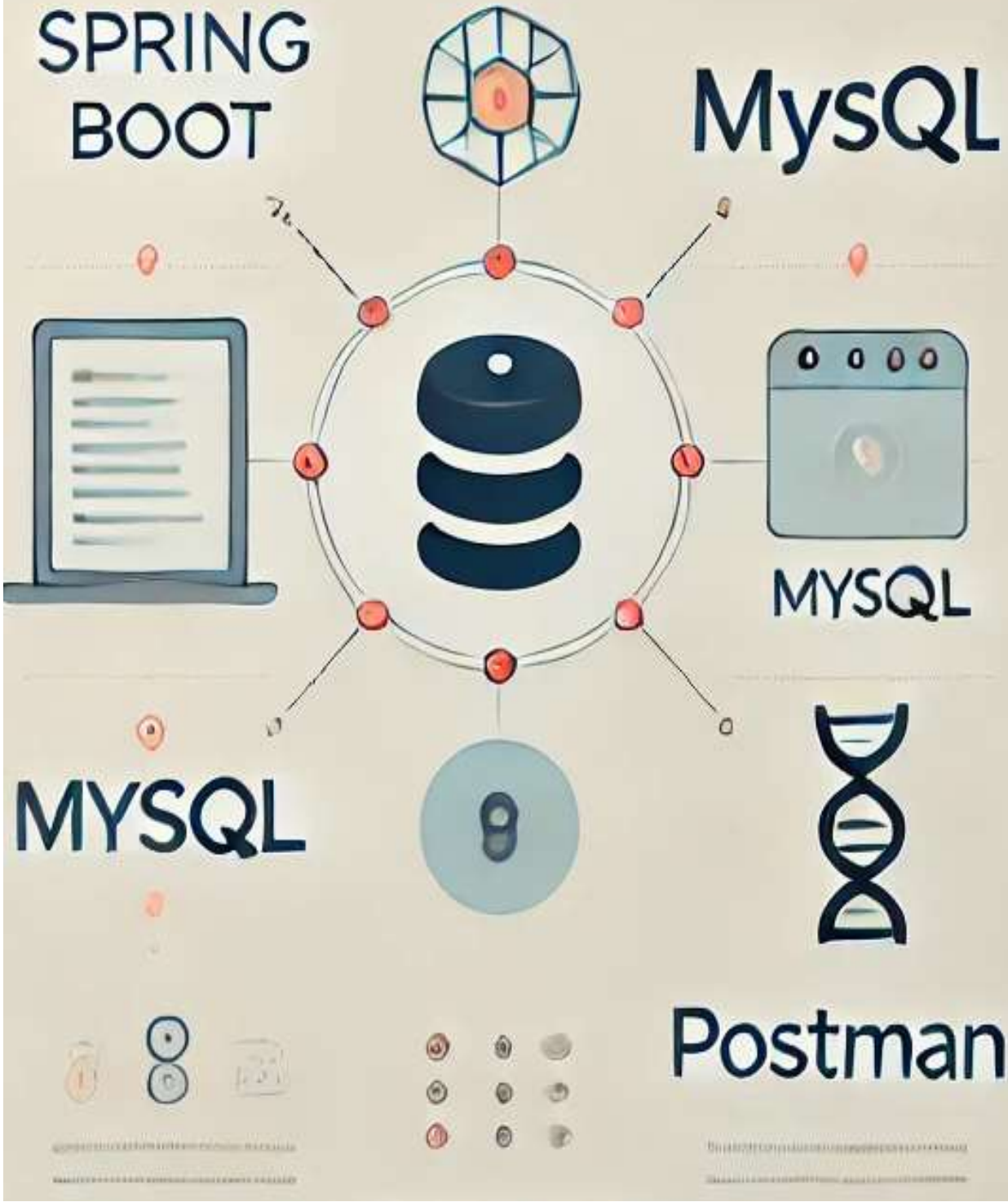
Spring Boot

SECURITY PROJECT



Project Technologies and Security Structure

Technologies Used

- **Java Version**: 17.0.7
- **Spring Boot Version**: 3.3.2
- **MySQL Version**: 8.0.40

Security Structure and Authentication

In this project, a comprehensive security infrastructure has been developed using **Spring Boot Security** to ensure secure user authentication and authorization processes.

JWT Integration

- **JSON Web Token (JWT)** integration enables secure and controlled access for users.
- Role-based access control is enforced through the **@PreAuthorize** annotation.
- User session information is securely stored via **cookies** to enhance security.

Token Expiration and Blacklist Usage

- Tokens provided to users are valid for a limited time.
- When a token expires or the user logs out, it is blacklisted to prevent reuse.

Access Limitations for Enhanced Security

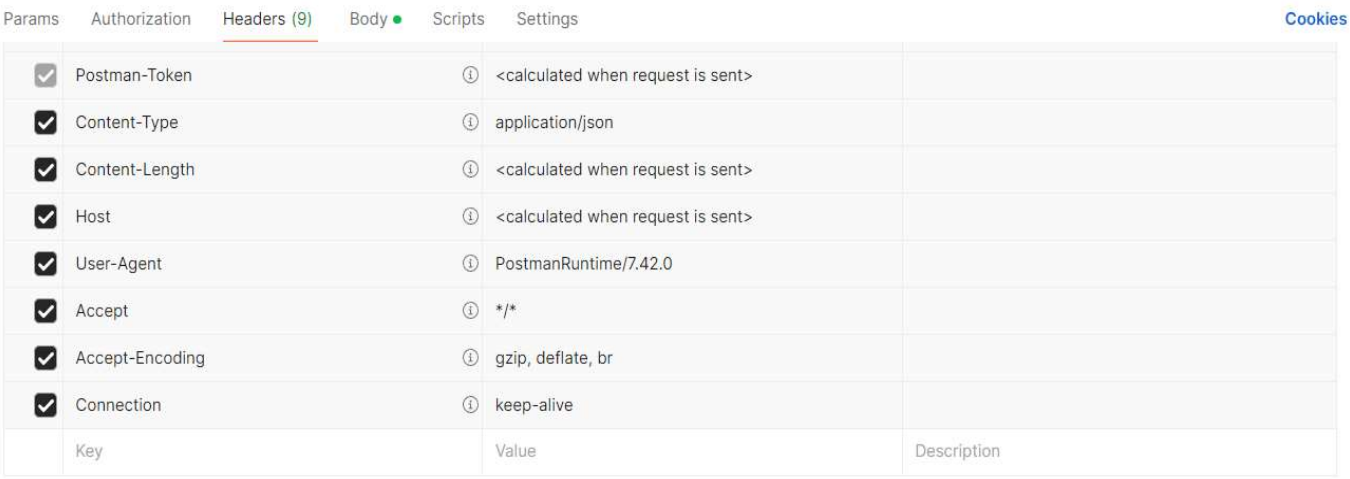
- Critical controllers are configured with a limited number of access rights within specific time frames.
- These limitations add an extra layer of security against brute force and dictionary attacks.
- Users are allowed a restricted number of login attempts or actions within a specified period.

Email Verification

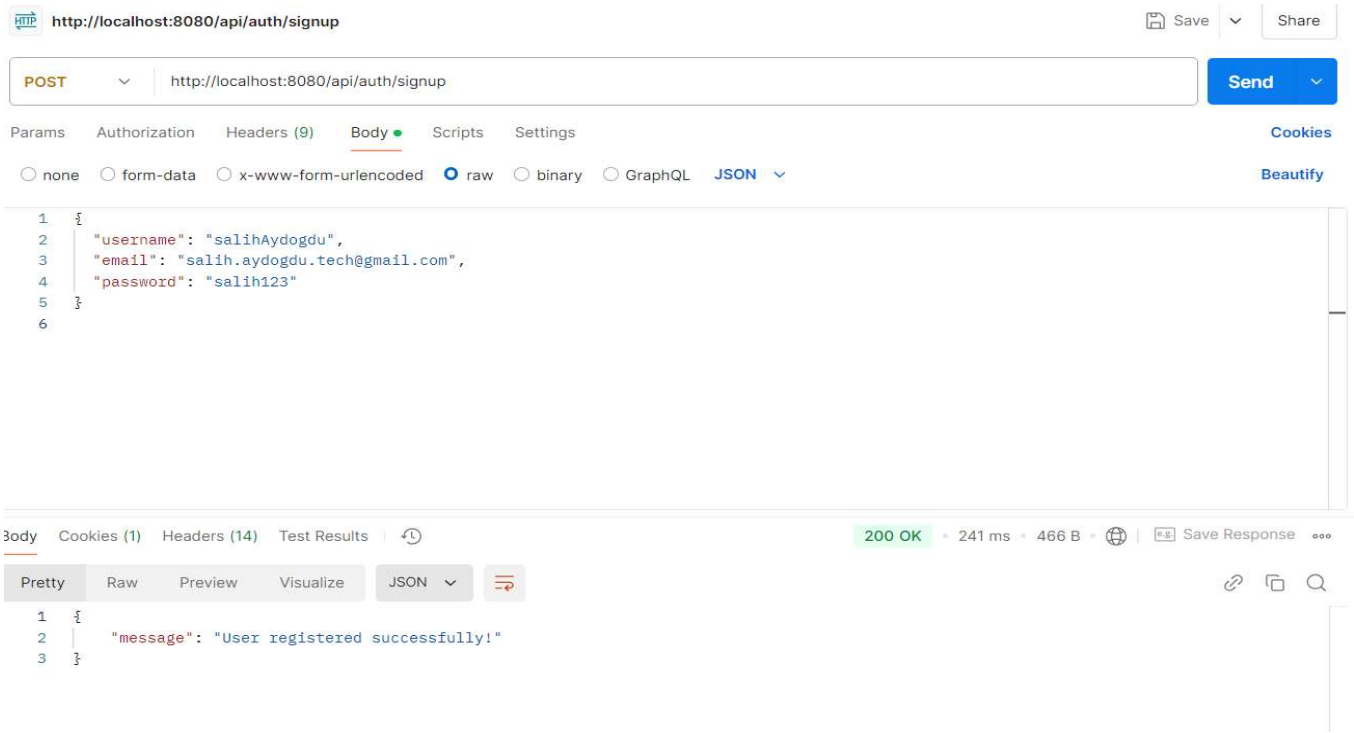
- Upon login, a **verification code is sent via SMTP** for security verification.
- Session access is granted only after successful verification.

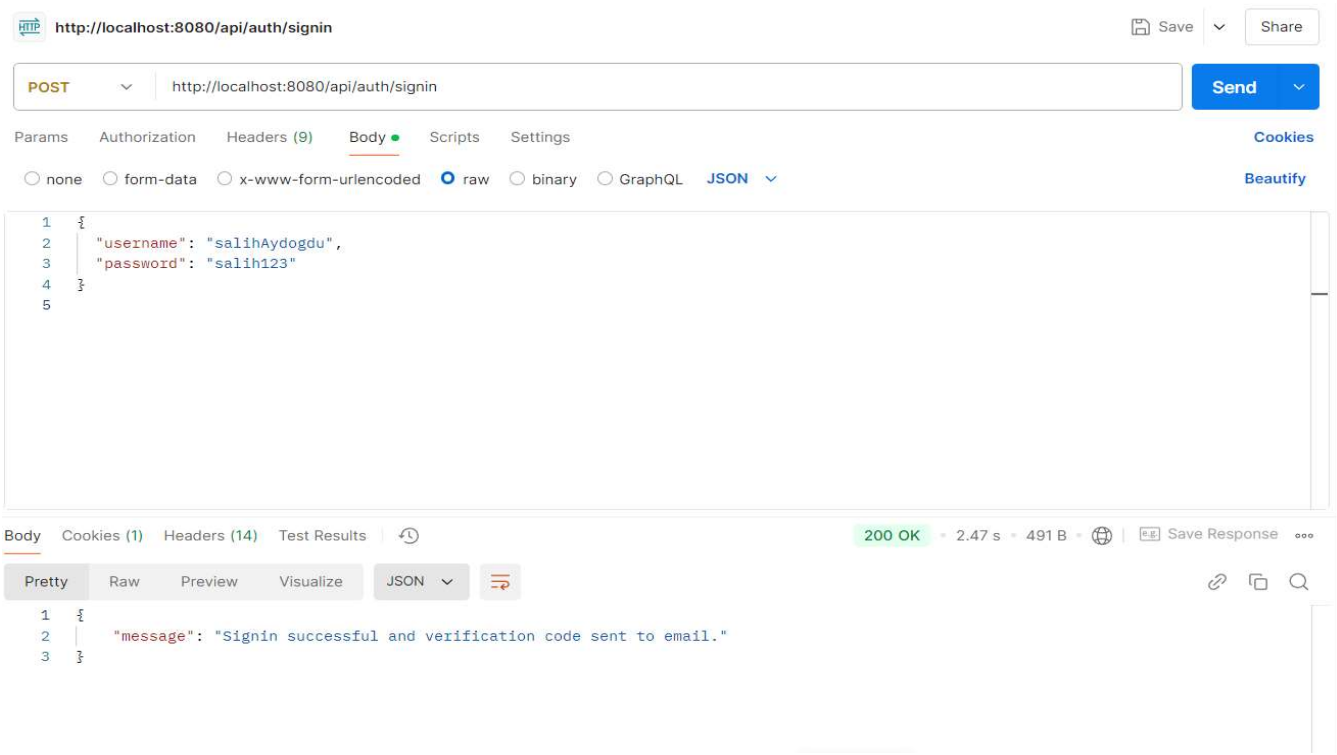
Purpose of the Security Structure This structure aims to protect user data, ensure secure session management, and provide a continuous security mechanism.

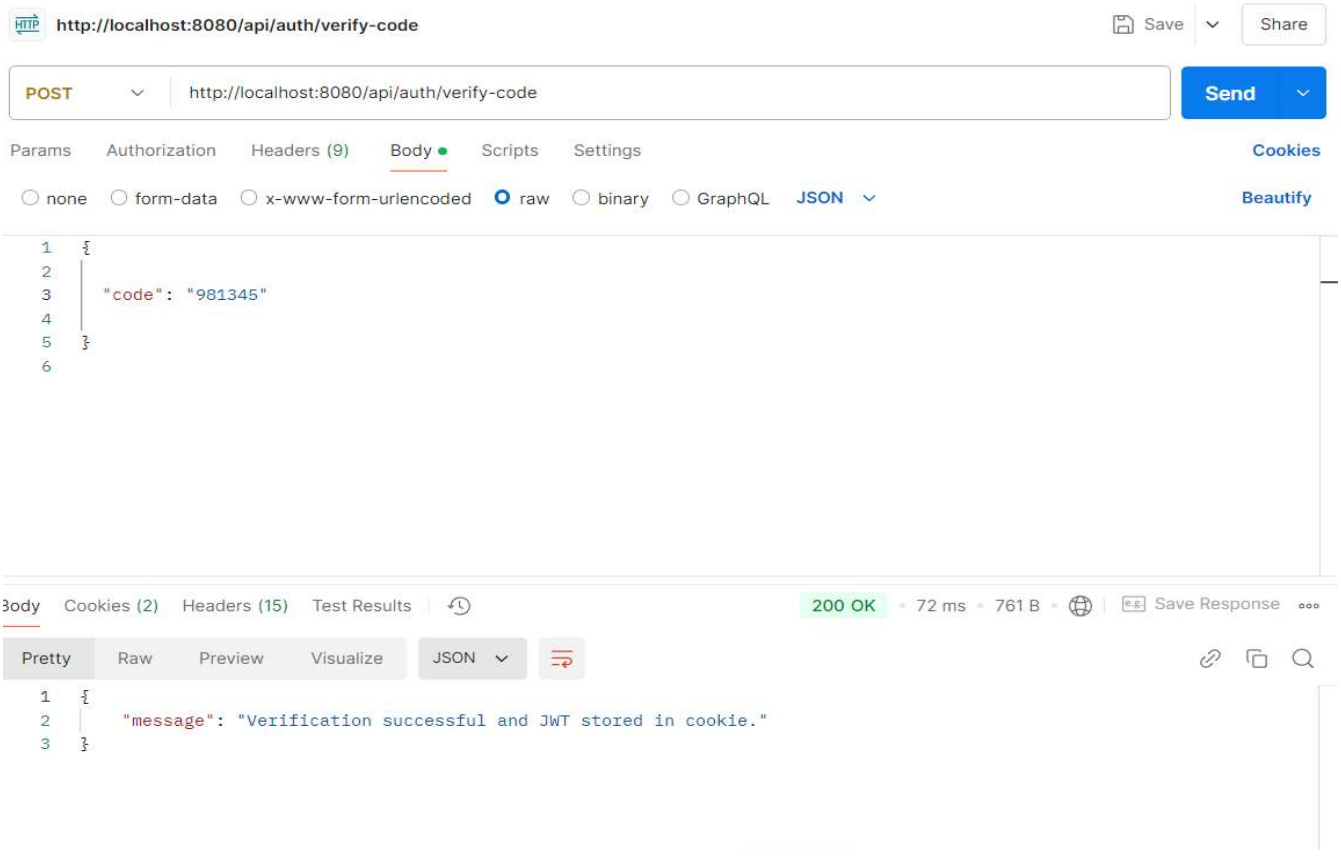
Postman API Test Cases and Results

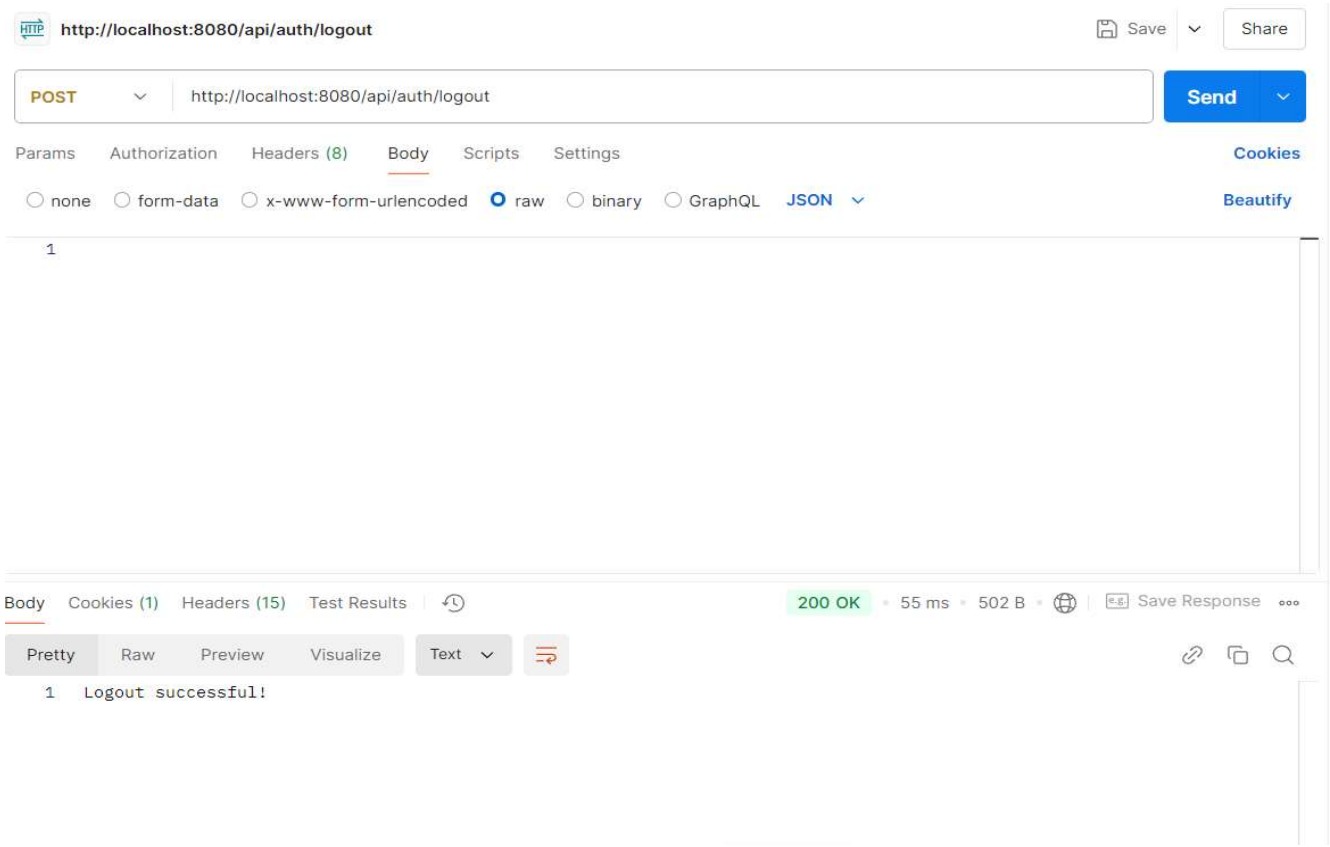


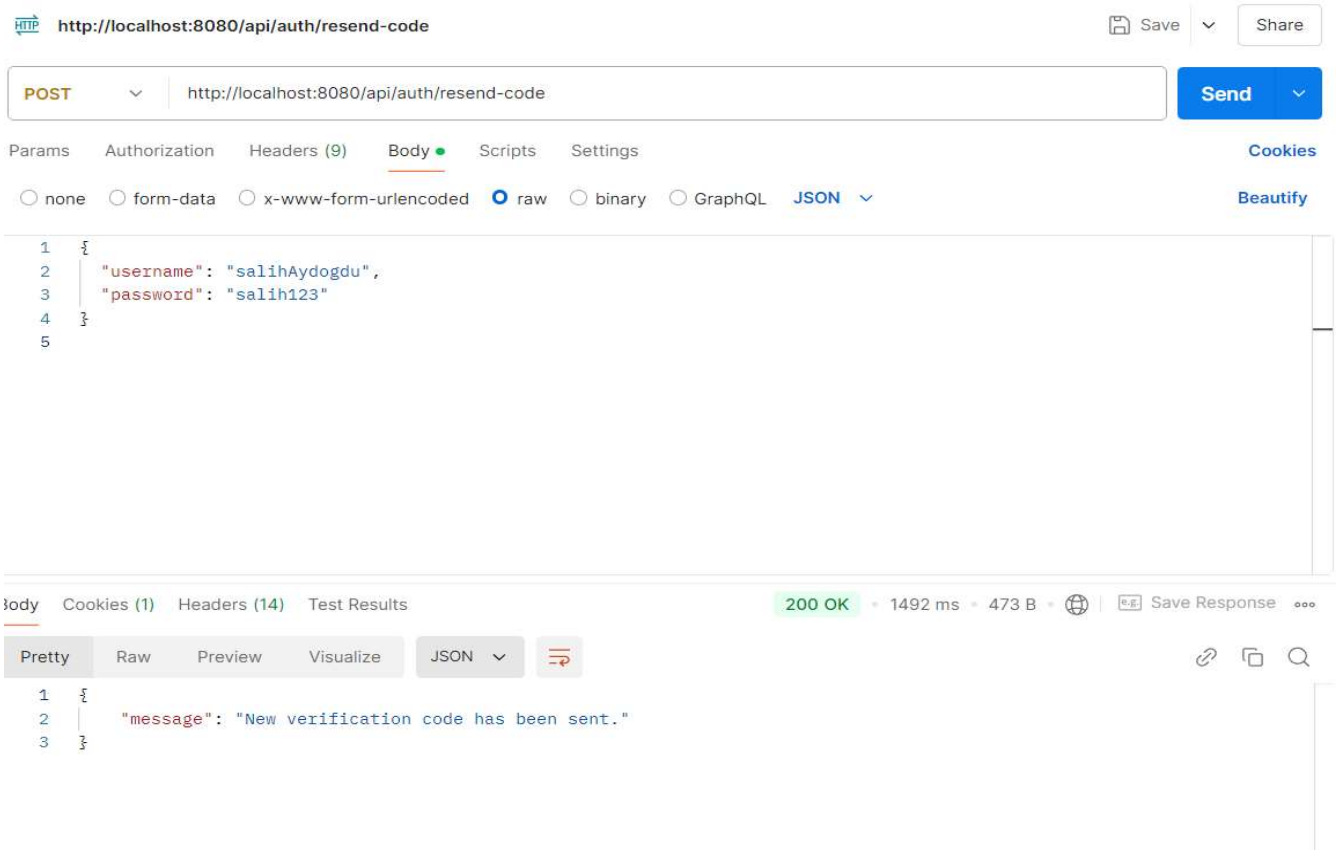
AuthController :

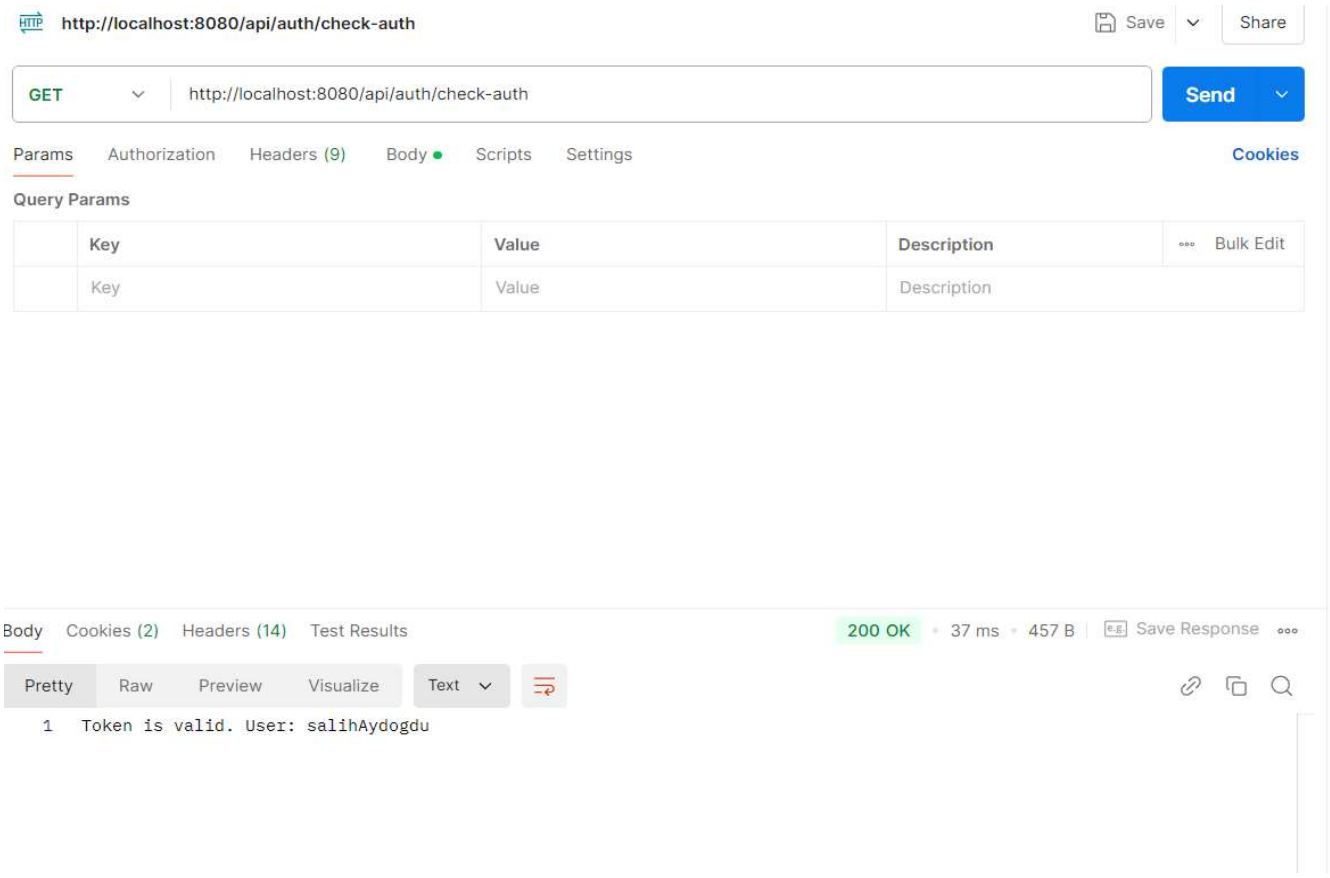



http://localhost:8080/api/auth/signin
Save
Share
POST
http://localhost:8080/api/auth/signin
Send
Params
Authorization
Headers (9)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"username": "salihAydogdu",
"password": "salih123"
}
Body
Cookies (1)
Headers (14)
Test Results
200 OK
2.47 s
491 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"message": "Signin successful and verification code sent to email."
}

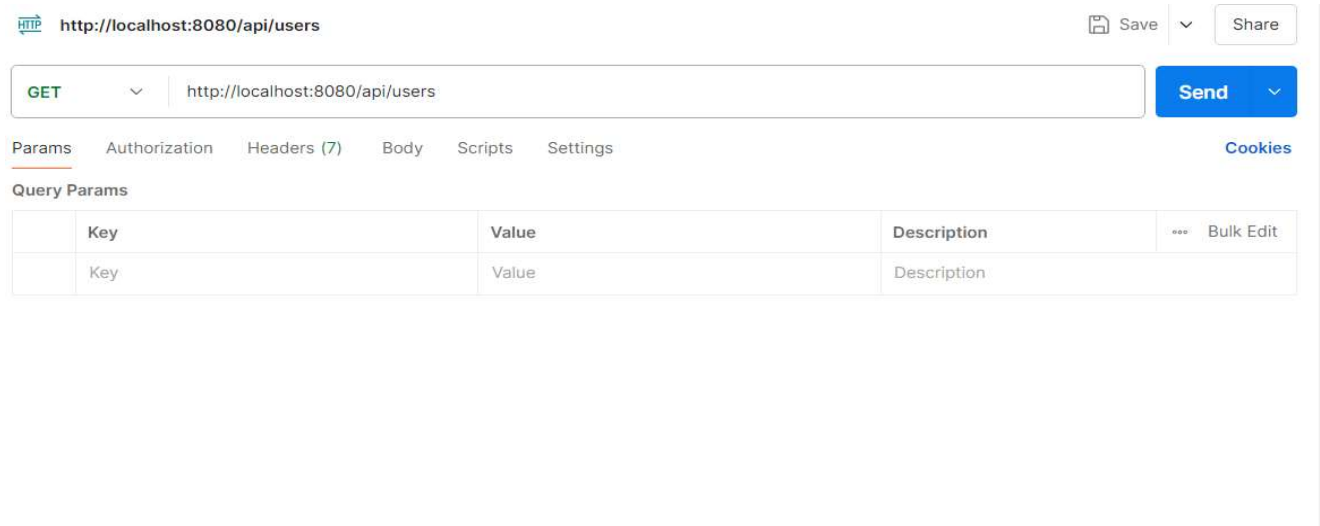

http://localhost:8080/api/auth/verify-code
Save
Share
POST
http://localhost:8080/api/auth/verify-code
Send
Params
Authorization
Headers (9)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"code": "981345"
}
Body
Cookies (2)
Headers (15)
Test Results
200 OK
72 ms
761 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"message": "Verification successful and JWT stored in cookie."
}

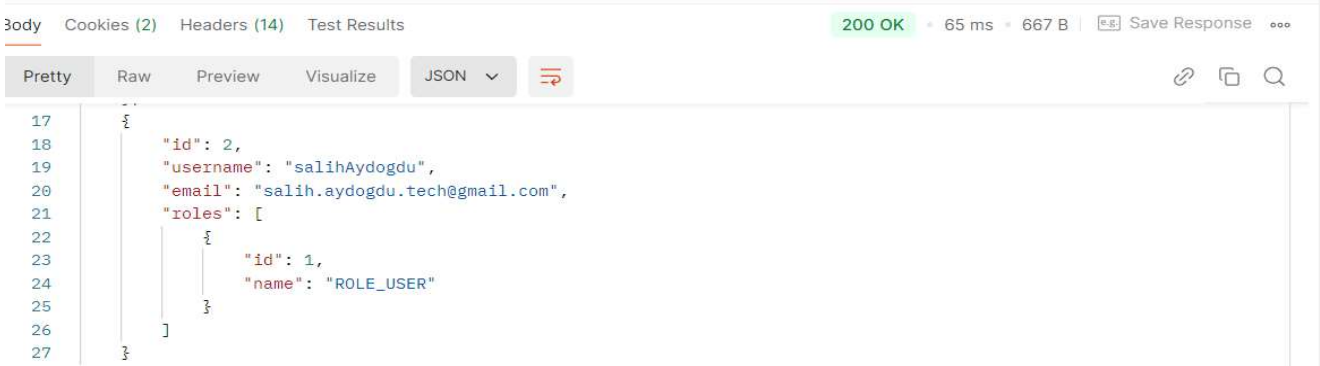

http://localhost:8080/api/auth/logout
Save
Share
POST
http://localhost:8080/api/auth/logout
Send
Params
Authorization
Headers (8)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
1
Body
Cookies (1)
Headers (15)
Test Results
200 OK
55 ms
502 B
Save Response
Pretty
Raw
Preview
Visualize
Text
1 Logout successful!

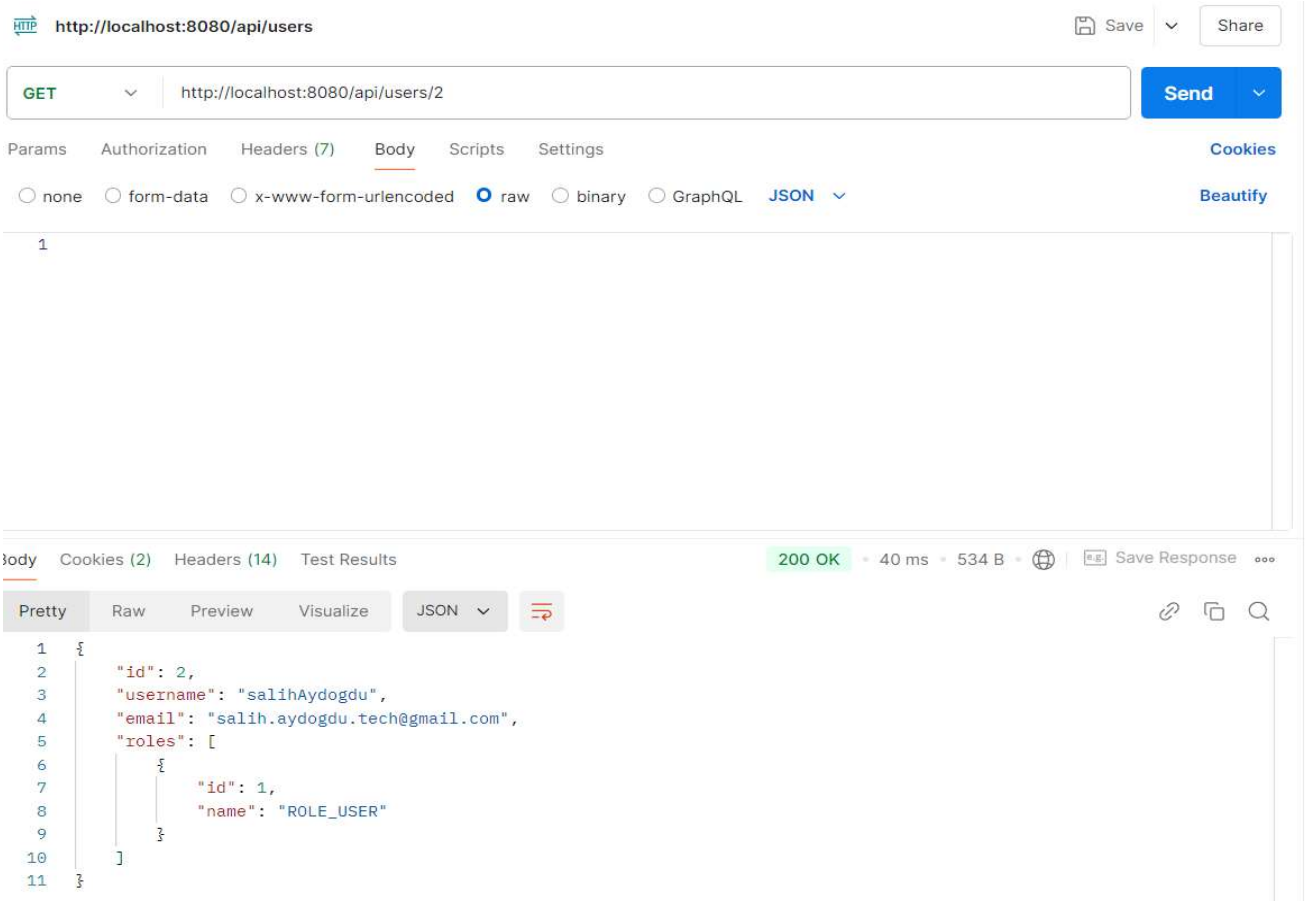

http://localhost:8080/api/auth/resend-code
Save
Share
POST
http://localhost:8080/api/auth/resend-code
Send
Params
Authorization
Headers (9)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
1 {
2 "username": "salihAydogdu",
3 "password": "salih123"
4 }
5
Body
Cookies (1)
Headers (14)
Test Results
200 OK
1492 ms
473 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
1 {
2 "message": "New verification code has been sent."
3 }

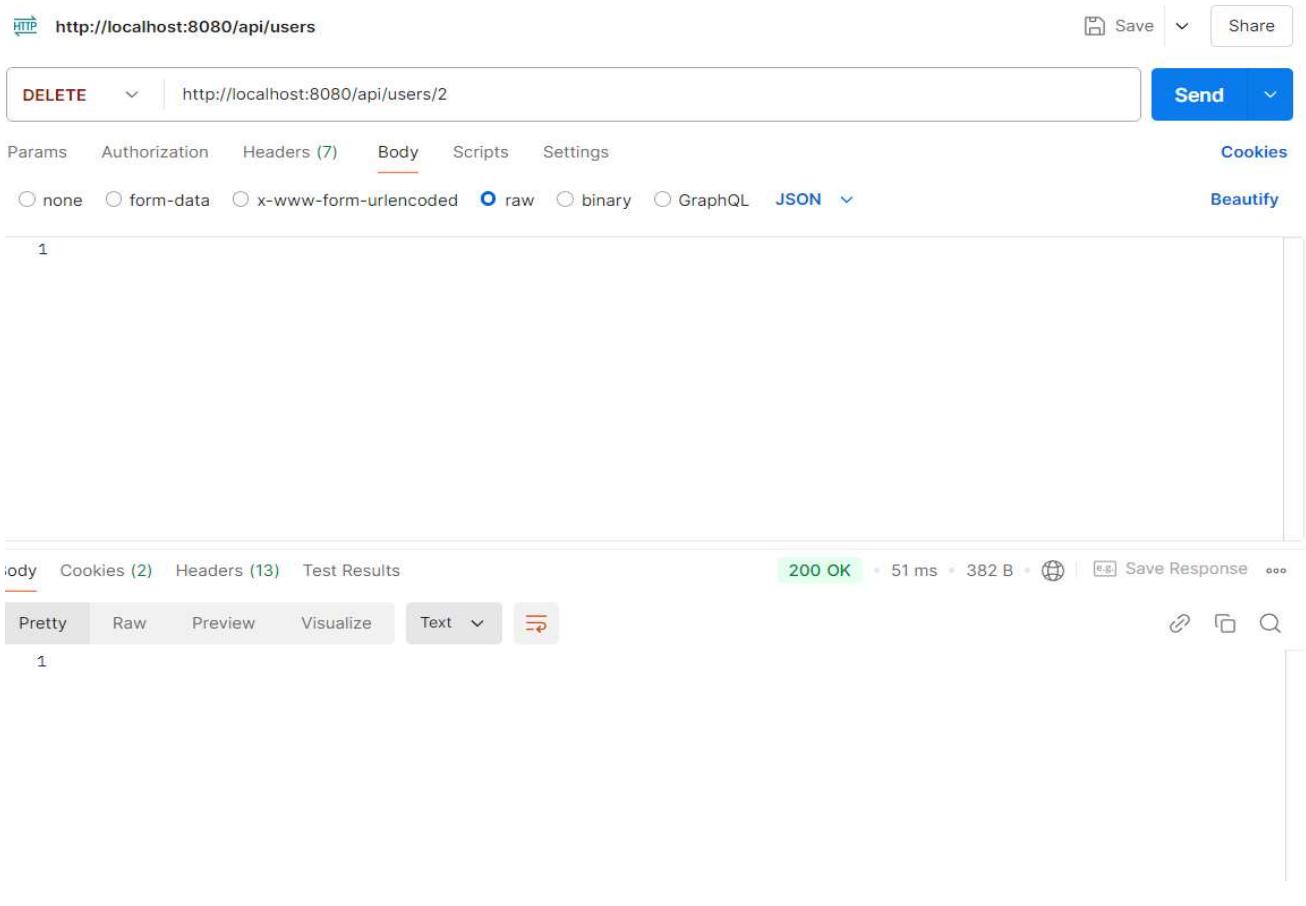


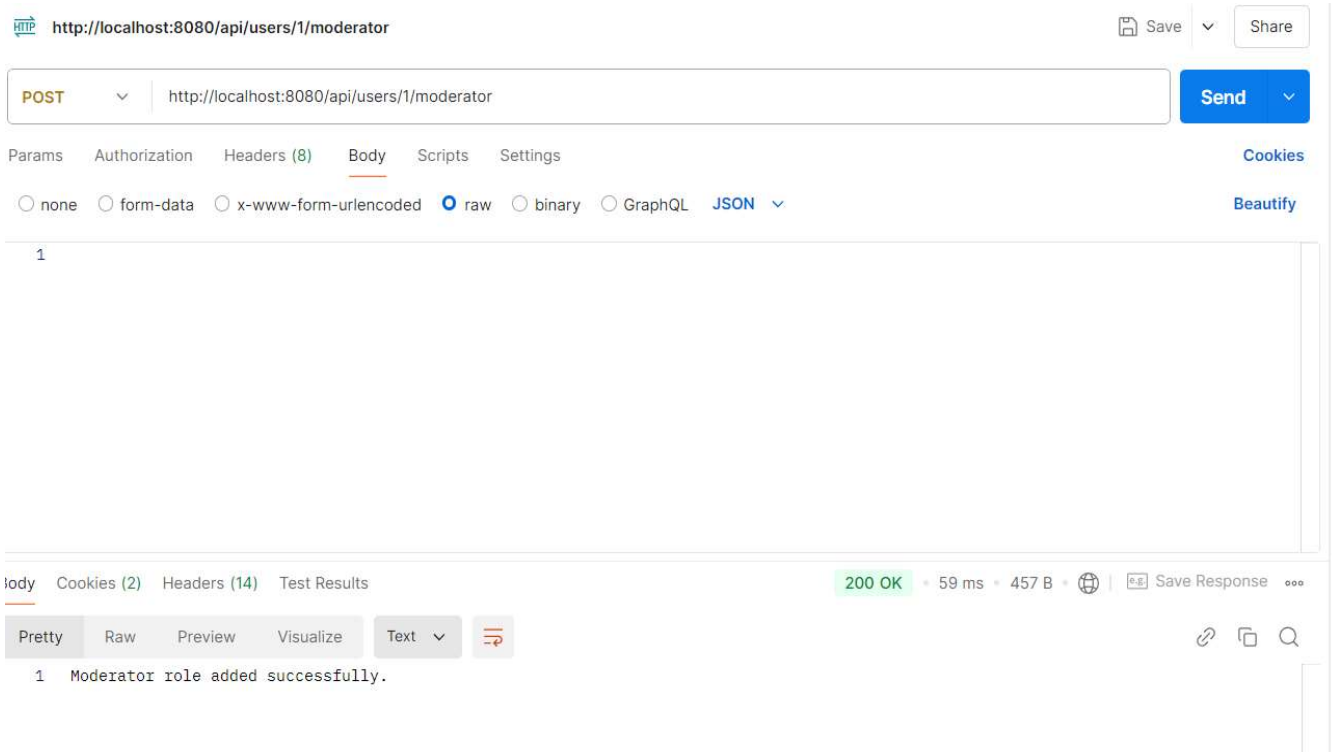
UserController :

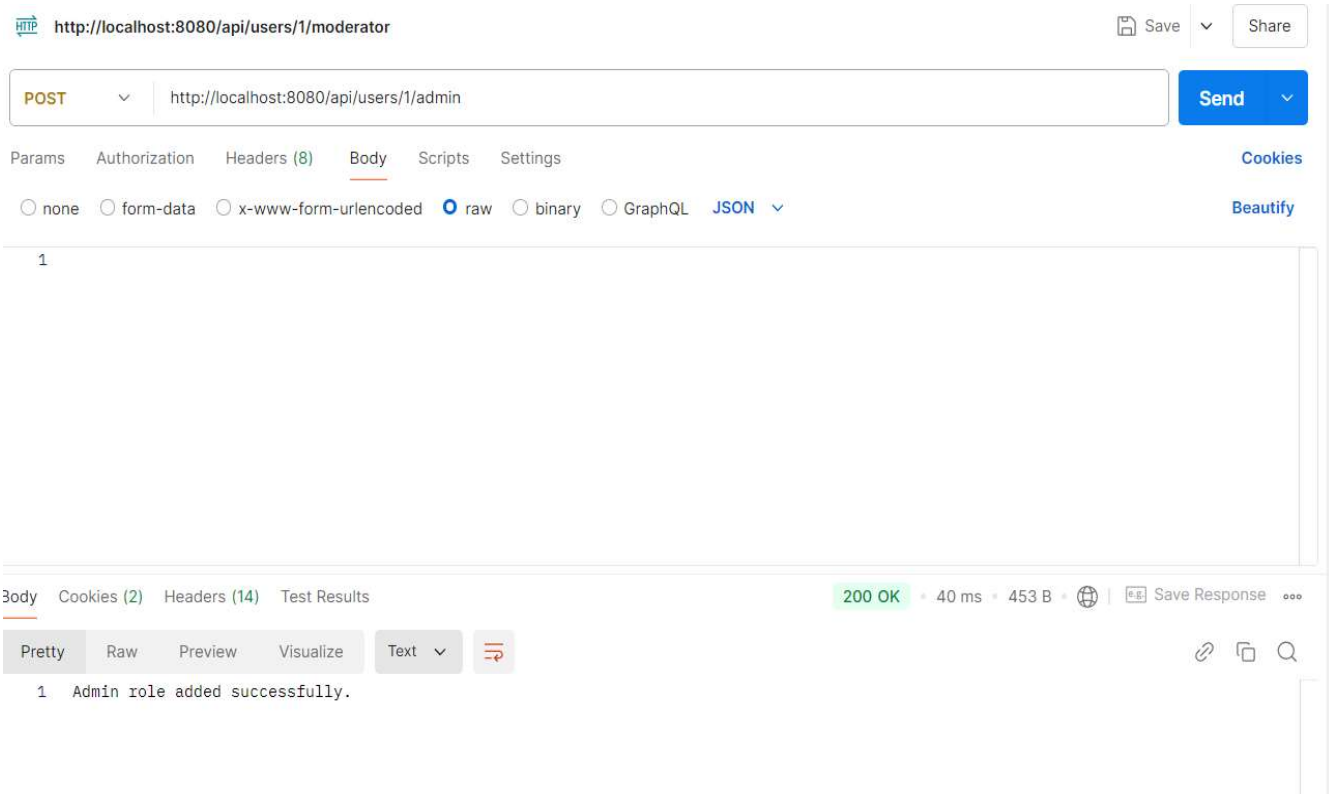


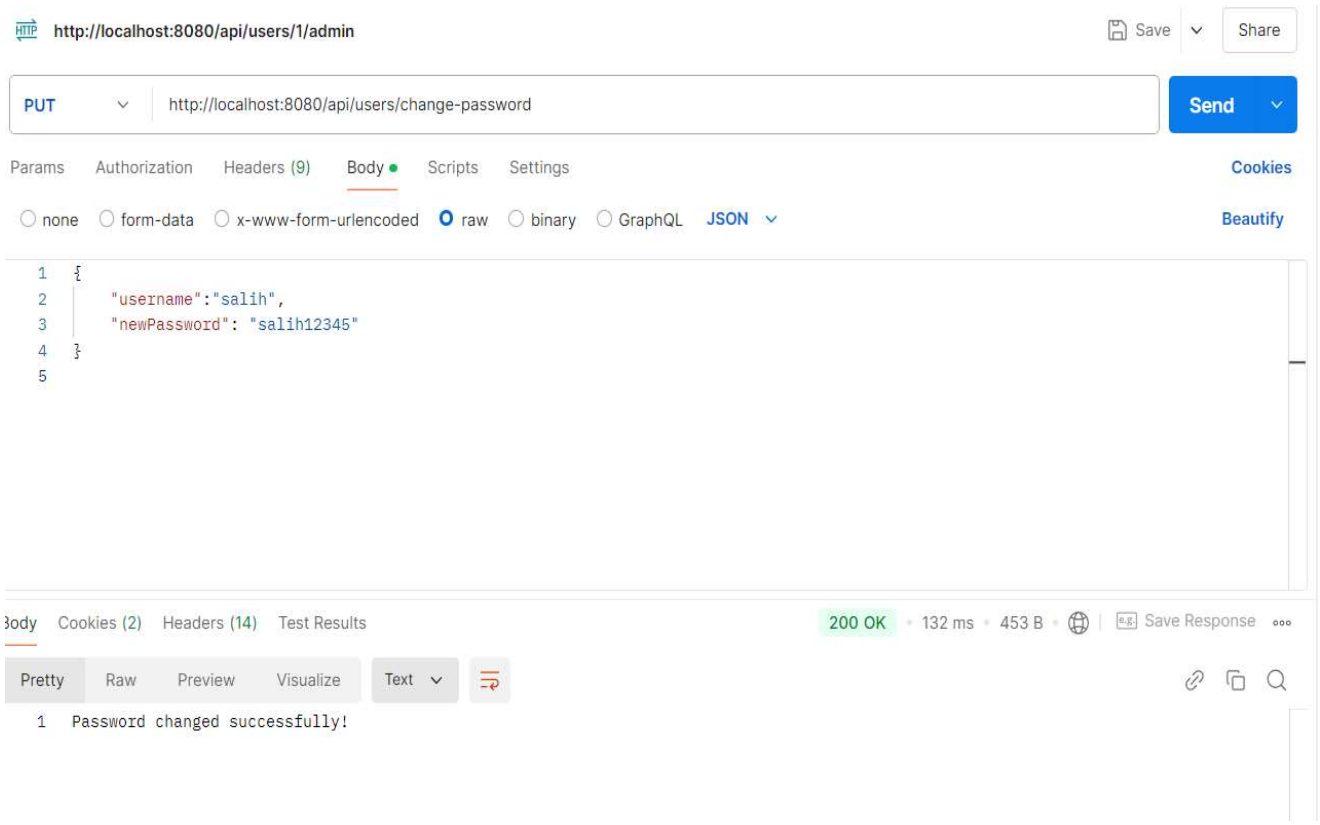



http://localhost:8080/api/users
Save
Share
GET
http://localhost:8080/api/users/2
Send
Params
Authorization
Headers (7)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
1
Body
Cookies (2)
Headers (14)
Test Results
200 OK
40 ms
534 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"id": 2,
"username": "salihAydogdu",
"email": "salih.aydogdu.tech@gmail.com",
"roles": [
{
"id": 1,
"name": "ROLE_USER"
}
]
}


http://localhost:8080/api/users
Save
Share
DELETE
http://localhost:8080/api/users/2
Send
Params
Authorization
Headers (7)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
1
Body
Cookies (2)
Headers (13)
Test Results
200 OK
51 ms
382 B
Save Response
Pretty
Raw
Preview
Visualize
Text
1


http://localhost:8080/api/users/1/moderator
Save
Share
POST
http://localhost:8080/api/users/1/moderator
Send
Params
Authorization
Headers (8)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
1
Body
Cookies (2)
Headers (14)
Test Results
200 OK
59 ms
457 B
Save Response
Pretty
Raw
Preview
Visualize
Text
1 Moderator role added successfully.


http://localhost:8080/api/users/1/moderator
Save
Share
POST
http://localhost:8080/api/users/1/admin
Send
Params
Authorization
Headers (8)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
1
Body
Cookies (2)
Headers (14)
Test Results
200 OK
40 ms
453 B
Save Response
Pretty
Raw
Preview
Visualize
Text
1 Admin role added successfully.



TestController :

This **TestController** class is an example of how to use Spring Security's **@PreAuthorize** annotation. The **@PreAuthorize** annotation provides authorization control at the method or class level, defining access permissions for specific user roles. This class contains four different **GetMapping** methods:

- **allAccess**: Returns content accessible to everyone, with no role control applied.
- **userAccess**: Allows access to users with the roles **hasRole('USER')**, **hasRole('MODERATOR')**, or **hasRole('ADMIN')**. This grants permission to users and higher-level roles (moderator and admin).
- **moderatorAccess**: Only users with **MODERATOR** and **ADMIN** roles can access this content.
- **adminAccess**: Accessible only to users with the **ADMIN** role.

This structure enables role-based access control in Spring Security. The @PreAuthorize annotation ensures that each method's authorization check is performed before the method is called, allowing access only to permitted user roles.

References :

https://github.com/bezkoder/spring-boot-login-example

https://www.youtube.com/watch?v=mq5oUXcAXL4

https://devdocs.io/openjdk~17/