Internationalization

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

```
hello.message=Hello
```

```
hello.message=Hallo
```

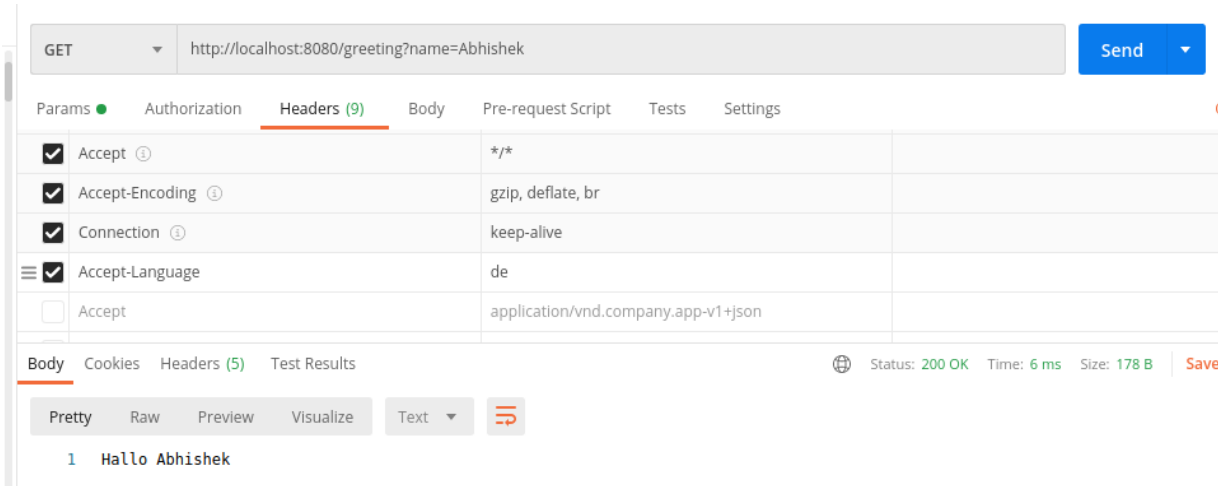
```
hello.message=Hallå
```

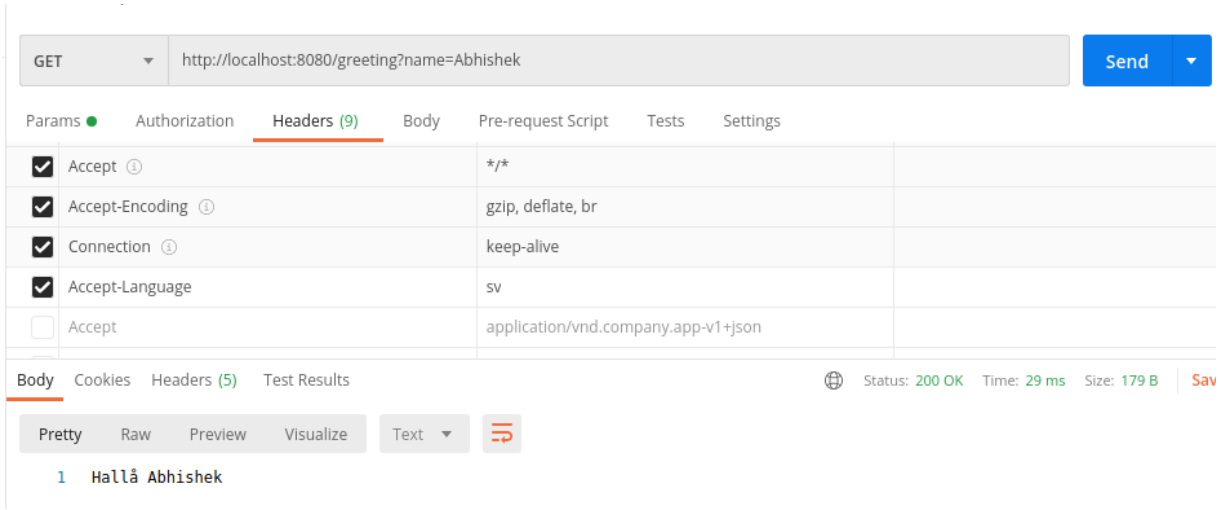
```
#management.endpoints.web.exposure.include=*
spring.messages.basename=messages
```

```
@Bean
public LocaleResolver localeResolver(){
    AcceptHeaderLocaleResolver localeResolver = new AcceptHeaderLocaleResolver();
    localeResolver.setDefaultLocale(Locale.US);
    return localeResolver;
}
```

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

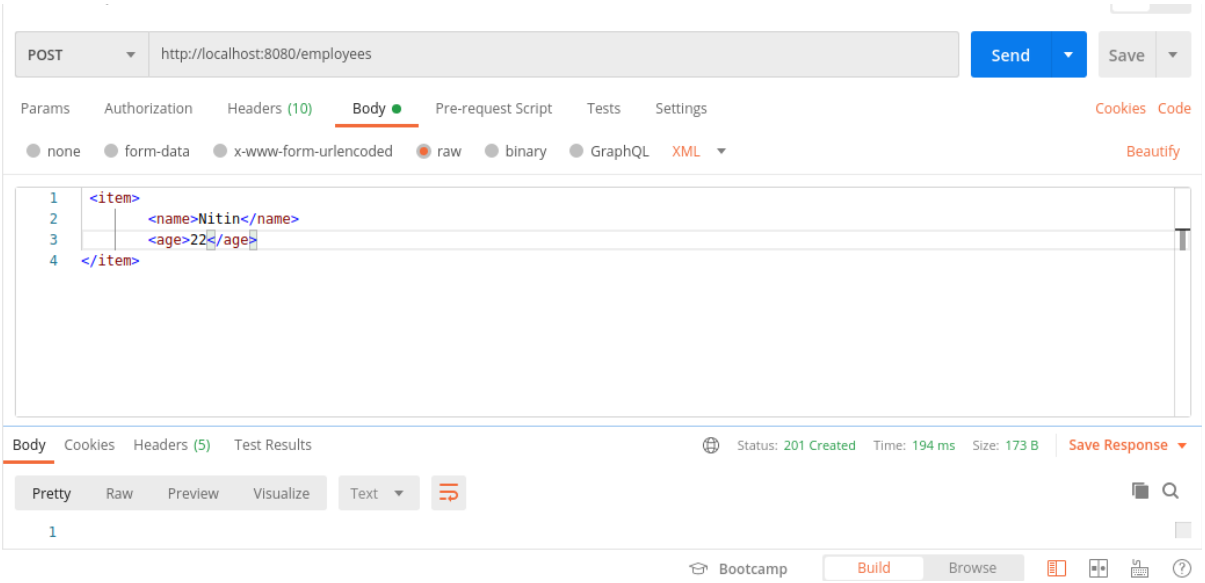
```
@GetMapping("/greeting")
@ApiModelProperty(notes = "Greeting according to language")
public String helloWorldInternationalized(@RequestParam(name = "name") String username){
    return messageSource.getMessage( s: "hello.message", objects: null, LocaleContextHolder.getLocale())+" "+username;
}
```



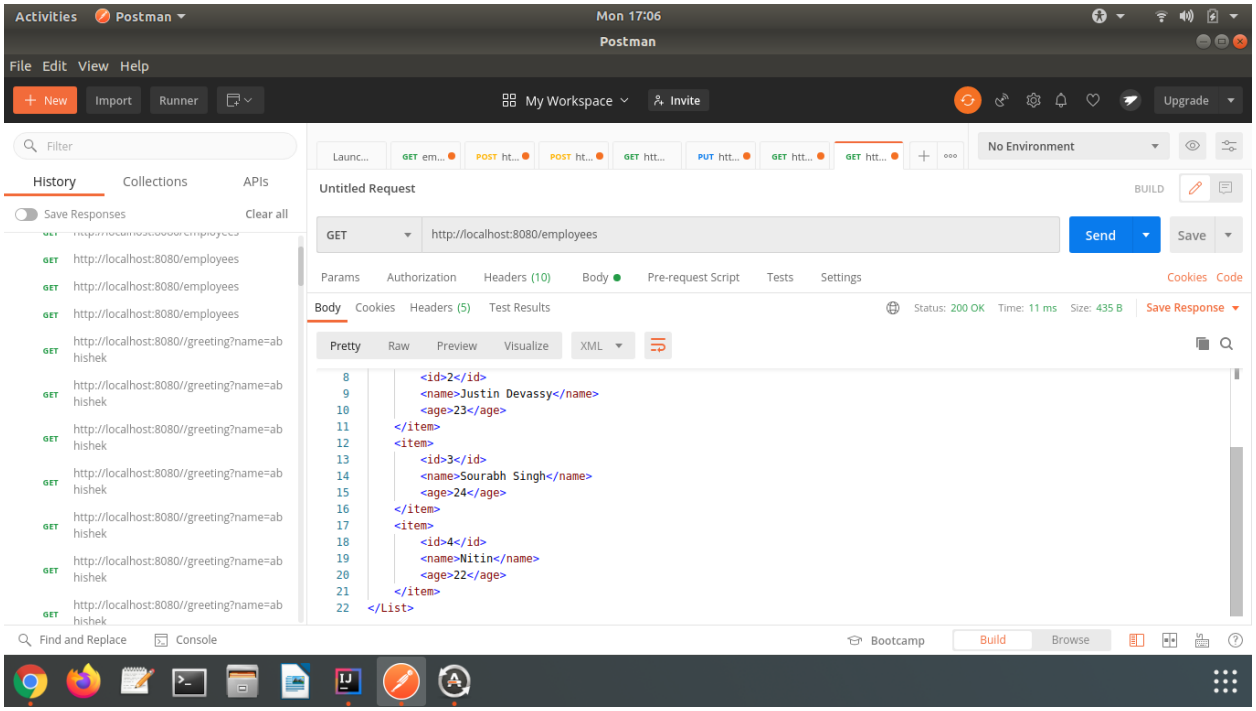


*Content Negotiation

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



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



*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	Response body

```
{
  "id": 1,
  "name": "Abhisehk Bhardwaj",
  "age": 23,
  "_links": {
    "List-all-employees": {
      "href": "http://localhost:8080/employees"
    }
  }
}
```

Download

Response headers

```
connection: keep-alive
content-type: application/hal+json
date: Tue09 Mar 2021 09:52:46 GMT
keep-alive: timeout=60
transfer-encoding: chunked
```

Save details of the user using POST request.

Request URL

```
http://localhost:8080/employees
```

Server response

Code	Details
201	Response headers

```
connection: keep-alive
content-length: 0
date: Tue09 Mar 2021 09:54:38 GMT
keep-alive: timeout=60
location: http://localhost:8080/employees/4
```

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	Response headers

```
connection: keep-alive
content-length: 0
date: Tue09 Mar 2021 09:56:07 GMT
keep-alive: timeout=60
```

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.

```java
@Autowired
private EmployeeDaoService service;
@GetMapping("/employees")
@ApiModelProperty(notes = "Fetch list of Employees in List")
public List<EmployeeBean> retrieveAllEmployee() { return service.getEmployee(); }
@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;
}
@PostMapping("/employees")
@ApiModelProperty(notes = "Add Employees in a List")
public ResponseEntity<Object> createEmployee(@Valid @RequestBody EmployeeBean emp){
```

```
com.Application.webService.EmployeeService
@GetMapping("/employees")
public List<EmployeeBean> retrieveAllEmployee()
```

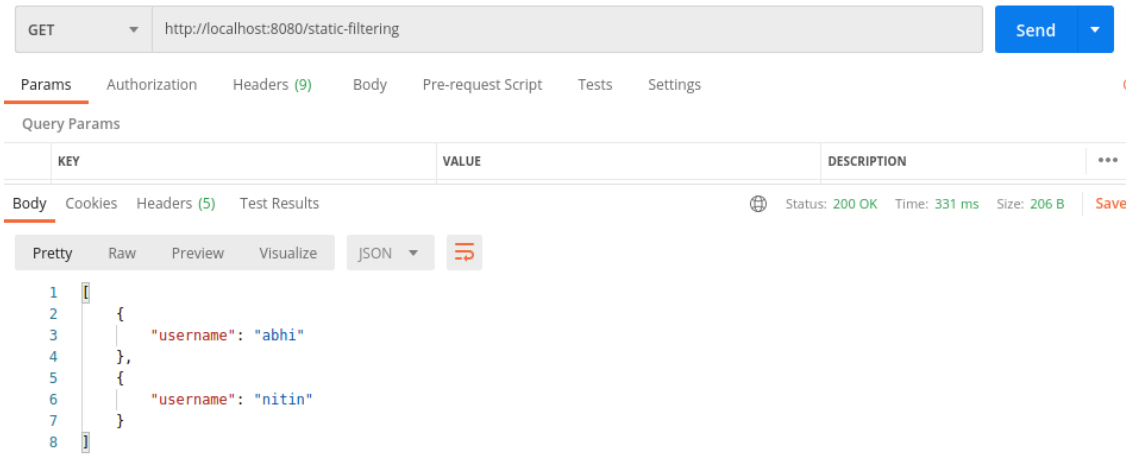
EmployeeBean {

description:	This is EmployeeBean class
age	integer($int32) *minimum: 18* *exclusiveMinimum: false* **Age should be more then 18 year**
id	integer($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)




```
    @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; }
```

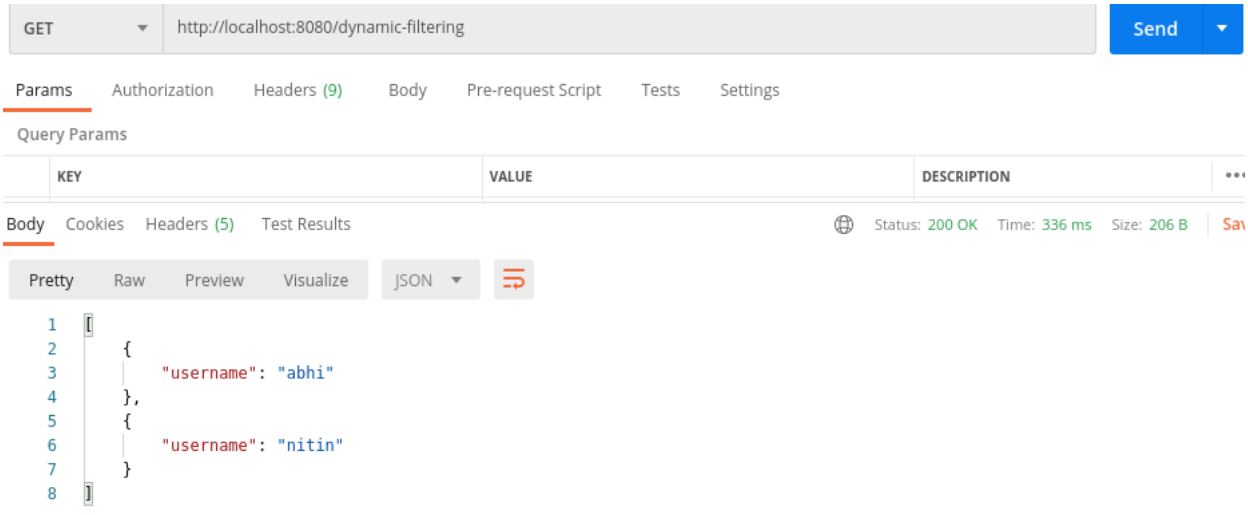
9. Create another API that does the same by using Dynamic Filtering.

```java
@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;
}
```

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

    public String getUsername() { return username; }
```



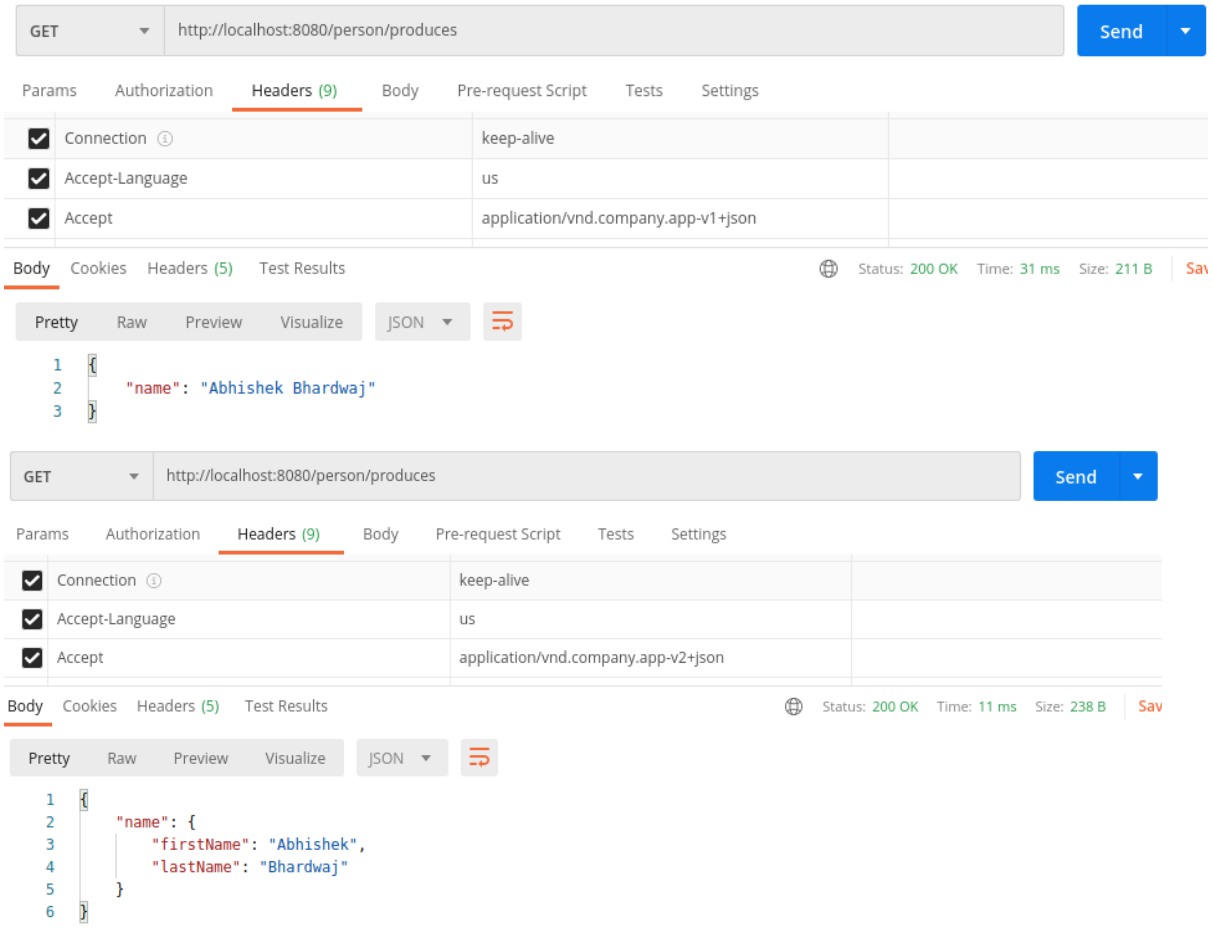
*Versioning Restful APIs

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:

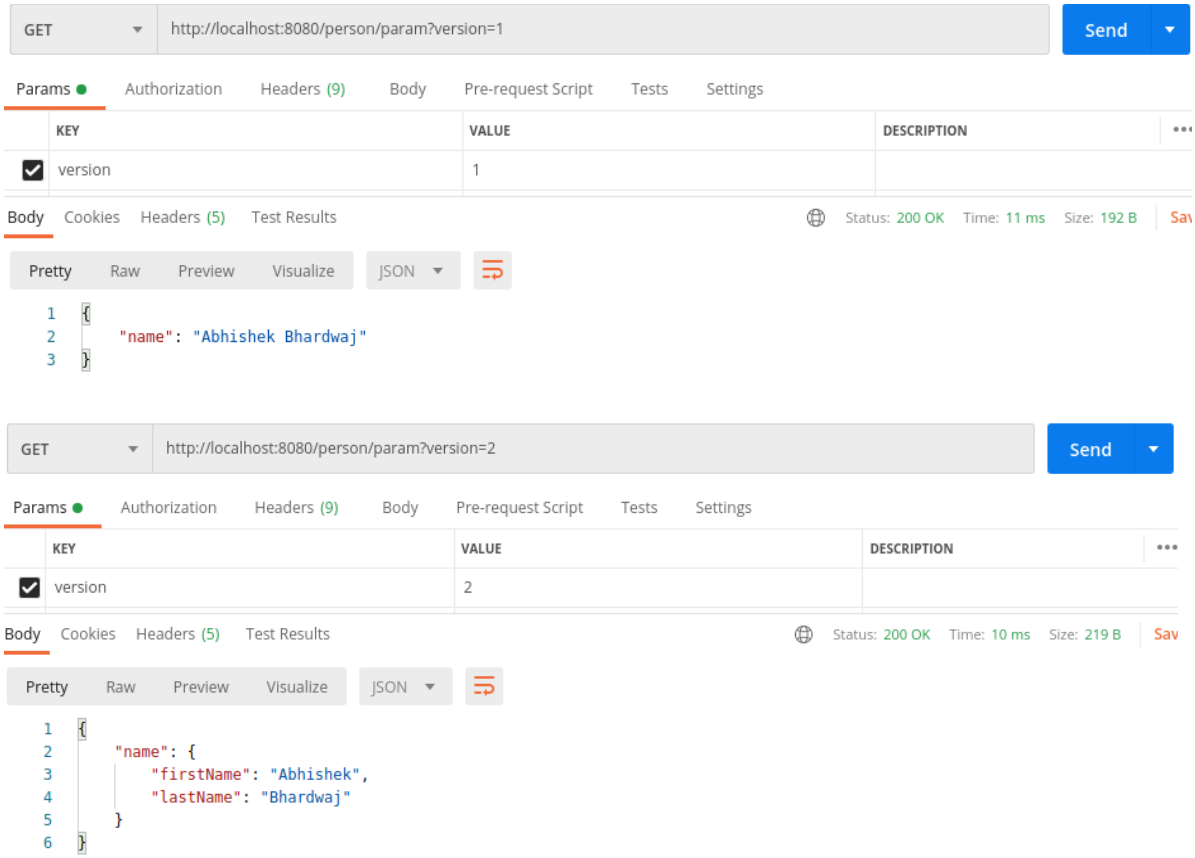
- MimeType Versioning

```java
@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")); }
```



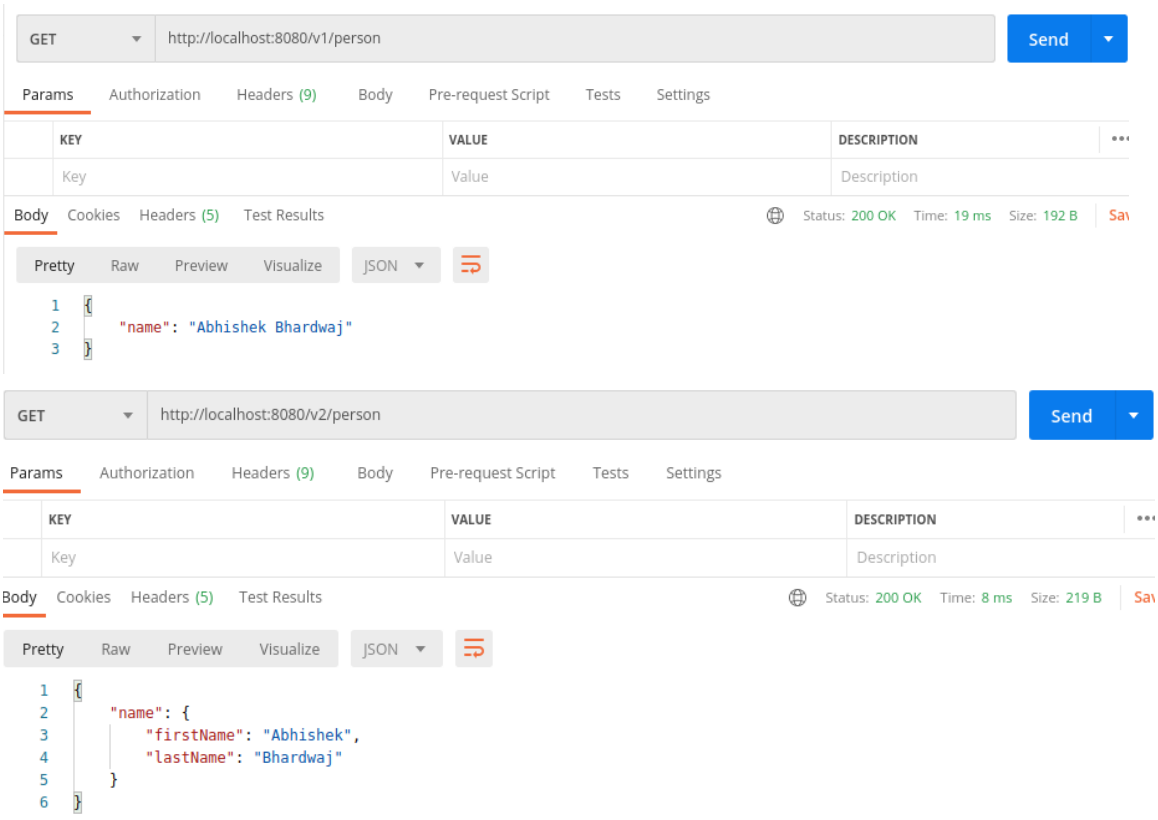
- Request Parameter versioning

```java
@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")); }
```



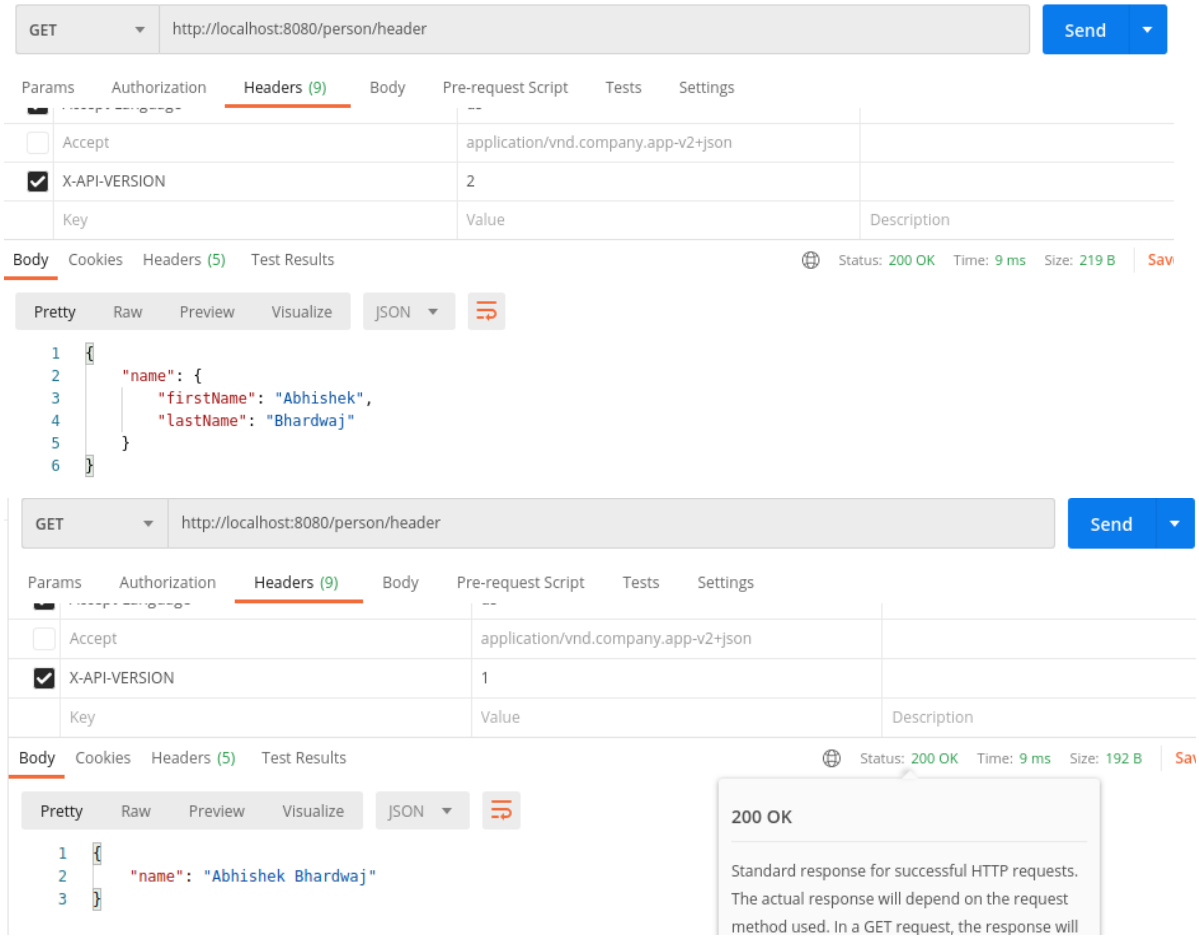
- URI versioning

```
@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")); }
```



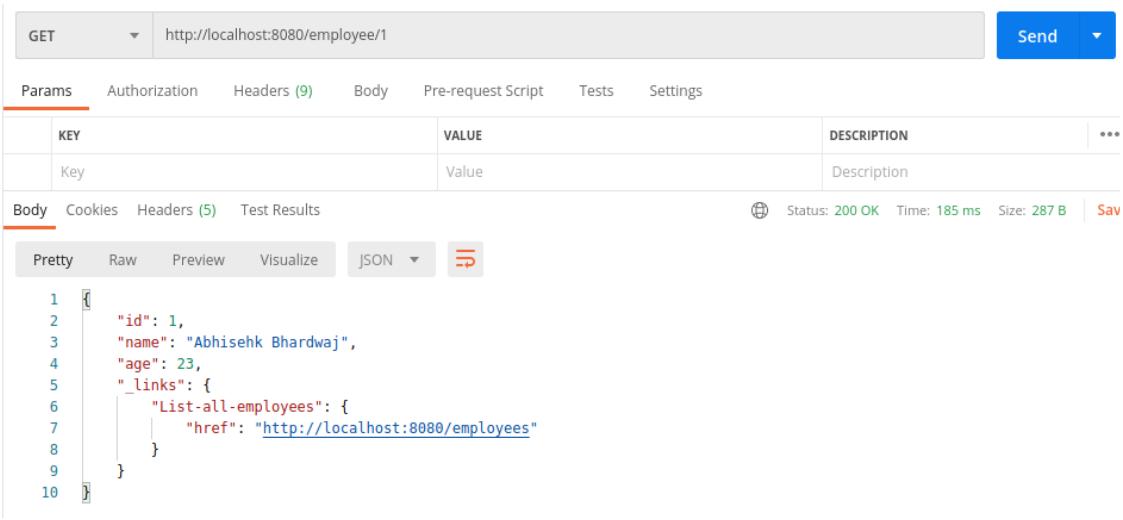
- Custom Header Versioning

```
@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")); }
```



*HATEOAS

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




```java
@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;
}
```