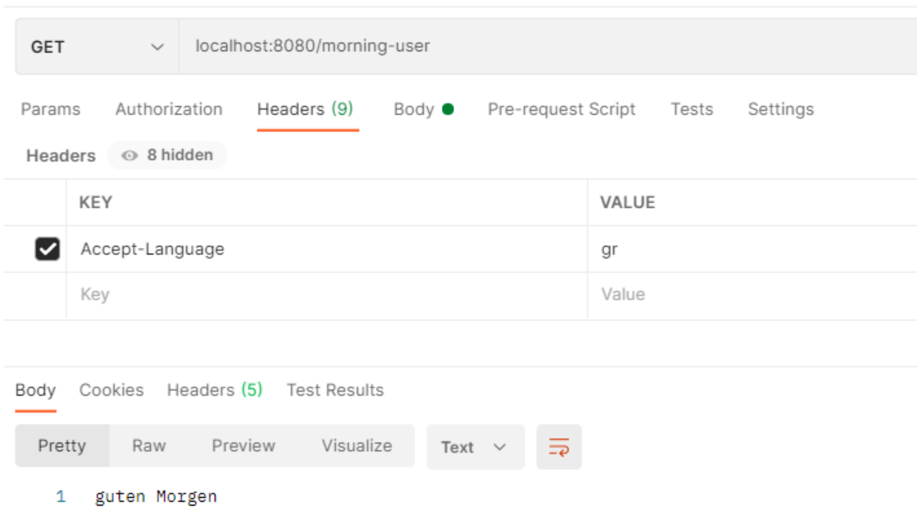
Session: Restful Web Service Part 2

Assignment

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

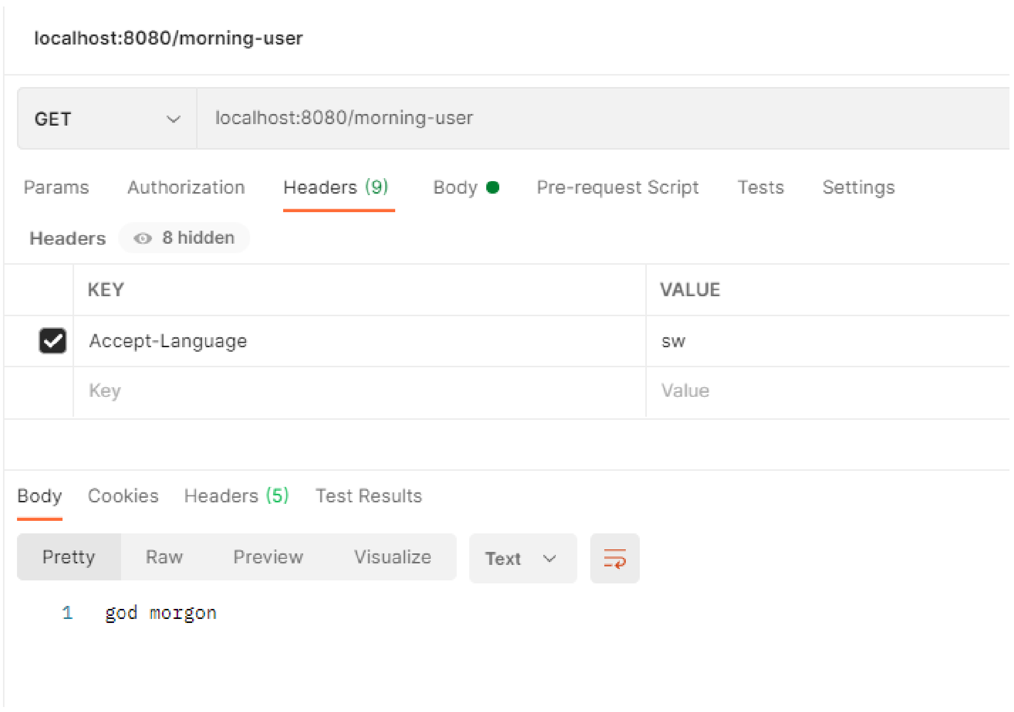


GET localhost:8080/morning-user

Headers (9)

KEY	VALUE
Accept-Language	gr

```
guten Morgen
```



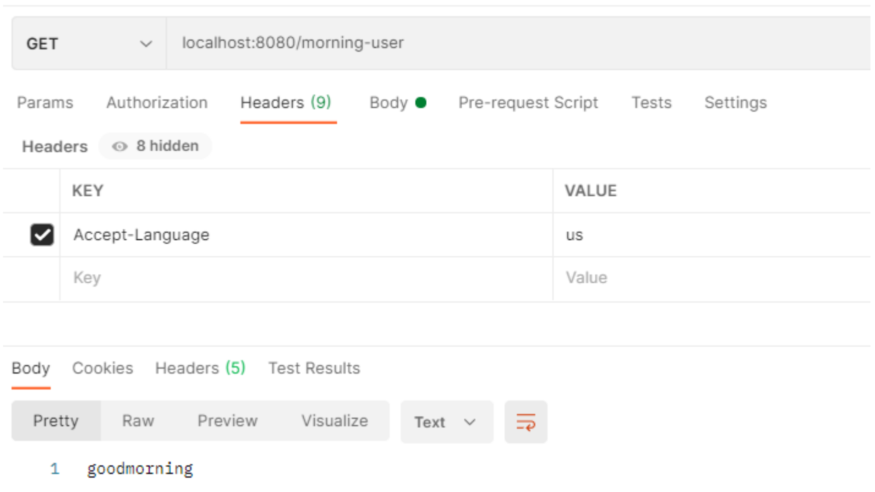
localhost:8080/morning-user

GET localhost:8080/morning-user

Headers (9)

KEY	VALUE
Accept-Language	sw

```
god morgon
```

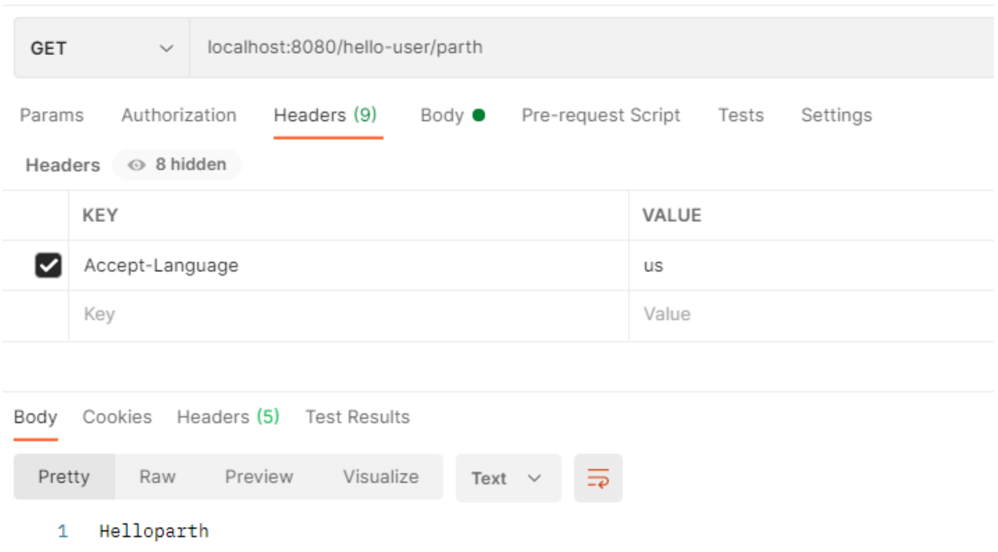

GET localhost:8080/morning-user

Headers

KEY	VALUE
Accept-Language	us

```
goodmorning
```

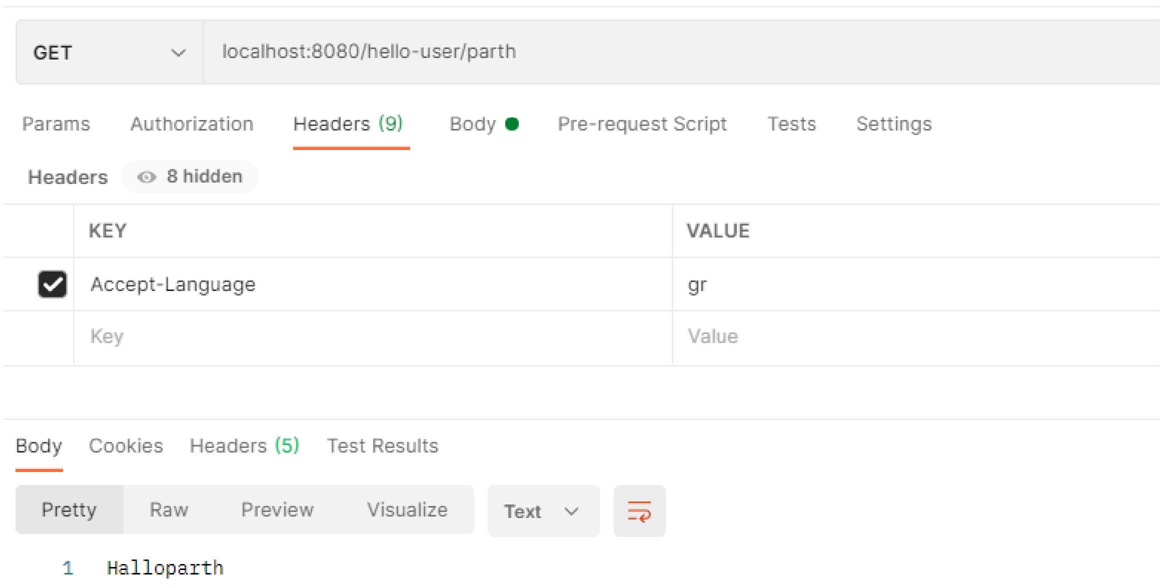
2. Create a GET request which takes "username" as param and shows a localized message "Hello Username". (Use parameters in message properties)


GET localhost:8080/hello-user/parth

Headers

KEY	VALUE
Accept-Language	us

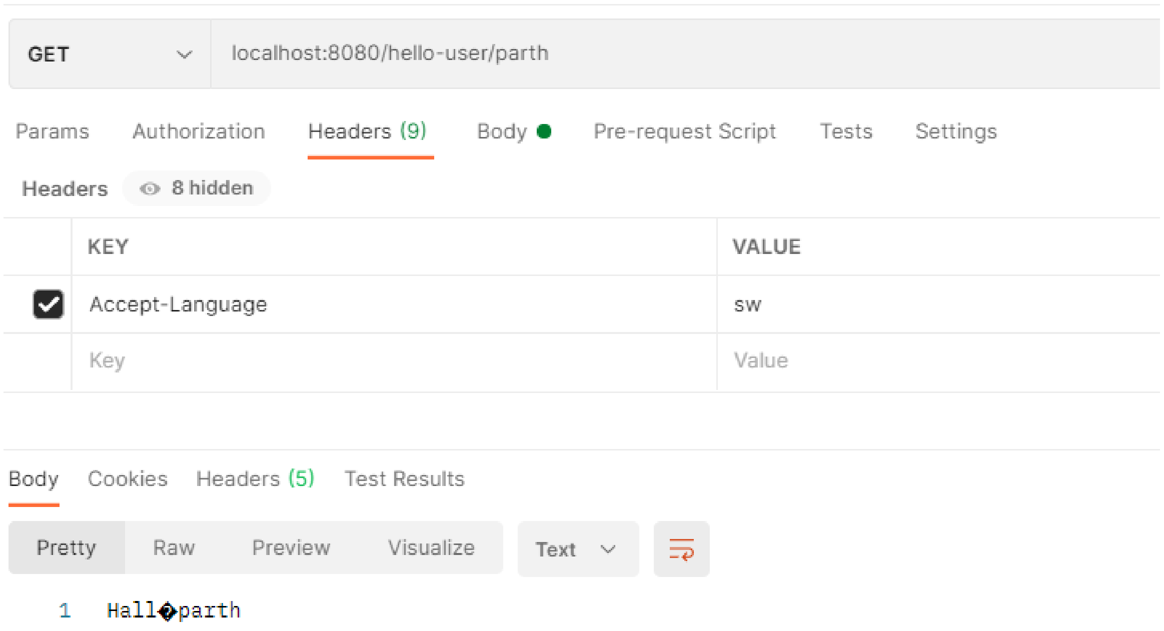
```
Helloparth
```



localhost:8080/hello-user/parth

KEY	VALUE
Accept-Language	gr
Key	Value

```
Halloparth
```

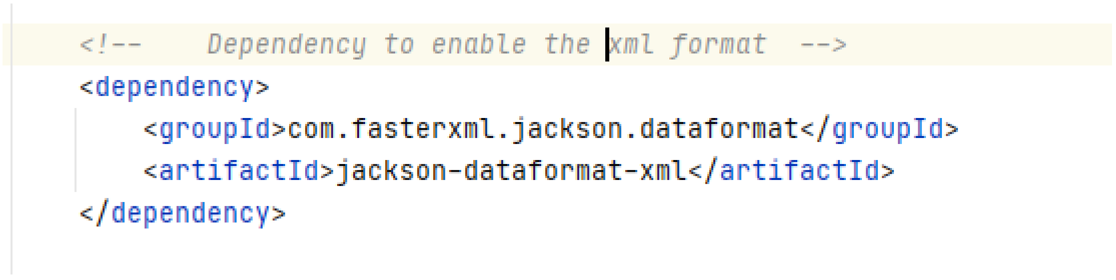


localhost:8080/hello-user/parth

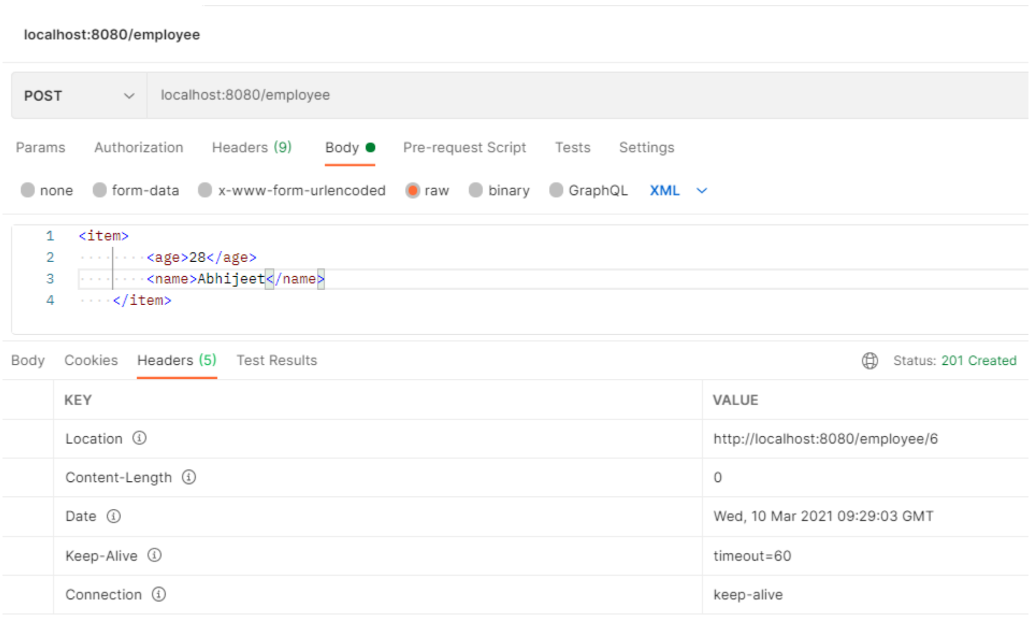
KEY	VALUE
Accept-Language	sw
Key	Value

```
Hall�parth
```

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>
```



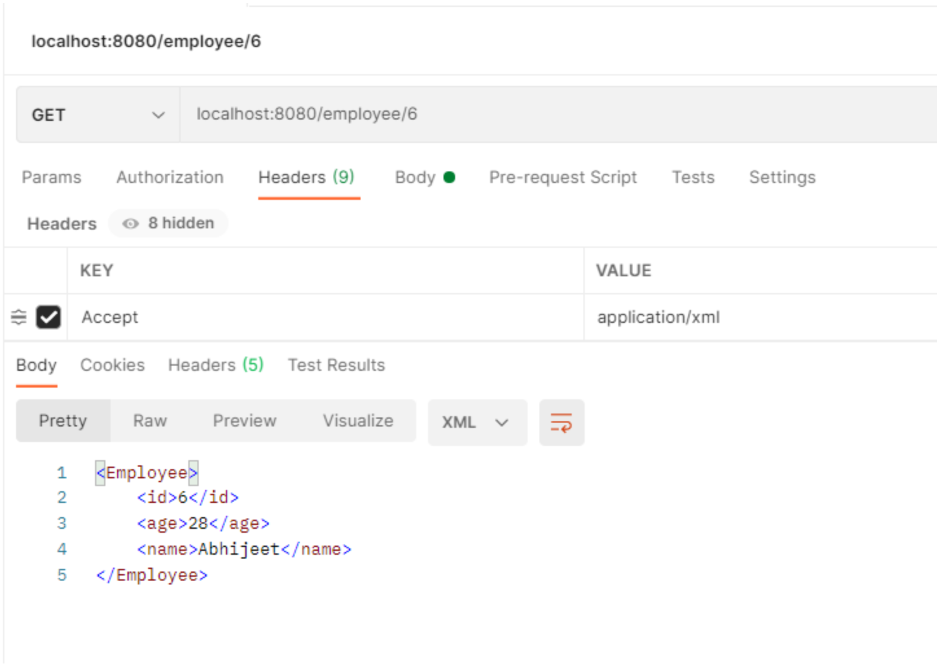
localhost:8080/employee

POST localhost:8080/employee

```
<item>
    <age>28</age>
    <name>Abhijeet</name>
</item>
```

Status: 201 Created

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



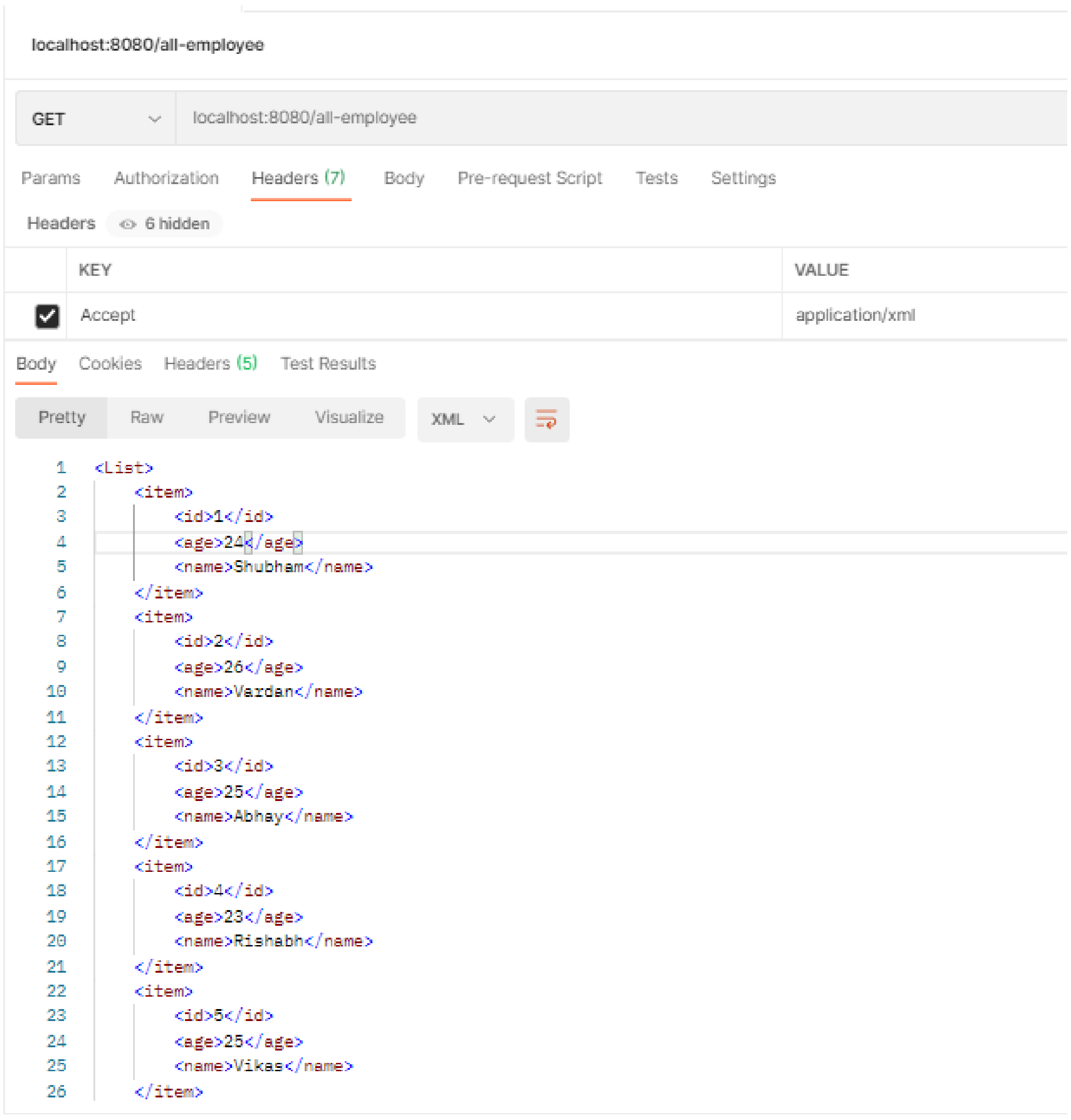
localhost:8080/employee/6

GET localhost:8080/employee/6

	KEY	VALUE
✓	Accept	application/xml

```
<Employee>
    <id>6</id>
    <age>28</age>
    <name>Abhijeet</name>
</Employee>
```

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



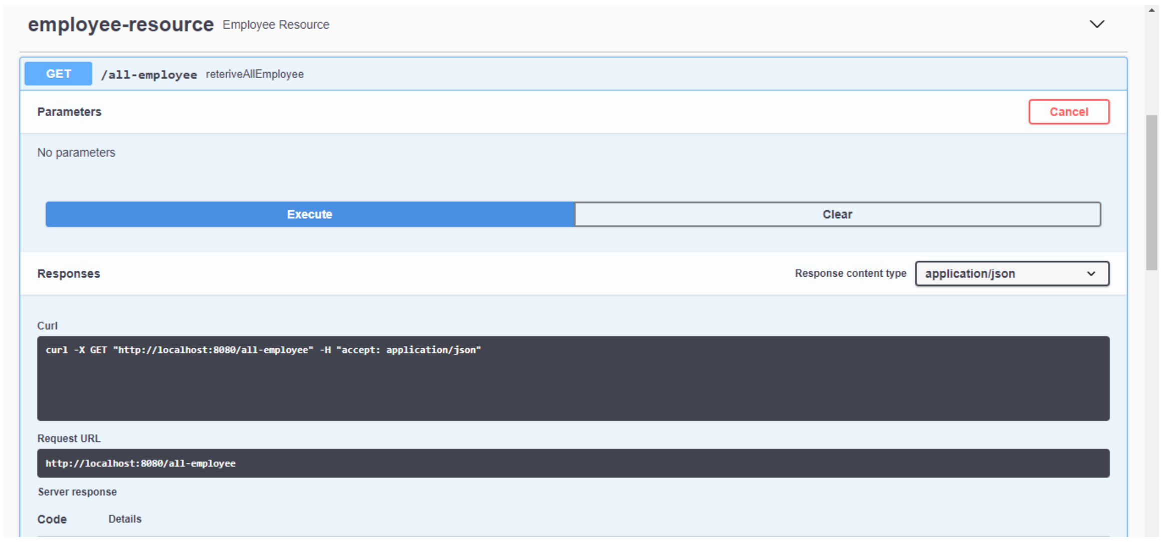
localhost:8080/all-employee

KEY	VALUE
✓ Accept	application/xml

```
<List>
    <item>
        <id>1</id>
        <age>24</age>
        <name>Shubham</name>
    </item>
    <item>
        <id>2</id>
        <age>26</age>
        <name>Vardan</name>
    </item>
    <item>
        <id>3</id>
        <age>25</age>
        <name>Abhay</name>
    </item>
    <item>
        <id>4</id>
        <age>23</age>
        <name>Rishabh</name>
    </item>
    <item>
        <id>5</id>
        <age>25</age>
        <name>Vikas</name>
    </item>
```

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

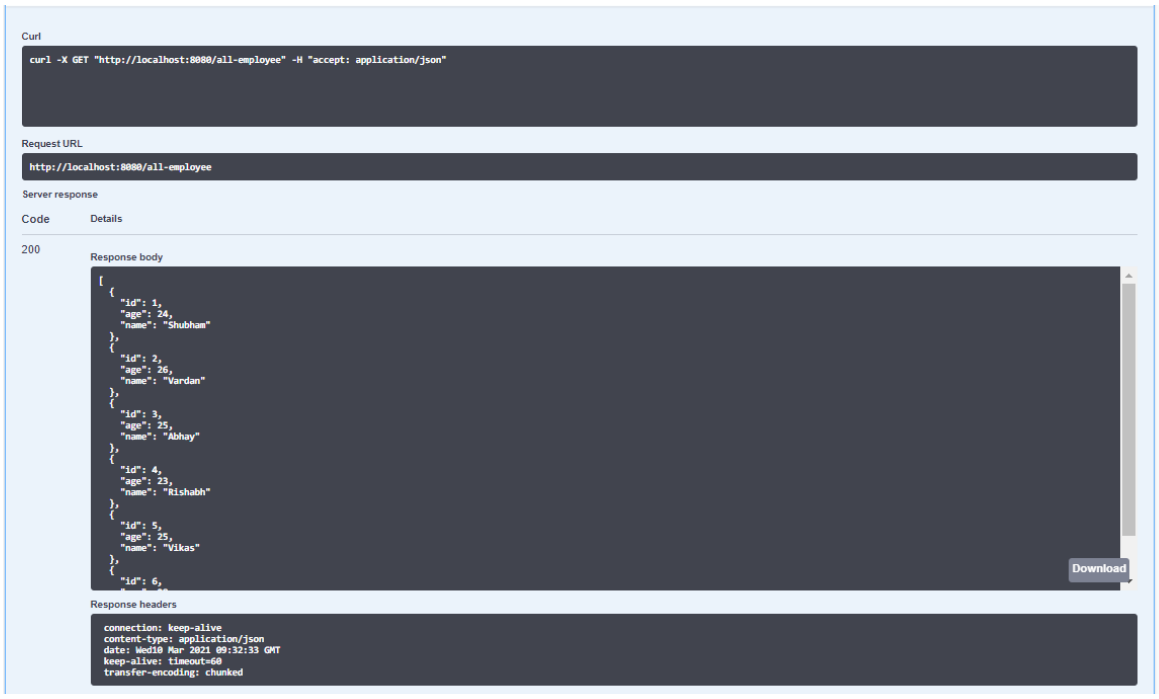
Get details of User using GET request.


employee-resource Employee Resource
GET /all-employee reteriveAllEmployee
Parameters
Cancel
No parameters
Execute
Clear
Responses
Response content type application/json
Curl

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

```
http://localhost:8080/all-employee
```

Server response
Code
Details



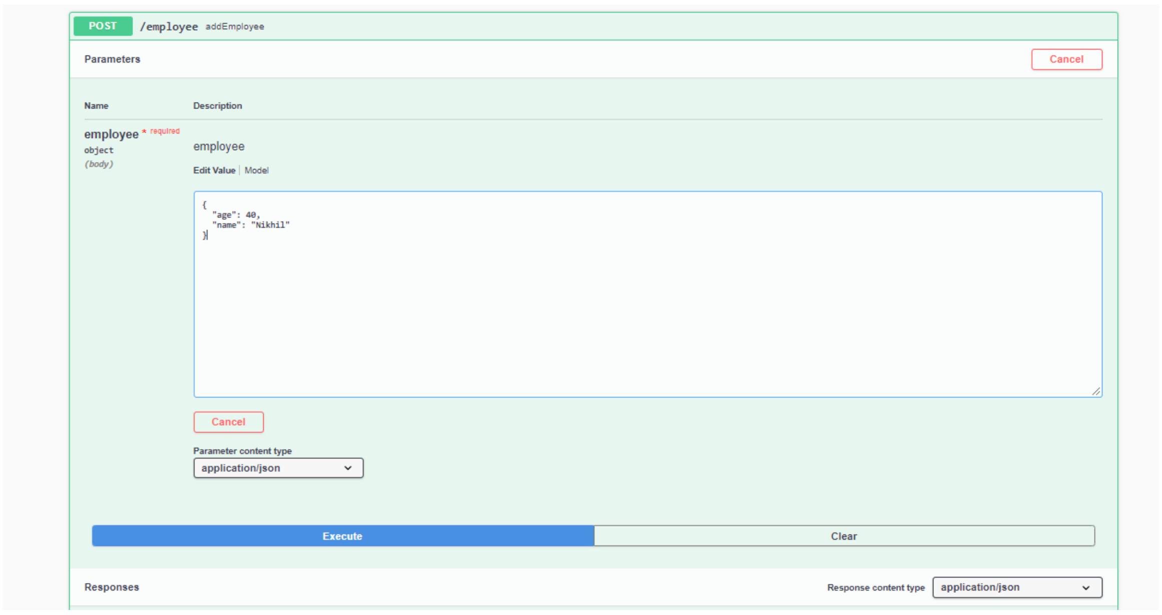
```
curl -X GET "http://localhost:8080/all-employee" -H "accept: application/json"
```

```
http://localhost:8080/all-employee
```

```
[
  {
    "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,
```

```
connection: keep-alive
content-type: application/json
date: Wed10 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

Name: employee * required, object (body)

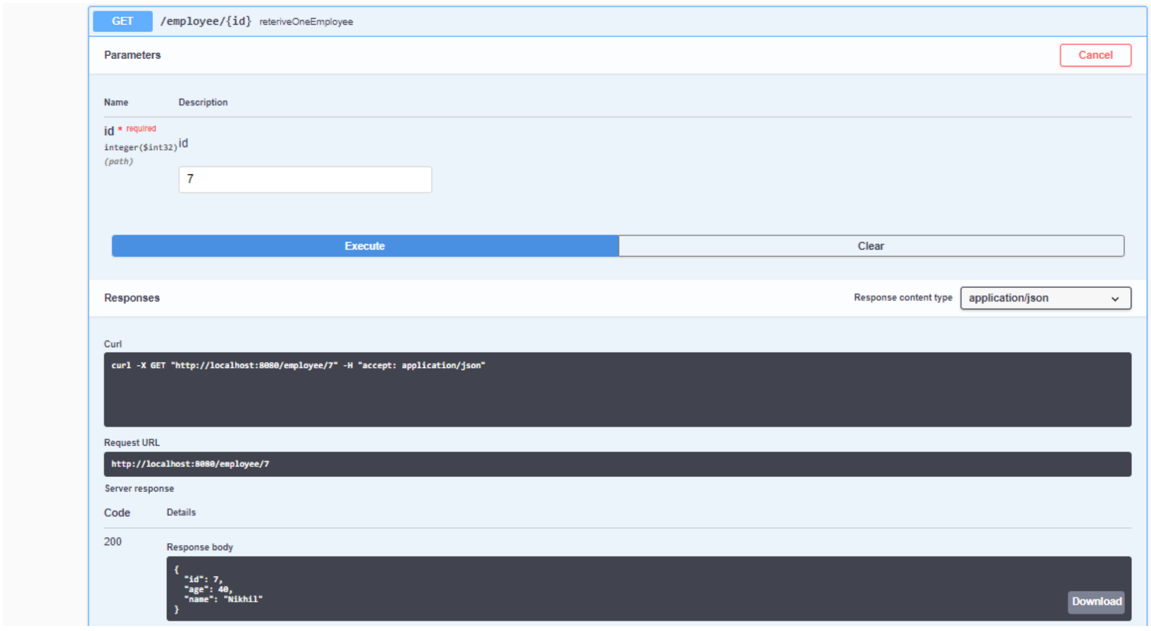
Description: employee

Edit Value | Model

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

Parameter content type: application/json

Responses — Response content type: application/json



GET /employee/{id} reteriveOneEmployee

Parameters

Name: id * required, integer($int32) (path)

Description: id

7

Responses — Response content type: application/json

Curl

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

Request URL

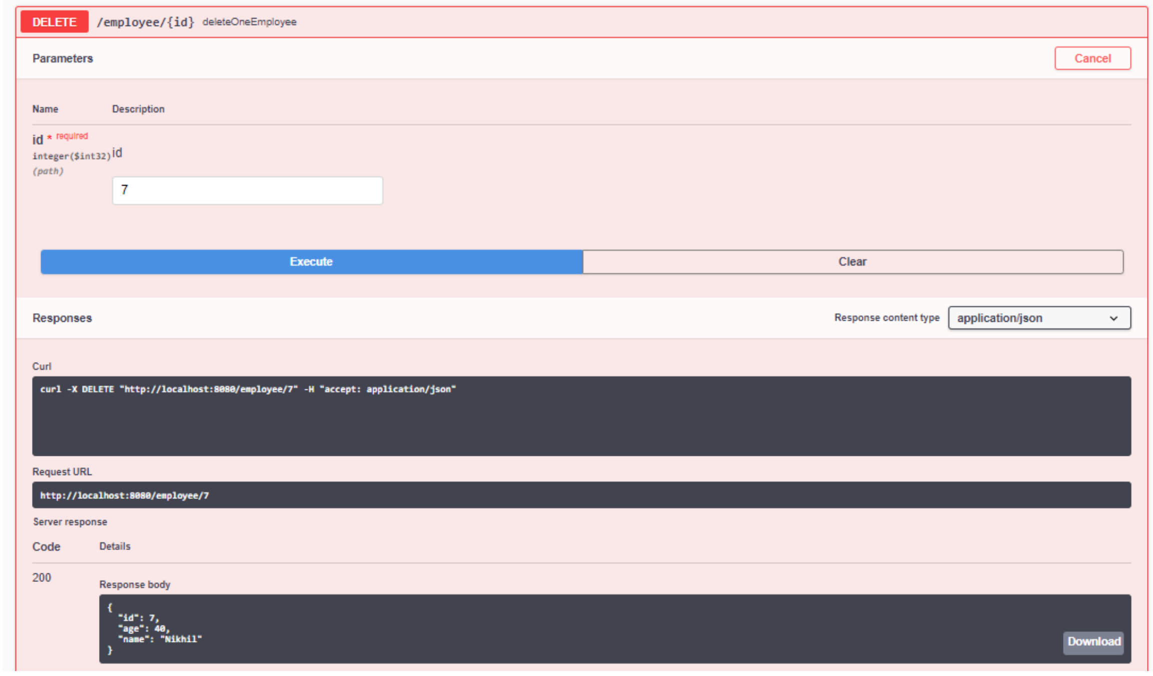
```
http://localhost:8080/employee/7
```

Server response

Code	Details
200	Response body

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

Delete a user using DELETE request.


DELETE /employee/{id} deleteOneEmployee

Parameters — Cancel

Name	Description
id * required integer($int32) (path)	id 7

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
```

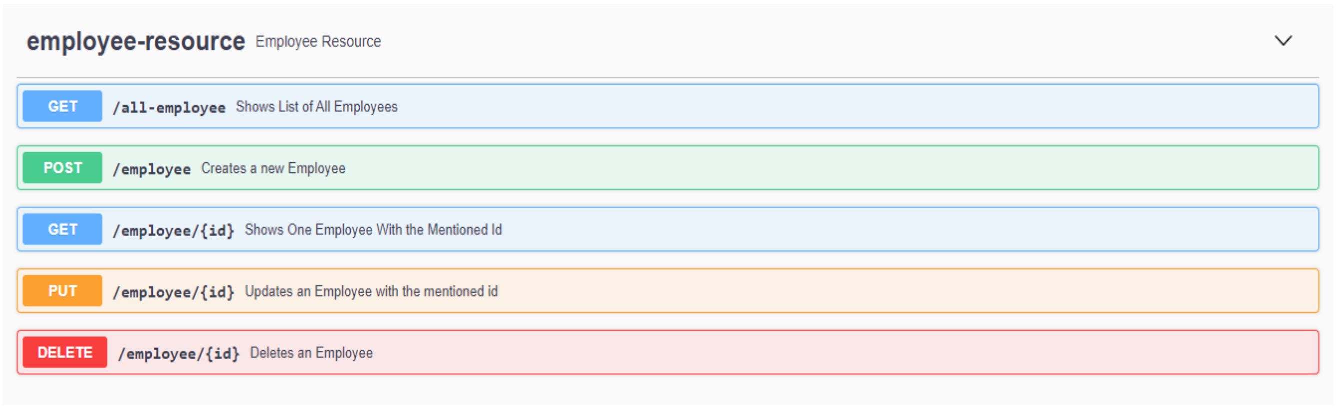
Server response

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.


employee-resource Employee Resource

GET /all-employee Shows List of All Employees

POST /employee Creates a new Employee

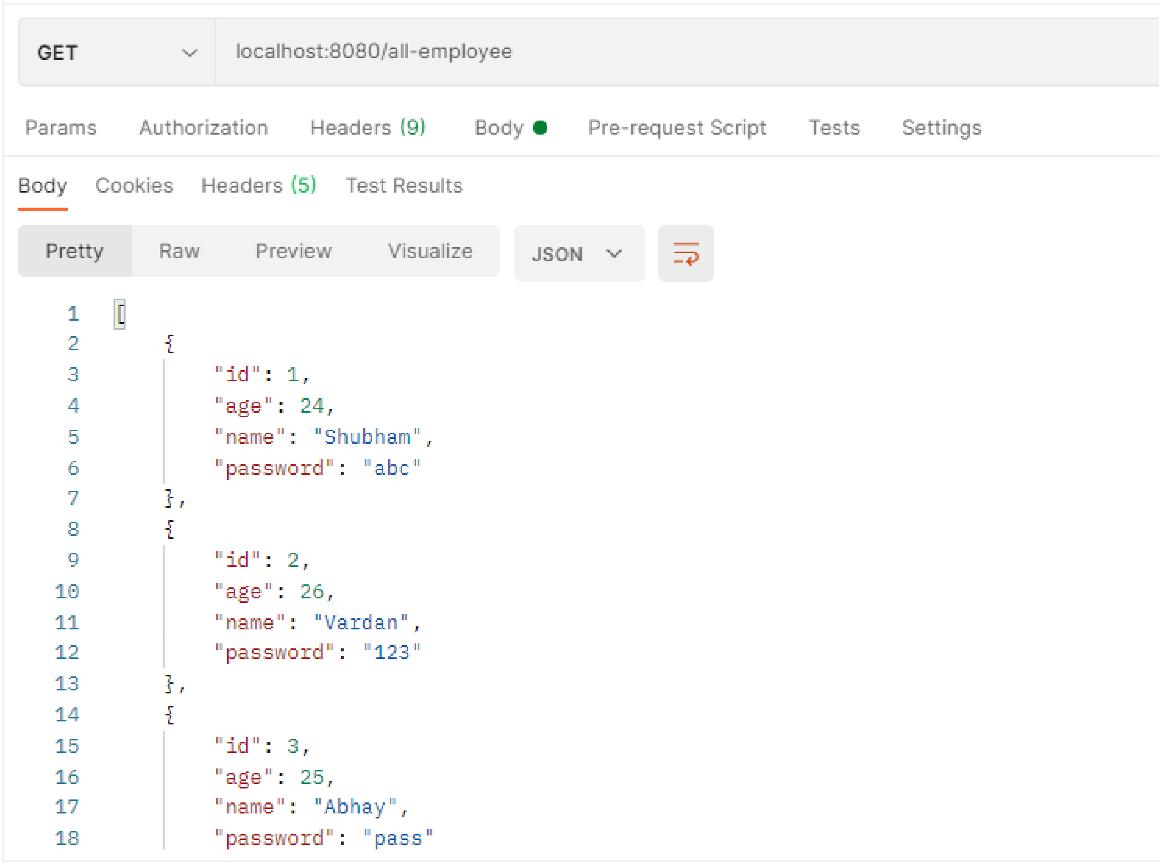
GET /employee/{id} Shows One Employee With the Mentioned Id

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)

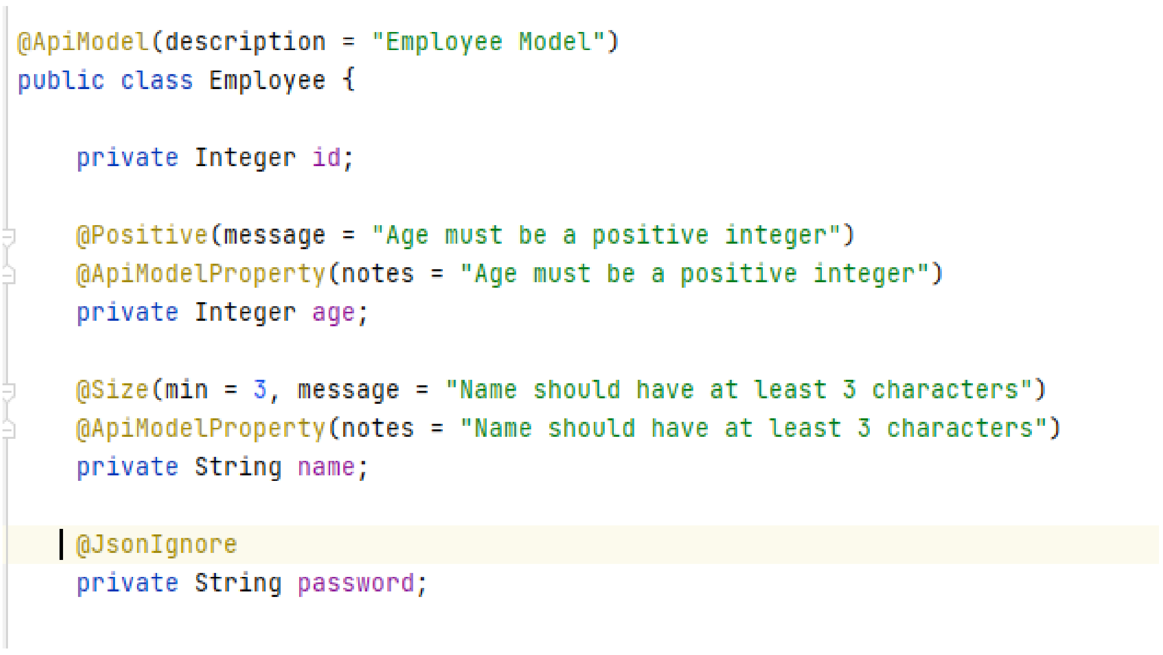
Before Filtering



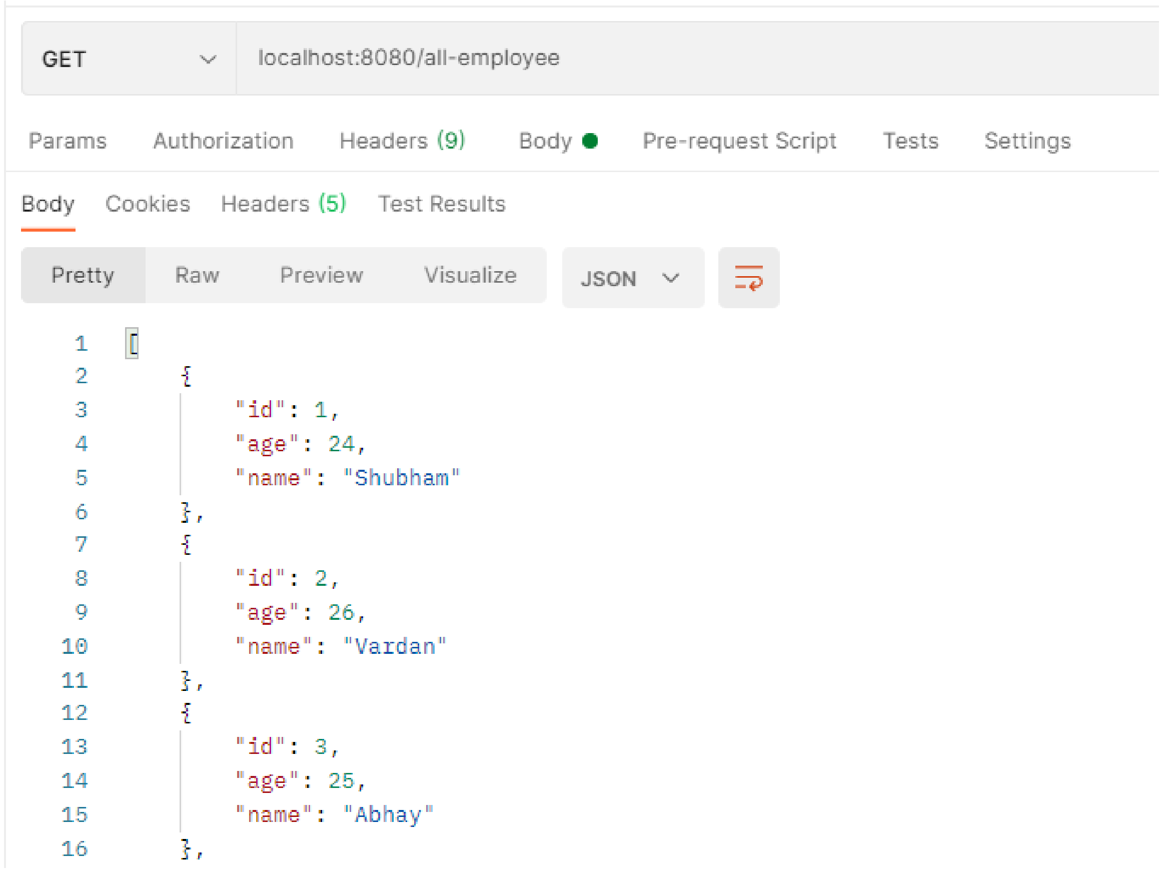
localhost:8080/all-employee

```
[
    {
        "id": 1,
        "age": 24,
        "name": "Shubham",
        "password": "abc"
    },
    {
        "id": 2,
        "age": 26,
        "name": "Vardan",
        "password": "123"
    },
    {
        "id": 3,
        "age": 25,
        "name": "Abhay",
        "password": "pass"
```

After Filtering

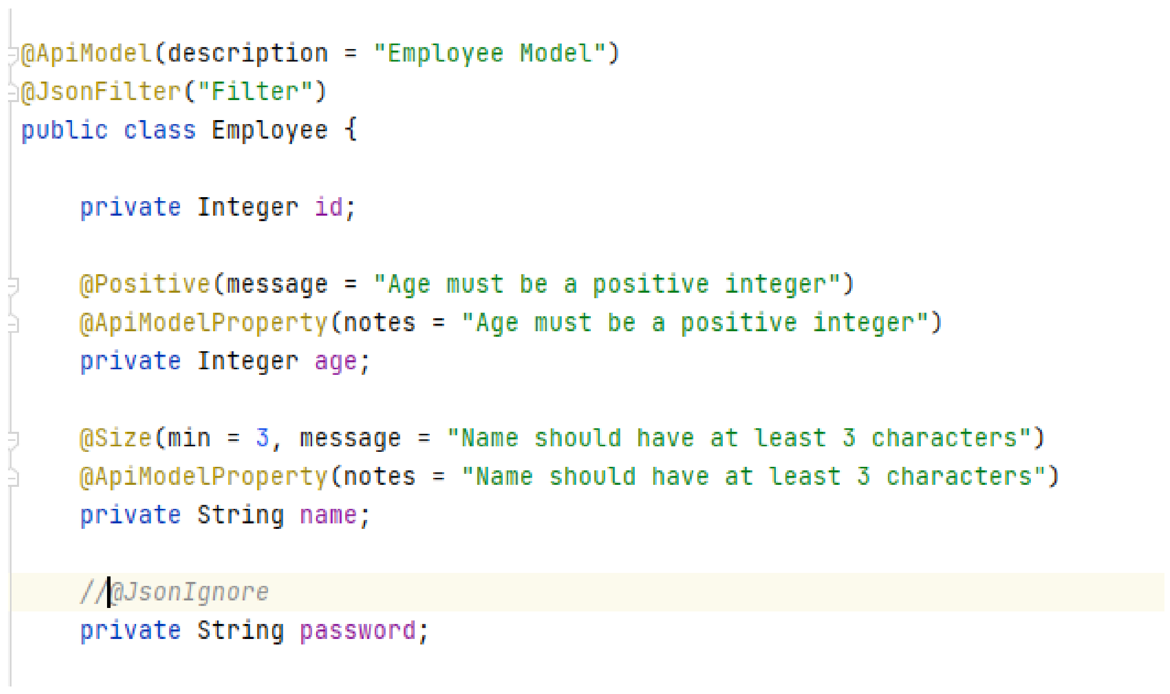


```java
@ApiModel(description = "Employee Model")
public class Employee {

    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;
```

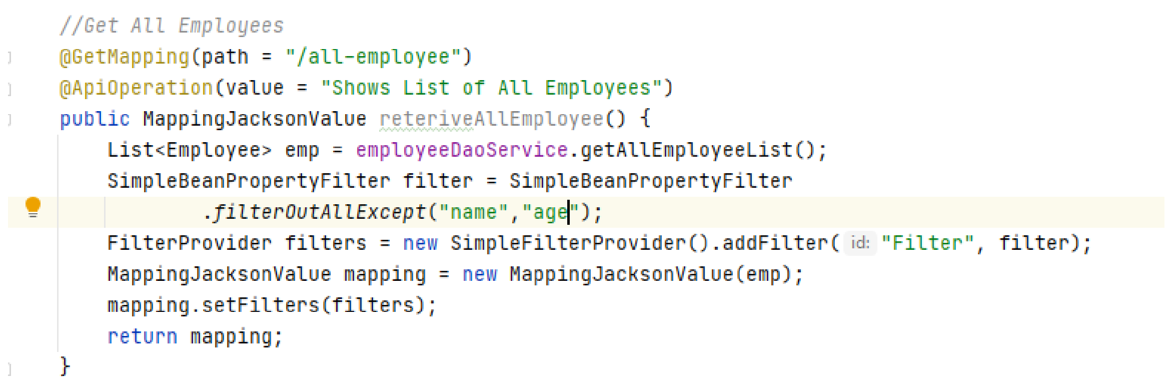


```json
[
    {
        "id": 1,
        "age": 24,
        "name": "Shubham"
    },
    {
        "id": 2,
        "age": 26,
        "name": "Vardan"
    },
    {
        "id": 3,
        "age": 25,
        "name": "Abhay"
    },
```

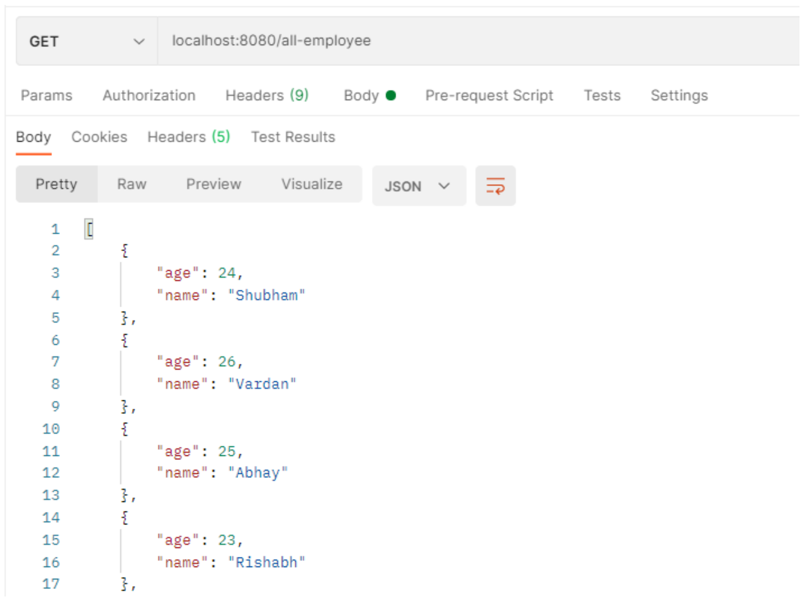
8. Create another API that does the same by using Dynamic Filtering.



```java
@ApiModel(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;
```



```java
    //Get All Employees
    @GetMapping(path = "/all-employee")
    @ApiOperation(value = "Shows List of All Employees")
    public MappingJacksonValue reteriveAllEmployee() {
        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;
    }
```



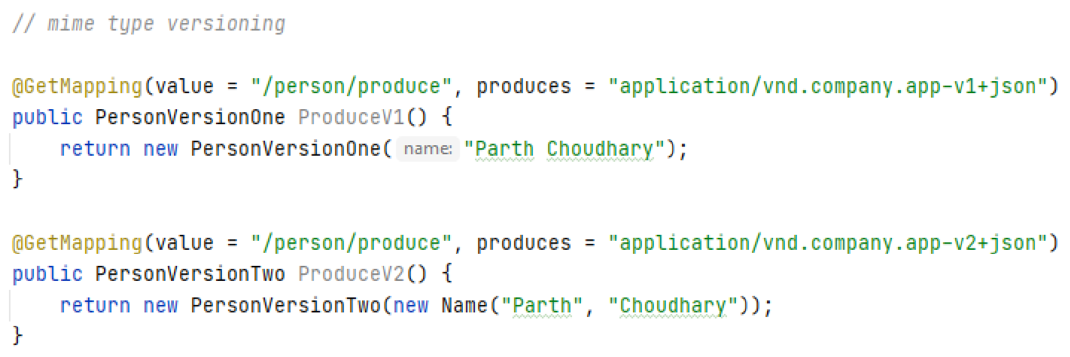
GET localhost:8080/all-employee

```json
[
    {
        "age": 24,
        "name": "Shubham"
    },
    {
        "age": 26,
        "name": "Vardan"
    },
    {
        "age": 25,
        "name": "Abhay"
    },
    {
        "age": 23,
        "name": "Rishabh"
    },
```

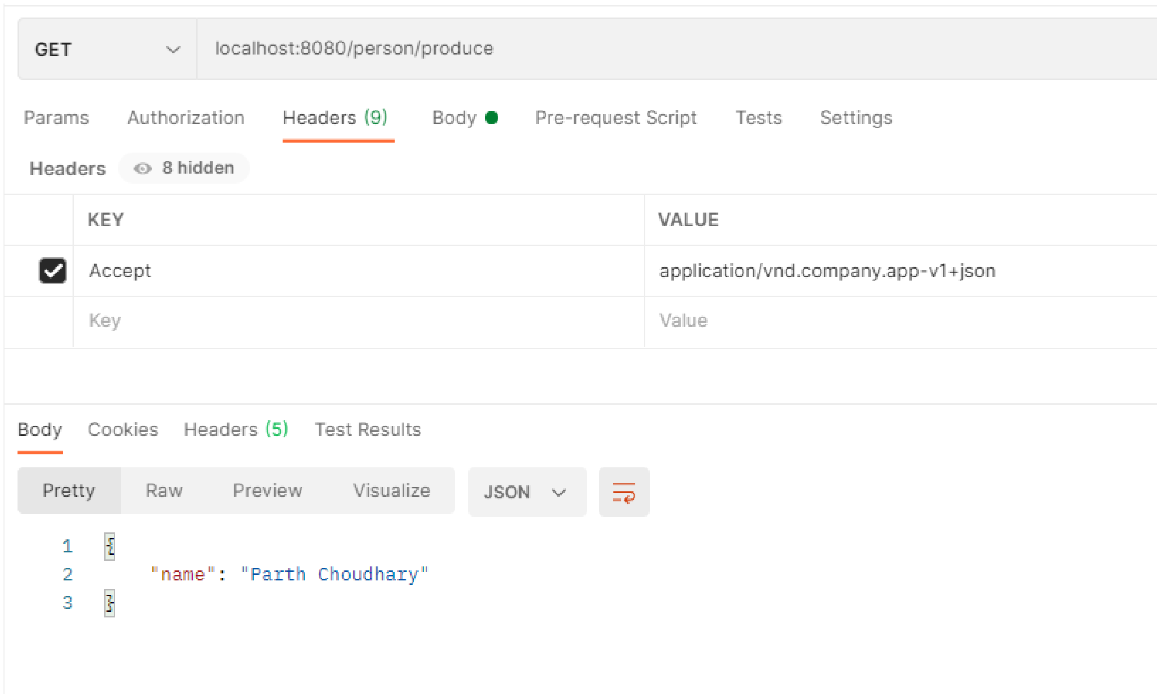
9. Create 2 API for showing user details. The first api should return only basic details of the user and the other API should return more/enhanced details of the user,

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

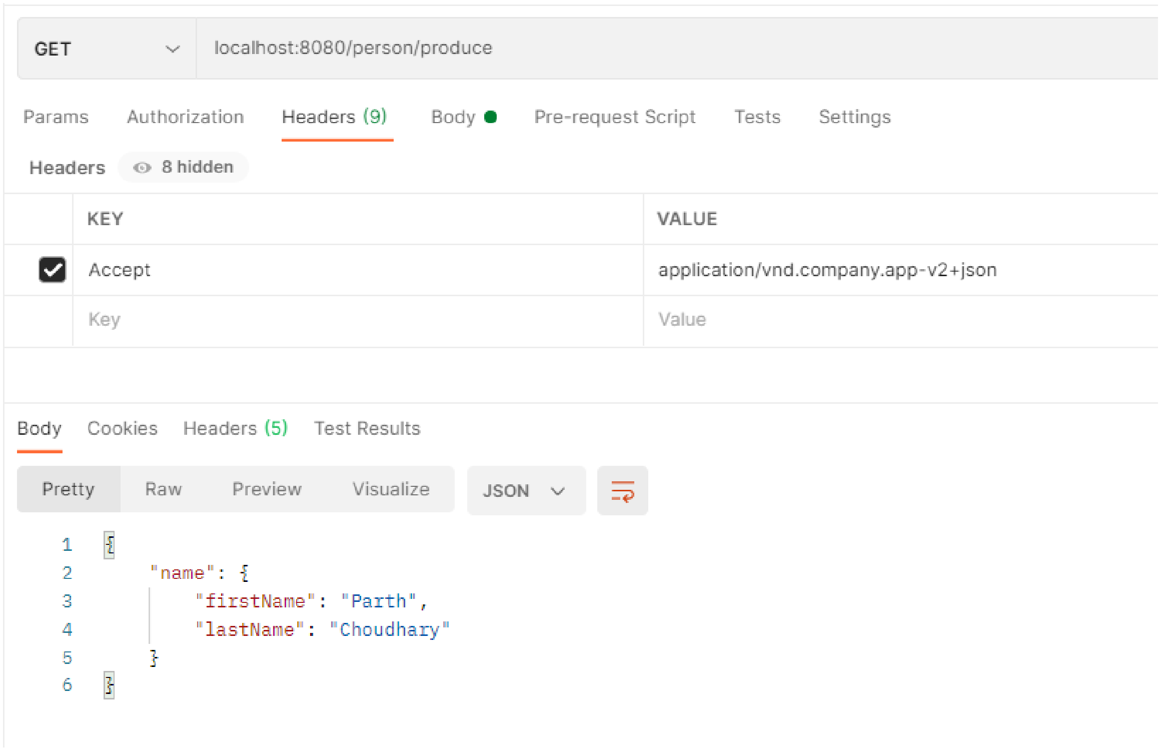
Headers 8 hidden

	KEY	VALUE
✓	Accept	application/vnd.company.app-v1+json
	Key	Value

Body | Cookies | Headers (5) | Test Results

Pretty | Raw | Preview | Visualize | JSON

```
{
    "name": "Parth Choudhary"
}
```



GET localhost:8080/person/produce

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

Headers 8 hidden

	KEY	VALUE
✓	Accept	application/vnd.company.app-v2+json
	Key	Value

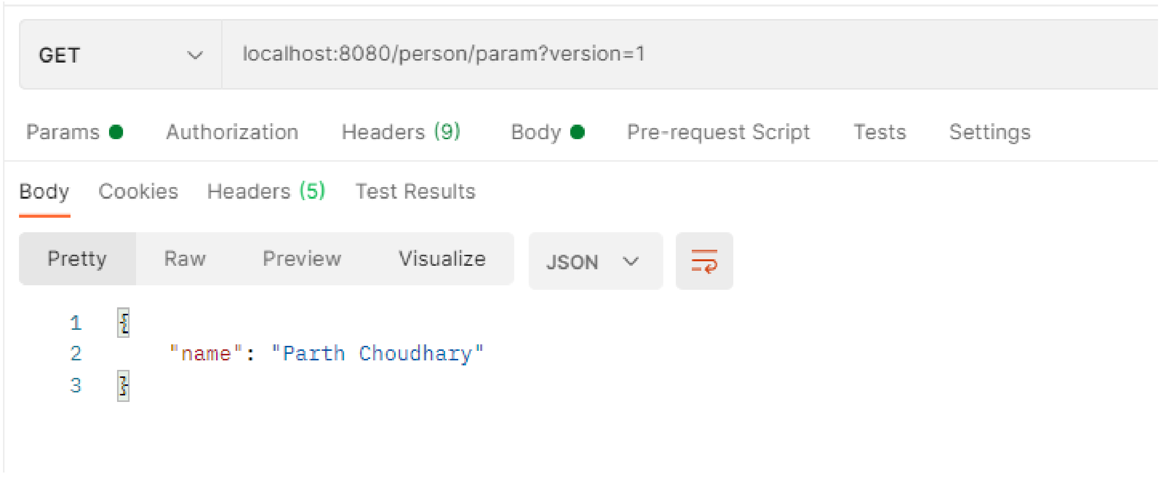
Body | Cookies | Headers (5) | Test Results

Pretty | Raw | Preview | Visualize | JSON

```
{
    "name": {
        "firstName": "Parth",
        "lastName": "Choudhary"
    }
}
```

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"));
}
```



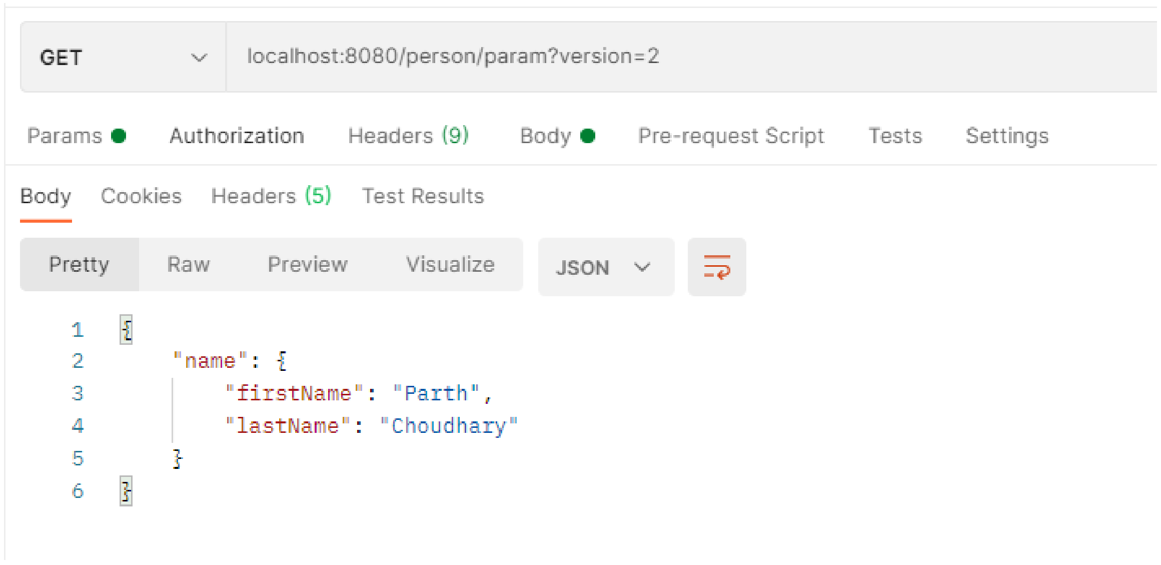
GET localhost:8080/person/param?version=1

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

Body | Cookies | Headers (5) | Test Results

Pretty | Raw | Preview | Visualize | JSON

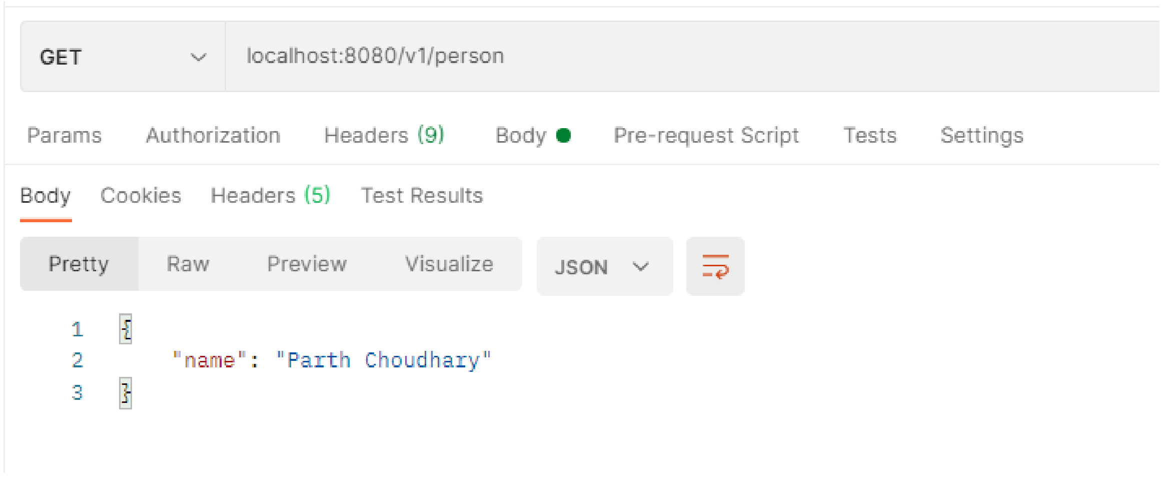
```
{
    "name": "Parth Choudhary"
}
```


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

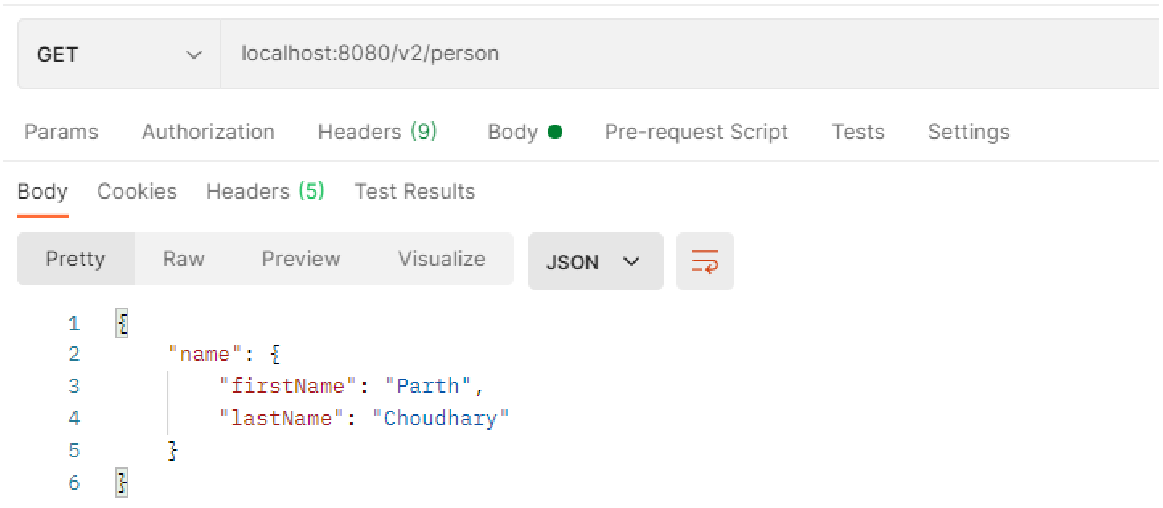
```
{
    "name": {
        "firstName": "Parth",
        "lastName": "Choudhary"
    }
}
```

URI versioning

```java
//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

```
{
    "name": "Parth Choudhary"
}
```



GET localhost:8080/v2/person

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

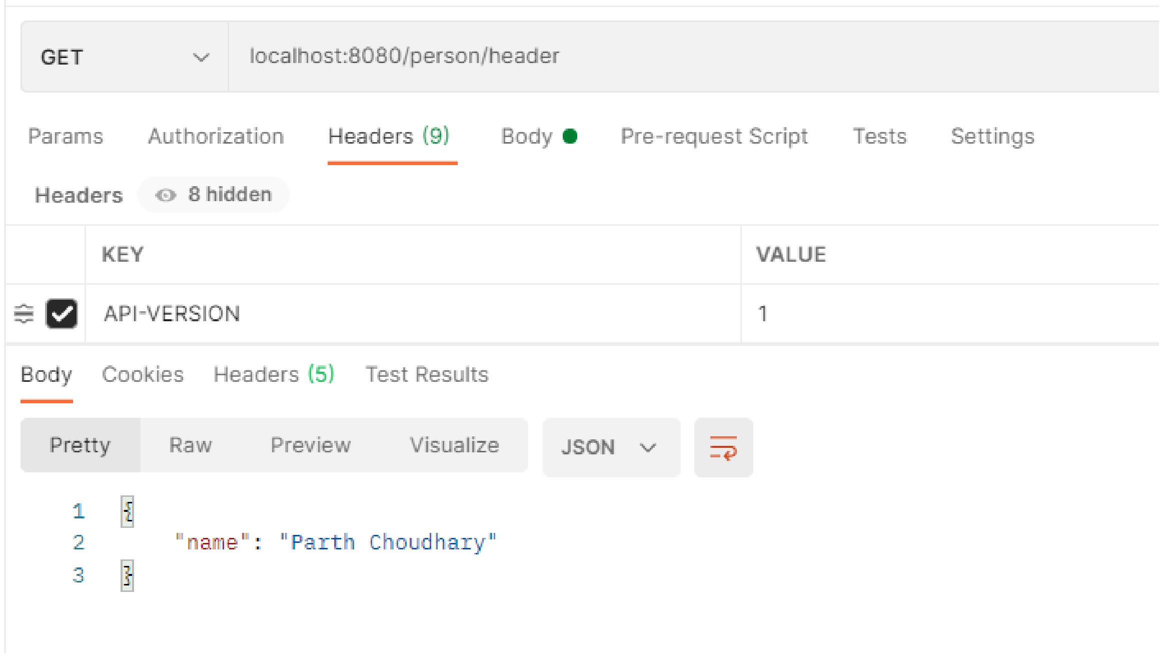
Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
{
    "name": {
        "firstName": "Parth",
        "lastName": "Choudhary"
    }
}
```

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

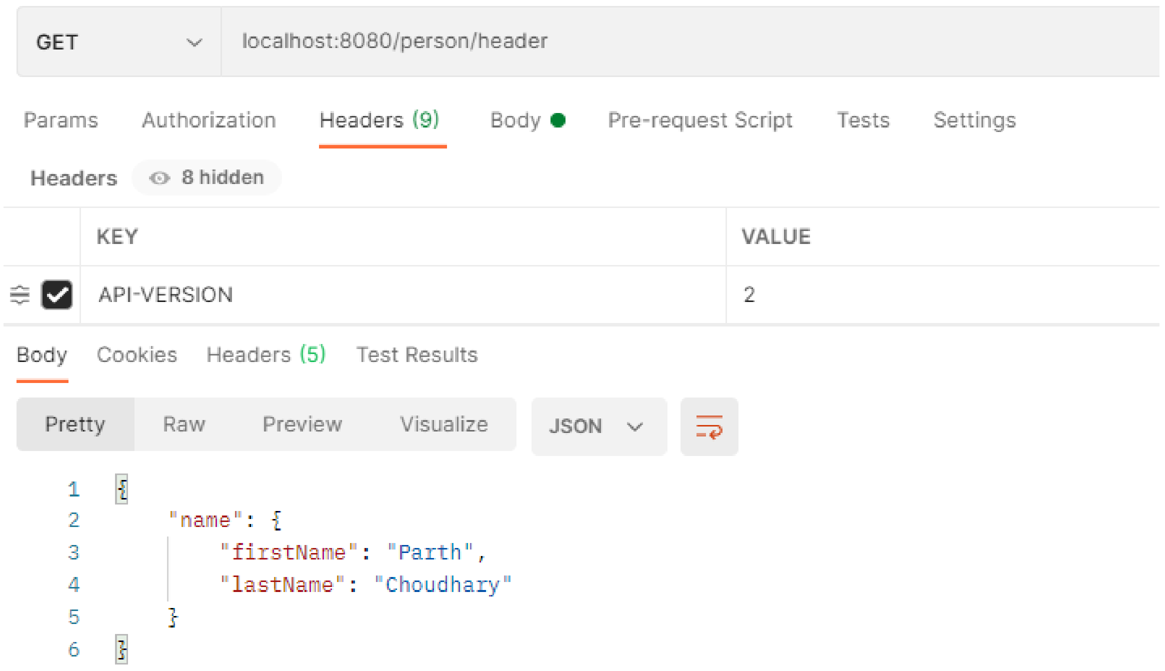
Headers 8 hidden

	KEY	VALUE
✓	API-VERSION	1

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
{
    "name": "Parth Choudhary"
}
```

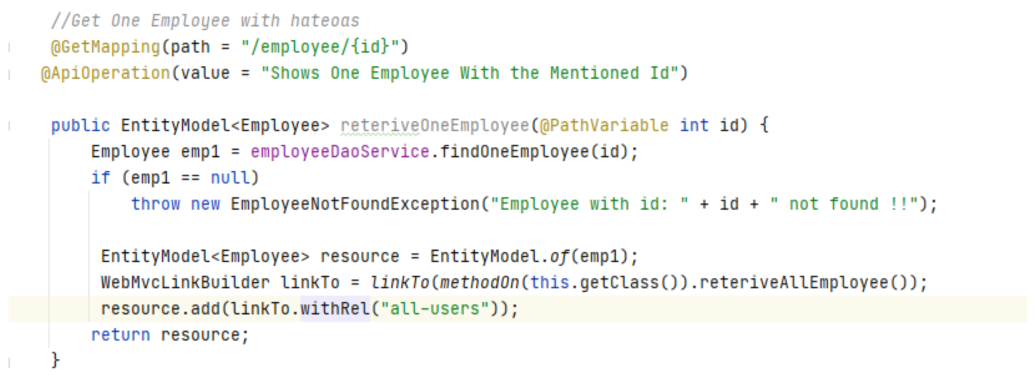


localhost:8080/person/header

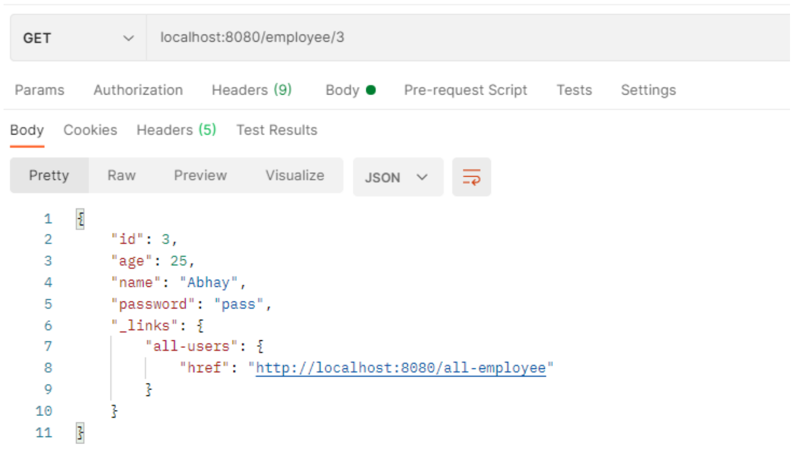
KEY	VALUE
API-VERSION	2

```
{
    "name": {
        "firstName": "Parth",
        "lastName": "Choudhary"
    }
}
```

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> reteriveOneEmployee(@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()).reteriveAllEmployee());
    resource.add(linkTo.withRel("all-users"));
    return resource;
}
```



localhost:8080/employee/3

```
{
    "id": 3,
    "age": 25,
    "name": "Abhay",
    "password": "pass",
    "_links": {
        "all-users": {
            "href": "http://localhost:8080/all-employee"
        }
    }
}
```