Session : RestFul Web Service Part 2

*Internationalization

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

Inside Person Controller.java

```
@RestController
public class PersonController {
    @Autowired
    PersonService personService;
    @Autowired
    MessageSource messageSource;
    // internationalization
    // allowing messages to be shown in English, German and
Swedish,
    // keeping English as default.
    // takes "username" as param

    @Operation(summary = "internationalization of string\n allowing
messages to be shown in English, German and Swedish,")

    @GetMapping("/users/international")
    public String EmployeeInternationalization()
    {
        //return "hello string";
        return messageSource.getMessage("good.morning.message",
null, LocaleContextHolder.getLocale());
    }


}
```

inside java.config

```
@Bean
// internationalization
// setting up locale and default locale for greetings in different
language
public LocaleResolver localeResolver()
{
    AcceptHeaderLocaleResolver localeResolver=new
AcceptHeaderLocaleResolver();
    localeResolver.setDefaultLocale(Locale.US);
    return localeResolver;
}    // also added spring.messages.basename=messages to
application.properties
```

Inside message.properties

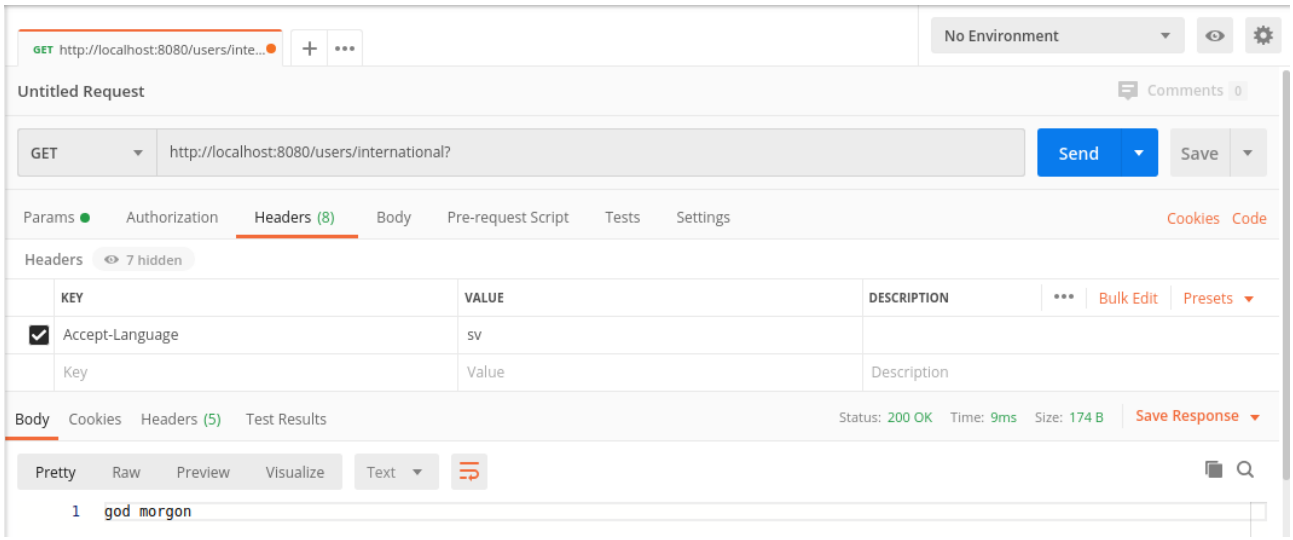
```
good.morning.message=Good morning
```

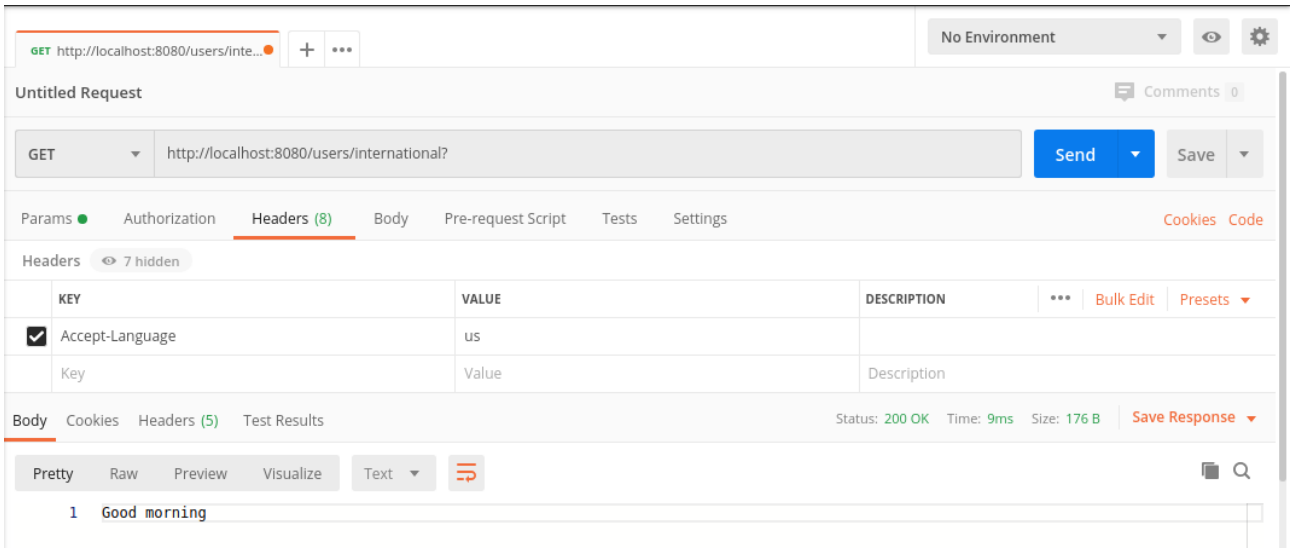
Inside message_sv.properties

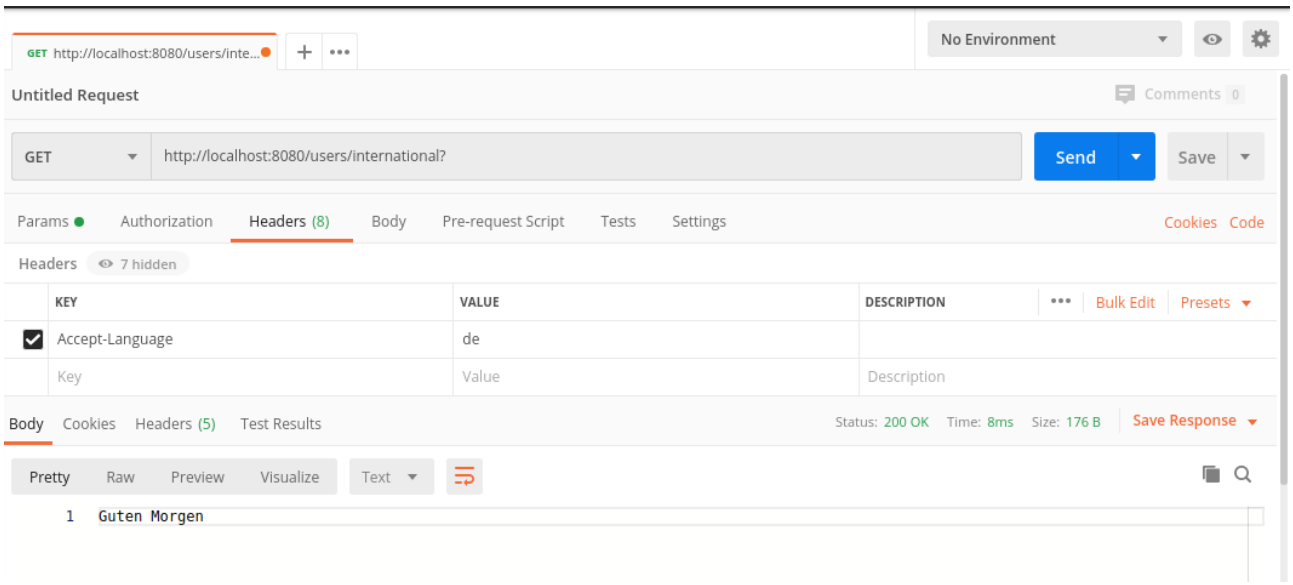
```
good.morning.message=god morgon
```

Inside message_de.properties

```
good.morning.message=Guten Morgen
```







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

Inside Person-Controller.java

```
@RestController
public class PersonController {
    @Autowired
    PersonService personService;
    @Autowired
    MessageSource messageSource;
    // internationalization
    // allowing messages to be shown in English, German and
Swedish,
    // keeping English as default.
    // takes "username" as param
    // shows a localized message "Hello Username". (Use parameters
in message properties)

    @Operation(summary = "internationalization of username\n
allowing messages to be shown in English, German and Swedish,")


    @GetMapping(path = "/greetings/{username}")
    public String sayHello(@PathVariable String username) {
        return messageSource.getMessage("hello.messages", new
String[]{username}, LocaleContextHolder.getLocale());
    }
```

inside java.config

```
@Bean
// internationalization
// setting up locale and default locale for greetings in different
language
public LocaleResolver localeResolver()
{
    AcceptHeaderLocaleResolver localeResolver=new
AcceptHeaderLocaleResolver();
    localeResolver.setDefaultLocale(Locale.US);
    return localeResolver;
}    // also added spring.messages.basename=messages to
application.properties
```

Inside message.properties

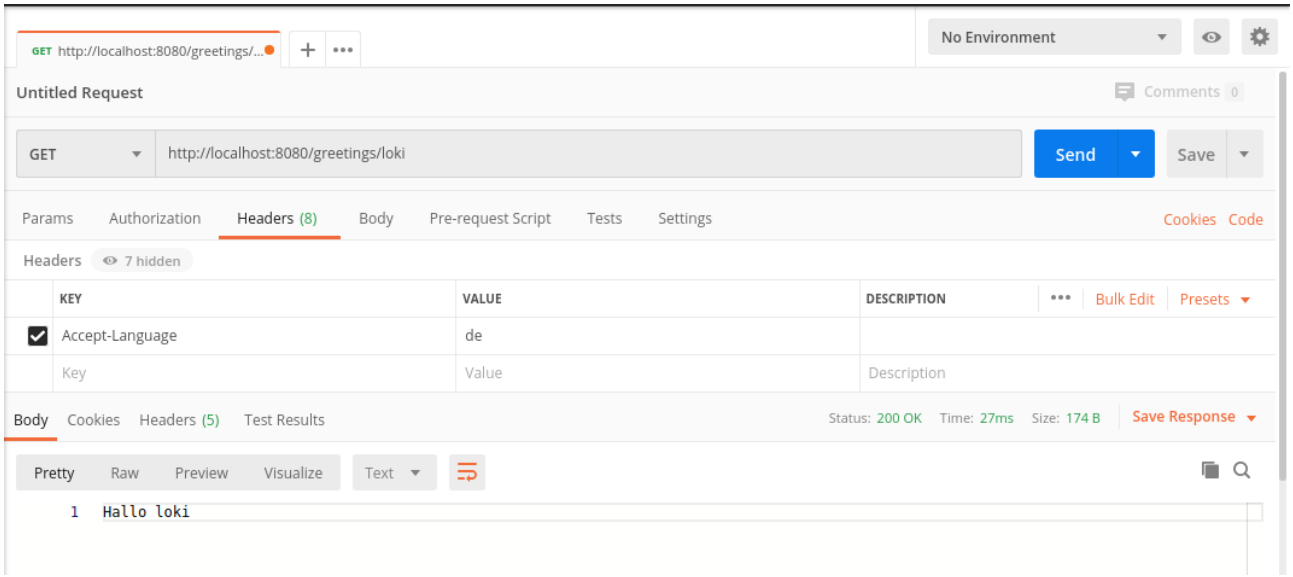
```
hello.messages=Hello {0}
```

Inside message_sv.properties

```
hello.messages=Hej {0}
```

Inside message_de.properties

```
hello.messages=Hallo {0}
```



GET http://localhost:8080/greetings/loki Send Save

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies Code

Headers 7 hidden

KEY	VALUE	DESCRIPTION
Accept-Language	sv	
Key	Value	Description

Bulk Edit Presets

Body Cookies Headers (5) Test Results Status: 200 OK Time: 21ms Size: 171 B Save Response

Pretty Raw Preview Visualize Text

```
1 Hej loki
```

Untitled Request Comments 0

GET http://localhost:8080/greetings/loki Send Save

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies Code

Headers 7 hidden

KEY	VALUE	DESCRIPTION
Accept-Language	us	
Key	Value	Description

Bulk Edit Presets

Body Cookies Headers (5) Test Results Status: 200 OK Time: 14ms Size: 174 B Save Response

Pretty Raw Preview Visualize Text

```
1 Hello loki
```

*Content Negotiation

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

Add following depenedency in

```
//  dependency to show data in xml format
   compile group: 'com.fasterxml.jackson.dataformat', name:
'jackson-dataformat-xml'
```

In PersonCotroller.java

Post Method

```
//*Content Negotiation
//POST Method to create user details which can accept XML for user
creation.
//by adding com.fasterxml.jackson.dataformat dependency to
build.gradle
@Operation(summary = "posting a specific person & accept the data
in XML format")
@ApiResponse(description = "Successful Operation", responseCode =
"200", content = @Content(mediaType = "application/json"))
@PostMapping(path = "/persons", consumes = "application/xml")
public ResponseEntity<Person> createPerson(@RequestBody Person
person) {
    // adding person
    Person savedPerson = personService.addPerson(person);
    return new ResponseEntity(savedPerson, HttpStatus.OK);
}
```

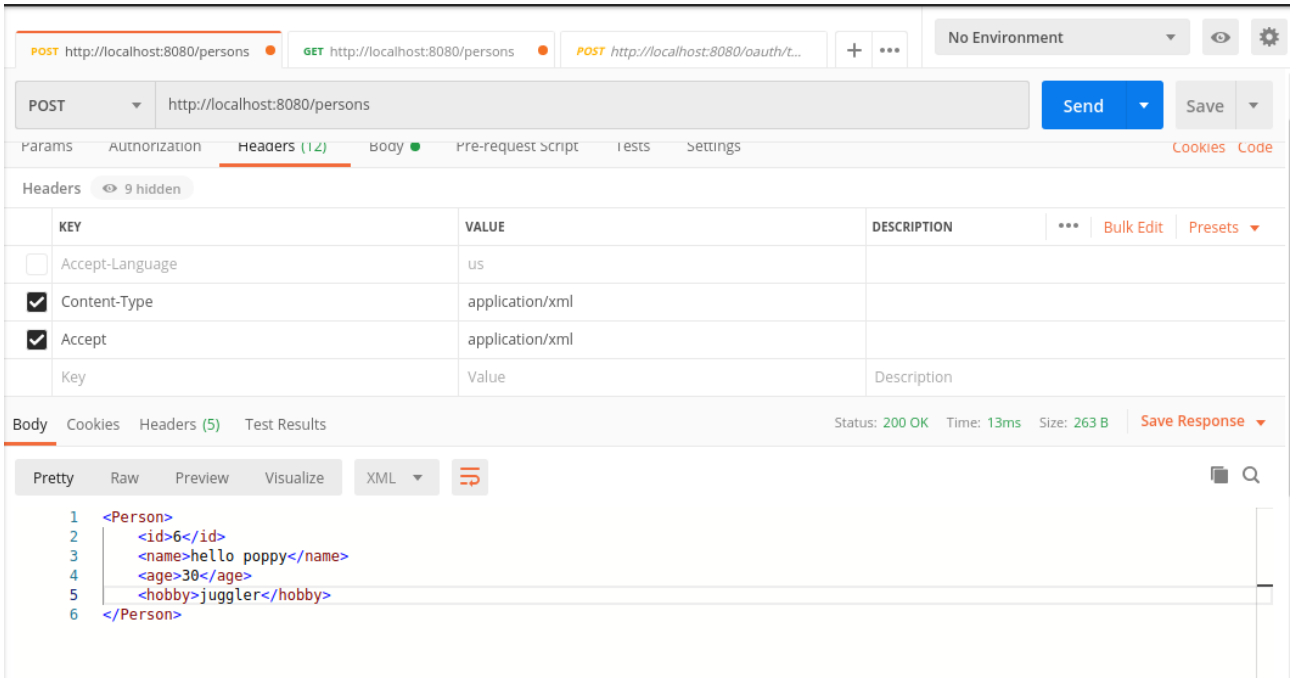
In PersonService.java

```
// logic to add a person to the list

public Person addPerson(Person person){
    if(person.getId()==null){
        person.setId(personCount+1);
    }
        personList.add(person);
        return person;
}

// while hiting the ponit end pont add following parameters in
headers
```

Content-Type application/xml

or

Accept application/xml



Q4. Create GET Method to fetch the list of users in XML format.

```
//  dependency to show data in xml format

   compile group: 'com.fasterxml.jackson.dataformat', name:
'jackson-dataformat-xml'
```

Inside PersonController.java

```
//getting the whole persons list
@Operation(summary = "Returns the whole list of persons")
@GetMapping(path = "/persons", consumes = "application/xml")
public List<Person> getPersonList() {
    return personService.getPersonList();
}
```

Inside PersonService.java

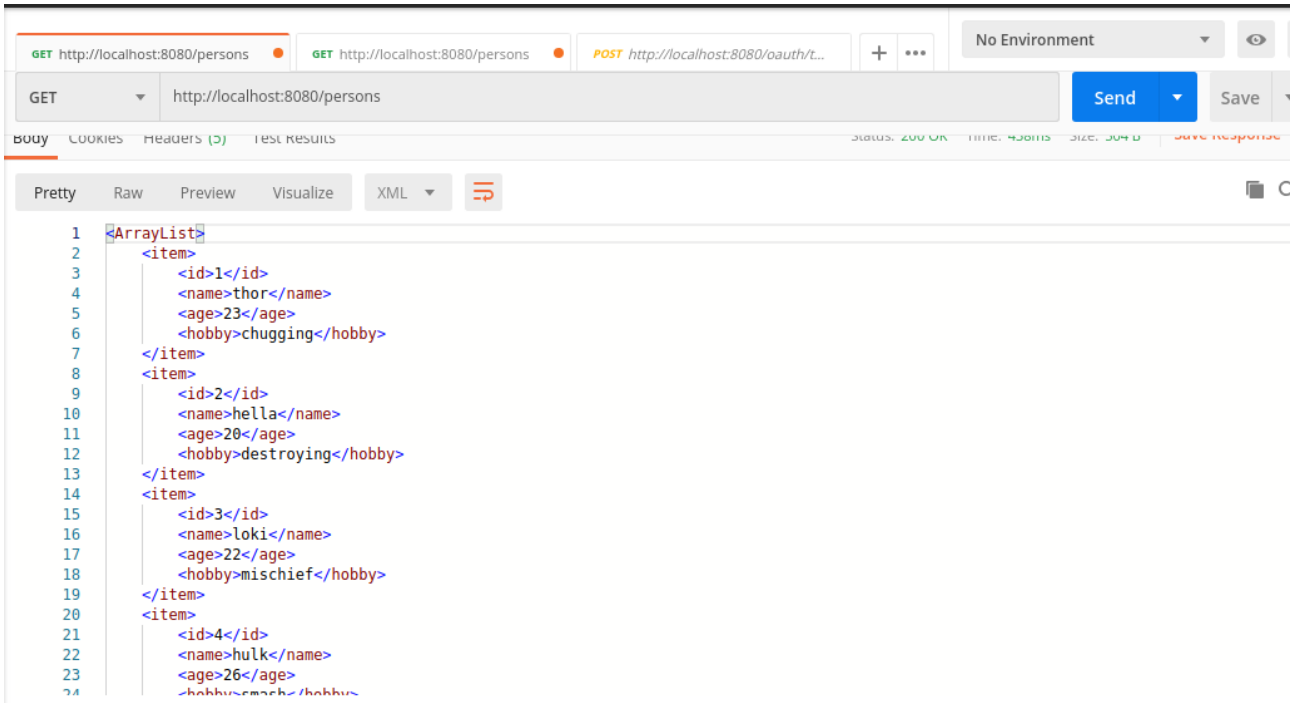
```
public List<Person> getPersonList() {

    return personList;
}
```

// while hiting the ponit end pont add following parameters in headers

Content-Type application/xml

or

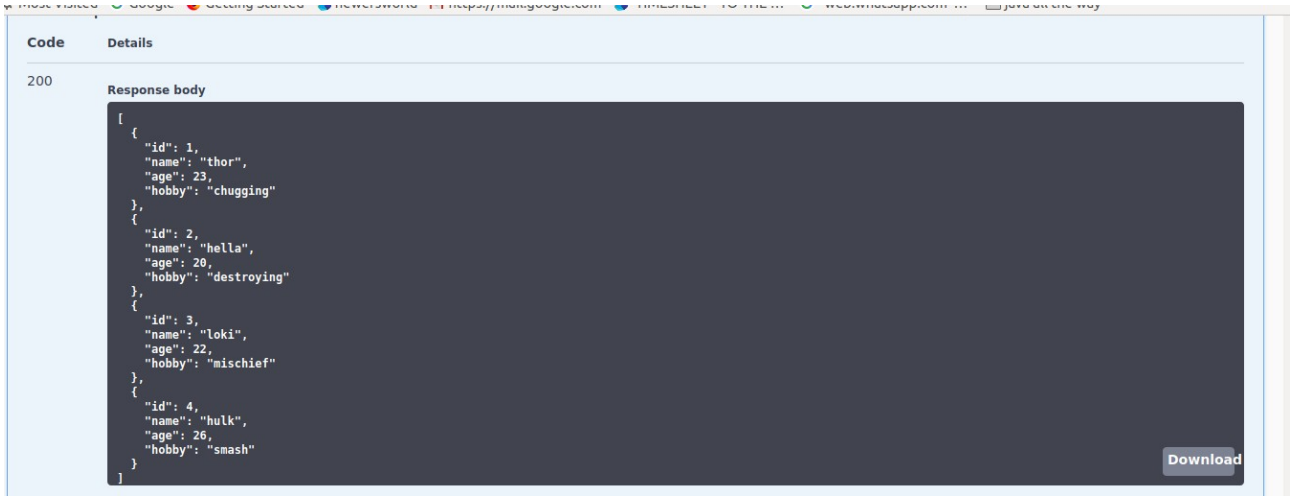
Accept application/xml



*Swagger

Q5. Configure swagger plugin and create document of following methods:

Get details of User using GET request.



Save details of the user using POST request.

POST /persons posting a specific person & accept the data in XML format

Parameters Cancel

No parameters

Request body application/json

```
{
    "name": "pegan",
    "password": "odin-son",
    "age": 23,
    "hobby": "worship"
}
```

201 Undocumented

Response body

```
{
  "id": 5,
  "name": "pegan",
  "age": 23,
  "hobby": "worship"
}
```

Download

Response headers

```
connection: keep-alive
content-type: application/json
date: Sun, 05 Apr 2020 08:34:04 GMT
keep-alive: timeout=60
transfer-encoding: chunked
```

Delete a user using DELETE request.

Parameters Cancel

Name	Description
id * required integer (path)	5

Execute Clear

Responses

200

Response body

```
    "hobby": "chugging"
  },
  {
    "id": 2,
    "name": "hella",
    "age": 20,
    "hobby": "destroying"
  },
  {
    "id": 3,
    "name": "loki",
    "age": 22,
    "hobby": "mischief"
  },
  {
    "id": 4,
    "name": "hulk",
    "age": 26,
    "hobby": "smash"
  },
  {
    "id": 6,
    "name": "pegan",
    "age": 23,
    "hobby": "worship"
  }
]
```

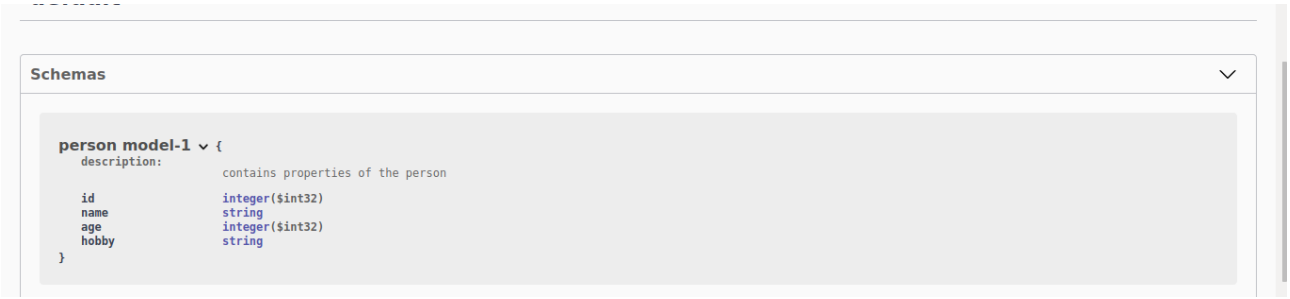
Download

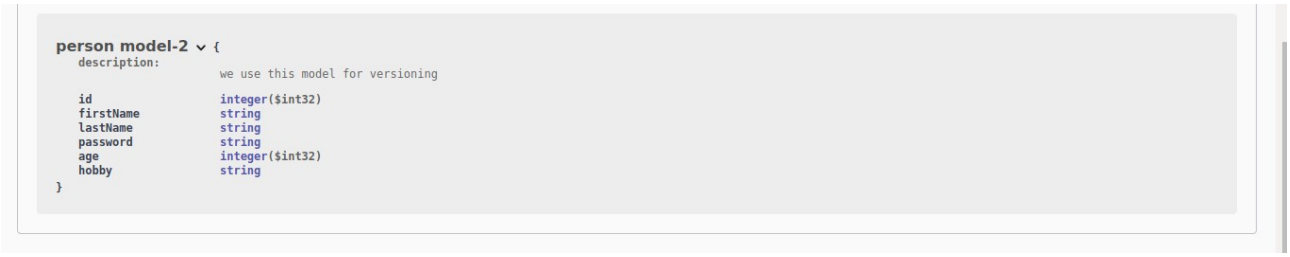
Q6. In swagger documentation, add the description of each class and URI so that in swagger UI the purpose of class and URI is clear.

Add

```
//  dependency for swagger2

   compile group: 'org.springdoc', name: 'springdoc-openapi-core',
version: '1.1.47'
// dependency for swagger-ui
   compile group: 'org.springdoc', name: 'springdoc-openapi-ui',
version: '1.1.47'
   compile group: 'org.springdoc', name: 'springdoc-openapi-
webflux-ui', version: '1.1.47'
```

```
//  dependency for swagger2
    compile group: 'org.springdoc', name: 'springdoc-openapi-core', version: '1.1.47'
//  dependency for swagger-ui
    compile group: 'org.springdoc', name: 'springdoc-openapi-ui', version: '1.1.47'
    compile group: 'org.springdoc', name: 'springdoc-openapi-webflux-ui', version: '1.1.47'
```





***Static and Dynamic filtering**

Q7 Create API which saves details of User (along with the password) but on successfully saving returns only non-critical data. (Use static filtering)

use of @jsonignore annotation on entity property

// inside person.java class

```
@Schema(title = "person model-1",description = "contains
properties of the person")
public class Person {
    private Integer id;
    private String name;
    @JsonIgnore // static filtering
    private String password;
    private int age;
    private String hobby;
    public Person(){
    }
    public Person(Integer id, String password,String name, int age,
String hobby){
        this.id=id;
        this.name=name;
        this.age=age;
        this.hobby=hobby;
        this.password=password;
    }

    // getter and setter
```

Inside PersonController.java

```
@Operation(summary = "Returns the whole list of persons")
@GetMapping(path = "/persons") // consumes = "application/xml"
public ResponseEntity<Person> getPersonList() {
    List<Person> personList= personService.getPersonList();
    return new ResponseEntity(personList,HttpStatus.OK);
}
```

inside PesronService.java

```
public List<Person1> getPersonList1() {
    return personList1;
}
```

// password field is being filtered out

Pretty Raw Preview Visualize JSON


```
[
    {
        "name": "thor",
        "hobby": "chugging"
    },
    {
        "name": "hella",
        "hobby": "destroying"
    },
    {
        "name": "loki",
        "hobby": "mischief"
    },
    {
        "name": "hulk",
        "hobby": "smash"
    }
]
```


Q8. Create another API that does the same by using Dynamic Filtering.

```
//Dynamic Filtering

// sending only the person name and its hobby in the response
@Operation(summary = "Dynamic Filtering sending only the person
name and its hobby in the response\n")
@GetMapping(path = "/persons/hobbies")
public MappingJacksonValue getHobbies() {
    List<Person> personList = personService.getPersonList(); //
return person list
    SimpleBeanPropertyFilter filter =
SimpleBeanPropertyFilter.filterOutAllExcept("name", "hobby");
    FilterProvider hobbyFilter = new
SimpleFilterProvider().addFilter("hobbyFilter", filter); //
creating the filter
    MappingJacksonValue hobbyMap = new
MappingJacksonValue(personList); // map the response base on the
filter
    hobbyMap.setFilters(hobbyFilter);
    return hobbyMap;
}

// make sure to add @JsonFilter("hobbyFilter")  above entity

Inside Person.java entity
```

```java
@Schema(title = "person model-1",description = "contains
properties of the person")
@JsonFilter("hobbyFilter") // to apply dynamic filtering uncomment
this

public class Person {

    private Integer id;
    private String name;
    @JsonIgnore // static filtering
    private String password;
    private int age;
    private String hobby;
    public Person(){
    }
    public Person(Integer id, String password,String name, int age,
String hobby){
        this.id=id;
        this.name=name;
        this.age=age;
        this.hobby=hobby;
        this.password=password;
    }

// getter and setter

public List<Person> getPersonList() {

    return personList;
}
```

Pretty Raw Preview Visualize JSON


```json
[
    {
        "name": "thor",
        "hobby": "chugging"
    },
    {
        "name": "hella",
        "hobby": "destroying"
    },
    {
        "name": "loki",
        "hobby": "mischief"
    },
    {
        "name": "hulk",
        "hobby": "smash"
    }
]
```

***Versioning Restful APIs**

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,

```
// Person Version 1 less detail version
@Schema(title = "person model-1",description = "contains
properties of the person")
public class Person {
    private Integer id;
    private String name;
    private String password;
    private int age;
    private String hobby;
    public Person(){
    }
    public Person(Integer id, String password,String name, int age,
String hobby){
        this.id=id;
        this.name=name;
        this.age=age;
        this.hobby=hobby;
        this.password=password;
    }

// Person Version 2 more detailed version
@Schema(title = "person model-2",description = "we use this model
for versioning")
public class Person1 {
    private Integer id;
    private String firstName;
    private String lastName;
    private String password;
    private int age;
    private String hobby;
    public Person1(Integer id, String firstName, String lastName,
String password, int age, String hobby) {
        this.id = id;
        this.firstName = firstName;
        this.lastName = lastName;
        this.password = password;
        this.age = age;
```

```
        this.hobby = hobby;
    }
```

Now apply versioning using the following methods:

- **MimeType Versioning**

```
     //  mime type versioning
// first uri gives basic detail version of the persons
@Operation(summary = "mime type versioning")
@GetMapping(value = "/persons/produces", produces =
"application/com.example.app-v1+json")
public ResponseEntity<List<Person>>  getPersonListV7() {
    List<Person> personList= personService.getPersonList();
    return new ResponseEntity<>(personList,HttpStatus.OK);
}
// second uri gives more enhanced details
@Operation(summary = "mime type versioning")
@GetMapping(value = "/persons/produces", produces =
"application/com.example.app-v2+json")
public ResponseEntity<List<Person1>>  getPersonListV8() {
    List<Person1> personList1= personService.getPersonList1();
    return new ResponseEntity<>(personList1,HttpStatus.OK);
}
```

Uri to hit

http://localhost:8080/persons/produces

with headrer

Accept: application/com.example.app-v1+json

http://localhost:8080/persons/produces

with headrer

Accept: application/com.example.app-v1+json

POST http://localhost:8080/persons | GET http://localhost:8080/persons/p... | POST http://localhost:8080/oauth/t...

No Environment

GET http://localhost:8080/persons/param?version=1 Send Save

Pretty Raw Preview Visualize JSON


```
[
    {
        "id": 1,
        "name": "thor",
        "age": 23,
        "hobby": "chugging"
    },
    {
        "id": 2,
        "name": "hella",
        "age": 20,
        "hobby": "destroying"
    },
    {
        "id": 3,
        "name": "loki",
        "age": 22,
        "hobby": "mischief"
    },
    {
        "id": 4,
        "name": "hulk",
        "age": 26,
        "hobby": "smash"
```


Bootcamp Build Browse

GET http://localhost:8080/persons/produces Send Save

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

Headers 7 hidden

KEY	VALUE	DESCRIPTION
API-VERSION	2	
Accept	application/com.example.app-v2+json	
Key	Value	Description

Bulk Edit Presets

Body Cookies Headers (5) Test Results Status: 200 OK Time: 8ms Size: 575 B Save Response

Pretty Raw Preview Visualize JSON

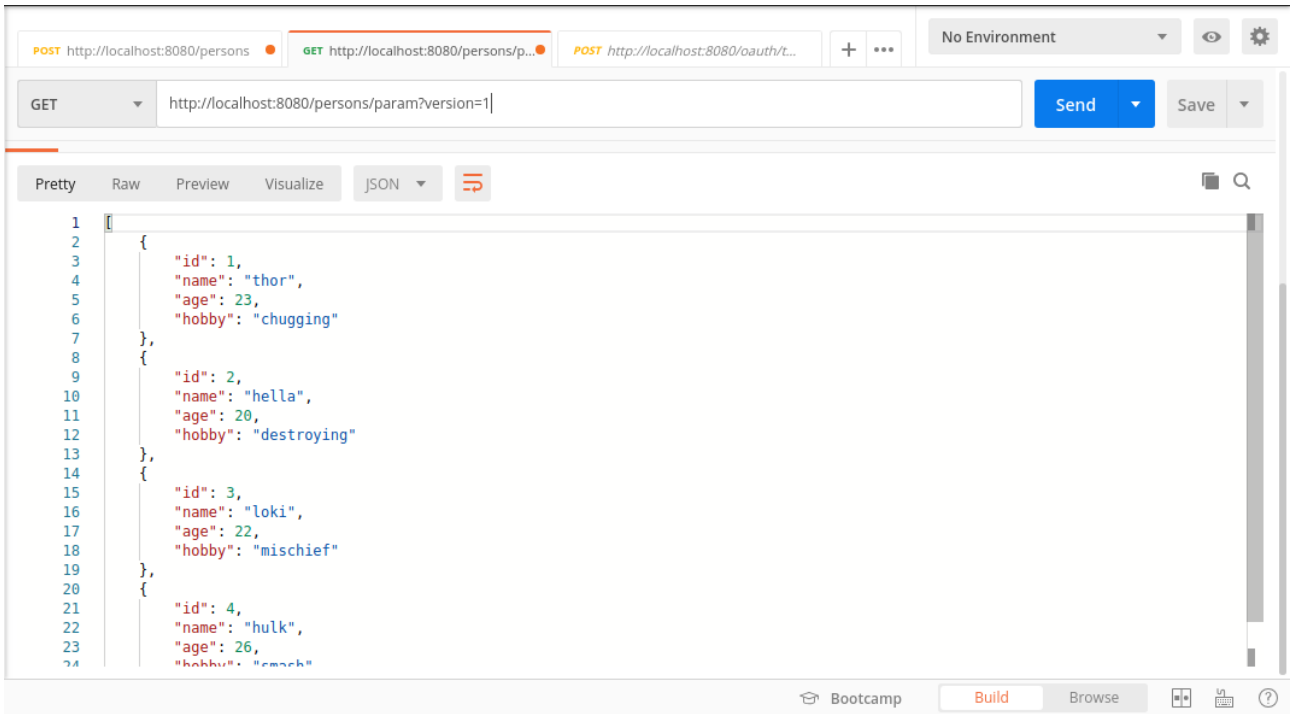

```
[
    {
        "id": 1,
        "firstName": "thor",
        "lastName": "odin son",
        "password": "pass1",
        "age": 23,
        "hobby": "chugging"
    },
    {
```

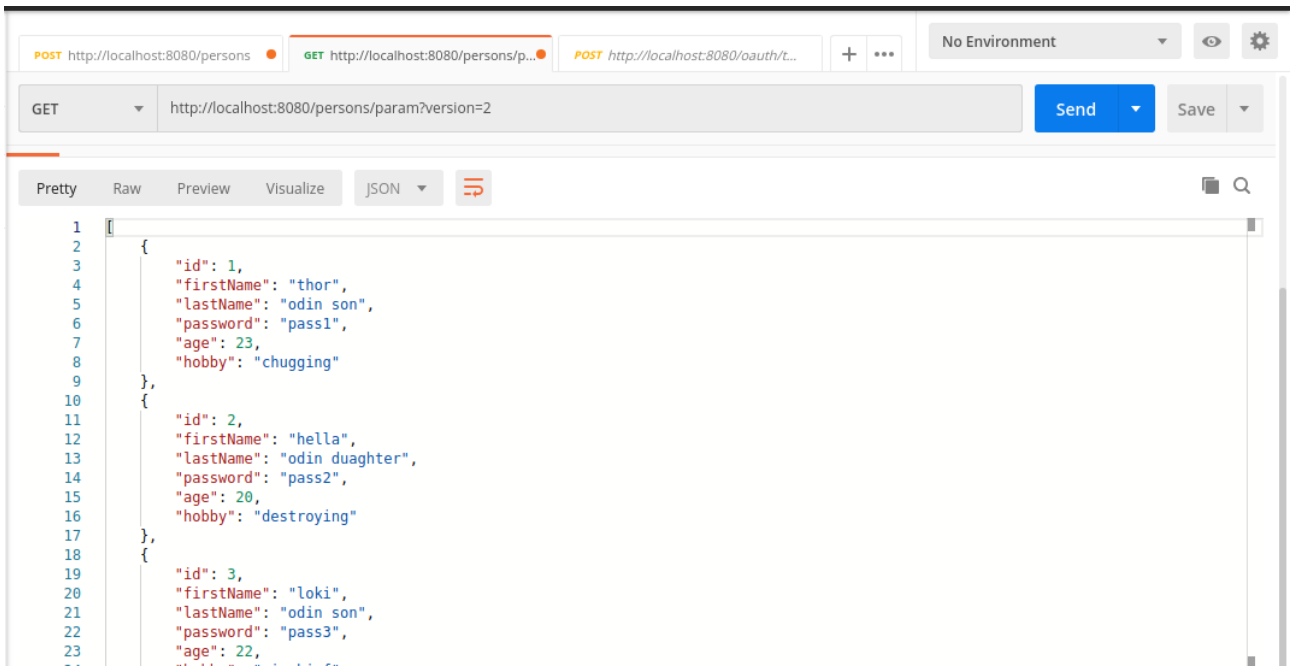
- **Request Parameter versioning**

```
    // parameter versioning
// first uri gives basic detail version of the persons
@Operation(summary = "parameter versioning")
@GetMapping(value = "/persons/param", params = "version=1")
public ResponseEntity<List<Person>>  getPersonListV3() {
    List<Person> personList= personService.getPersonList();
    return new ResponseEntity<>(personList,HttpStatus.OK);
}
// second uri gives more enhanced details
@Operation(summary = "parameter versioning")
@GetMapping(value = "/persons/param", params = "version=2")
public ResponseEntity<List<Person1>>  getPersonListV4()
{
    List<Person1> personList1 =personService.getPersonList1();
    return new ResponseEntity<>(personList1,HttpStatus.OK);
}
```

uri to hit are-

- **http://localhost:8080/persons/param?version=1**
- **http://localhost:8080/persons/param?version=2**





- **URI versioning**

```
    // uri versioning
// first uri gives basic detail version of the persons
@Operation(summary = "uri versioning first uri gives basic detail
version of the persons")
@GetMapping(path = "/persons/version1")
public ResponseEntity<List<Person>>  getPersonListV1() {
    List<Person> personList= personService.getPersonList();
    return new ResponseEntity<>(personList,HttpStatus.OK);
}
// second uri gives more enhanced details
@Operation(summary = "uri versioning second uri gives more
enhanced details of person")
@GetMapping(path = "/persons/version2")
public ResponseEntity<List<Person1>>  getPersonListV2()
{
    List<Person1> personList1 =personService.getPersonList1();
    return new ResponseEntity<>(personList1,HttpStatus.OK);
}
```

GET http://localhost:8080/persons/version1 Send Save

Pretty Raw Preview Visualize JSON


```
[
    {
        "id": 1,
        "name": "thor",
        "age": 23,
        "hobby": "chugging"
    },
    {
        "id": 2,
        "name": "hella",
        "age": 20,
        "hobby": "destroying"
    },
    {
        "id": 3,
        "name": "loki",
        "age": 22,
        "hobby": "mischief"
    },
```


GET http://localhost:8080/persons/produces Send Save

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

Headers 7 hidden

KEY	VALUE	DESCRIPTION
API-VERSION	2	
Accept	application/com.example.app-v2+json	
Key	Value	Description

Body Cookies Headers (5) Test Results Status: 200 OK Time: 8ms Size: 575 B Save Response

Pretty Raw Preview Visualize JSON


```
[
    {
        "id": 1,
        "firstName": "thor",
        "lastName": "odin son",
        "password": "pass1",
        "age": 23,
        "hobby": "chugging"
    },
    {
```

- **Custom Header Versioning**

```
// header parameter versioning

// first uri gives basic detail version of the persons
@Operation(summary = "header parameter versioning")
@GetMapping(value = "/persons/header", headers = "api-version=1")
public ResponseEntity<List<Person>>  getPersonListV5() {
    List<Person> personList= personService.getPersonList();
    return new ResponseEntity<>(personList,HttpStatus.OK);
}
// second uri gives more enhanced details
@Operation(summary = "header parameter versioning")
@GetMapping(value = "/persons/header", headers = "api-version=1")
public ResponseEntity<List<Person1>>  getPersonListV6() {
    List<Person1> personList1= personService.getPersonList1();
    return new ResponseEntity<>(personList1,HttpStatus.OK);
}
```

uri to hit

http://localhost:8080/persons/header

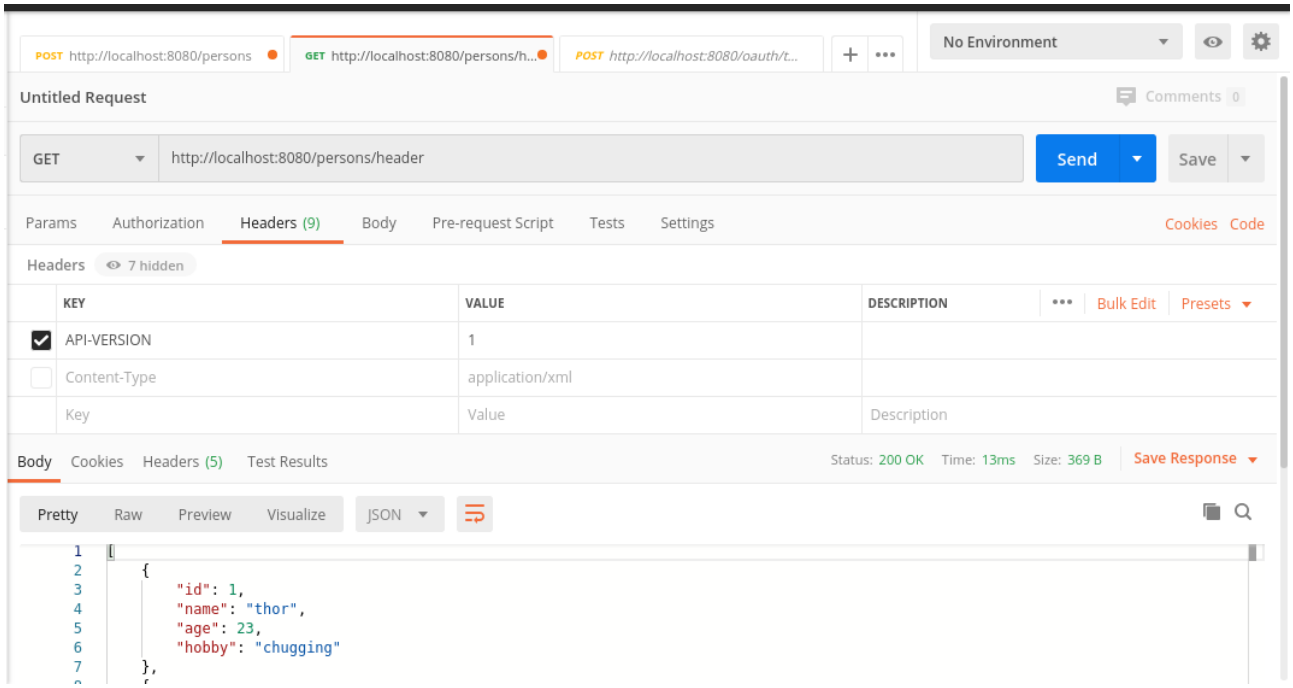
also pass in header

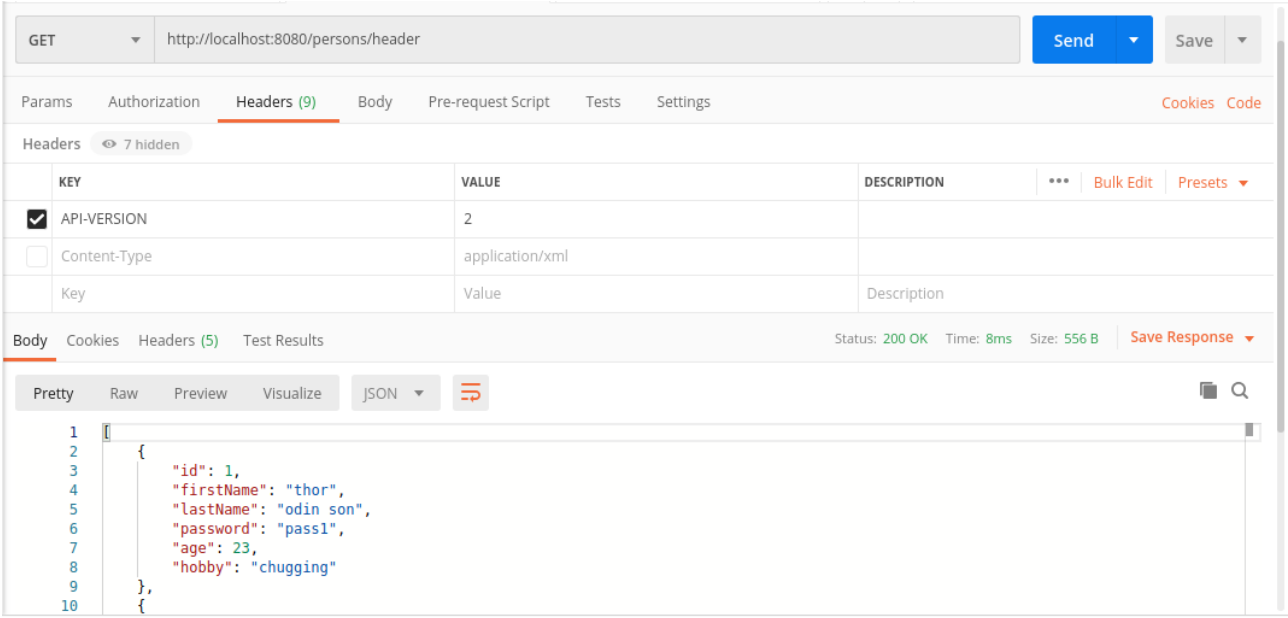
API-VERSION: 1

http://localhost:8080/persons/header

also pass in header

API-VERSION: 2





*HATEOAS

Q10.Configure hateoas with your springboot application. Create an api which returns User Details along with url to show all topics.

Q10 inside PersonController.java

```
@GetMapping(path = "/persons/{id}")    // getting a specific
person
public EntityModel<Person> getPerson(@PathVariable("id") Integer
id) {
    Person p = personService.getPerson(id);
    if (p == null) {
        throw new UserNotFoundException("user not found with id->" +
id);
    }
    //*HATEOAS
    //Configuring hateoas with your springboot application.
    //it returns User Details along with link to show all other
users.
    EntityModel<Person> resource = new EntityModel<>(p);
    WebMvcLinkBuilder linkTo =
linkTo(methodOn(this.getClass()).getPersonList());
    resource.add(linkTo.withRel("find-all-persons"));
    return resource;
} //end of getPerson method
```

GET http://localhost:8080/persons/1 Send Save

Pretty Raw Preview Visualize JSON


```
{
    "id": 1,
    "name": "thor",
    "age": 23,
    "hobby": "chugging",
    "links": [
        {
            "rel": "find-all-persons",
            "href": "http://localhost:8080/persons"
        }
    ]
}
```