

Restful WebServices Part 2

*Internationalization

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

The screenshot displays a REST client interface with the following components:

- URL Bar:** `http://localhost:8080/message`
- Method:** `GET`
- Send Button:** A blue button labeled "Send".
- Tabs:** Params, Authorization, **Headers (7)**, Body, Pre-request Script, Tests, Settings, Cookies.
- Headers Table:**

	Key	Value	Description	...	Bulk Edit	Presets
<input type="checkbox"/>	Accept-Language					
	Key	Value	Description			
- Body Tab:** Shows the response `1 My App Welcome to My App`.
- Status Bar:** `200 OK 87 ms 188 B`
- Save as Example:** A button to save the response.

http://localhost:8080/message

Save

GET

http://localhost:8080/message

Send

Params

Authorization

Headers (7)

Body

Pre-request Script

Tests

Settings

Cookies

Headers

6 hidden

	Key	Value	Description	...	Bulk Edit	Presets
<input checked="" type="checkbox"/>	Accept-Language	de				
	Key	Value	Description			

Body

Cookies

Headers (5)

Test Results

200 OK

9 ms

198 B

Save as Example

Pretty

Raw

Preview

Visualize

Text

1

Meine App Willkommen bei Meine App

http://localhost:8080/message

Save

GET

http://localhost:8080/message

Send

Params

Authorization

Headers (7)

Body

Pre-request Script

Tests

Settings

Cookies

Headers

6 hidden

	Key	Value	Description	...	Bulk Edit	Presets
<input checked="" type="checkbox"/>	Accept-Language	sv				
	Key	Value	Description			

Body

Cookies

Headers (5)

Test Results

200 OK

124 ms

194 B

Save as Example

Pretty

Raw

Preview

Visualize

Text

1

Min App Valkommen till Min App

2.

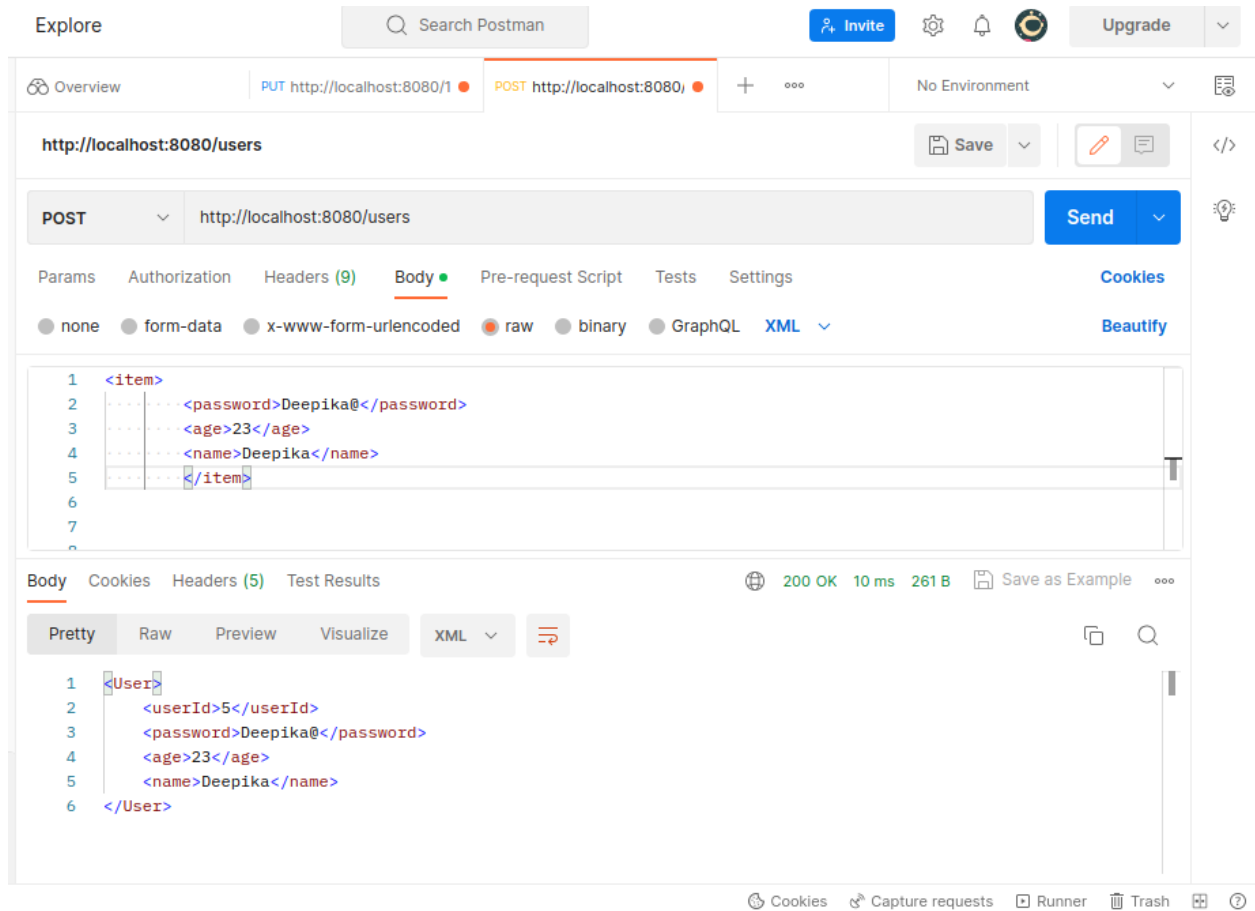
Create a GET request which takes "username" as param and shows a localized message "Hello Username". (Use parameters in message properties)

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

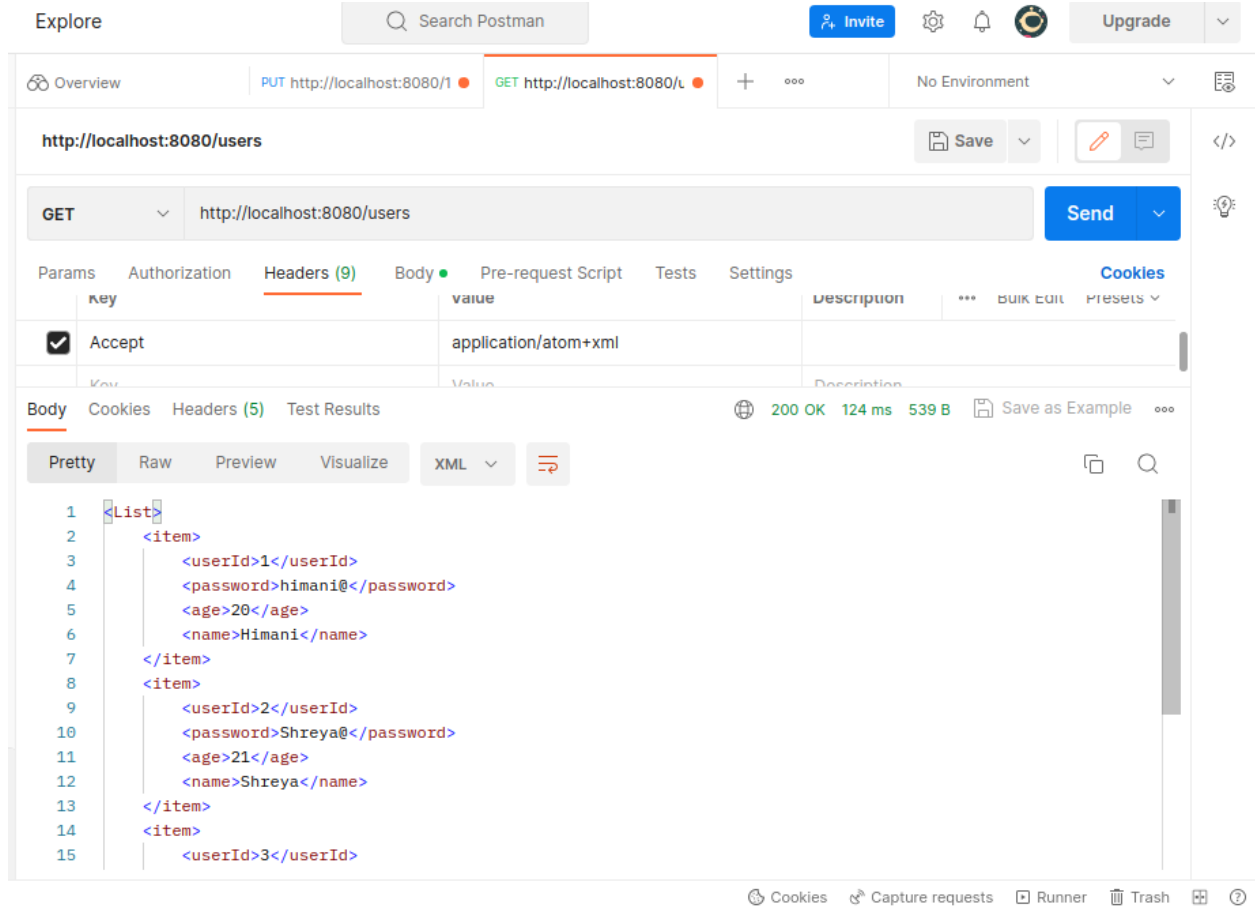
- URL:** `http://localhost:8080/greeting?username=Himani`
- Method:** GET
- Headers (7):** A table with columns Key, Value, and Description. The first row is empty, and the second row has the headers Key, Value, and Description.
- Body:** Pretty, Raw, Preview, Visualize. The response is `1 Hello Himani`.
- Status:** 200 OK, 93 ms, 176 B
- Actions:** Save, Send, Bulk Edit, Presets, Save as Example.

*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.



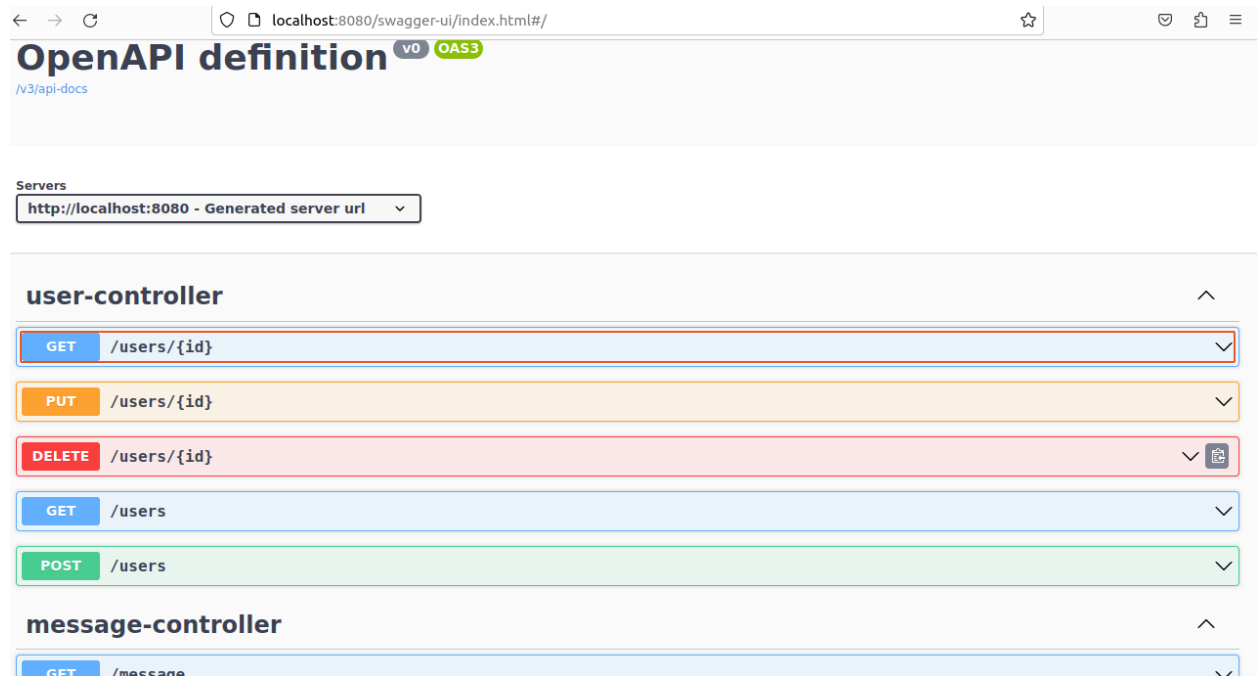
*Swagger

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

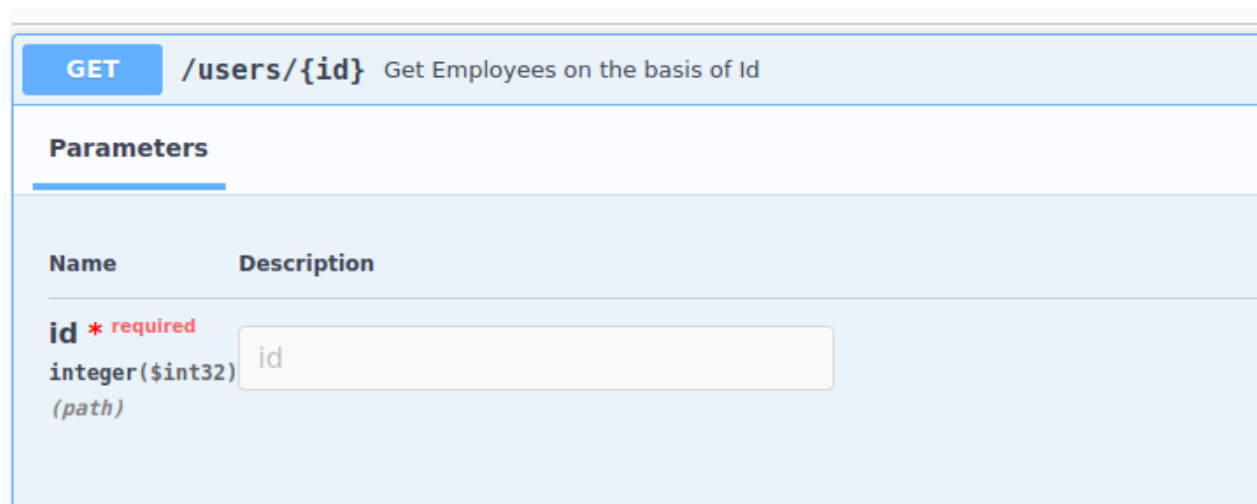
Get details of User using GET request.

Save details of the user using POST request.

Delete a user using DELETE request.



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.



Code	Description
200	<p>User Found</p> <p>Media type</p> <div>application/json</div> <p>Controls Accept header.</p> <p>Example Value Schema</p> <pre>{ "userId": 0, "password": "string", "age": 0, "name": "string" }</pre>
400	Invalid id supplied
404	User not found

*Static and Dynamic filtering

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

http://localhost:8080/users

Save

POST http://localhost:8080/users Send

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

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
1 {
2   "password": "devansh@",
3   "age": 20,
4   "name": "Devansh"
5 }
```

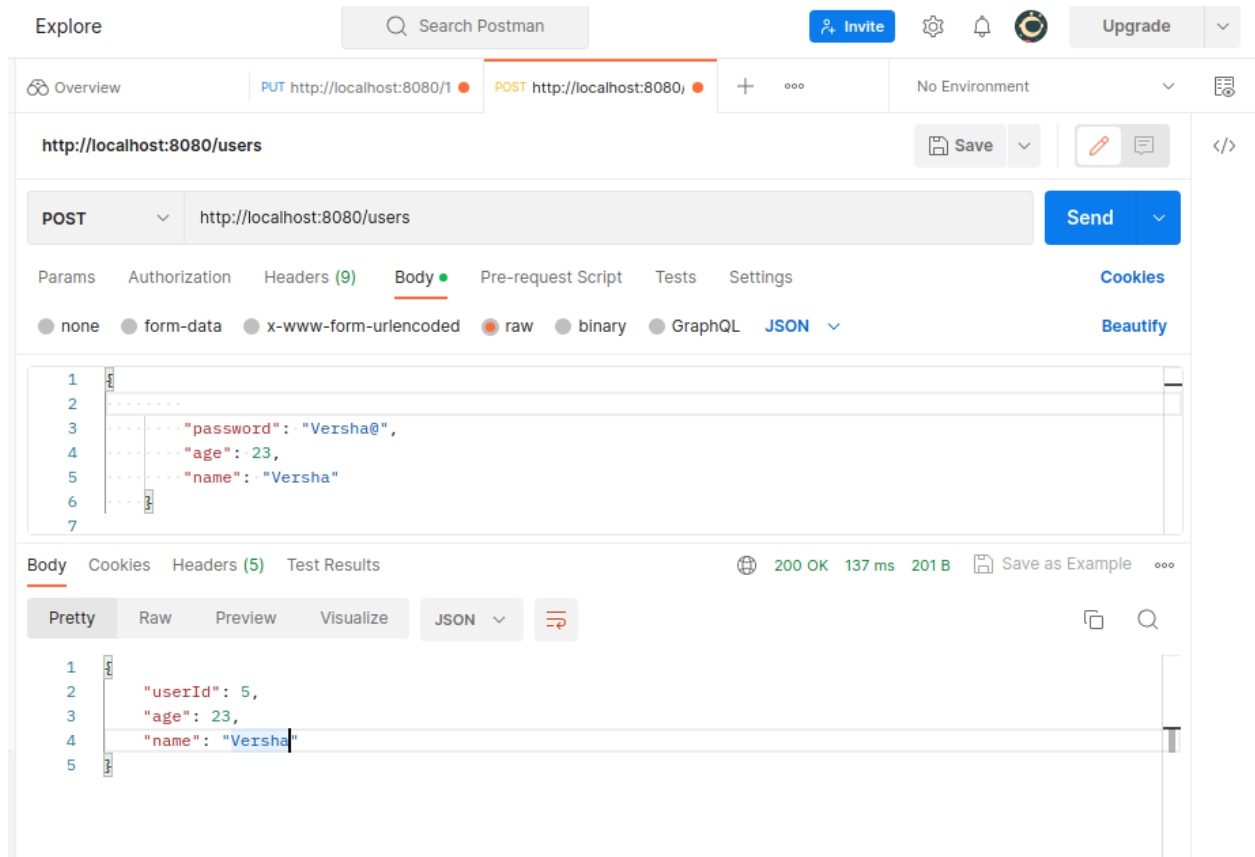
Body Cookies Headers (5) Test Results 200 OK 136 ms 202 B Save as Example

Pretty Raw Preview Visualize JSON

```
1 {
2   "userId": 5,
3   "age": 20,
4   "name": "Devansh"
5 }
```

Cookies Capture requests Runner Trash

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



*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,

Now apply versioning using the following methods:

- Media Type Versioning

GET http://localhost:8080/user/accept Send

Params Authorization Headers (10) Body • Pre-request Script Tests Settings Cookies

	Key	Value	Description	...	Bulk Edit	Presets
<input type="checkbox"/>	X-API-VERSION	2				
<input checked="" type="checkbox"/>	Accept	application/user.app-v1+json				
	Key	Value	Description			

Body Cookies Headers (5) Test Results 200 OK 89 ms 216 B Save as Example

Pretty Raw Preview Visualize JSON Copy Search

```
1 {
2   "userId": 2345,
3   "name": "Himani Bhardwaj"
4 }
```

Cookies Capture requests Runner Trash Grid Help

GET http://localhost:8080/user/accept Send

Params Authorization Headers (10) Body • Pre-request Script Tests Settings Cookies

	Key	Value	Description	...	Bulk Edit	Presets
<input type="checkbox"/>	X-API-VERSION	2				
<input checked="" type="checkbox"/>	Accept	application/user.app-v2+json				
	Key	Value	Description			

Body Cookies Headers (5) Test Results 200 OK 6 ms 252 B Save as Example

Pretty Raw Preview Visualize JSON Copy Search

```
1 {
2   "name": {
3     "firstName": "Himani",
4     "lastName": "Bhardwaj"
5   },
6   "userId": 4567,
7   "age": 21
8 }
```

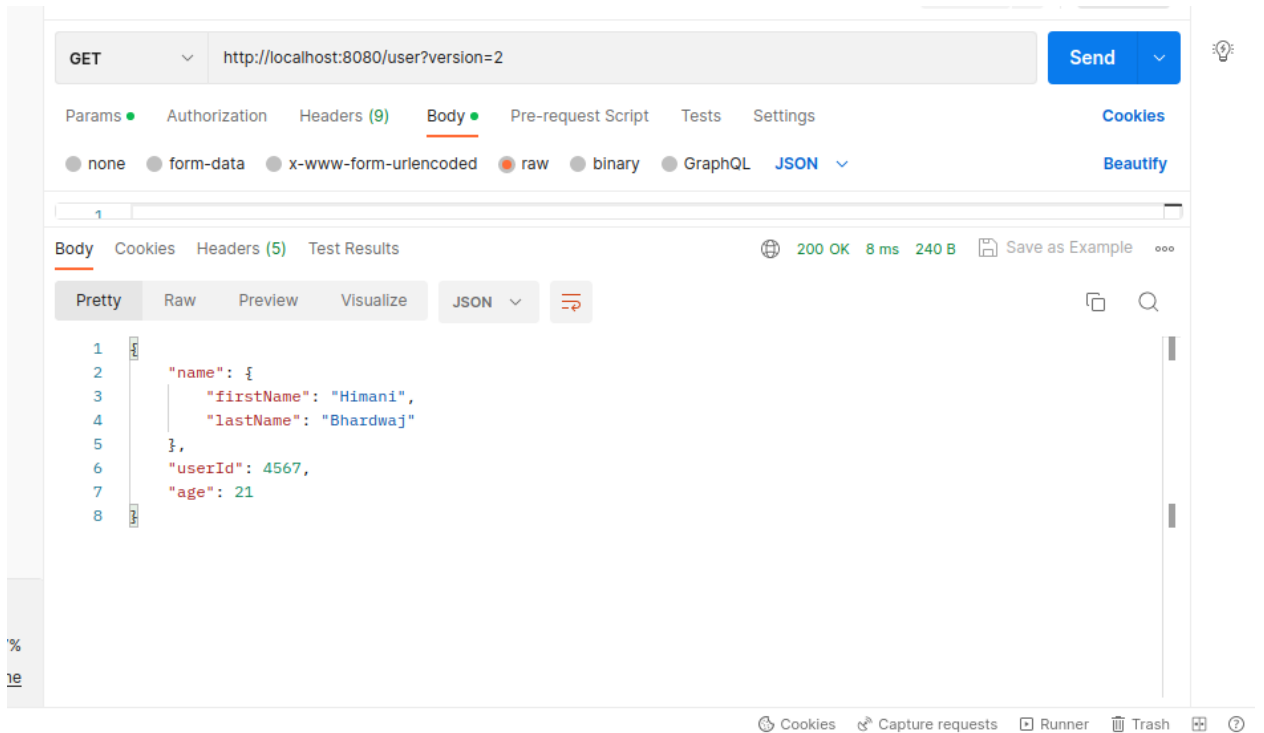
Cookies Capture requests Runner Trash Grid Help

Request Parameter versioning

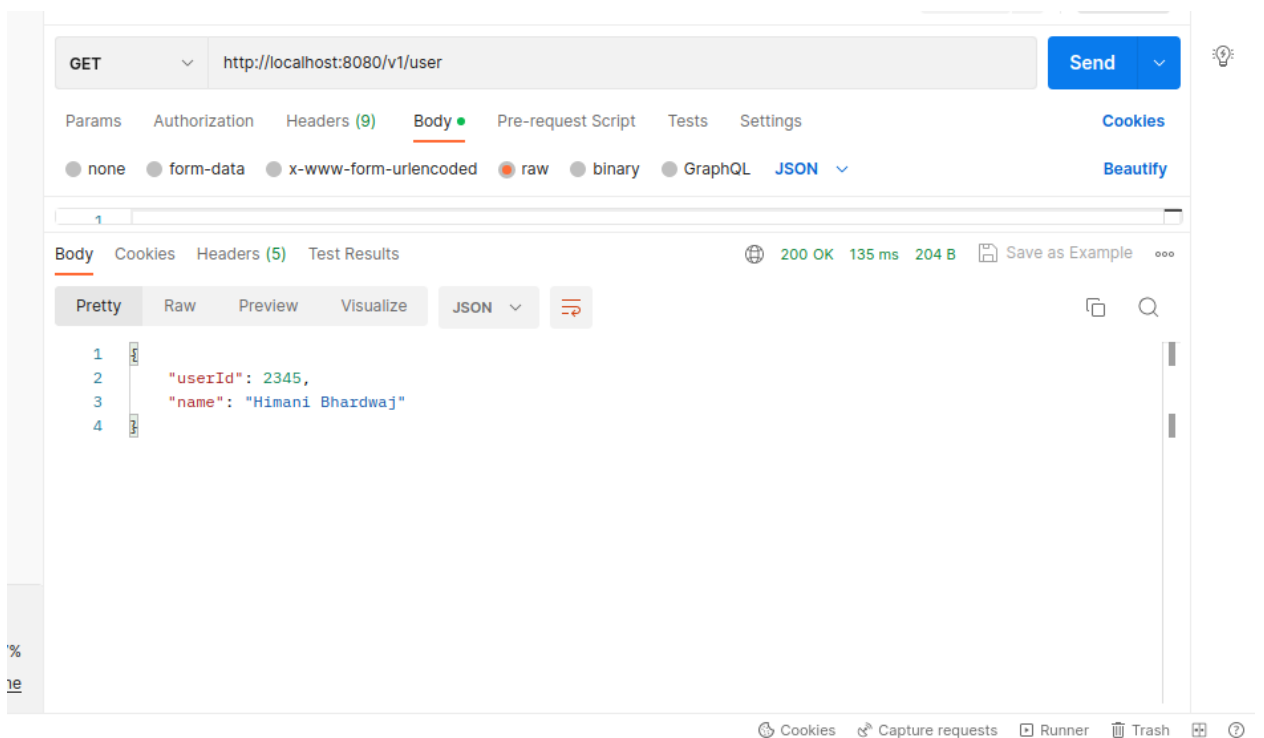
The screenshot displays a REST client interface with the following details:

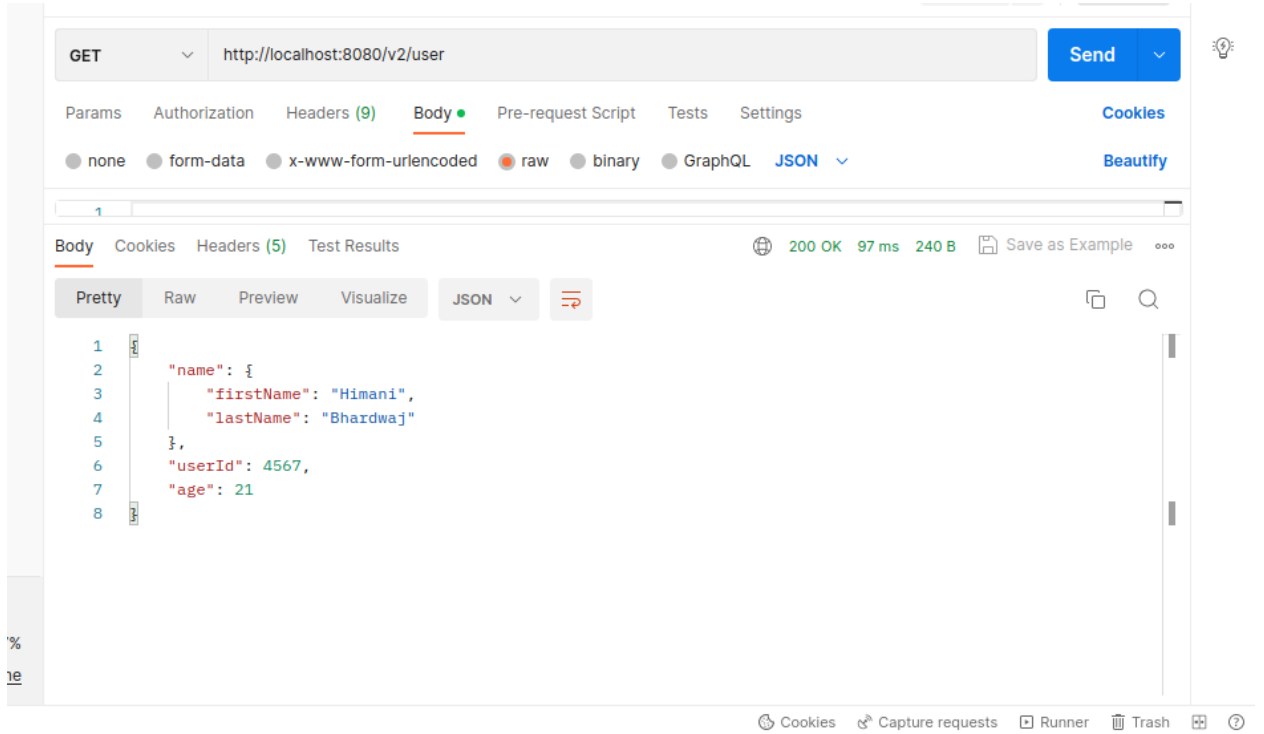
- Request Method:** GET
- URL:** http://localhost:8080/user?version=1
- Send Button:** A blue button labeled "Send" with a dropdown arrow.
- Request Body Type:** raw (selected), with other options like none, form-data, x-www-form-urlencoded, binary, and GraphQL.
- Response Status:** 200 OK, 85 ms, 204 B.
- Response Body:** A JSON object:

```
{  "userId": 2345,  "name": "Himani Bhardwaj"}
```
- Response Format:** JSON (selected), with other options like Pretty, Raw, Preview, and Visualize.
- Bottom Bar:** Includes buttons for Cookies, Capture requests, Runner, Trash, and a help icon.



- ## URI versioning





Custom Header Versioning

GET http://localhost:8080/user/header Send

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

Headers 8 hidden

	Key	Value	Description	...	Bulk Edit	Presets
<input checked="" type="checkbox"/>	X-API-VERSION	1				
	Key	Value	Description			

Body Cookies Headers (5) Test Results 200 OK 14 ms 204 B Save as Example

Pretty Raw Preview Visualize JSON

```
1
2  "userId": 2345,
3  "name": "Himani Bhardwaj"
4
```

Cookies Capture requests Runner Trash

GET http://localhost:8080/user/header Send

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

Headers 8 hidden

	Key	Value	Description	...	Bulk Edit	Presets
<input checked="" type="checkbox"/>	X-API-VERSION	2				
	Key	Value	Description			

Body Cookies Headers (5) Test Results 200 OK 7 ms 240 B Save as Example

Pretty Raw Preview Visualize JSON

```
1
2  "name": {
3    "firstName": "Himani",
4    "lastName": "Bhardwaj"
5  },
6  "userId": 4567,
7  "age": 21
8
```

Cookies Capture requests Runner Trash

*HATEOAS

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

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

- URL:** `http://localhost:8080/users/1`
- Method:** `GET`
- Status:** `200 OK`, `41 ms`, `288 B`
- Body (JSON):**

```
1 {
2   "userId": 1,
3   "password": "himani@",
4   "age": 20,
5   "name": "Himani",
6   "_links": {
7     "all-users": {
8       "href": "http://localhost:8080/users"
9     }
10  }
11 }
```

The interface includes tabs for Params, Authorization, Headers (6), Body, Pre-request Script, Tests, Settings, and Cookies. The Body tab is active, showing the JSON response in a Pretty view. The bottom status bar includes options for Cookies, Capture requests, Runner, Trash, and a help icon.