

Session: Restful Web Service Part 2

Assignment

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

GET localhost:8080/morning-user

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

Headers 8 hidden

	KEY	VALUE
<input checked="" type="checkbox"/>	Accept-Language	gr
	Key	Value

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize Text ▾

```
1 guten Morgen
```

localhost:8080/morning-user

GET localhost:8080/morning-user

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

Headers 8 hidden

	KEY	VALUE
<input checked="" type="checkbox"/>	Accept-Language	sw
	Key	Value

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize Text ▾

```
1 god morigon
```

GET localhost:8080/morning-user

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

Headers
8 hidden

KEY	VALUE
<input checked="" type="checkbox"/> Accept-Language	us
Key	Value

Body
Cookies
Headers (5)
Test Results

Pretty
Raw
Preview
Visualize
Text

```
1 goodmorning
```

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

GET localhost:8080/hello-user/parth

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

Headers
8 hidden

KEY	VALUE
<input checked="" type="checkbox"/> Accept-Language	us
Key	Value

Body
Cookies
Headers (5)
Test Results

Pretty
Raw
Preview
Visualize
Text

```
1 Helloparth
```

GET

localhost:8080/hello-user/parth

ParamsAuthorizationHeaders (9)Body ●Pre-request ScriptTestsSettings

Headers

8 hidden

	KEY	VALUE
<input checked="" type="checkbox"/>	Accept-Language	gr
	Key	Value

BodyCookiesHeaders (5)Test Results

PrettyRawPreviewVisualizeText

1 Halloparth

GET

localhost:8080/hello-user/parth

ParamsAuthorizationHeaders (9)Body ●Pre-request ScriptTestsSettings

Headers

8 hidden

	KEY	VALUE
<input checked="" type="checkbox"/>	Accept-Language	sw
	Key	Value

BodyCookiesHeaders (5)Test Results

PrettyRawPreviewVisualizeText

1 Halloparth

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

```
<!-- Dependency to enable the xml format -->
<dependency>
  <groupId>com.fasterxml.jackson.dataformat</groupId>
  <artifactId>jackson-dataformat-xml</artifactId>
</dependency>
```

localhost:8080/employee

POST localhost:8080/employee

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

none form-data x-www-form-urlencoded raw binary GraphQL XML

```
1 <item>
2   <age>28</age>
3   <name>Abhijeet</name>
4 </item>
```

Body Cookies Headers (5) Test Results Status: 201 Created

KEY	VALUE
Location ①	http://localhost:8080/employee/6
Content-Length ①	0
Date ①	Wed, 10 Mar 2021 09:29:03 GMT
Keep-Alive ①	timeout=60
Connection ①	keep-alive

localhost:8080/employee/6

GET localhost:8080/employee/6

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

Headers 8 hidden

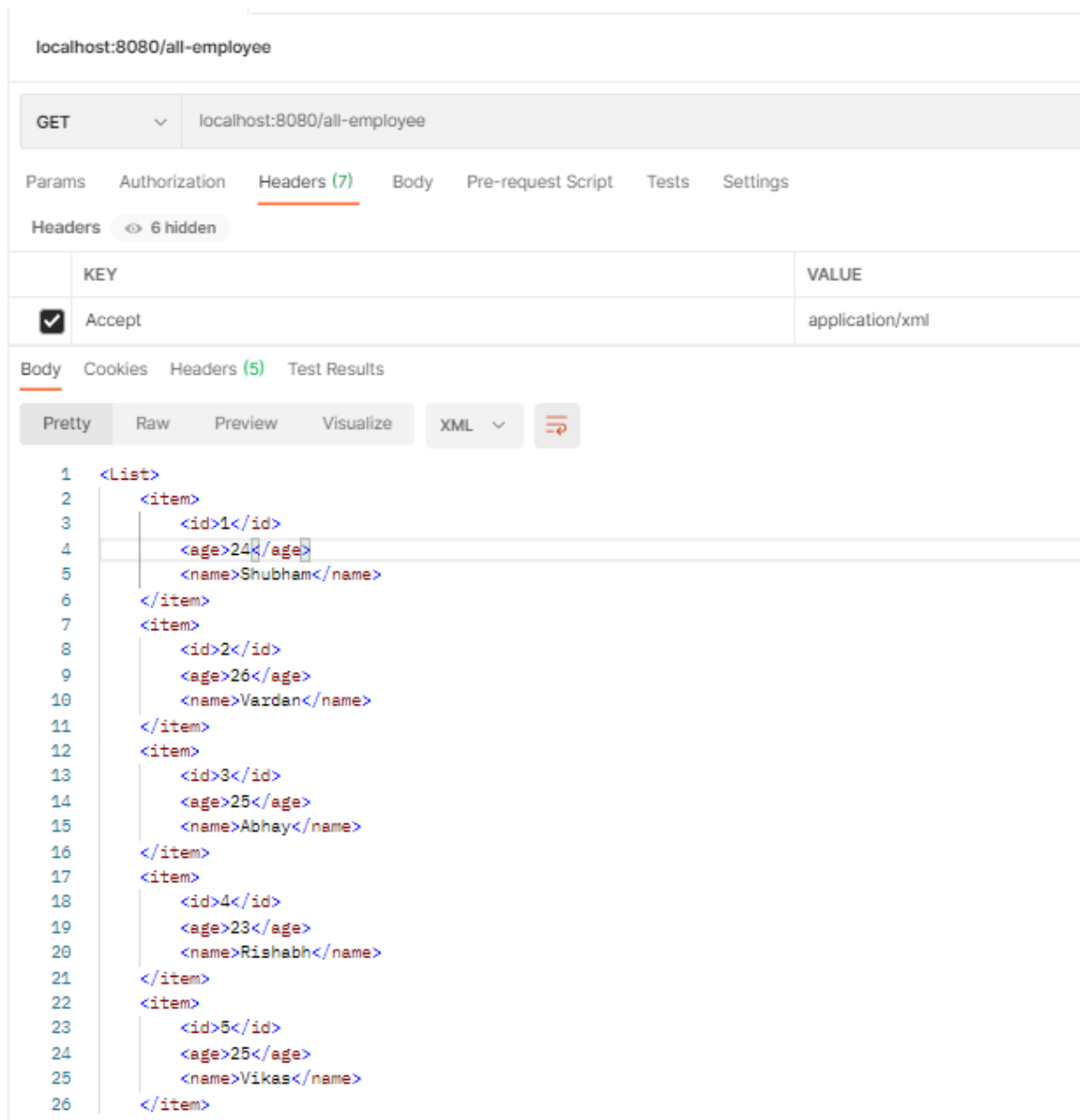
KEY	VALUE
Accept	application/xml

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize XML

```
1 <Employee>
2   <id>6</id>
3   <age>28</age>
4   <name>Abhijeet</name>
5 </Employee>
```

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



The screenshot shows a REST client interface with the following details:

- URL:** localhost:8080/all-employee
- Method:** GET
- Headers:** 6 hidden. One header is visible: **Accept** with value **application/xml**.
- Body:** XML format. The response is a list of 5 users.

```
1 <List>
2   <item>
3     <id>1</id>
4     <age>24</age>
5     <name>Shubham</name>
6   </item>
7   <item>
8     <id>2</id>
9     <age>26</age>
10    <name>Vardan</name>
11  </item>
12  <item>
13    <id>3</id>
14    <age>25</age>
15    <name>Abhay</name>
16  </item>
17  <item>
18    <id>4</id>
19    <age>23</age>
20    <name>Rishabh</name>
21  </item>
22  <item>
23    <id>5</id>
24    <age>25</age>
25    <name>Vikas</name>
26  </item>
```

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

Get details of User using GET request.

employee-resourceEmployee Resource

GET

/all-employee

reteriveAllEmployee

Parameters

Cancel

No parameters

Execute

Clear

Responses

Response content typeapplication/json

Curl

curl -X GET "http://localhost:8080/all-employee" -H "accept: application/json"

Request URL

http://localhost:8080/all-employee

Server response

Code

Details

Curl

curl -X GET "http://localhost:8080/all-employee" -H "accept: application/json"

Request URL

http://localhost:8080/all-employee

Server response

Code

Details

200

Response body

```
{
  {
    "id": 1,
    "age": 24,
    "name": "Shubham"
  },
  {
    "id": 2,
    "age": 26,
    "name": "Vardan"
  },
  {
    "id": 3,
    "age": 25,
    "name": "Abhay"
  },
  {
    "id": 4,
    "age": 23,
    "name": "Rishabh"
  },
  {
    "id": 5,
    "age": 25,
    "name": "Vikas"
  },
  {
    "id": 6,
    "age": 24,
    "name": "Vishal"
  }
}
```

Download

Response headers

connection: keep-alive
content-type: application/json
date: Wed 10 Mar 2021 09:32:33 GMT
keep-alive: timeout=60
transfer-encoding: chunked

Save details of the user using POST request.

POST /employee addEmployee

Parameters Cancel

Name	Description
employee required	employee
object (body)	Edit Value Model

```
{  "age": 40,  "name": "Nikhil"}
```

Cancel

Parameter content type
application/json

Execute Clear

Responses Response content type application/json

GET /employee/{id} retrieveOneEmployee

Parameters Cancel

Name	Description
id required	Integer(\$int32) id (path)
	<input type="text" value="7"/>

Execute Clear

Responses Response content type application/json

Curl

```
curl -X GET "http://localhost:8080/employee/7" -H "accept: application/json"
```

Request URL

```
http://localhost:8080/employee/7
```

Server response

Code	Details
200	<p>Response body</p> <pre>{ "id": 7, "age": 40, "name": "Nikhil"}</pre> Download

Delete a user using DELETE request.

The image shows a Swagger UI interface for a DELETE endpoint. At the top, it says 'DELETE /employee/{id} deleteOneEmployee'. Below this is a 'Parameters' section with a table. The table has two columns: 'Name' and 'Description'. There is one parameter named 'id' with a description 'integer(\$int32) id (path)'. The 'id' parameter is marked as 'required' and has a value of '7' entered in the input field. Below the parameters section are 'Execute' and 'Clear' buttons. The 'Responses' section shows the 'Response content type' as 'application/json'. Below this is a 'Curl' section with the command 'curl -X DELETE "http://localhost:8080/employee/7" -H "accept: application/json"'. The 'Request URL' section shows 'http://localhost:8080/employee/7'. The 'Server response' section shows a '200' status code and a 'Response body' containing a JSON object: {'id': 7, 'age': 40, 'name': 'Nikhil'}. There is a 'Download' button next to the response body.

Name	Description
id	integer(\$int32) id (path)

Execute Clear

Responses Response content type: application/json

Curl

```
curl -X DELETE "http://localhost:8080/employee/7" -H "accept: application/json"
```

Request URL

```
http://localhost:8080/employee/7
```

Server response

Code Details

200

Response body

```
{  "id": 7,  "age": 40,  "name": "Nikhil"}
```

Download

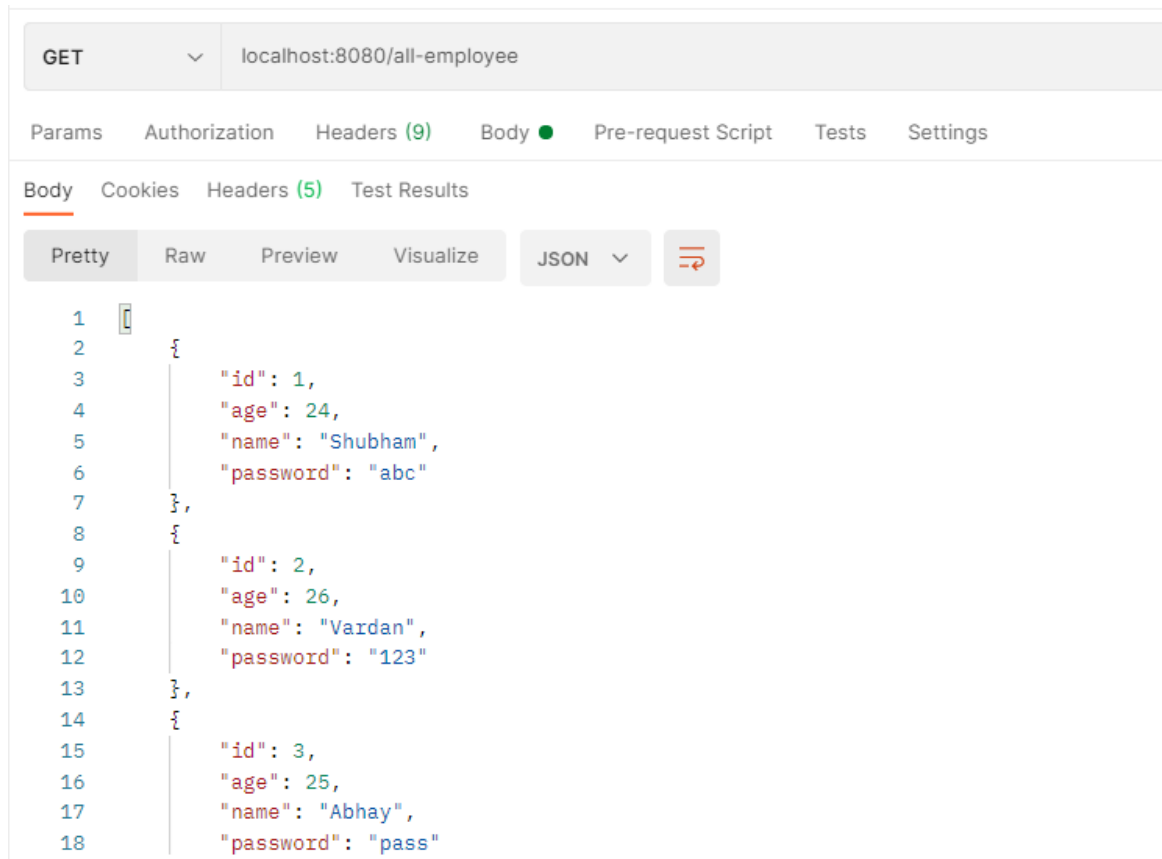
6. In swagger documentation, add the description of each class and URI so that in swagger UI the purpose of class and URI is clear.

The image shows a Swagger UI interface for the 'employee-resource' API. The title is 'employee-resource Employee Resource'. Below this are five API endpoints listed in a table. Each row has a colored button indicating the HTTP method (GET, POST, PUT, DELETE) and a description of the endpoint's purpose.

Method	Endpoint	Description
GET	/all-employee	Shows List of All Employees
POST	/employee	Creates a new Employee
GET	/employee/{id}	Shows One Employee With the Mentioned Id
PUT	/employee/{id}	Updates an Employee with the mentioned id
DELETE	/employee/{id}	Deletes an Employee

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

Before Filtering



After Filtering

```

@ApiModel(description = "Employee Model")
public class Employee {

    private Integer id;

    @Positive(message = "Age must be a positive integer")
    @ApiModelProperty(notes = "Age must be a positive integer")
    private Integer age;

    @Size(min = 3, message = "Name should have at least 3 characters")
    @ApiModelProperty(notes = "Name should have at least 3 characters")
    private String name;

    @JsonIgnore
    private String password;

```

GET localhost:8080/all-employee

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

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON ↕

```

1  [
2    {
3      "id": 1,
4      "age": 24,
5      "name": "Shubham"
6    },
7    {
8      "id": 2,
9      "age": 26,
10     "name": "Vardan"
11   },
12   {
13     "id": 3,
14     "age": 25,
15     "name": "Abhay"
16   },

```

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

```
@ApiModelProperty(description = "Employee Model")
@JsonFilter("Filter")
public class Employee {

    private Integer id;

    @Positive(message = "Age must be a positive integer")
    @ApiModelProperty(notes = "Age must be a positive integer")
    private Integer age;

    @Size(min = 3, message = "Name should have at least 3 characters")
    @ApiModelProperty(notes = "Name should have at least 3 characters")
    private String name;

    // @JsonIgnore
    private String password;
```

```
//Get All Employees
@GetMapping(path = "/all-employee")
@ApiOperation(value = "Shows List of All Employees")
public MappingJacksonValue retrieveAllEmployee() {
    List<Employee> emp = employeeDaoService.getAllEmployeeList();
    SimpleBeanPropertyFilter filter = SimpleBeanPropertyFilter
        .filterOutAllExcept("name", "age");
    FilterProvider filters = new SimpleFilterProvider().addFilter(id: "Filter", filter);
    MappingJacksonValue mapping = new MappingJacksonValue(emp);
    mapping.setFilters(filters);
    return mapping;
}
```

The screenshot shows a REST client interface with a GET request to `localhost:8080/all-employee`. The response is displayed in JSON format, showing a list of employees with their IDs, names, and ages. The response is filtered to only include the 'name' and 'age' fields.

```
1 {
2   {
3     "age": 24,
4     "name": "Shubham"
5   },
6   {
7     "age": 26,
8     "name": "Vardan"
9   },
10  {
11    "age": 25,
12    "name": "Abhay"
13  },
14  {
15    "age": 23,
16    "name": "Rishabh"
17  },
18 }
```

9. 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

```
// mime type versioning

@GetMapping(value = "/person/produce", produces = "application/vnd.company.app-v1+json")
public PersonVersionOne ProduceV1() {
    return new PersonVersionOne( name: "Parth Choudhary");
}

@GetMapping(value = "/person/produce", produces = "application/vnd.company.app-v2+json")
public PersonVersionTwo ProduceV2() {
    return new PersonVersionTwo(new Name("Parth", "Choudhary"));
}
```

GET localhost:8080/person/produce

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

Headers 8 hidden

KEY	VALUE
<input checked="" type="checkbox"/> Accept	application/vnd.company.app-v1+json
Key	Value

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "name": "Parth Choudhary"
3 }
```

GET localhost:8080/person/produce

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

Headers 8 hidden

KEY	VALUE
<input checked="" type="checkbox"/> Accept	application/vnd.company.app-v2+json
Key	Value

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```

1  {
2    "name": {
3      "firstName": "Parth",
4      "lastName": "Choudhary"
5    }
6  }
```

Request Parameter versioning

//Request Parameter

```

@GetMapping(value = "/person/param", params = "version=1")
public PersonVersionOne ParamV1() {
    return new PersonVersionOne( name: "Parth Choudhary");
}

@GetMapping(value = "/person/param", params = "version=2")
public PersonVersionTwo Param2() {
    return new PersonVersionTwo(new Name("Parth", "Choudhary"));
}

```

GET localhost:8080/person/param?version=1

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

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```

1  {
2    "name": "Parth Choudhary"
3  }
```

GET localhost:8080/person/param?version=2

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

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```

1  {
2    "name": {
3      "firstName": "Parth",
4      "lastName": "Choudhary"
5    }
6  }

```

URI versioning

//uri version

```

@GetMapping("/v1/person")
public PersonVersionOne PersonV1() {
    return new PersonVersionOne( name: "Parth Choudhary");
}

@GetMapping("/v2/person")
public PersonVersionTwo PersonV2() {
    return new PersonVersionTwo(new Name("Parth", "Choudhary"));
}

```

GET localhost:8080/v1/person

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

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```

1  {
2    "name": "Parth Choudhary"
3  }

```

GET localhost:8080/v2/person

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

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON ↕

```

1  {
2    "name": {
3      "firstName": "Parth",
4      "lastName": "Choudhary"
5    }
6  }

```

Custom Header Versioning

//header Versioning

```

@GetMapping(value = "/person/header", headers = "API-VERSION=1")
public PersonVersionOne HeaderV1() {
    return new PersonVersionOne( name: "Parth Choudhary");
}

@GetMapping(value = "/person/header", headers = "API-VERSION=2")
public PersonVersionTwo HeaderV2() {
    return new PersonVersionTwo(new Name("Parth", "Choudhary"));
}

```

GET localhost:8080/person/header

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

Headers 8 hidden

	KEY	VALUE
<input checked="" type="checkbox"/>	API-VERSION	1

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON ↕

```


1  {
2    "name": "Parth Choudhary"
3  }

```


GET localhost:8080/person/header

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

Headers 8 hidden

	KEY	VALUE
 <input checked="" type="checkbox"/>	API-VERSION	2

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON 

```

1  {
2    "name": {
3      "firstName": "Parth",
4      "lastName": "Choudhary"
5    }
6  }

```

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

```

//Get One Employee with hateoas
@GetMapping(path = "/employee/{id}")
@ApiOperation(value = "Shows One Employee With the Mentioned Id")

public EntityModel<Employee> retrieveOneEmployee(@PathVariable int id) {
    Employee emp1 = employeeDaoService.findOneEmployee(id);
    if (emp1 == null)
        throw new EmployeeNotFoundException("Employee with id: " + id + " not found !!");


    EntityModel<Employee> resource = EntityModel.of(emp1);
    WebMvcLinkBuilder linkTo = linkTo(methodOn(this.getClass()).retrieveAllEmployee());
    resource.add(linkTo.withRel("all-users"));
    return resource;
}

```

GET localhost:8080/employee/3

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

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON 

```

1  {
2    "id": 3,
3    "age": 25,
4    "name": "Abhay",
5    "password": "pass",
6    "_links": {
7      "all-users": {
8        "href": "http://localhost:8080/all-employee"
9      }
10   }
11 }

```