

## Internationalization

1. Add support for Internationalization in your application allowing messages to be shown in English, German and Swedish, keeping English as default.

```
1 hello.message=Hello  
1 hello.message=Hallo  
1 hello.message=Hallå  
1 #management.endpoints.web.exposure.include=*<br/>2 spring.messages.basename=messages  
  
@Bean  
public LocaleResolver localeResolver(){  
    AcceptHeaderLocaleResolver localeResolver = new AcceptHeaderLocaleResolver();  
    localeResolver.setDefaultLocale(Locale.US);  
    return localeResolver;  
}  
1
```

2. Create a GET request which takes "username" as param and shows a localized message "Hello Username". (Use parameters in message properties)

```
@GetMapping("/greeting")  
@ApiModelProperty(notes = "Greeting according to language")  
public String helloWorldInternationalized(@RequestParam(name = "name") String username){  
    return messageSource.getMessage("hello.message", null, LocaleContextHolder.getLocale())+" "+username;  
}
```

GET http://localhost:8080/greeting?name=Abhishek

Headers (9)

Header	Value
Accept	*
Accept-Encoding	gzip, deflate, br
Connection	keep-alive
Accept-Language	de
Accept	application/vnd.company.app-v1+json

Body Cookies Headers (5) Test Results

Status: 200 OK Time: 6 ms Size: 178 B Save

Pretty Raw Preview Visualize Text

1 Hallo Abhishek

GET http://localhost:8080/greeting?name=Abhishek

Headers (9)

Header	Value
Accept	*
Accept-Encoding	gzip, deflate, br
Connection	keep-alive
Accept-Language	sv
Accept	application/vnd.company.app-v1+json

Body Cookies Headers (5) Test Results

Status: 200 OK Time: 29 ms Size: 179 B Save

Pretty Raw Preview Visualize Text

1 Hallå Abhishek

\*Content Negotiation

3. Create POST Method to create user details which can accept XML for user creation.

The screenshot shows the Postman interface after sending a POST request to `http://localhost:8080/employees`. The request body contains the following XML:

```

1 <item>
2   <name>Nitin</name>
3   <age>22</age>
4 </item>

```

The response status is `201 Created`, time `194 ms`, and size `173 B`.

#### 4. Create GET Method to fetch the list of users in XML format.

The screenshot shows the Postman interface after sending a GET request to `http://localhost:8080/employees`. The response body contains the following XML list of users:

```

8   <id>2</id>
9   <name>Justin Devassy</name>
10  <age>23</age>
11 </item>
12 <item>
13   <id>3</id>
14   <name>Sourabh Singh</name>
15   <age>24</age>
16 </item>
17 <item>
18   <id>4</id>
19   <name>Nitin</name>
20   <age>22</age>
21 </item>
22 </List>

```

\*Swagger

#### 5. Configure swagger plugin and create document of following methods:

Get details of User using GET request.

Request URL  
`http://localhost:8080/employee/1`

Server response

Code	Details
200	<p>Response body</p> <pre>{   "id": 1,   "name": "Abhisehk Bhardwaj",   "age": 23,   "links": {     "List-all-employees": {       "href": "http://localhost:8080/employees"     }   } }</pre> <p><a href="#">Download</a></p> <p>Response headers</p> <pre>connection: keep-alive content-type: application/hal+json date: Tue09 Mar 2021 09:52:46 GMT keep-alive: timeout=60 transfer-encoding: chunked</pre>

Save details of the user using POST request.

Request URL  
`http://localhost:8080/employees`

Server response

Code	Details
201	<p>Response headers</p> <pre>connection: keep-alive content-length: 0 date: Tue09 Mar 2021 09:54:38 GMT keep-alive: timeout=60 location: http://localhost:8080/employees/4</pre>

Delete a user using DELETE request.

Responses

Response content type `*/*`

Curl

```
curl -X DELETE "http://localhost:8080/employee/4" -H "accept: */*"
```

Request URL  
`http://localhost:8080/employee/4`

Server response

Code	Details
200	<p>Response headers</p> <pre>connection: keep-alive content-length: 0 date: Tue09 Mar 2021 09:56:07 GMT keep-alive: timeout=60</pre>

7. In swagger documentation, add the description of each class and URI so that in swagger UI the purpose of class and URI is clear.

```
16     @Autowired
17     private EmployeeDaoService service;
18     @GetMapping("/employees")
19     @ApiModelProperty(notes = "Fetch list of Employees in List")
20     public List<EmployeeBean> retrieveAllEmployee() { return service.getEmployee(); }
21     @GetMapping("/employee/{id}")
22     @ApiModelProperty(notes = "Fetch Employees by Id in List")
23     public EntityModel<EmployeeBean> retrieveEmployeeById(@PathVariable int id)
24     {
25         EmployeeBean emp= service.getEmployeeById(id);
26         if(emp==null)
27             throw new EmployeeNotFoundException("id-"+id);
28         EntityModel<EmployeeBean> resource =EntityModel.of(emp);
29         WebMvcLinkBuilder linkTo= WebMvcLinkBuilder.linkTo(WebMvcLinkBuilder.
30             methodOn(this.getClass()).retrieveAllEmployee());
31         resource.add(linkTo.withRel("List-all-employees"));
32         return resource;
33     }
34     @PostMapping("/employees")
35     @ApiModelProperty(notes = "Add Employees in a List")
36     public ResponseEntity<Object> createEmployee(@Valid @RequestBody EmployeeBean emp){
37         EmployeeBean empSaved=service.saveEmployee(emp);
38     }
39
40
```

The screenshot shows the Swagger UI interface for the EmployeeBean model. The model is defined as follows:

```
EmployeeBean {
    description: This is EmployeeBean class
    age: integer($int32)
        minimum: 18
        exclusiveMinimum: false
        Age should be more than 18 year
    id: integer($int32)
    name: string
        pattern: [a-zA-Z][a-zA-Z ]*
        Name should only contains Alphabets
}
```

\*Static and Dynamic filtering

8. Create API which saves details of User (along with the password) but on successfully saving returns only non-critical data. (Use static filtering)

GET http://localhost:8080/static-filtering

Send

Params Authorization Headers (9) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION	...

Body Cookies Headers (5) Test Results

Status: 200 OK Time: 331 ms Size: 206 B Save

Pretty Raw Preview Visualize JSON

```

1 [
2   {
3     "username": "abhi"
4   },
5   {
6     "username": "nitin"
7   }
8 ]

```

```

} @GetMapping("/static-filtering")
} @ApiModelProperty(notes = "Static filtering of Password field")
public List<StudentBean> retrieveStudentDetails(){
    return Arrays.asList(new StudentBean( username: "abhi", password: "abhipass"),
}           new StudentBean( username: "nitin", password: "nitinpass"));
}
}

```

---

```

public class StudentBean {
    private String username;
    @JsonIgnore
    private String password;

    public String getUsername() { return username; }

    public StudentBean(String username, String password) {
        this.username = username;
        this.password = password;
    }

    public void setUsername(String username) { this.username = username; }
}

```

9. Create another API that does the same by using Dynamic Filtering.

```

    @GetMapping("/dynamic-filtering")
    @ApiModelProperty(notes = "Dynamic filtering of Password field")
    public MappingJacksonValue retrieveStudent(){
        List<StudentBean> list = Arrays.asList(new StudentBean( username: "abhi", password: "abhipass"),
            new StudentBean( username: "nitin", password: "nitinpass"));
        SimpleBeanPropertyFilter filter = SimpleBeanPropertyFilter.filterOutAllExcept("username");
        FilterProvider filters =new SimpleFilterProvider().addFilter( id: "StudentBeanFilter",filter);
        MappingJacksonValue mapping= new MappingJacksonValue(list);
        mapping.setFilters(filters);

        return mapping;
    }

```

```

    @JsonFilter("StudentBeanFilter")
    public class StudentBean {
        private String username;
        // @JsonIgnore
        private String password;

        public String getUsername() { return username; }
    }

```

GET http://localhost:8080/dynamic-filtering Send

Params	Authorization	Headers (9)	Body	Pre-request Script	Tests	Settings
Query Params						

KEY	VALUE	DESCRIPTION	...

Body Cookies Headers (5) Test Results Status: 200 OK Time: 336 ms Size: 206 B Save

Pretty Raw Preview Visualize JSON

```

1 [
2   {
3     "username": "abhi"
4   },
5   {
6     "username": "nitin"
7   }
8 ]

```

\*Versioning Restful APIs

10. Create 2 API for showing user details. The first api should return only basic details of the user and the other API should return more/enhanced details of the user,

Now apply versioning using the following methods:

- MimeType Versioning

```

    @GetMapping(value = "/person/produces", produces = "application/vnd.company.app-v1+json")
    @ApiModelProperty(notes = "MimeType based versioning v1")
    public PersonOne personProducesV1() { return new PersonOne( name: "Abhishek Bhardwaj"); }

    @GetMapping(value = "/person/produces", produces = "application/vnd.company.app-v2+json")
    @ApiModelProperty(notes = "MimeType based versioning v1")
    public PersonTwo personProducesV2() { return new PersonTwo(new Name("Abhishek", "Bhardwaj")); }

```

The screenshot shows two separate API requests in Postman:

- Request 1 (Top):** GET http://localhost:8080/person/produces. Headers: Connection (keep-alive), Accept-Language (us), Accept (application/vnd.company.app-v1+json). Response: Status: 200 OK, Time: 31 ms, Size: 211 B. JSON response: {"name": "Abhishek Bhardwaj"}
- Request 2 (Bottom):** GET http://localhost:8080/person/produces. Headers: Connection (keep-alive), Accept-Language (us), Accept (application/vnd.company.app-v2+json). Response: Status: 200 OK, Time: 11 ms, Size: 238 B. JSON response: {"name": {"firstName": "Abhishek", "lastName": "Bhardwaj"}}

- Request Parameter versioning

```

    @GetMapping(value = "/person/param", params = "version=1")
    @ApiModelProperty(notes = "Param based versioning v1")
    public PersonOne personParamV1() { return new PersonOne( name: "Abhishek Bhardwaj"); }

    @GetMapping(value = "/person/param", params = "version=2")
    @ApiModelProperty(notes = "Param based versioning v2")
    public PersonTwo personParamV2() { return new PersonTwo(new Name("Abhishek", "Bhardwaj")); }

```

The screenshot shows two API requests in Postman:

- Request 1 (version=1):**
  - Method: GET
  - URL: http://localhost:8080/person/param?version=1
  - Params:
 

KEY	VALUE	DESCRIPTION
version	1	
  - Body (Pretty):
 

```

1 [
2   "name": "Abhishek Bhardwaj"
3 ]

```
- Request 2 (version=2):**
  - Method: GET
  - URL: http://localhost:8080/person/param?version=2
  - Params:
 

KEY	VALUE	DESCRIPTION
version	2	
  - Body (Pretty):
 

```

1 [
2   "name": {
3     "firstName": "Abhishek",
4     "lastName": "Bhardwaj"
5   }
6 ]

```

- URI versioning

```

@GetMapping("/v1/person")
@ApiModelProperty(notes = "URI based versioning v1")
public PersonOne personV1() { return new PersonOne( name: "Abhishek Bhardwaj"); }

@GetMapping("/v2/person")
@ApiModelProperty(notes = "URI based versioning v2")
public PersonTwo personV2() { return new PersonTwo(new Name("Abhishek","Bhardwaj")); }

```

The screenshot shows two API requests in Postman:

- Request 1 (GET to v1):**
  - URL: `http://localhost:8080/v1/person`
  - Body (Pretty):
 

```

1 [
2   "name": "Abhishek Bhardwaj"
3 ]
```
- Request 2 (GET to v2):**
  - URL: `http://localhost:8080/v2/person`
  - Body (Pretty):
 

```

1 [
2   "name": {
3     "firstName": "Abhishek",
4     "lastName": "Bhardwaj"
5   }
6 ]
```

- Custom Header Versioning

```

@ApiModelProperty(notes = "Header based versioning v1")
public PersonOne personHeaderV1() { return new PersonOne( name: "Abhishek Bhardwaj"); }
@GetMapping(value = "/person/header",headers = "X-API-VERSION=2")
@ApiModelProperty(notes = "Header based versioning v2")
public PersonTwo personHeaderV2() { return new PersonTwo(new Name("Abhishek", "Bhardwaj")); }
```

GET http://localhost:8080/person/header

Key	Value	Description
Accept	application/vnd.company.app-v2+json	
<input checked="" type="checkbox"/> X-API-VERSION	2	

```

1 [
2   "name": {
3     "firstName": "Abhishek",
4     "lastName": "Bhardwaj"
5   }
6 ]

```

GET http://localhost:8080/person/header

Key	Value	Description
Accept	application/vnd.company.app-v2+json	
<input checked="" type="checkbox"/> X-API-VERSION	1	

```

1 [
2   "name": "Abhishek Bhardwaj"
3 ]

```

**200 OK**

Standard response for successful HTTP requests.  
The actual response will depend on the request method used. In a GET request, the response will

## \*HATEOAS

11. Configure hateoas with your springboot application. Create an api which returns User Details along with url to show all topics.

GET http://localhost:8080/employee/1 Send

Params	Authorization	Headers (9)	Body	Pre-request Script	Tests	Settings
						...
KEY		VALUE		DESCRIPTION		
Key		Value		Description		

Body Cookies Headers (5) Test Results Status: 200 OK Time: 185 ms Size: 287 B Sav

Pretty Raw Preview Visualize JSON

```

1  {
2    "id": 1,
3    "name": "Abhisehk Bhardwaj",
4    "age": 23,
5    "_links": {
6      "List-all-employees": {
7        "href": "http://localhost:8080/employees"
8      }
9    }
10 }
```

```

@GetMapping("/employee/{id}")
@ApiModelProperty(notes = "Fetch Employees by Id in List")
public EntityModel<EmployeeBean> retrieveEmployeeById(@PathVariable int id)
{
    EmployeeBean emp= service.getEmployeeById(id);
    if(emp==null)
        throw new EmployeeNotFoundException("id-"+id);
    EntityModel<EmployeeBean> resource =EntityModel.of(emp);
    WebMvcLinkBuilder linkTo= WebMvcLinkBuilder.linkTo(WebMvcLinkBuilder.
        methodOn(this.getClass()).retrieveAllEmployee());
    resource.add(linkTo.withRel("List-all-employees"));

    return resource;
}

```