Cách thức hoạt động của Spring Boot: Trường hợp exception chưa được handle trong code thì Spring Boot sẽ tự access path request "/error" để xử lý các exception và return ra response mặc định tương ứng
* Vậy chỉ cần define lại nội dung controller của request path "/error" là đã handle được vấn đề này

ErrorPageController.java

|  |
| --- |
| @RestController  public class ErrorPageController implements ErrorController {  @RequestMapping("/error")  public ErrorResponse HandlerErrorNotFoundUrl() {  return new ErrorResponse(new ErrorResponse.Error(HttpStatus.NOT\_FOUND.value(), "Page Not Found"));  }  @Override  public String getErrorPath() {  return null;  }  } |

* Xử lý như trên thì đã giải quyết được vấn đề Hanlde **HttpStatus.NOT\_FOUND** khi access sai path. Nhưng nó cũng làm ảnh hưởng đến HttpStatus error khác.
* VD: Nếu xảy ra HttpStatus error khác như HttpStatus.BAD\_REQUEST, thì nó cũng return ra HttpStatus.NOT\_FOUND, bởi nó chỉ có 1 dòng code màu xanh để xử lý cho tất cả các error
* **Cách giải quyết gồm 2 step:**
* **Step 1**: Phân loại HttpStatus error -> bằng cách add thêm ErrorPage cho HttpStatus.NOT\_FOUND (nghĩa là trước đây chỉ có 1 errorPage “/error” xử lý cho tất cả HttpStatus, thì bây giờ mình tạo riêng 1 ErrorPage cho HttpStatus.NOT\_FOUND)

ErrorCustomizer.java

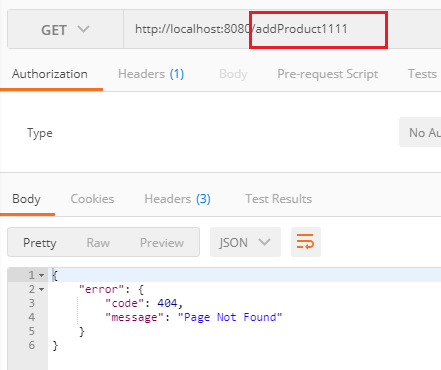
|  |
| --- |
| @Configuration  public class ErrorCustomizer implements WebServerFactoryCustomizer<ConfigurableWebServerFactory> {  @Override  public void customize(ConfigurableWebServerFactory factory) {  factory.addErrorPages(new ErrorPage(HttpStatus.NOT\_FOUND, "/404"));  }  } |

Với xử lý trên thì mình đã add thêm 1 ErrorPage "/404" cho HttpStatus.NOT\_FOUND

* **Step 2**: Tạo ErrorController cho request "/404"

ErrorPageController.java

|  |
| --- |
| @RestController  public class ErrorPageController implements ErrorController {  @RequestMapping("/404")  public ErrorResponse HandlerErrorNotFoundUrl() {  return new ErrorResponse(new ErrorResponse.Error(HttpStatus.NOT\_FOUND.value(), "Page Not Found"));  }  @Override  public String getErrorPath() {  return null;  }  } |



# **Filter**

Filter là step sàng lọc thông tin (chứng thực, phân quyền, …) trước khi vào controller.

Cách tạo Filter trong Spring Boot

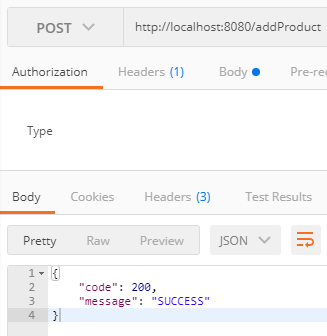
ApiFirstFilter.java

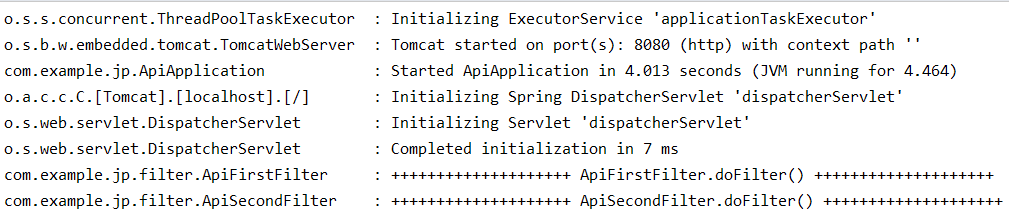
|  |
| --- |
| @Slf4j  @Component  @Order(1)  public class ApiFirstFilter implements Filter {  @Override  public void **init**(FilterConfig filterConfig) throws ServletException {  // Do something ...  }  @Override  public void **doFilter**(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain)  throws IOException, ServletException {  HttpServletRequest request = (HttpServletRequest) servletRequest;  HttpServletResponse response = (HttpServletResponse) servletResponse;  // Do something: Check authentication, authorization, ...  String header = request.getHeader("accessToken");  // ...  log.info("++++++++++++++++++++ ApiFirstFilter.doFilter() ++++++++++++++++++++");  //call next filter in the filter chain  filterChain.doFilter(request, response);  }  @Override  public void **destroy**() {  // Do something ...  }  } |

* @Component: đánh dấu class ApiFirstFilter thuộc sự quản lý của Spring boot
* @Order(1): thứ tự ưu tiên các Filter class. Trường hợp có nhiều filter thì mỗi filter là 1 class riêng. Nếu chỉ có 1 filter thì không cần @Order
* @Slf4j: là annotation phục vụ việc ghi log (không liên quan đến Filter)
* Interface Filter có 3 method, nhưng chỉ cần xử lý method doFilter() là đủ, 2 method còn lại nếu không cần xử lý thì có thể cho qua.
* Thứ tự ưu tiên các method: init() -> doFilter() -> destroy()
* filterChain: là đối tượng chứa tất cả các filter tong đó

ApiSecondFilter.java

|  |
| --- |
| @Slf4j  @Component  @Order(2)  public class ApiSecondFilter implements Filter {  @Override  public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain)  throws IOException, ServletException {  HttpServletRequest request = (HttpServletRequest) servletRequest;  HttpServletResponse response = (HttpServletResponse) servletResponse;  // Do something: Check authentication, authorization, ...  String header = request.getHeader("accessToken");  // ...  log.info("++++++++++++++++++++ ApiSecondFilter.doFilter() ++++++++++++++++++++");  //call next filter in the filter chain  filterChain.doFilter(request, response);  }  } |





**Lưu ý:**

* Trường hợp không đánh order cho Filter thì sẽ run theo thứ tự alphabet của Filter name
* Spring boot sẽ ưu tiên run các Filter có đánh @Order trước. Các filter không đánh @Order sẽ ưu tiên sau và theo thứ tự alphabet class name
* Các Filter cùng thứ tự @Order (VD cả 2 filter cùng có @Order(1)) thì độ ưu tiên giữa 2 Filter này dựa theo thứ tự alphabet class name
* filterChain.doFilter(request, response) là lệnh để run đối tượng kế tiếp
  + Nếu ApiFirstFilter không có filterChain.doFilter(request, response) thì sẽ không di chuyển qua ApiSecondFilter -> Application dừng ở ApiFirstFilter.java
  + Nếu ApiSecondFilter không có filterChain.doFilter(request, response) thì sẽ không di chuyển qua Controller -> Application dừng ở ApiSecondFilter.java