**API Flow**

**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
2. 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

1. Validation input parameter

validate trên request object:

+ String: null, blank, length, size, format

+ Number: min, max, parse error

+ List: empty

validate trên argment trực tiếp từ method controller

1. Factory -> service -> repository -> response

+ Custom response by inner class

1. Flow exception -> error response
2. Spring boot Filter
3. Access database
4. Access data from api (RestTemplate)

**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; |

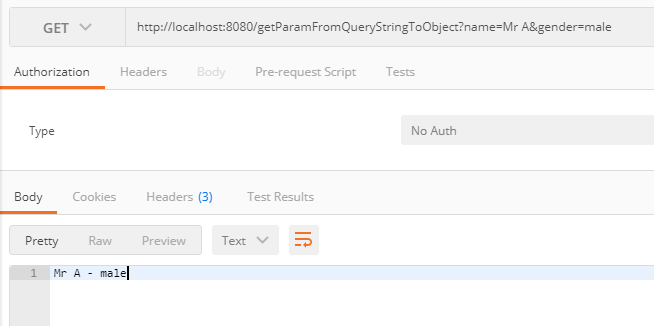
* ***Query string (Paramters on Url)***

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();  } |



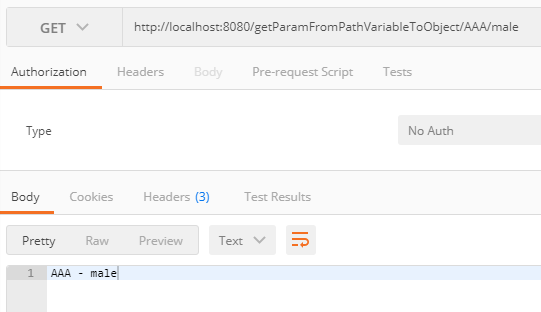
* ***Path Variables (Paramters on Uri)***

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();  } |



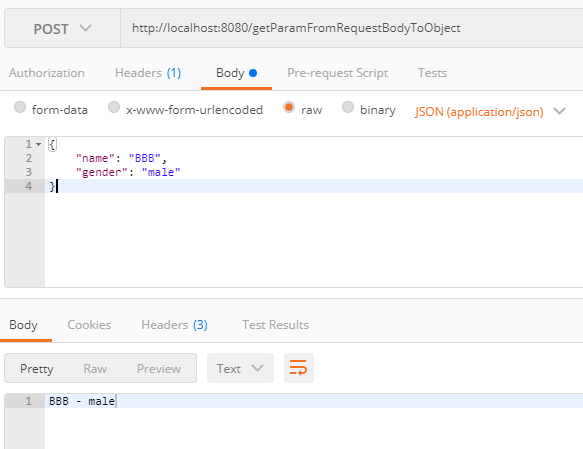
* ***Request Body***

StudentRequestBody.java

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

StudentController.java

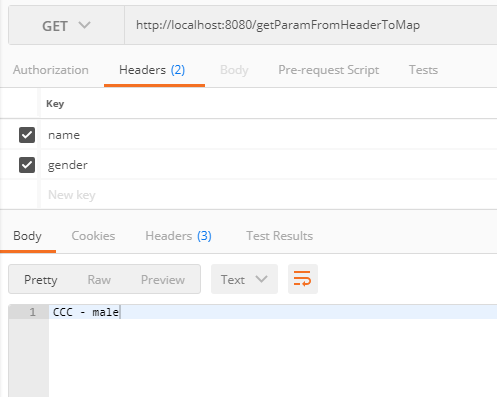
|  |
| --- |
| @PostMapping("/getParamFromRequestBodyToObject")  public String getParamFromRequestBodyToObject(@RequestBody StudentRequestBody studentRequestBody) {  return studentRequestBody.getName() + " - " + studentRequestBody.getGender();  } |



* ***Request Header (to Map)***

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:*

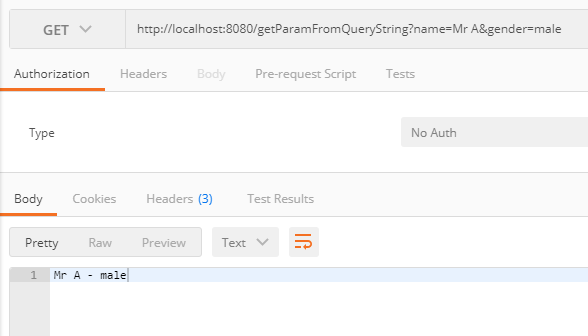
|  |
| --- |
| @RequestParam("student\_id") int id |

* ***Query string (Paramters on Url)***

StudentController.java

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

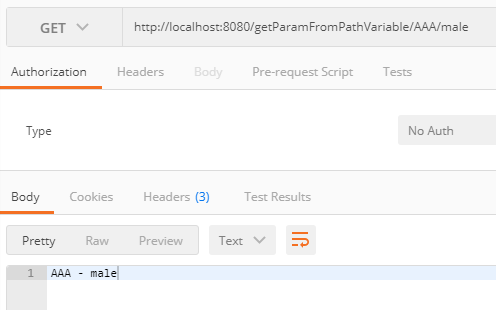
**Lưu ý:** (required = false) -> có ý nghĩa là parameter “gender” có thể truyền vào hoặc không truyền -> nếu không truyền vào sẽ nhận giá trị Null



* ***Path Variables (Paramters on Uri)***

StudentController.java

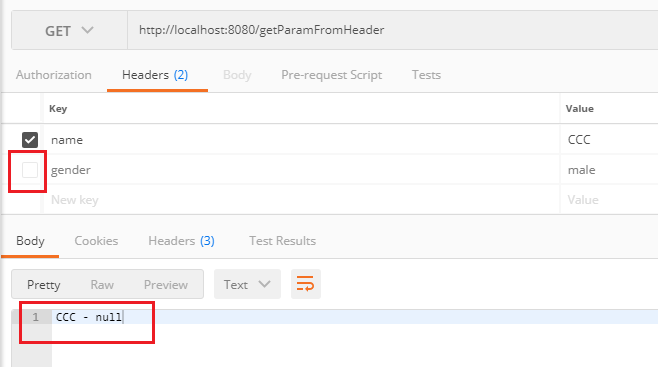
|  |
| --- |
| @GetMapping("/getParamFromPathVariable/{name}/{gender}")  public String getParamFromPathVariable(@PathVariable String name,  @PathVariable String gender) {  return name + " - " + gender;  } |

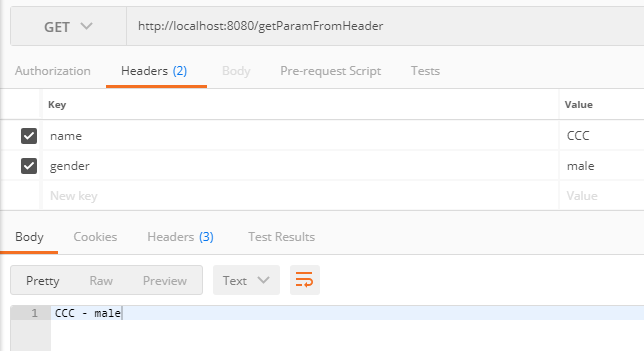


* ***Request Header***

StudentController.java

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





1. **Get List data from Request**

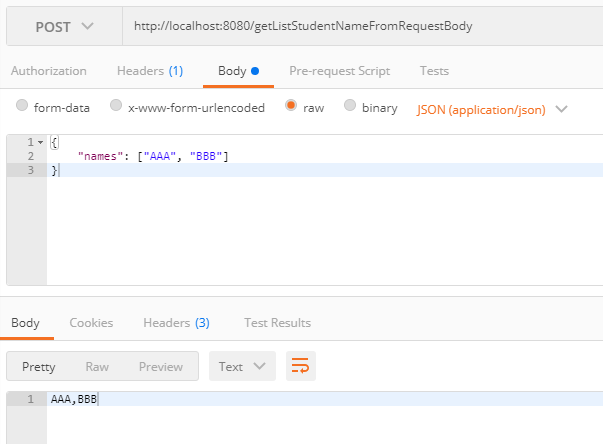
* ***List String***

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());  } |



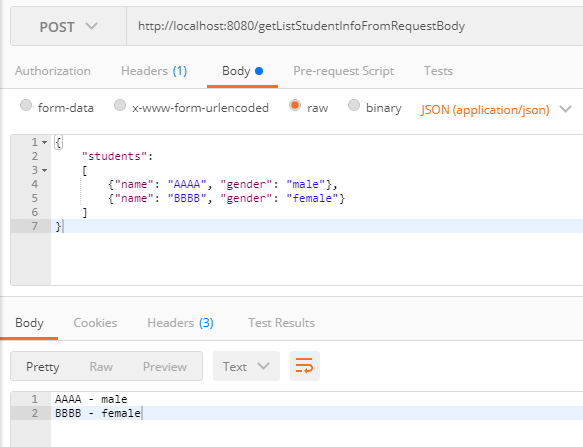
* ***List Object***

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"));  } |



**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 argment 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)

**1.1 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

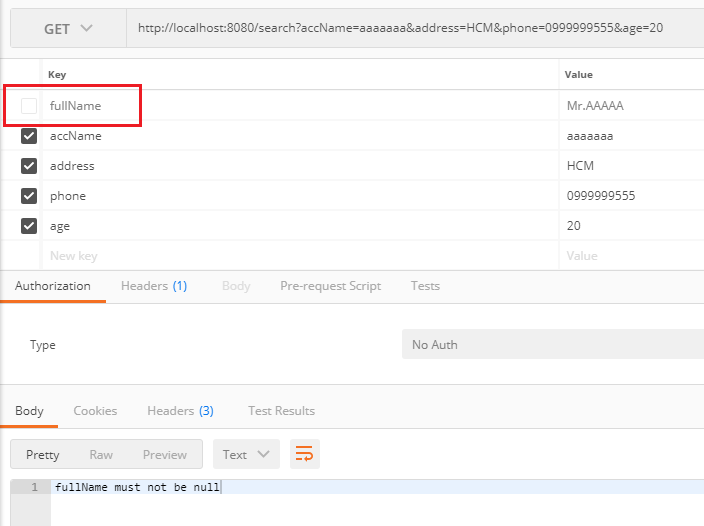
|  |
| --- |
| @Data  public class EmployeeSearchRequest {  @NotNull  private String fullName;  private String accName;  private String address;  private String phone;  } |

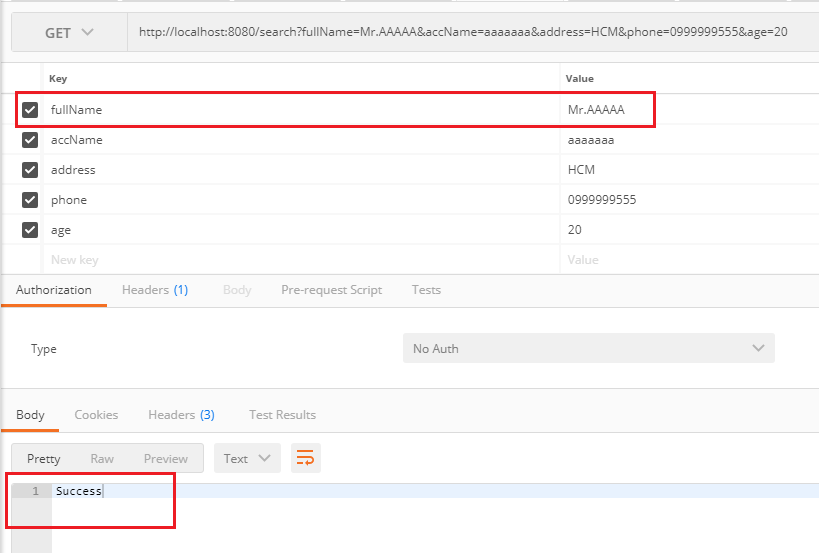
Theo đoạn code trên chỉ validate cho **“fullName” NotNull**

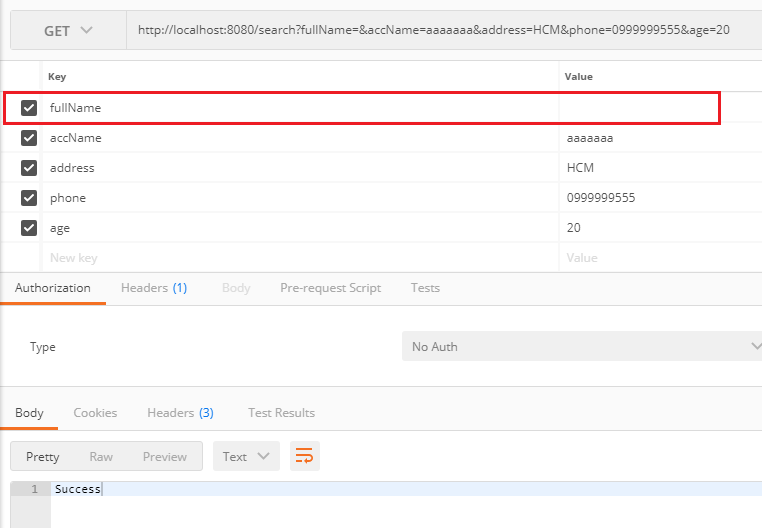
EmployeeController.java

|  |
| --- |
| @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

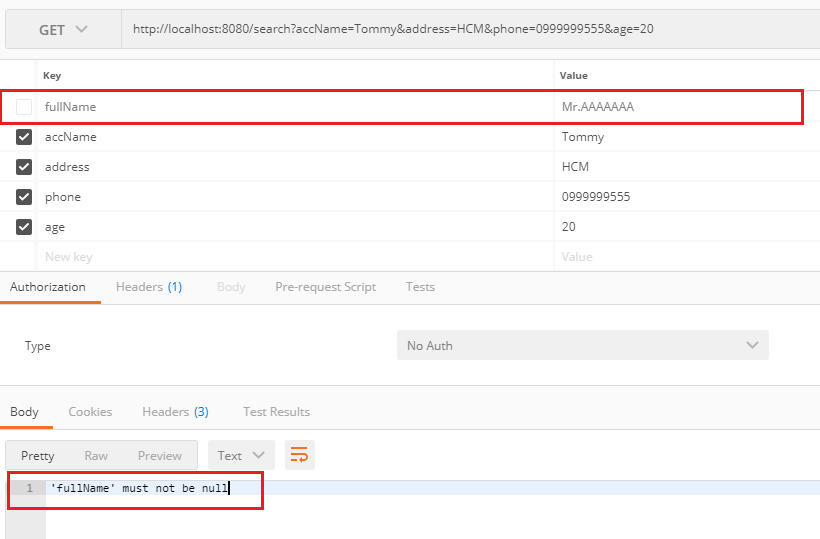
|  |
| --- |
| @Data  public class EmployeeSearchRequest {  @NotNull(message = "'fullName' must not be null")  private String fullName;  private String accName;  private String address;  private String phone;  } |

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***

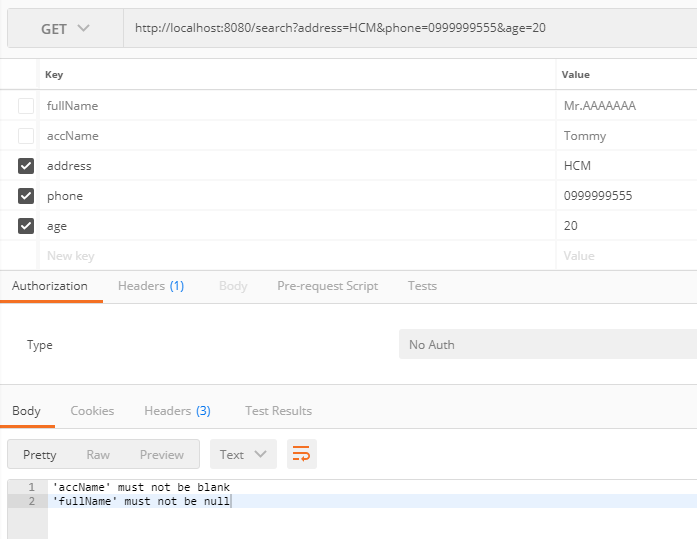
|  |  |
| --- | --- |
| @NotBlank | Default message |
| @ NotBlank (message = "accName must not be blank") | Custom message |

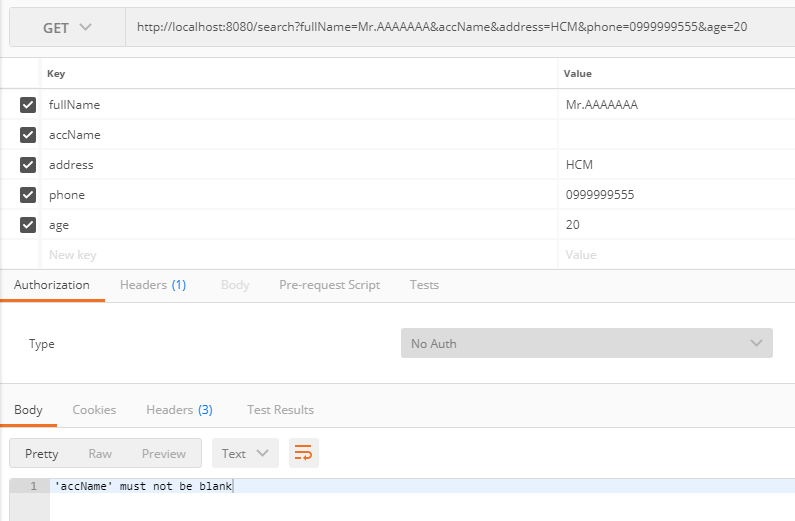
**Lưu ý:** Chỉtrường hợp String có ít nhất 1 ký tự thì mới hợp lệ. String null và empty không hợp lệ

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 của kiểu dữ liệu làm giá trị max. Ở đây 2147483647 là 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:**

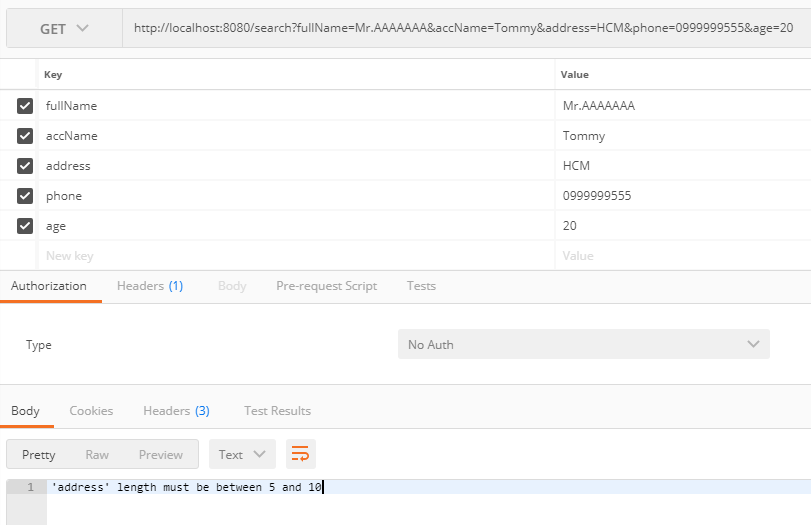
+ Nếu định nghĩa cùng 1 loại annotation trên 1 item thì sẽ chỉ check annotation sau cùng, các annotation cùng loại trước đó vô nghĩa. VD @Length, nếu chia min và max ra thành 2 dòng @Length thì chỉ nhận check annotation sau cùng

+ 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;  } |



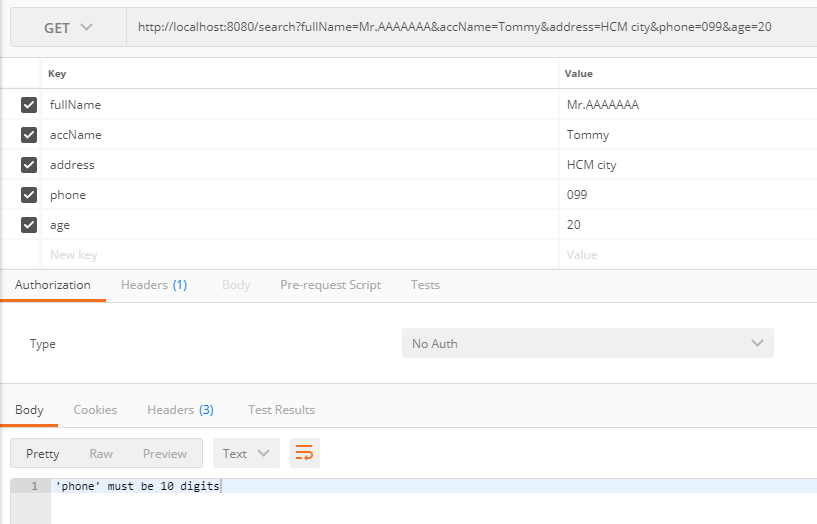
* ***Check String Pattern***

|  |  |
| --- | --- |
| @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;  } |



To be continue …