Session : RestFul Web Service Part 2

1. **Add support for Internationalization in your application allowing messages to be shown in English, German and Swedish, keeping English as default.**
2. **Create a GET request which takes "username" as param and shows a localized message "Hello Username". (Use parameters in message properties)**

CODE

RestFullWebServicesAssignment2Application.java

```
@SpringBootApplication
public class RestFullWebServicesAssignment2Application {

  public static void main(String[] args) {
    SpringApplication.run(RestFullWebServicesAssignment2Application.class, args);
  }

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

  @Bean
  public ResourceBundleMessageSource bundleMessageSource(){
    ResourceBundleMessageSource messageSource = new ResourceBundleMessageSource();
    messageSource.setBasename("messages");
    return messageSource;
  }
}
```

UserController.java

```
@RestController
public class UserController {

   @Autowired
   private MessageSource messageSource;

   @Autowired
   private UserDao obj;
```

```
  @GetMapping(path="/Users/{id}")
  public User findOne(@PathVariable Integer id)
  {
     User user = obj.findOne(id);
     return user;
  }

  @GetMapping(path = "/Users-internationalized/{id}")
  public String UserInternationalized(@PathVariable Integer id,@RequestHeader(name =
"Accept-Language",required = false) Locale locale){
     User user = obj.findOne(id);
     String username = user.getUsername();
     return messageSource.getMessage("hello.message",null,locale) +" " +username;
  }
```

UserDao.java

```
@Component
public class UserDao {

  List<User> ls = new ArrayList<User>();

  //Get Single User
  public User findOne(Integer id)
  {
     for (User user:ls)
     {
        if (user.getId()==id)
           return user;
     }
     return null;
  }
}
```

messages.properties

```
hello.message=Hello
```

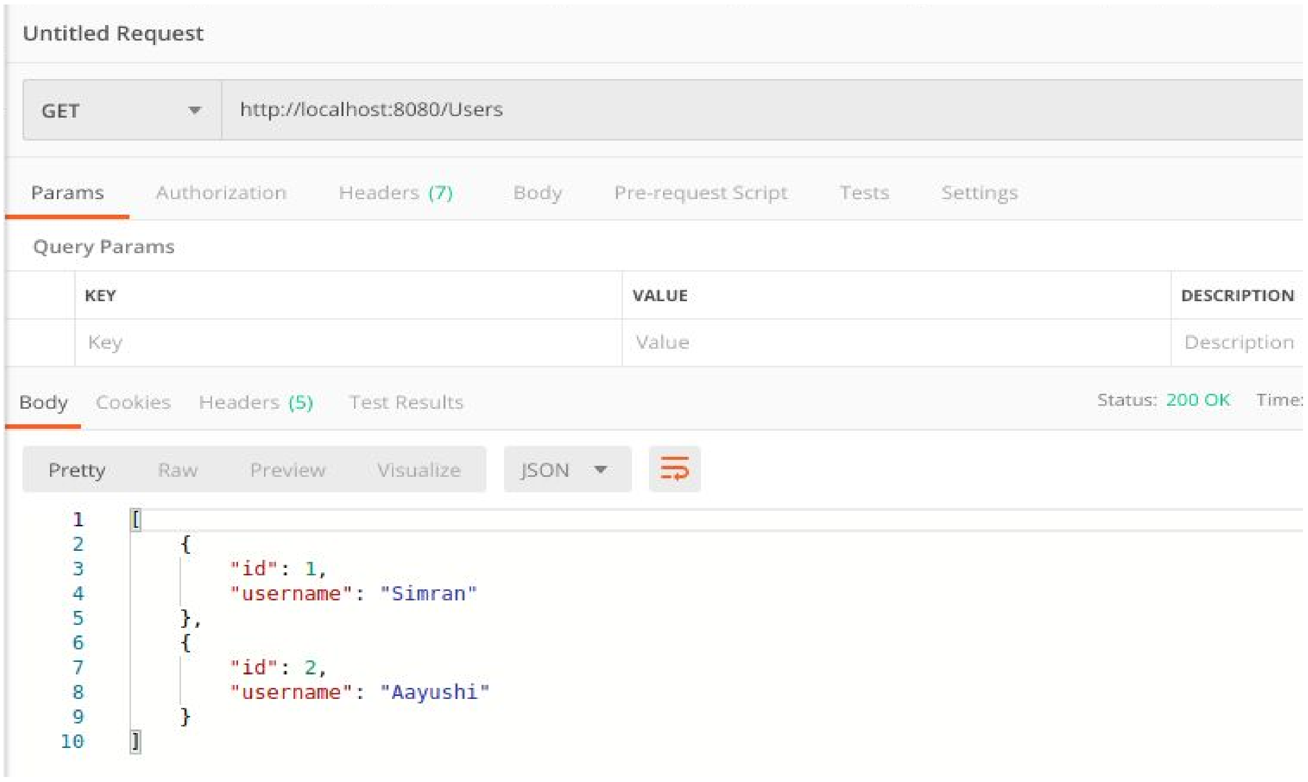
messages_de.properties

```
hello.message=Hello
```

messages_sv.properties

```
hello.message=Hello
```

OUTPUT



Untitled Request

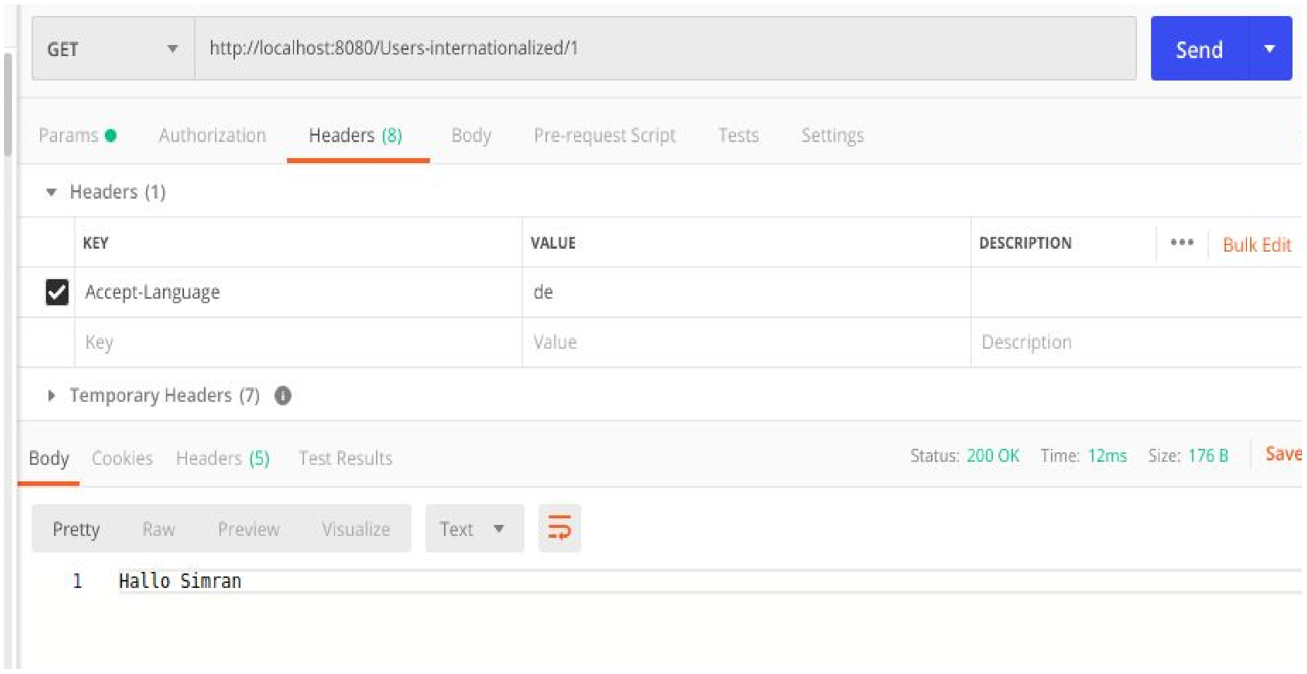
GET http://localhost:8080/Users

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Status: 200 OK

```
[
    {
        "id": 1,
        "username": "Simran"
    },
    {
        "id": 2,
        "username": "Aayushi"
    }
]
```



GET http://localhost:8080/Users-internationalized/1

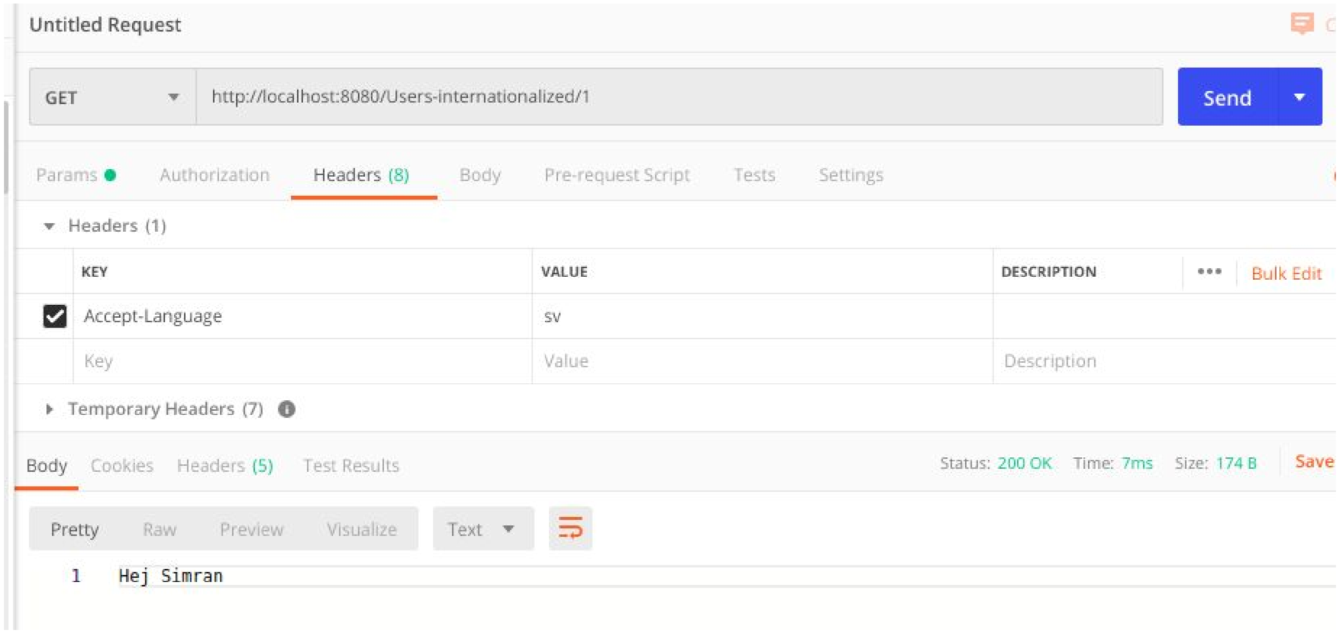
Headers (1)

	KEY	VALUE	DESCRIPTION
✓	Accept-Language	de	
	Key	Value	Description

Temporary Headers (7)

Status: 200 OK Time: 12ms Size: 176 B

```
Hallo Simran
```

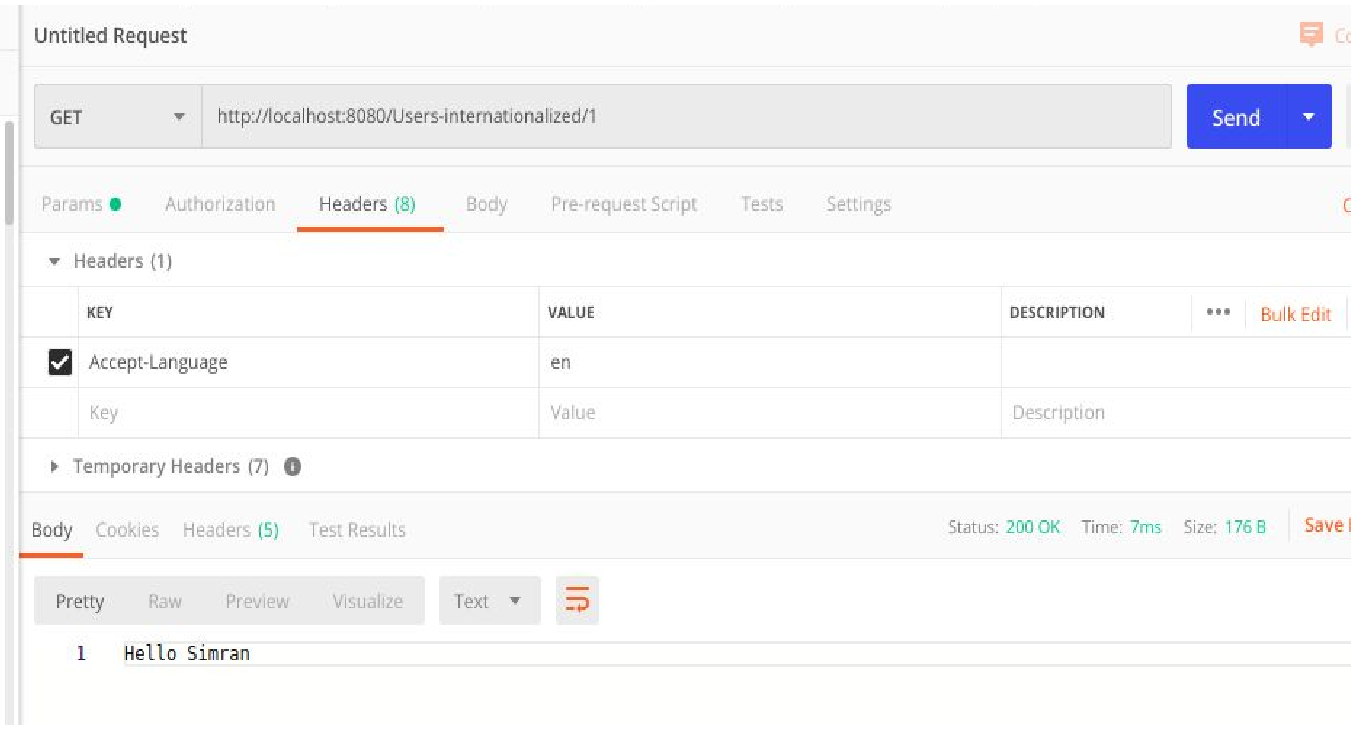


http://localhost:8080/Users-internationalized/1

KEY	VALUE	DESCRIPTION
✓ Accept-Language	sv	
Key	Value	Description

Status: 200 OK Time: 7ms Size: 174 B

```
Hej Simran
```



http://localhost:8080/Users-internationalized/1

KEY	VALUE	DESCRIPTION
✓ Accept-Language	en	
Key	Value	Description

Status: 200 OK Time: 7ms Size: 176 B

```
Hello Simran
```

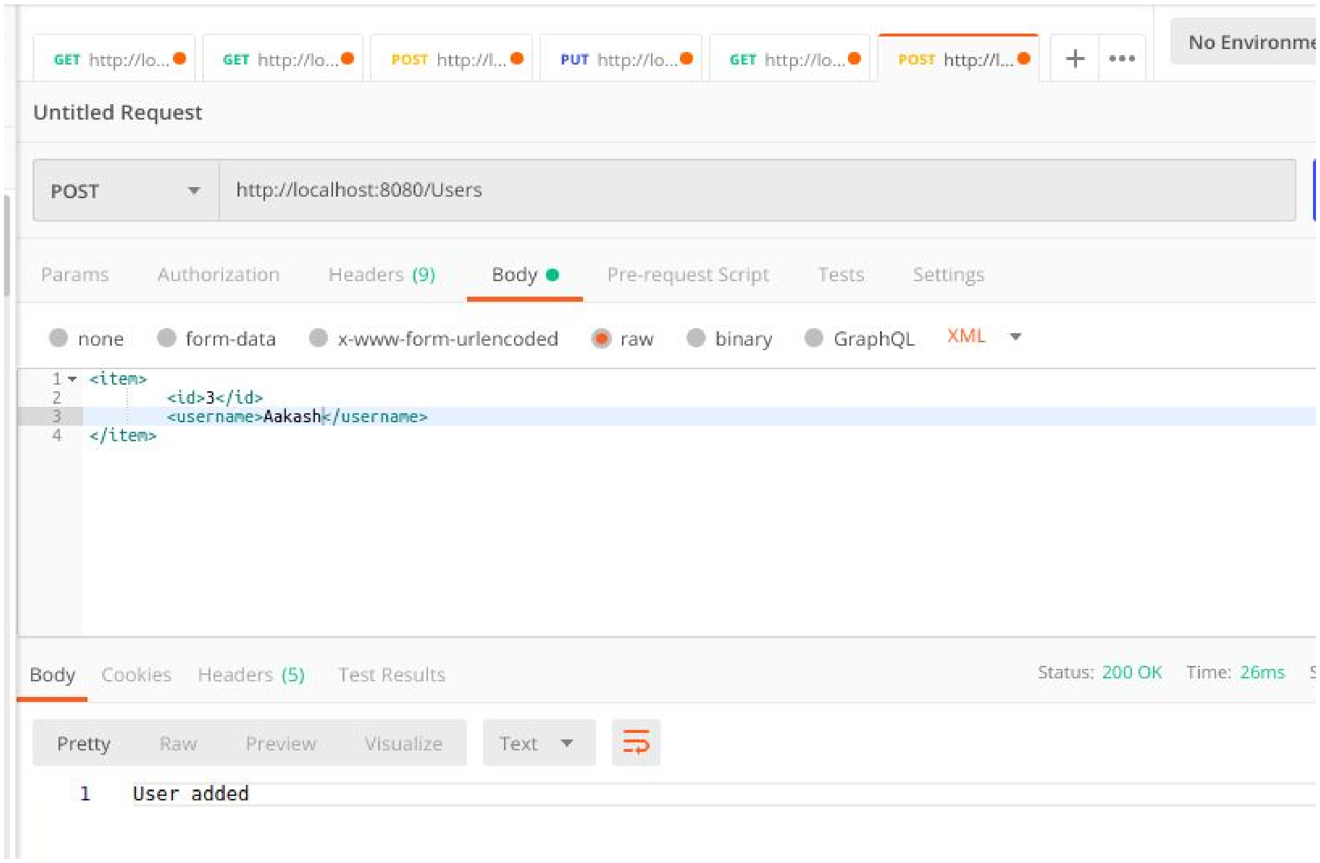
3. Create POST Method to create user details which can accept XML for user creation.

<u>CODE</u>

<u>Build.gradle</u>

```
dependencies {
  compile group: 'com.fasterxml.jackson.dataformat', name: 'jackson-dataformat-xml', version:
'2.10.2'
}
```

<u>OUTPUT</u>



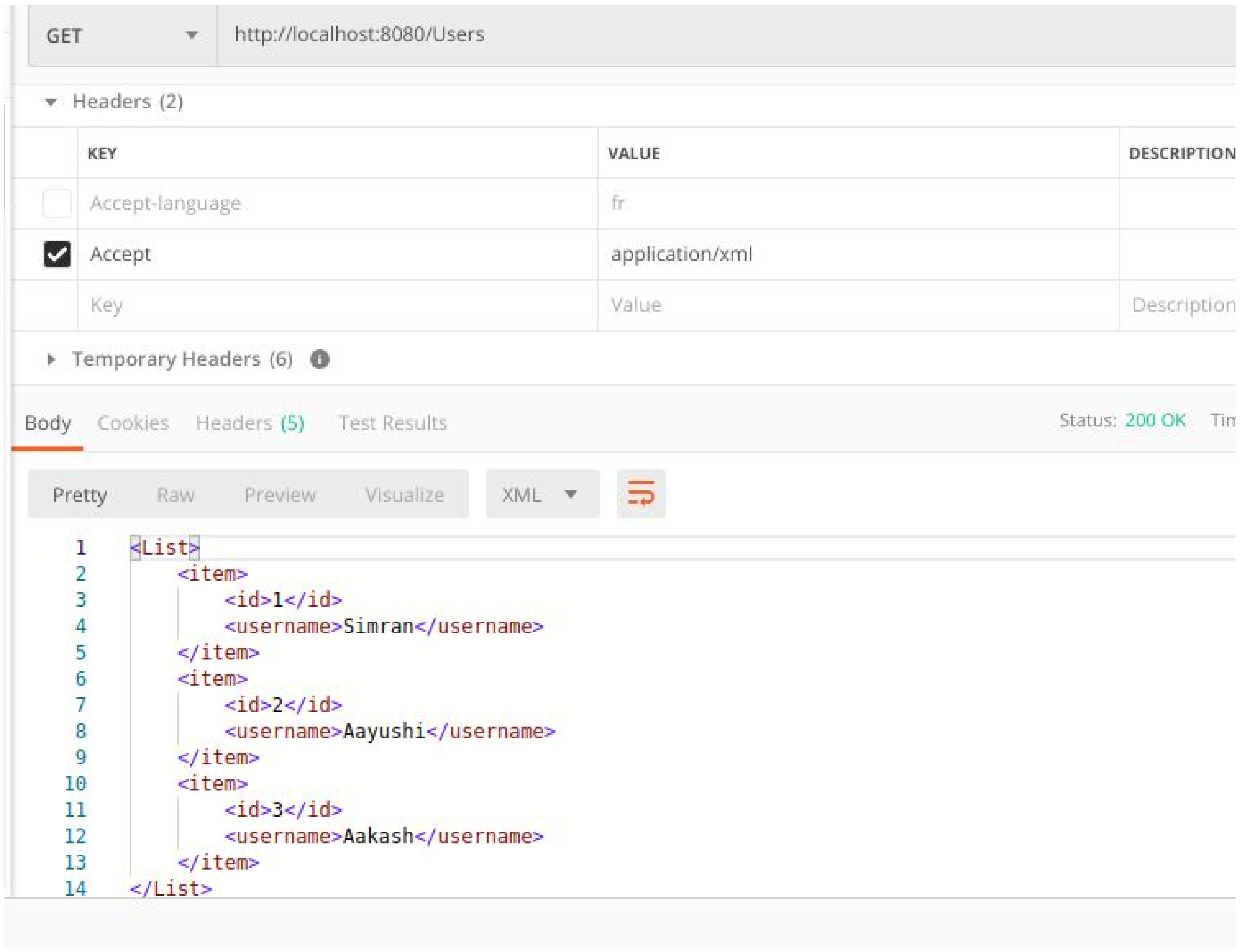
POST http://localhost:8080/Users

Body: raw XML

```
<item>
        <id>3</id>
        <username>Aakash</username>
</item>
```

Status: 200 OK Time: 26ms

```
User added
```


GET http://localhost:8080/Users

Headers (2)

KEY	VALUE	DESCRIPTION
Accept-language	fr	
✓ Accept	application/xml	
Key	Value	Description

Temporary Headers (6)

Body | Cookies | Headers (5) | Test Results — Status: 200 OK

Pretty | Raw | Preview | Visualize | XML

```
<List>
    <item>
        <id>1</id>
        <username>Simran</username>
    </item>
    <item>
        <id>2</id>
        <username>Aayushi</username>
    </item>
    <item>
        <id>3</id>
        <username>Aakash</username>
    </item>
</List>
```

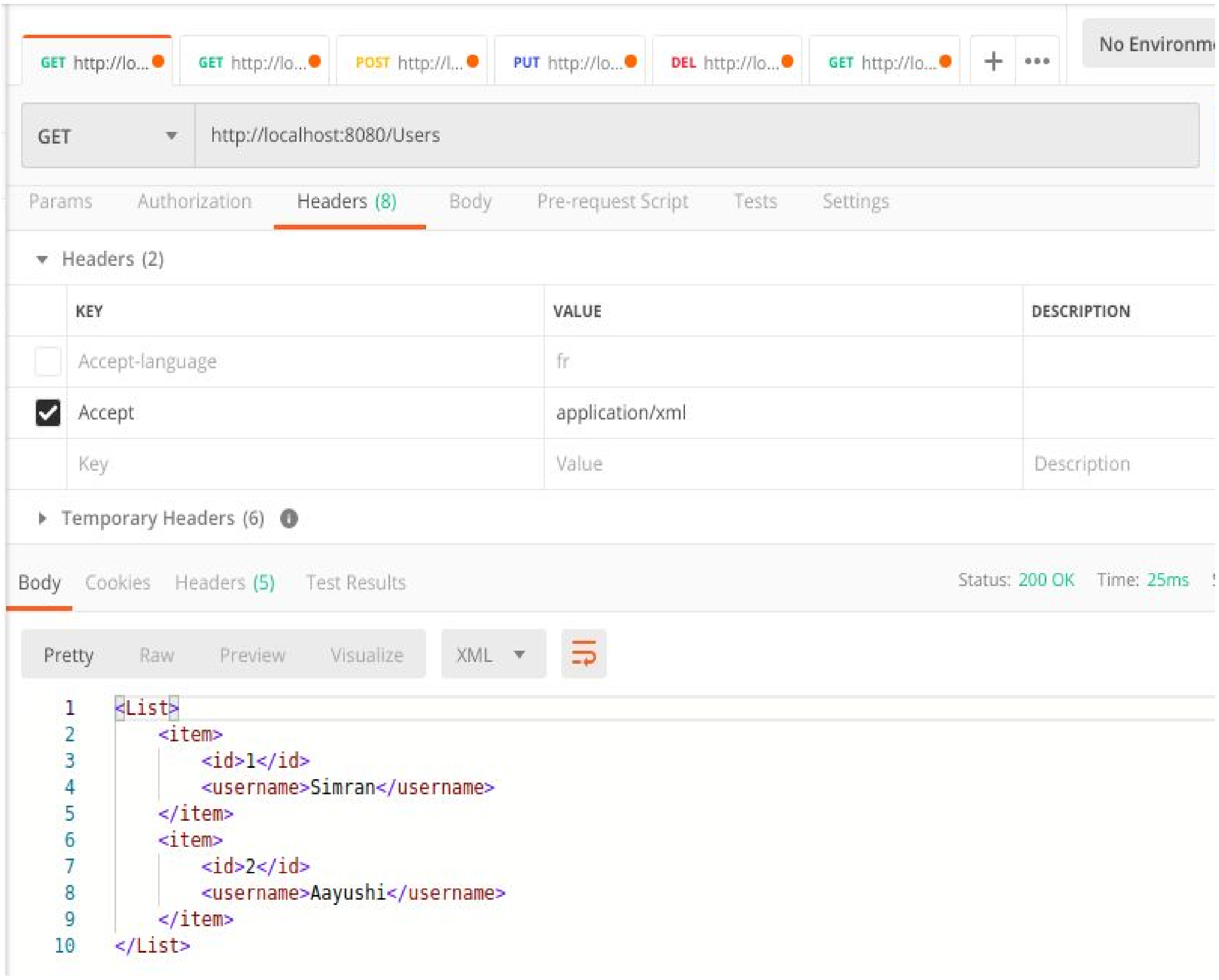
4. Create GET Method to fetch the list of users in XML format.

CODE

Build.gradle

```
dependencies {
  compile group: 'com.fasterxml.jackson.dataformat', name: 'jackson-dataformat-xml', version:
'2.10.2'
}
```

OUTPUT



	KEY	VALUE	DESCRIPTION
	Accept-language	fr	
✓	Accept	application/xml	
	Key	Value	Description

```
<List>
    <item>
        <id>1</id>
        <username>Simran</username>
    </item>
    <item>
        <id>2</id>
        <username>Aayushi</username>
    </item>
</List>
```

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

Get details of User using GET request.

Save details of the user using POST request.

Delete a user using DELETE request.

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.

CODE

Build.gradle

```
dependencies {
  compile group: 'io.springfox', name: 'springfox-swagger2', version: '2.9.2'
  compile group: 'io.springfox', name: 'springfox-swagger-ui', version: '2.9.2'
}
```

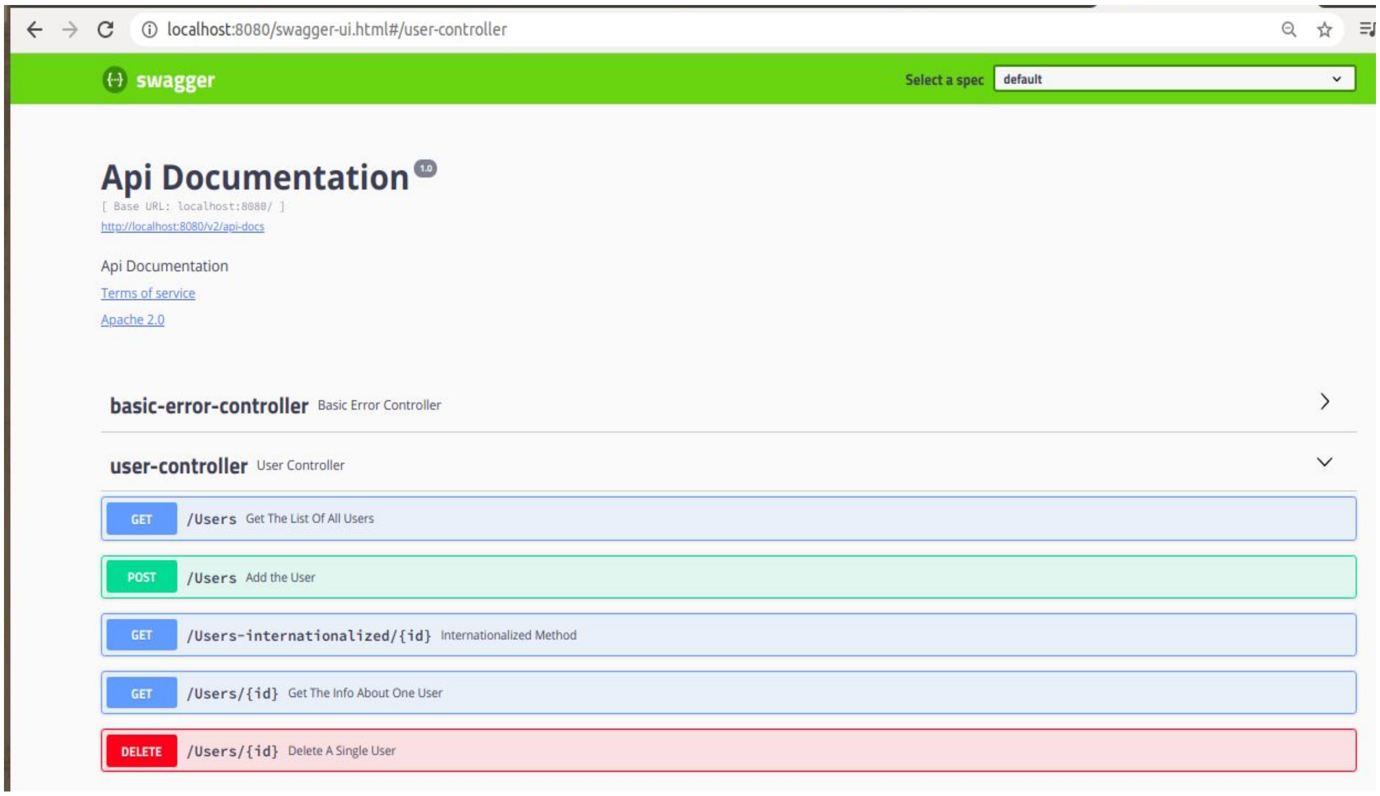
SwaggerConfig.java

```
@Configuration
@EnableSwagger2
public class SwaggerConfig {

   @Bean
   public Docket api(){
       return new Docket(DocumentationType.SWAGGER_2);
   }
}
```

UserController.java

```
@ApiModel(description = "User Controller Class")
@RestController
public class UserController {

   @Autowired
   private MessageSource messageSource;

   @Autowired
   private UserDao obj;

   @ApiOperation(value = "Get The List Of All Users")
   @GetMapping(path="/Users")
   public List<User> getAllUser()
   {
       return obj.getUserList();
   }

   @ApiOperation(value = "Get The Info About One User")
   @GetMapping(path="/Users/{id}")
   public User findOne(@PathVariable Integer id)
   {
       User user = obj.findOne(id);
       return user;
   }
   @ApiOperation(value = "Add the User")
   @PostMapping(path="/Users")
   public String addUser(@RequestBody User user)
   {
       String message = obj.addUser(user);
       return message;
   }

   @ApiOperation(value = "Internationalized Method")
   @GetMapping(path = "/Users-internationalized/{id}")
   public String UserInternationalized(@PathVariable Integer id,@RequestHeader(name =
"Accept-Language",required = false) Locale locale){
       User user = obj.findOne(id);
       String username = user.getUsername();
       return messageSource.getMessage("hello.message",null,locale) +" " +username;
   }

   @ApiOperation(value = "Delete A Single User")
   @DeleteMapping(path="/Users/{id}")
   public String deleteUser(@PathVariable Integer id)
   {
       String message = obj.deleteUser(id);
       return message;
   }
}
```

<u>OUTPUT</u>



localhost:8080/swagger-ui.html#/user-controller

Api Documentation 1.0

[Base URL: localhost:8080/]

http://localhost:8080/v2/api-docs

Api Documentation

Terms of service

Apache 2.0

basic-error-controller Basic Error Controller

user-controller User Controller

GET /Users Get The List Of All Users

POST /Users Add the User

GET /Users-internationalized/{id} Internationalized Method

GET /Users/{id} Get The Info About One User

DELETE /Users/{id} Delete A Single User

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

<u>CODE</u>

<u>UserController.java</u>

```
@RestController
public class UserController {

  @Autowired
  private MessageSource messageSource;

  @Autowired
  private UserDao obj;

  @ApiOperation(value = "Add the User")
  @PostMapping(path="/Users")
  public User addUser(@RequestBody User user)
  {
        User user1 = obj.addUser(user);
        return user1;
    }
}
```

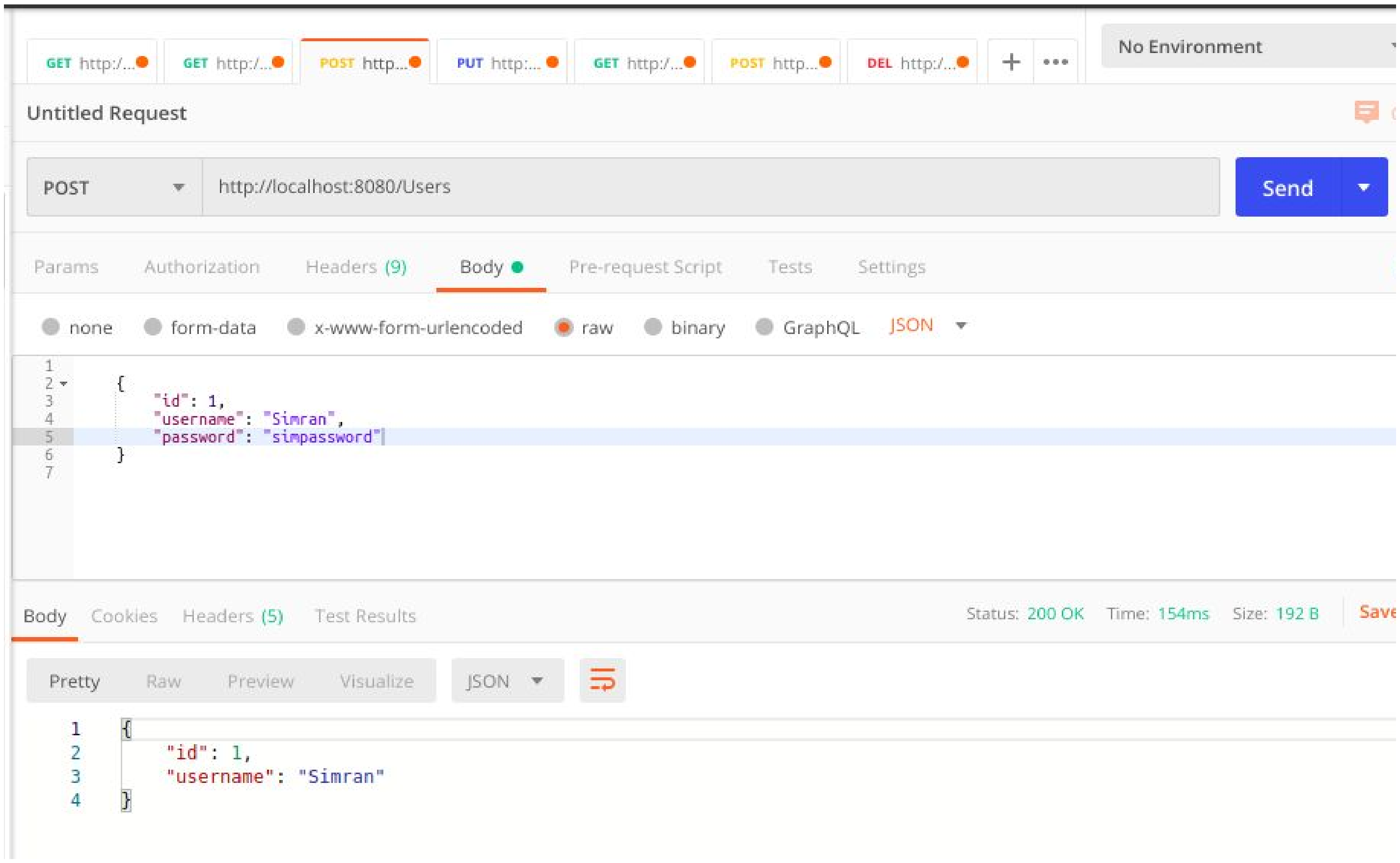
UserDao.java

```
//Post a Single User
public User addUser(User user)
{
    ls.add(user);
    return user;
}
```

User.java

```
@JsonIgnoreProperties(value = {"password"})
@ApiModel(description = "User Service Class")
public class User {
    private Integer id;
    private String username;
    private String password;

    public User(Integer id, String username, String password) {
        this.id = id;
        this.username = username;
        this.password = password;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }
    public String getUsername() {
        return username;
    }

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

    @Override
    public String toString() {
        return "User{" +
                "id=" + id +
                ", username='" + username + '\'' +
                ", password='" + password + '\'' +
                '}';
    }
}
```

OUTPUT



Untitled Request

POST http://localhost:8080/Users

```
{
    "id": 1,
    "username": "Simran",
    "password": "simpassword"
}
```

Status: 200 OK Time: 154ms Size: 192 B

```
{
    "id": 1,
    "username": "Simran"
}
```

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

CODE

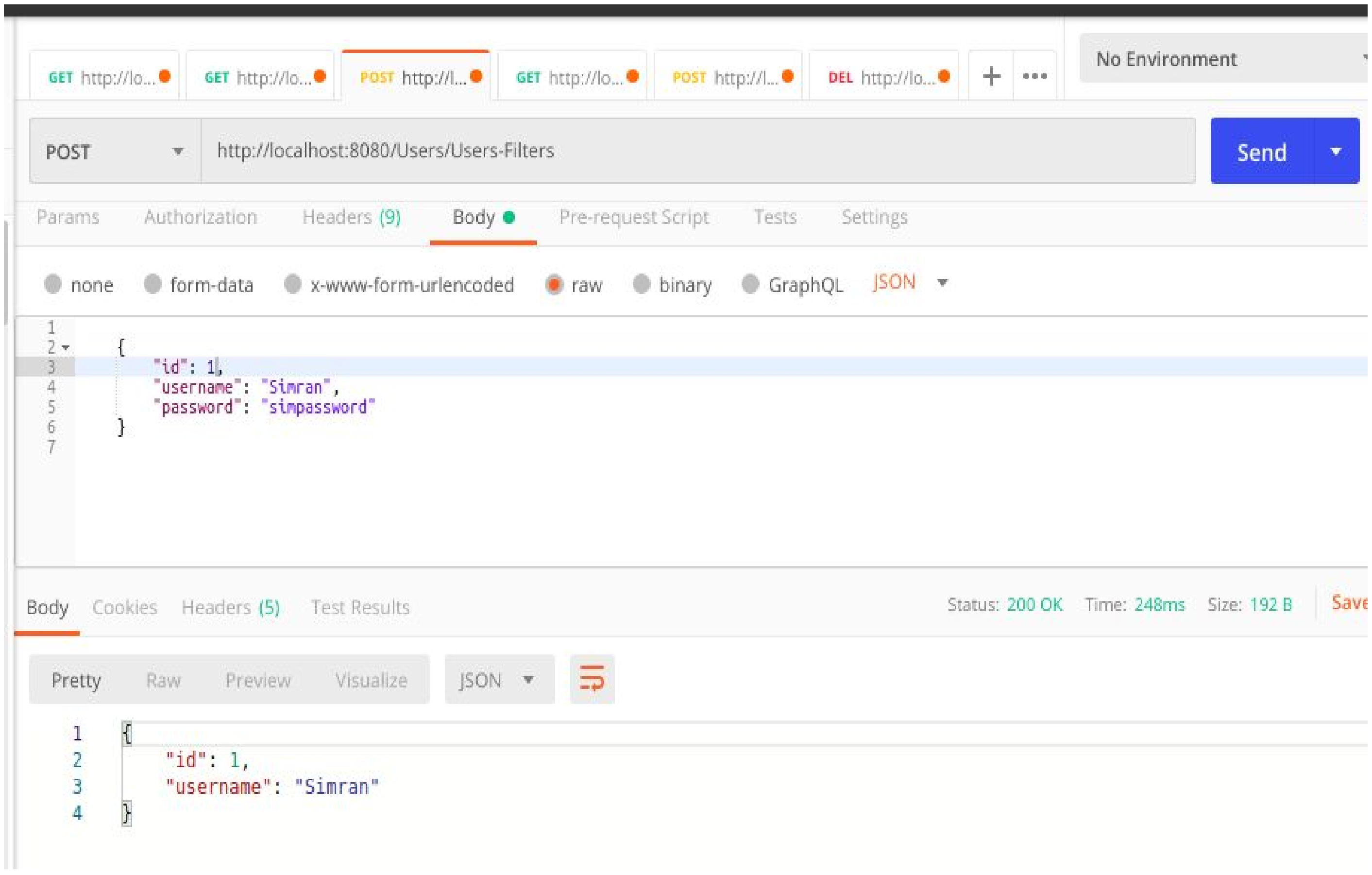
UserController.java

```
@ApiOperation(value = "Add the User using Dynamic Filter")
@PostMapping(path="/Users/Users-Filters")
public MappingJacksonValue addUserDynamicFilter(@RequestBody User user)
{
   User user1 = obj.addUser(user);
   SimpleBeanPropertyFilter filter =
SimpleBeanPropertyFilter.filterOutAllExcept("id","username");

   FilterProvider filters = new SimpleFilterProvider().addFilter("PostDynamicFilter",filter);

   MappingJacksonValue mapping = new MappingJacksonValue(user1);

   mapping.setFilters(filters);
   return mapping;
}
```

User.java

```
@JsonFilter("PostDynamicFilter")
@ApiModel(description = "User Service Class")
public class User {
   private Integer id;
   private String username;
   private String password;

   public User(Integer id, String username, String password) {
       this.id = id;
       this.username = username;
       this.password = password;
   }
}
```

OUTPUT



POST http://localhost:8080/Users/Users-Filters

```

    {
        "id": 1,
        "username": "Simran",
        "password": "simpassword"
    }

```

Status: 200 OK Time: 248ms Size: 192 B

```
{
    "id": 1,
    "username": "Simran"
}
```

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:

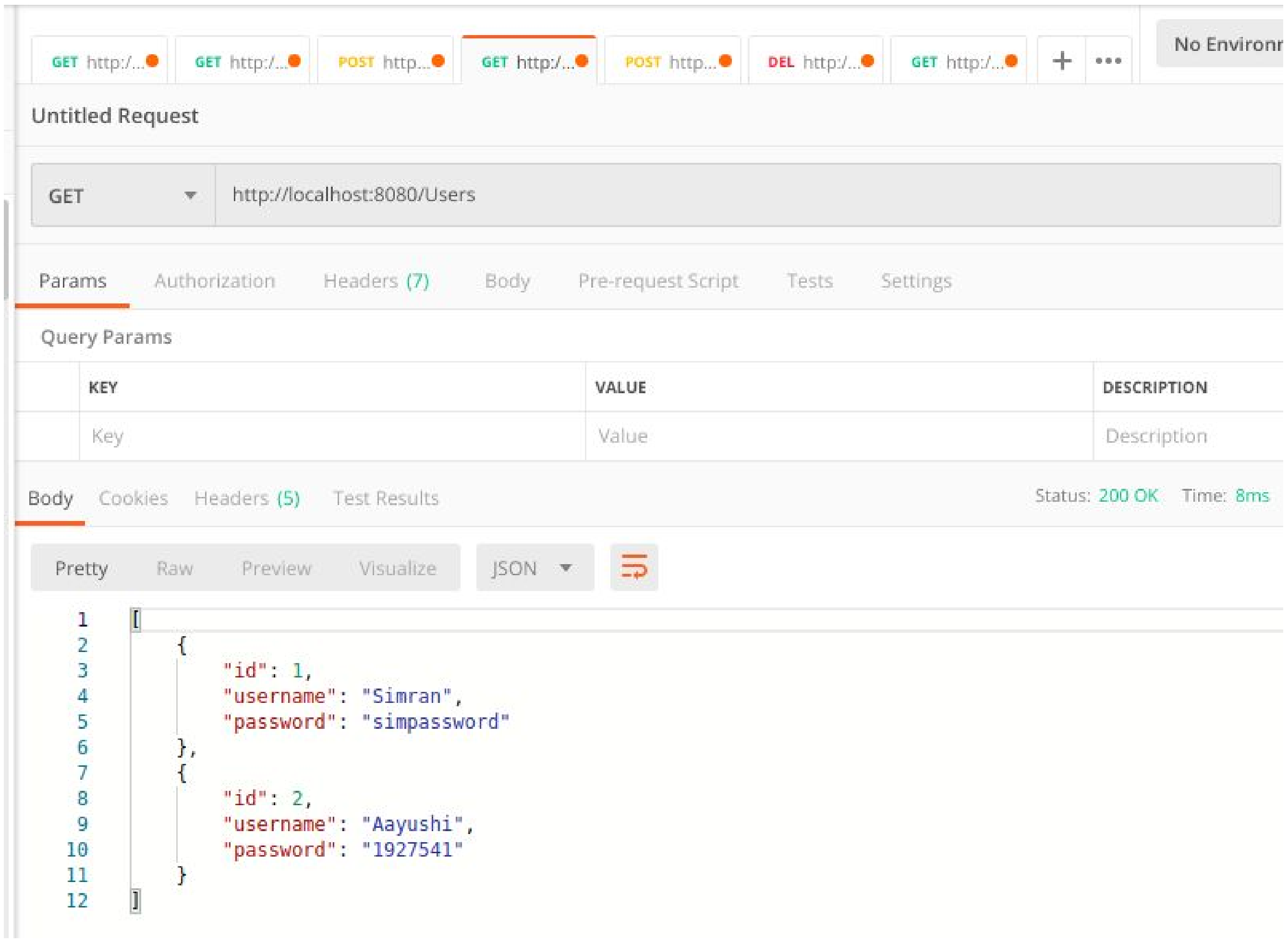
- **MimeType Versioning**
- **Request Parameter versioning**
- **URI versioning**
- **Custom Header Versioning**

1. <u>URI versioning</u>

```
@GetMapping("Users/v1/{id}")
public String personV1(@PathVariable Integer id){
  User user = obj.findOne(id);
  String username = user.getUsername();
  return "The Username is: "+username;
}
```

```
@GetMapping("Users/v2/{id}")
public String personV2(@PathVariable Integer id){
   User user = obj.findOne(id);
   String username = user.getUsername();
   String id1 = user.getId().toString();
   return "The User id is: "+ id1 + " The Username is: "+username;
}
```

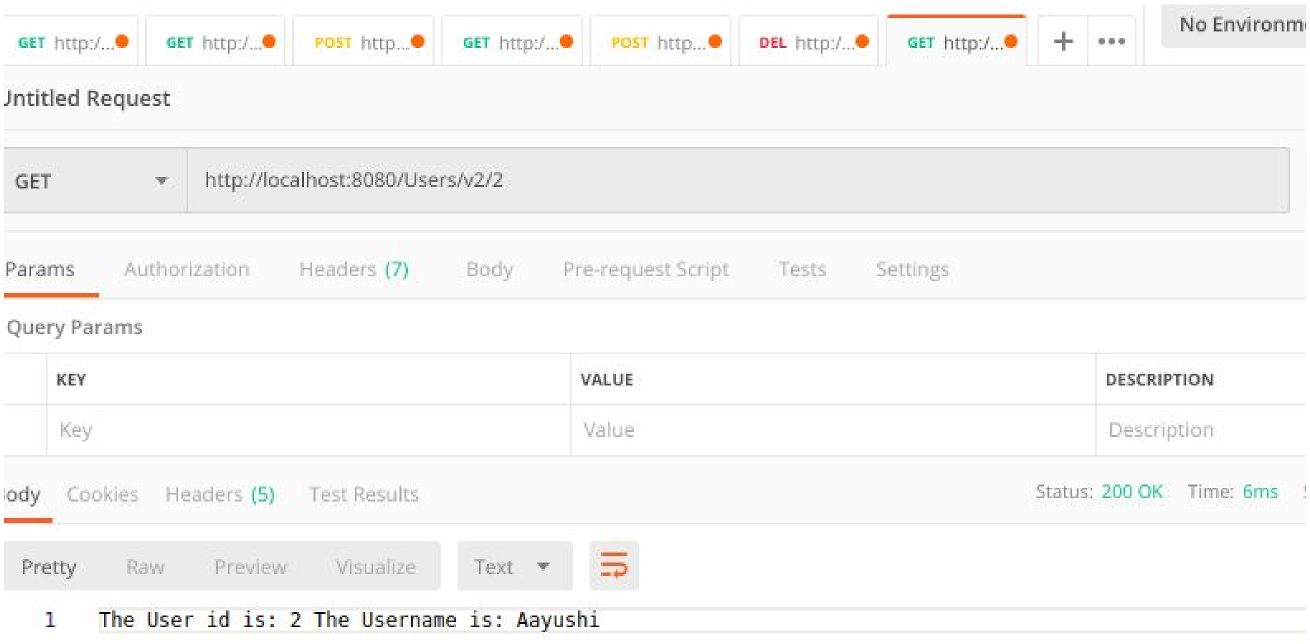
OUTPUT



http://localhost:8080/Users

Status: 200 OK Time: 8ms

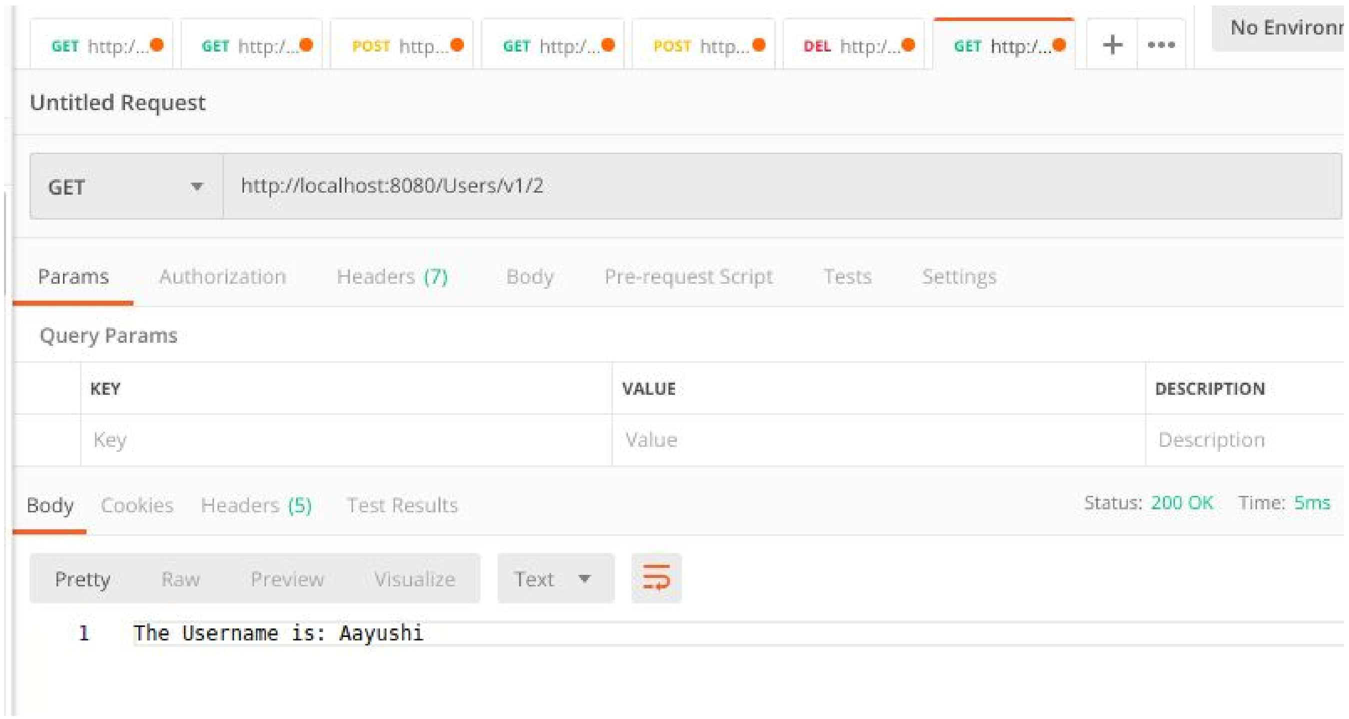
```
[
    {
        "id": 1,
        "username": "Simran",
        "password": "simpassword"
    },
    {
        "id": 2,
        "username": "Aayushi",
        "password": "1927541"
    }
]
```



GET http://localhost:8080/Users/v2/2

Status: 200 OK Time: 6ms

```
The User id is: 2 The Username is: Aayushi
```



GET http://localhost:8080/Users/v1/2

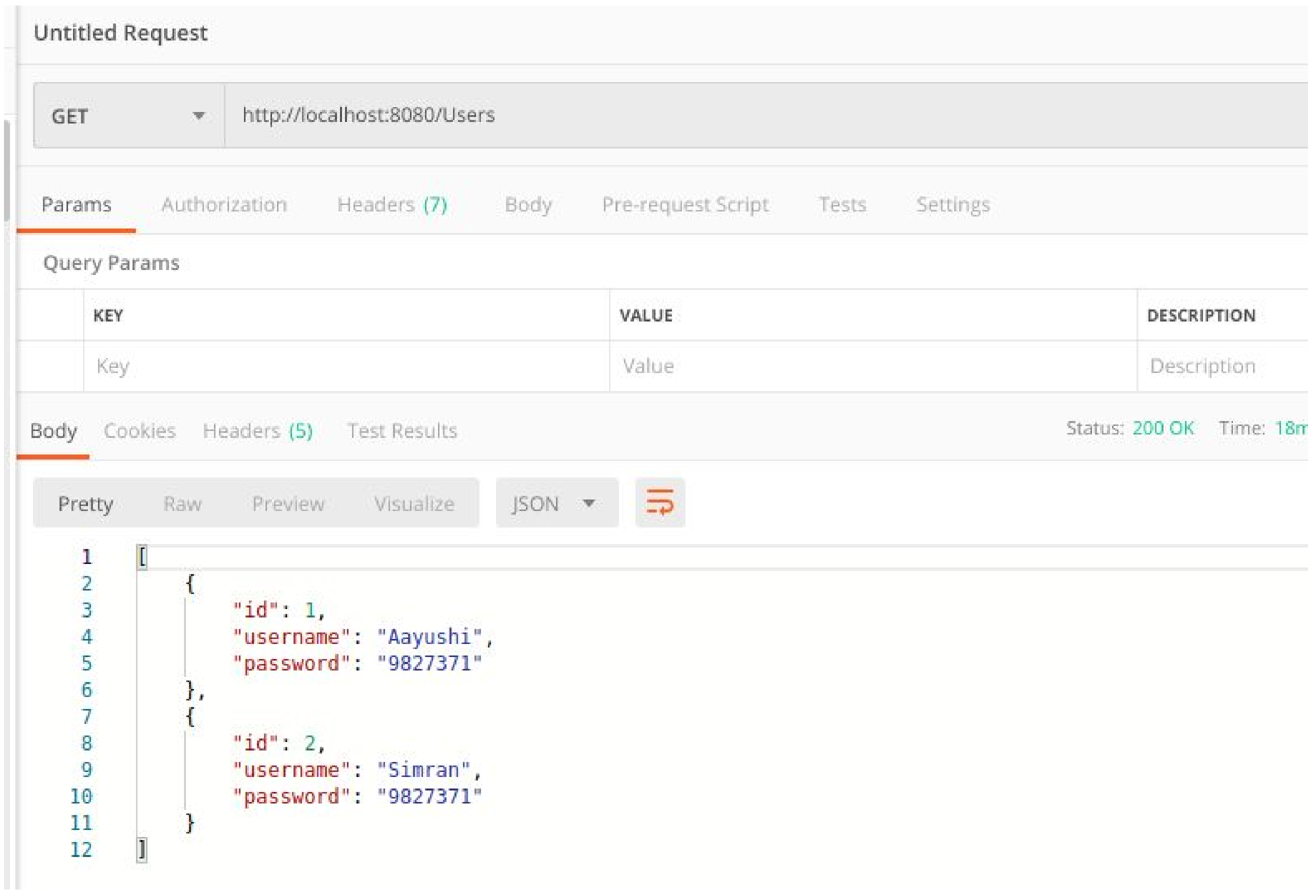
Status: 200 OK Time: 5ms

```
The Username is: Aayushi
```

2. Request Parameter versioning

```
@GetMapping(value = "Users/{id}/param", params = "version=1")
public String userparamV1(@PathVariable Integer id){
  User user = obj.findOne(id);
  String username = user.getUsername();
  return "The Username is: "+username;
}

@GetMapping(value = "Users/{id}/param", params = "version=2")
public String userparamV2(@PathVariable Integer id){
  User user = obj.findOne(id);
  String username = user.getUsername();
  String id1 = user.getId().toString();
  return "The User id is: "+ id1 + " The Username is: "+username;
}
```

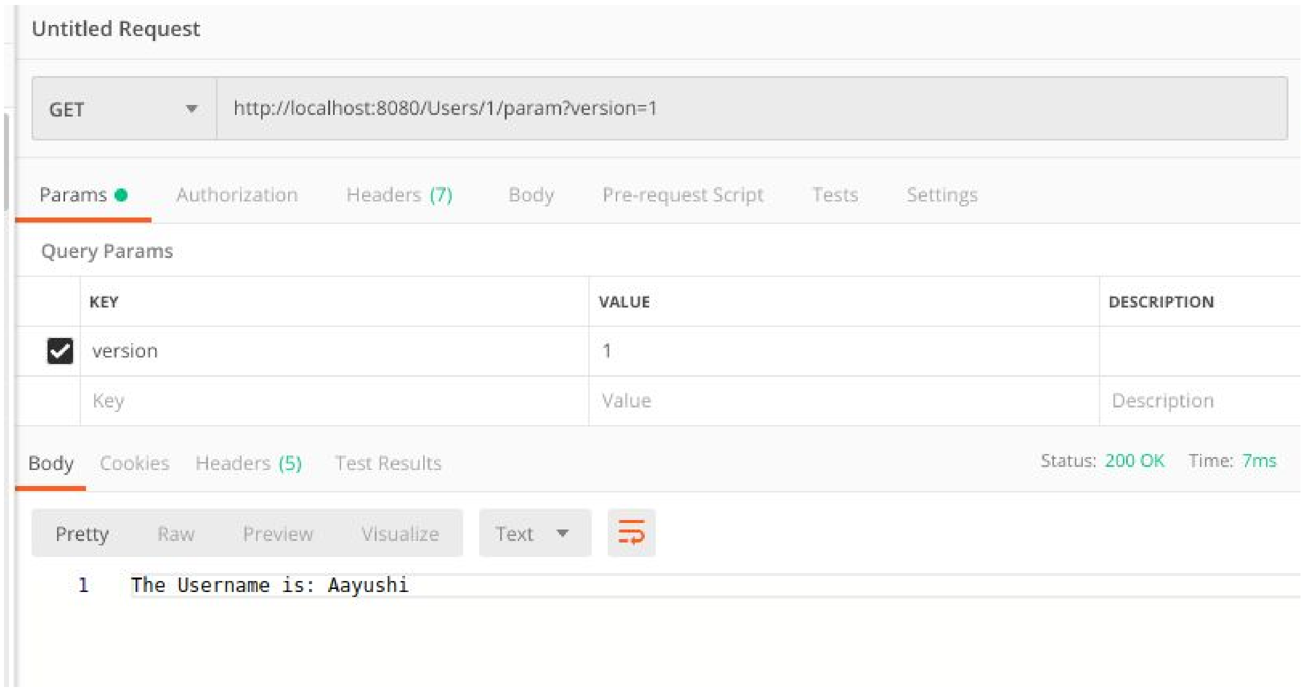
OUTPUT



http://localhost:8080/Users

Status: 200 OK Time: 18m

```
[
    {
        "id": 1,
        "username": "Aayushi",
        "password": "9827371"
    },
    {
        "id": 2,
        "username": "Simran",
        "password": "9827371"
    }
]
```



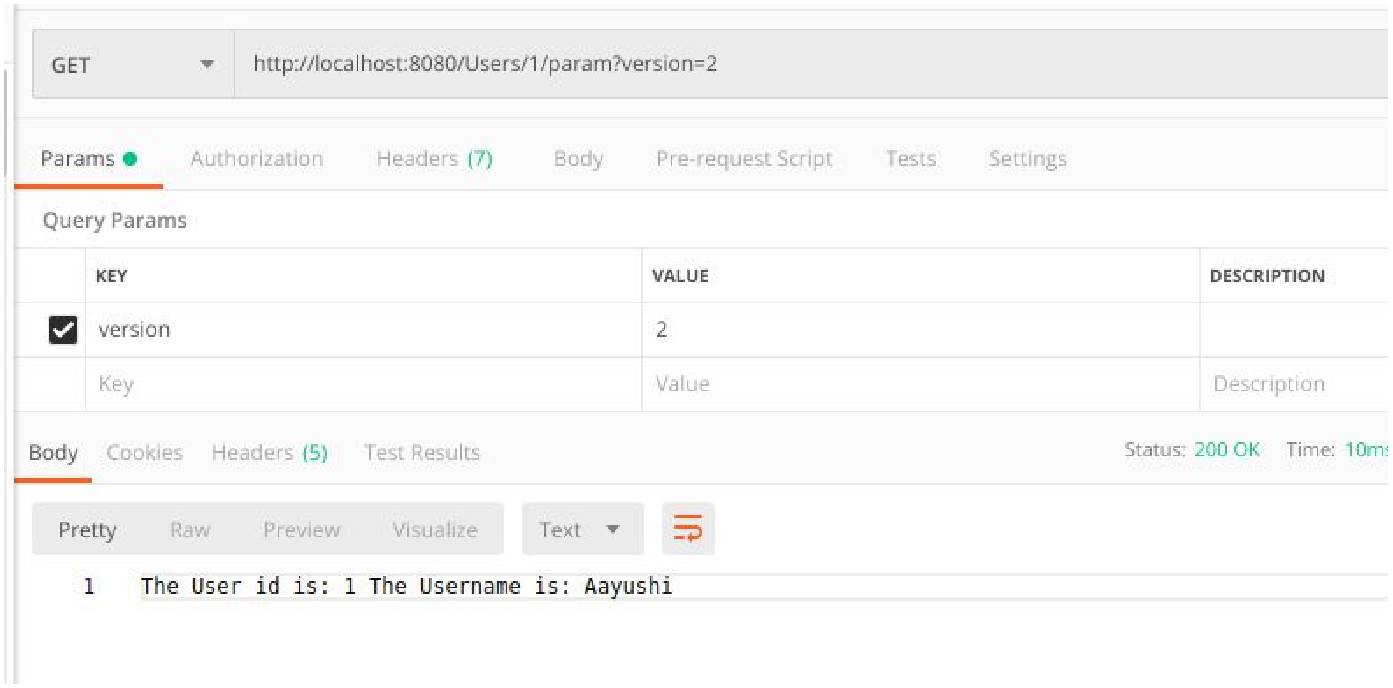
http://localhost:8080/Users/1/param?version=1

Query Params

	KEY	VALUE	DESCRIPTION
✓	version	1	
	Key	Value	Description

Status: 200 OK Time: 7ms

```
The Username is: Aayushi
```



http://localhost:8080/Users/1/param?version=2

Query Params

	KEY	VALUE	DESCRIPTION
✓	version	2	
	Key	Value	Description

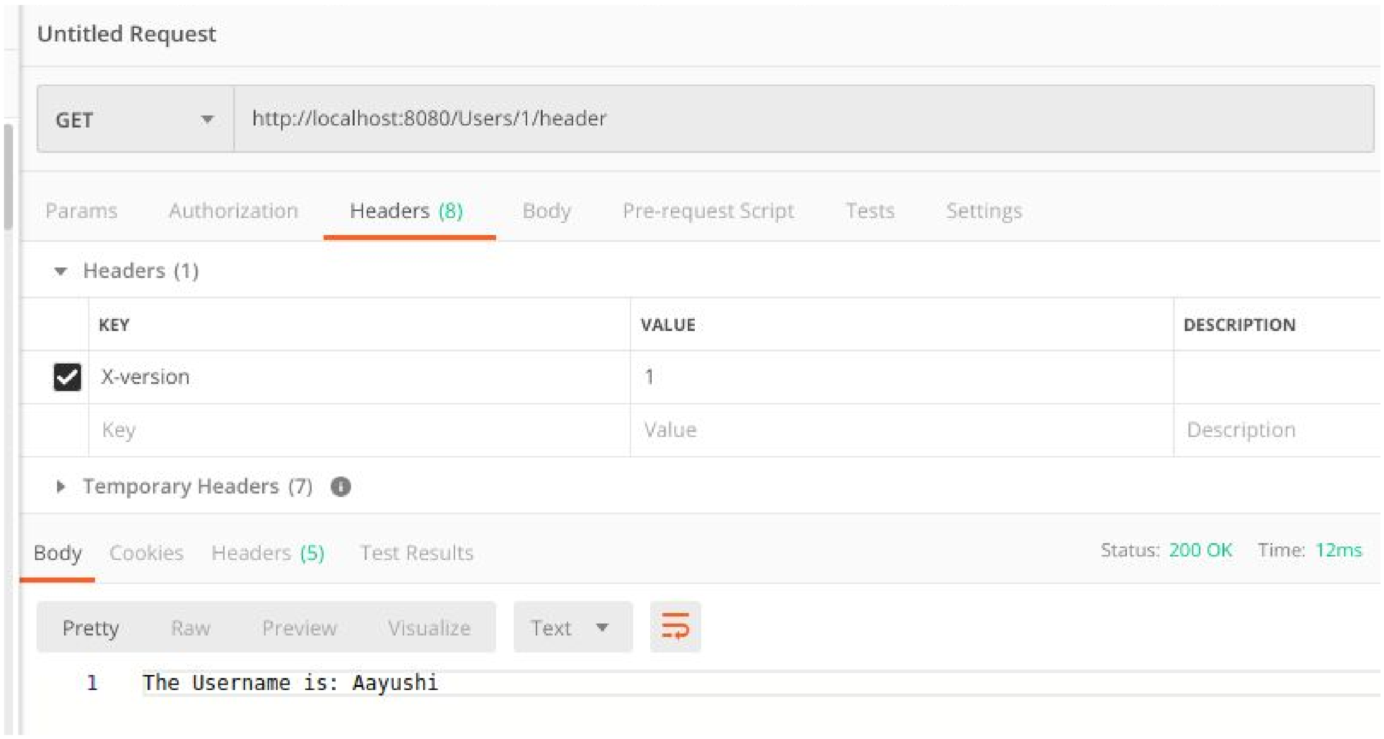
Status: 200 OK Time: 10ms

```
The User id is: 1 The Username is: Aayushi
```

3. Custom Header Versioning

```
//Custom Header Versioning
@GetMapping(value = "Users/{id}/header", headers = "X-version=1")
public String userheaderV1(@PathVariable Integer id){
  User user = obj.findOne(id);
  String username = user.getUsername();
  return "The Username is: "+username;
}

@GetMapping(value = "Users/{id}/header", headers = "X-version=2")
public String userheaderV2(@PathVariable Integer id){
  User user = obj.findOne(id);
  String username = user.getUsername();
  String id1 = user.getId().toString();
  return "The User id is: "+ id1 + " The Username is: "+username;
}
```

OUTPUT

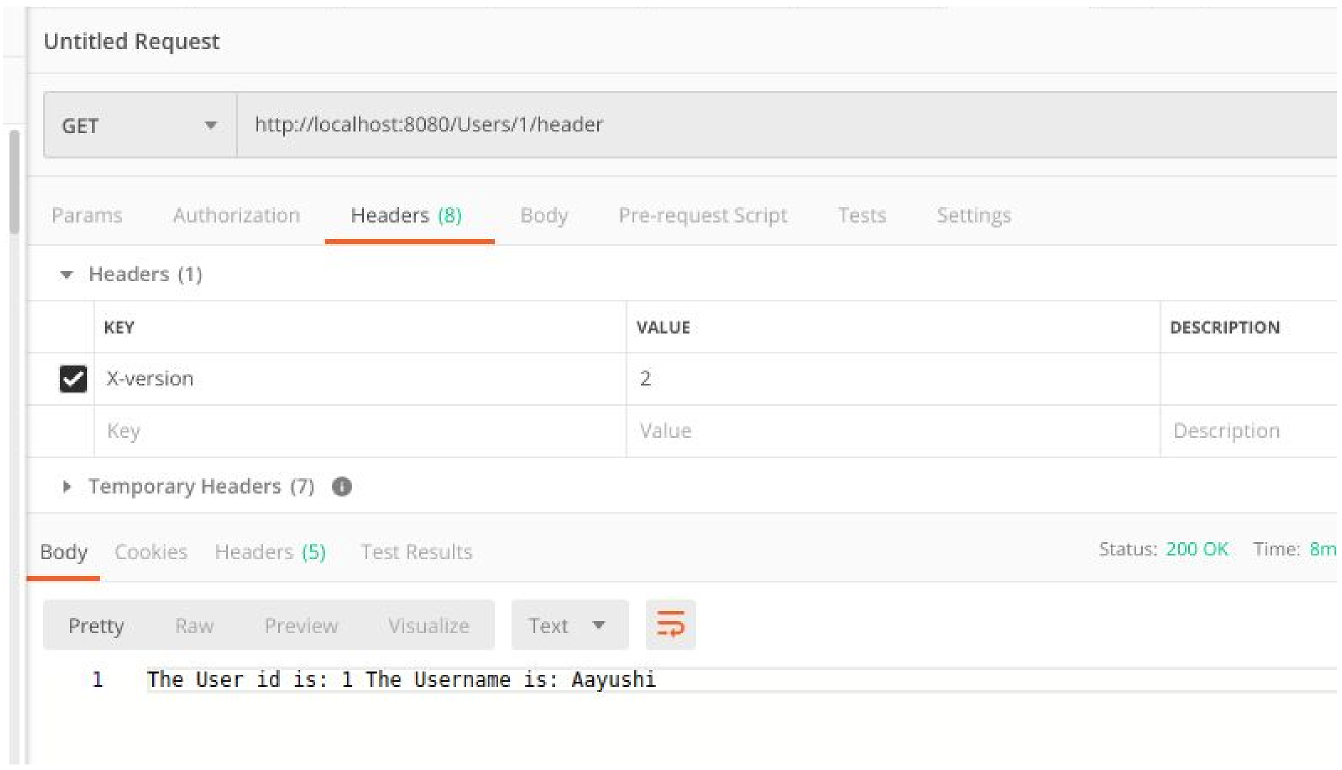


http://localhost:8080/Users/1/header

KEY	VALUE	DESCRIPTION
✓ X-version	1	
Key	Value	Description

Status: 200 OK Time: 12ms

```
The Username is: Aayushi
```



http://localhost:8080/Users/1/header

Headers (1)

	KEY	VALUE	DESCRIPTION
✓	X-version	2	
	Key	Value	Description

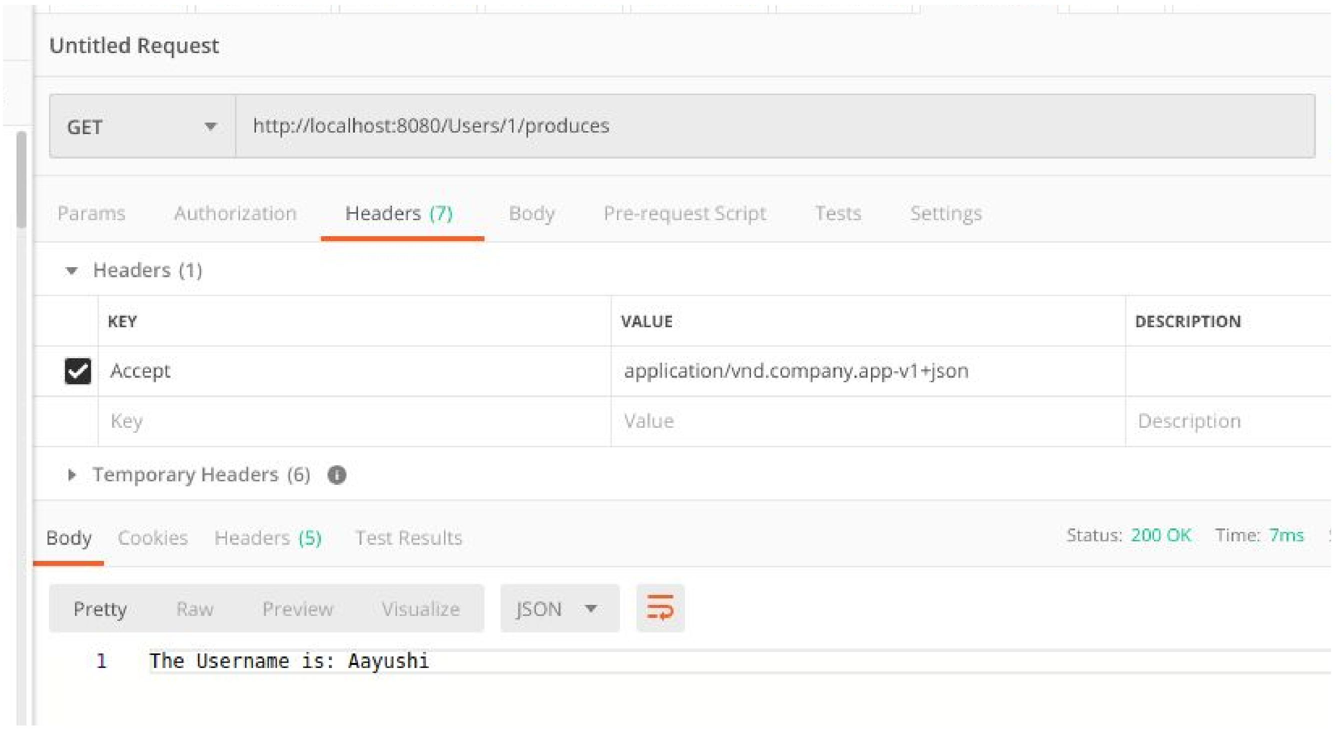
Status: 200 OK Time: 8m

```
The User id is: 1 The Username is: Aayushi
```

4. <u>MimeType Versioning</u>

```
//MimeType Versioning
@GetMapping(value = "Users/{id}/produces", produces =
"application/vnd.company.app-v1+json")
public String userproducesV1(@PathVariable Integer id){
  User user = obj.findOne(id);
  String username = user.getUsername();
  return "The Username is: "+username;
}

@GetMapping(value = "Users/{id}/produces", produces =
"application/vnd.company.app-v2+json")
public String userproducesV2(@PathVariable Integer id){
  User user = obj.findOne(id);
  String username = user.getUsername();
  String id1 = user.getId().toString();
  return "The User id is: "+ id1 + " The Username is: "+username;
}
```

OUTPUT


Untitled Request

GET http://localhost:8080/Users/1/produces

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Headers (1)

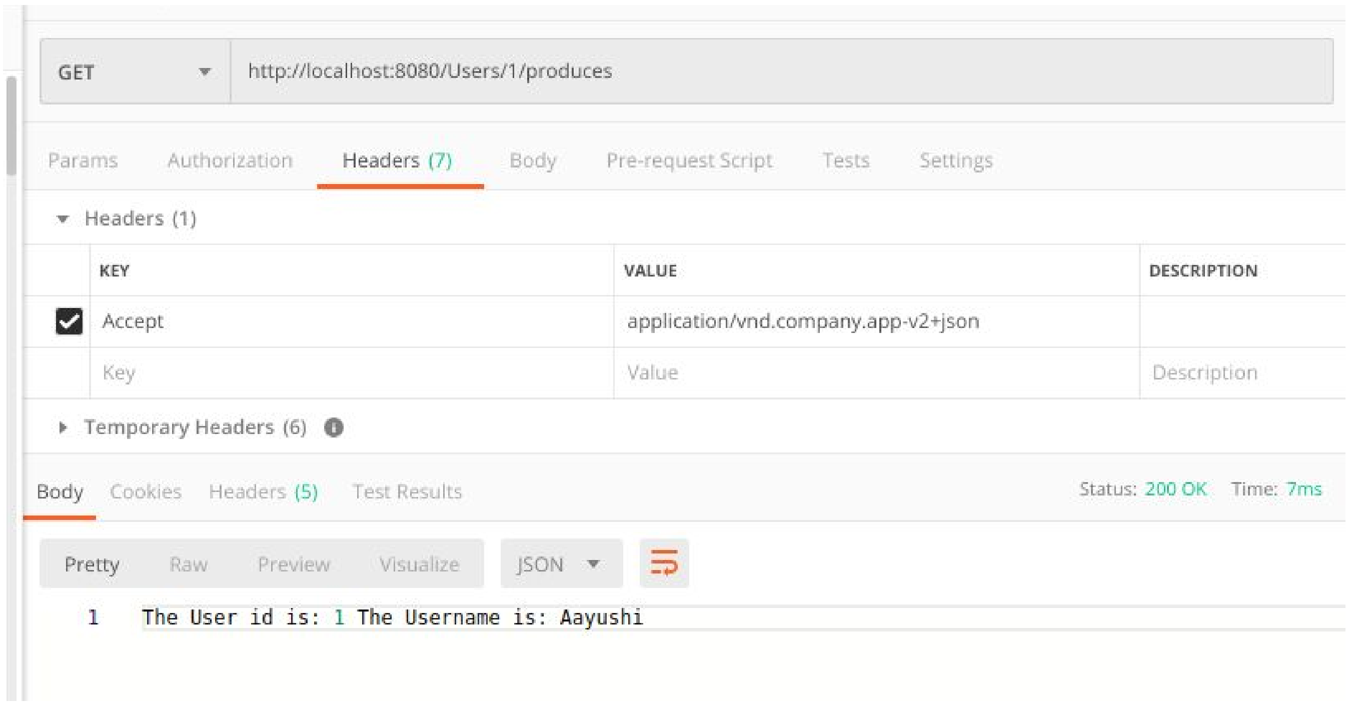
	KEY	VALUE	DESCRIPTION
✓	Accept	application/vnd.company.app-v1+json	
	Key	Value	Description

Temporary Headers (6)

Body Cookies Headers (5) Test Results Status: 200 OK Time: 7ms

Pretty Raw Preview Visualize JSON

```
The Username is: Aayushi
```


GET http://localhost:8080/Users/1/produces

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Headers (1)

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

Temporary Headers (6)

Body Cookies Headers (5) Test Results Status: 200 OK Time: 7ms

Pretty Raw Preview Visualize JSON

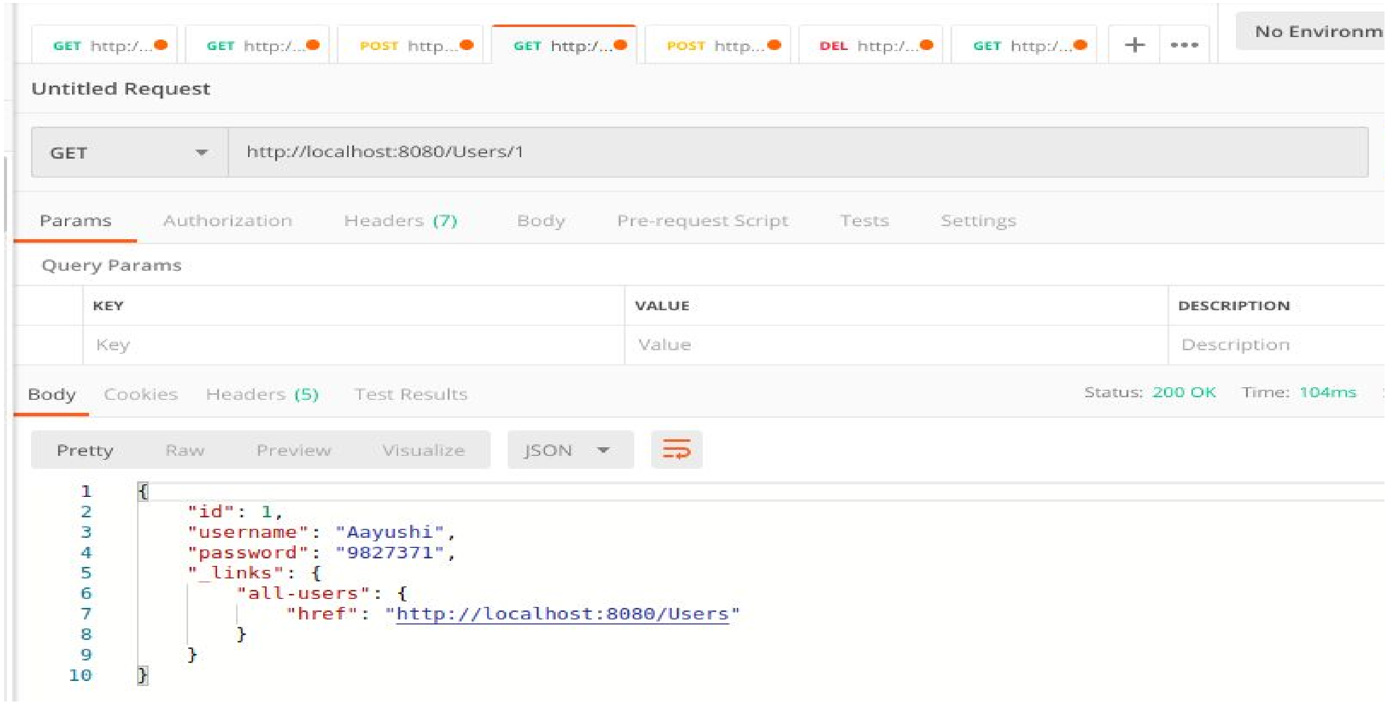
```
The User id is: 1 The Username is: Aayushi
```

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

CODE

```
//HATEOAS IMPLEMENTATION
@GetMapping(path="/Users/{id}")
public EntityModel<User> findOne(@PathVariable Integer id)
{
   User user = obj.findOne(id);

   EntityModel<User> model = new EntityModel<>(user);
   WebMvcLinkBuilder linkTo = linkTo(methodOn(this.getClass()).getAllUser());
   model.add(linkTo.withRel("all-users"));

   return model;
}
```

OUTPUT



GET http://localhost:8080/Users/1

Status: 200 OK Time: 104ms

```
{
    "id": 1,
    "username": "Aayushi",
    "password": "9827371",
    "_links": {
        "all-users": {
            "href": "http://localhost:8080/Users"
        }
    }
}
```