📘 Social Media - Project Report

Title: “Social Media RESTful API using Spring Boot + MongoDB & MySQL”

Name: Dhruvin Kheni GH1037940

University Name: GISMA University of Applied Sciences, Potsdam

Module Name: M605 - Advanced Databases

Professor: Dr. Mazhar Hameed

GitHub Link: https://github.com/Khenidhruvin2001/Social-Media.git

Video Link: https://screenrec.com/share/Kbho57BgqE

Table of Contents

1. Introduction

This project is a **full-stack Social Media RESTful API**, developed using **Spring Boot**, with a hybrid database approach combining **MySQL (SQL)** and **MongoDB (NoSQL)** for optimal flexibility and scalability.

The goal of the application is to simulate core social media functionalities such as user registration, post creation, commenting, liking, following, messaging, and notification systems—all accessible through RESTful endpoints and documented using Swagger UI.

2. System Architecture

The project uses a layered architecture:

1. **REST API Layer**:
   - Controllers (e.g., UserController, PostController) receive and respond to HTTP requests.
2. **Service Layer**:
   - Encapsulates business logic, validations, and processing.
3. **Data Access Layer**:
   - Uses both JpaRepository for SQL entities (Users, Posts, Follows, Tags, etc.)
   - Uses MongoRepository for NoSQL documents (Comments, Likes, Messages, Notifications).

4. **Database Layer**:
    - **MySQL** for structured data (Users, Posts, Tags, Reports, etc.)
    - **MongoDB** for scalable, unstructured social interactions (Comments, Likes, Messages, Notifications).
5. **Swagger UI**:
    - Full API testing and documentation provided at **/swagger-ui/index.html.**

3. Key Components

1. Backend Components

- **Controllers**: Handle HTTP requests for Users, Posts, Comments, Follows, Likes, Notifications, Tags, Messages, Media, Reports, etc.

- **Services**: Business logic including validation and transformation.

- **Repositories**:
    - **MySQL**: JPA Repositories for relational data.
    - **MongoDB**: Spring Data MongoDB repositories for unstructured data.

- **Models**: Java classes and MongoDB documents representing entities like User, Post, Comment, Follow, Notification, etc.

- **Exception Handlers**: Custom exceptions mapped to HTTP status codes using @ResponseStatus

2. Database Design

- **MySQL**: Used for structured entities (Users, Posts, Tags, Reports, PostTag mapping, etc.) with foreign key constraints.
- **MongoDB**: Used for dynamic NoSQL entities (Comments, Likes, Messages, Notifications).

4. API Design

Follows REST conventions:

- **GET** for fetching resources
- **POST** for creating new resources
- **PUT** for updating existing resources

- **DELETE** for removing resources

5. Implementation

1. **Backend Development**

    - **Spring Boot Version**: 3.1+
    - **Java Version**: 21
    - **MongoDB**: Used for Comments, Likes, Notifications, Messages collections.
    - **MySQL**: Used for Users, Posts, Tags, Reports, Media.

2. **Highlights:**

    1. **Entities**:
        - User, Post, Tag, Report, PostTag (MySQL)
        - Comment, Like, Message, Notification (MongoDB)
    2. **Data Flow**:
        - User sends request → Controller → Service → Repository → Database
    3. **Exception Handling**:
        - UserNotFoundException, InvalidInputException, etc. with proper response status

6. API Documentation and Testing

Swagger UI is an interactive API documentation tool that I have incorporated into this project. With the help of Swagger UI, developers can see the structure and functionality of the API through a web-based interface.

- View all endpoints that are accessible along with their HTTP methods.
- Recognize each endpoint's necessary and optional parameters.
- Run API queries straight from the web browser.
- Review the status codes and response formats.
- For testing APIs, this has been a crucial tool.
- After starting the server, we can access this dynamic documentation by going to **http://localhost:8080/swagger-ui/index.html**

Database Integration

This project uses a hybrid database architecture involving both **MySQL** and **MongoDB**:

MySQL Integration (SQL):

Spring Boot with Spring Data JPA handles SQL entities like:

- Users
- Posts
- Tags
- Reports
- Follows
- Media

Each SQL entity is mapped to a table using JPA annotations. Relationships (e.g., OneToMany, ManyToOne) are defined with proper foreign keys.

MongoDB Integration (NoSQL):

Spring Data MongoDB is used for managing NoSQL collections such as:

- Comments
- Likes
- Notifications
- Messages

MongoDB operations are handled using repository interfaces for each collection.

API Development

The REST API is designed using RESTful principles and includes endpoints for both SQL and NoSQL components:

SQL-based Endpoints:

Method	Endpoint	Description
GET	/api/users	Get all users
GET	/api/users/{id}	Get user by ID
POST	/api/users	Create a new user
PUT	/api/users/{id}	Update user
DELETE	/api/users/{id}	Delete user
POST	/api/posts	Create a new post
POST	/api/tags	Create a new tag
POST	/api/post-tags	Add tag to a post
POST	/api/media	Upload media to post
POST	/api/follows/follow	Follow a user
POST	/api/follows/unfollow	Unfollow a user

POST	/api/reports	Report a post or user

MongoDB-based Endpoints:

Method	Endpoint	Description
POST	/api/comments	Add a comment
GET	/api/comments/post/{postId}	Get comments by post
PUT	/api/notifications/read/{id}	Mark notification as read
POST	/api/notifications	Add a notification
GET	/api/notifications/user/{userId}	Get user's notifications
POST	/api/messages	Send a message
GET	/api/messages/user/{userId}	Get user's messages
POST	/api/likes	Add a like
GET	/api/likes/post/{postId}	Get likes for post

Notes:

- All endpoints follow RESTful standards.
- Proper error handling and validation are implemented.
- Swagger UI has been essential in testing all endpoints interactively.
- MongoDB Compass was used for visual inspection and CRUD operations for NoSQL collections

Social Media App - ER Diagram & Data Flow Diagram (DFD)

The following diagram represents the high-level Data Flow Diagram (DFD) for the Social Media project. It illustrates how data moves between components like Controllers, Services, Repositories, and Databases.

- **Data Flow Diagram**

Client

→ UserController

→ UserService

→ UserRepository

→ MySQL

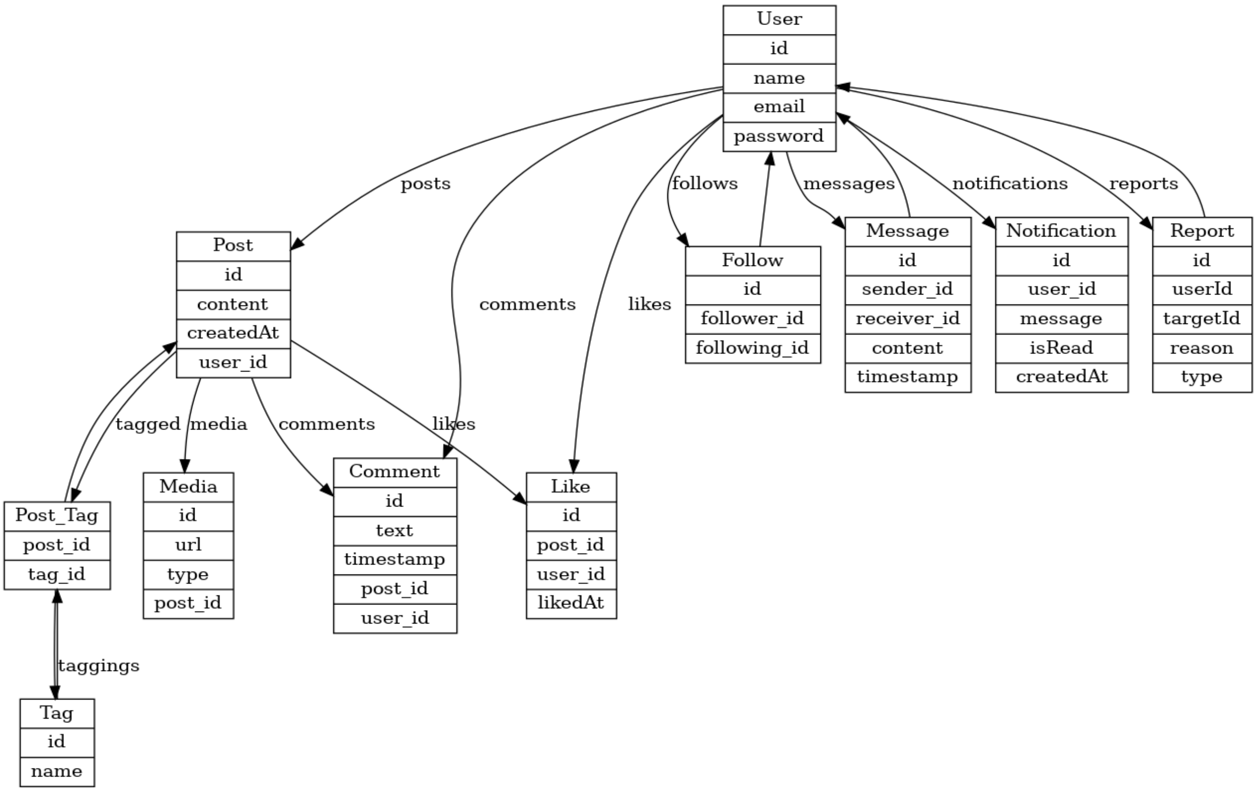
```
→ PostController
      → PostService
            → PostRepository
                  → MySQL
→ CommentController
      → CommentService
            → CommentRepository
                  → MongoDB
→ LikeController
      → LikeService
            → LikeRepository
                  → MongoDB
→ FollowController
      → FollowService
            → FollowRepository
                  → MySQL
→ NotificationController
      → NotificationService
            → NotificationRepository
                  → MongoDB
→ MessageController
      → MessageService
            → MessageRepository
                  → MongoDB
```

Data Flow Diagram (DFD) Overview:

- User interacts with the system via frontend
- User can create, update, delete posts
- User can follow/unfollow others

- Users receive notifications/messages
- Users can like or comment on posts
- Post data is processed and stored in MongoDB
- Notification and Message services operate using NoSQL
- Tagging feature is supported on posts

- **Entity Relationship Diagram (ERD):**
  The following diagram represents the complex ER (Entity Relationship) structure for the Social Media application. It includes all major entities such as User, Post, Comment, Like, Follow, Message, Notification, Media, Tag, Post_Tag, and Report, along with their relationships



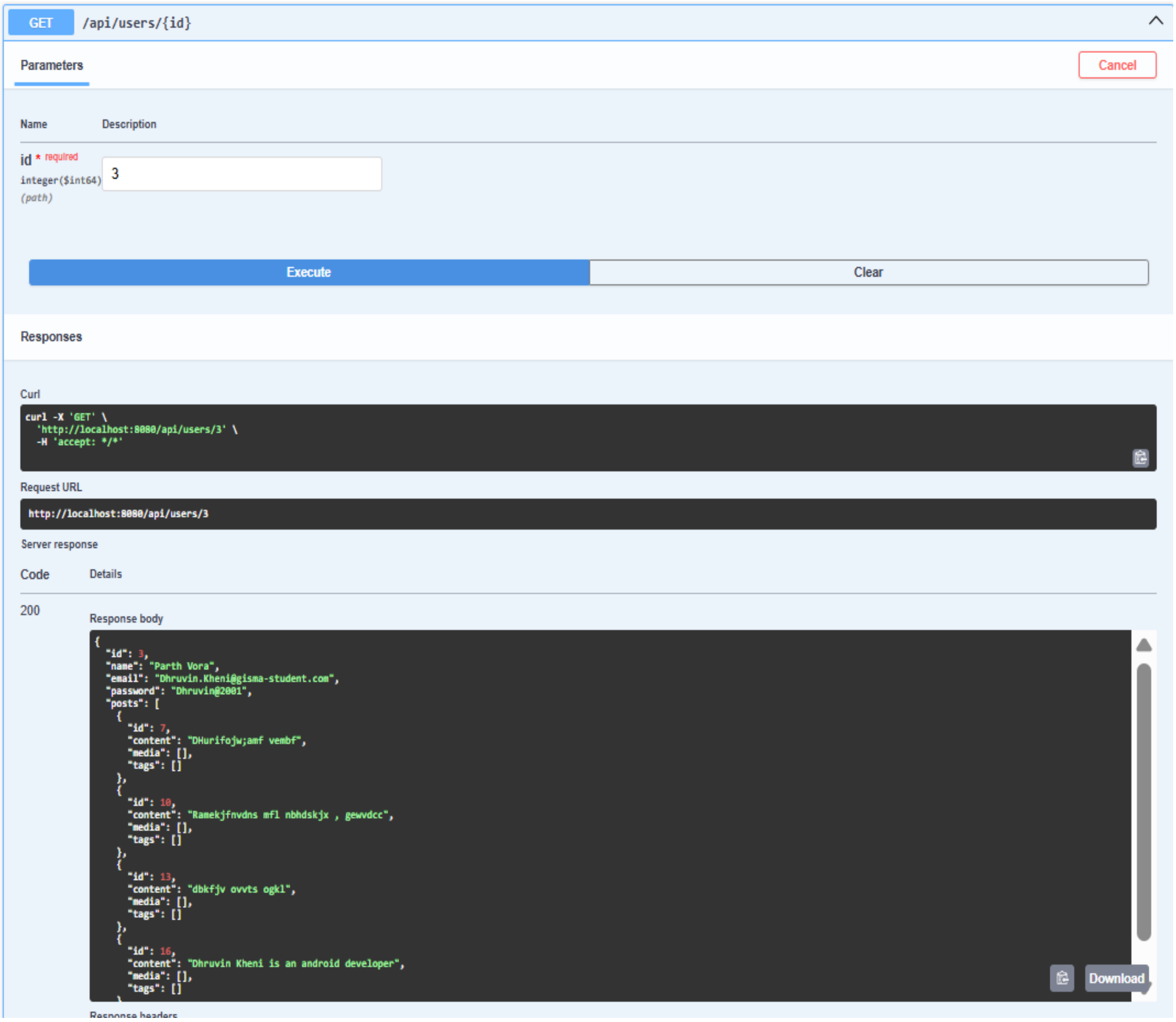
Results

The Social Media Application successfully implements all planned features as demonstrated through comprehensive Swagger UI testing:

1. User Management

- Create, Read, Update, Delete operations tested successfully.

- Users can be registered with posts.



Name	Description
id * required integer($int64) *(path)*	3

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:8080/api/users/3' \
  -H 'accept: */*'
```

Request URL

```
http://localhost:8080/api/users/3
```

Server response

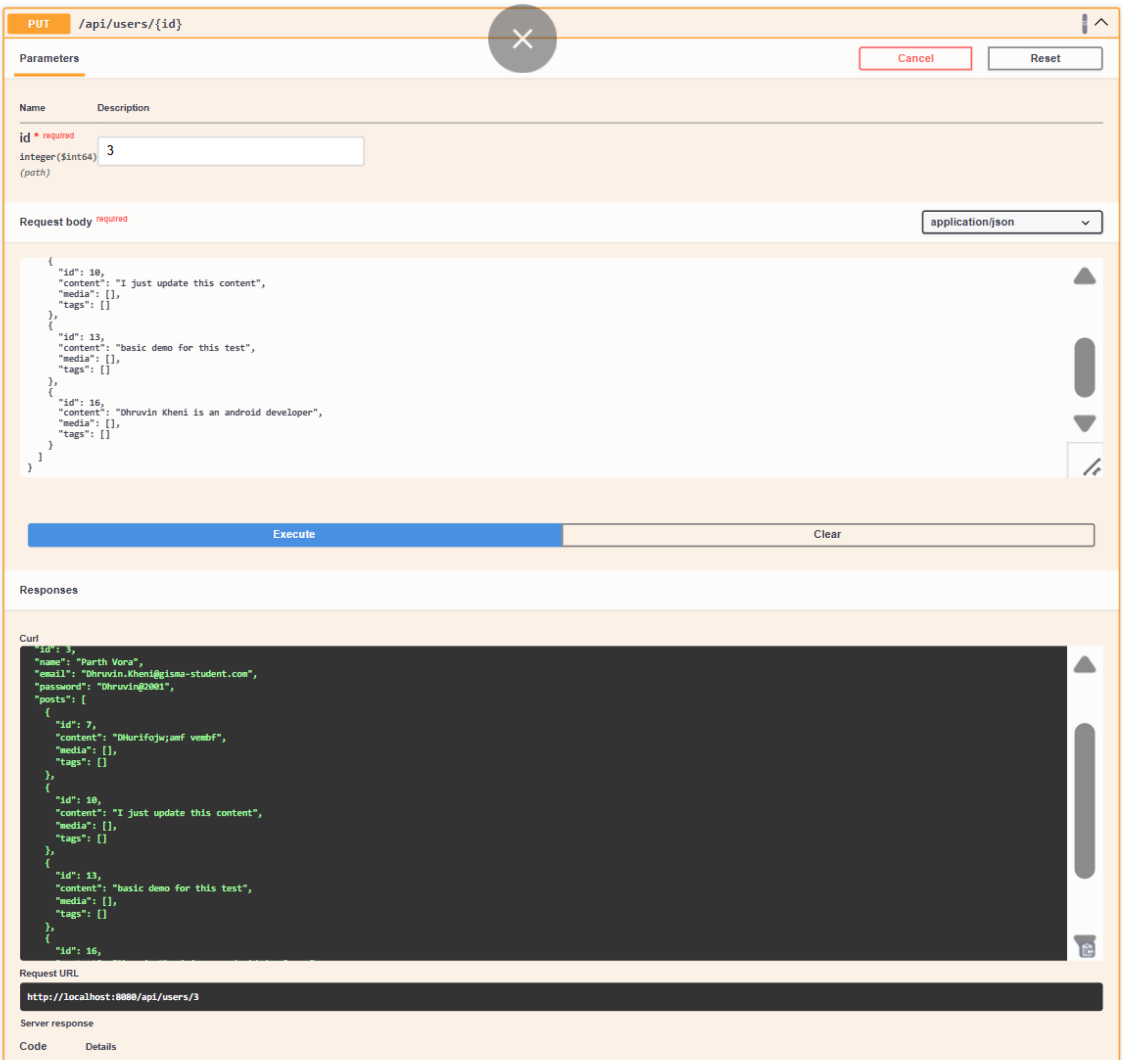
Code Details

200 Response body

```
{
  "id": 3,
  "name": "Parth Vora",
  "email": "Dhruvin.Kheni@gisma-student.com",
  "password": "Dhruvin@2001",
  "posts": [
    {
      "id": 7,
      "content": "DHurifojw;amf vembf",
      "media": [],
      "tags": []
    },
    {
      "id": 10,
      "content": "Ramekjfnvdns mfl nbhdskjx , gewvdcc",
      "media": [],
      "tags": []
    },
    {
      "id": 13,
      "content": "dbkfjv ovvts ogkl",
      "media": [],
      "tags": []
    },
    {
      "id": 16,
      "content": "Dhruvin Kheni is an android developer",
      "media": [],
      "tags": []
```

Response headers

A GET request with ID to show a particular user and their associated posts.

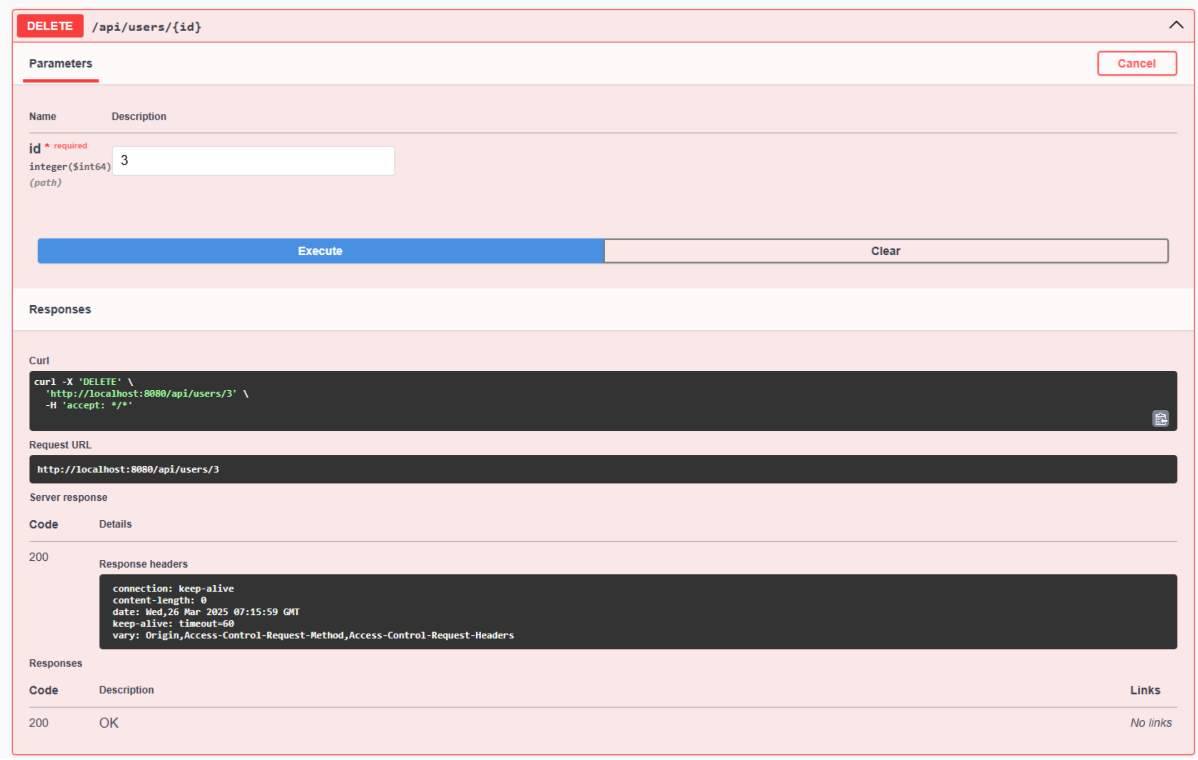


```
    {
      "id": 10,
      "content": "I just update this content",
      "media": [],
      "tags": []
    },
    {
      "id": 13,
      "content": "basic demo for this test",
      "media": [],
      "tags": []
    },
    {
      "id": 16,
      "content": "Dhruvin Kheni is an android developer",
      "media": [],
      "tags": []
    }
  ]
}
```

```
  "id": 3,
  "name": "Parth Vora",
  "email": "Dhruvin.Kheni@gisma-student.com",
  "password": "Dhruvin@2001",
  "posts": [
    {
      "id": 7,
      "content": "DHurifojw;amf vembf",
      "media": [],
      "tags": []
    },
    {
      "id": 10,
      "content": "I just update this content",
      "media": [],
      "tags": []
    },
    {
      "id": 13,
      "content": "basic demo for this test",
      "media": [],
      "tags": []
    },
    {
      "id": 16,
```

```
http://localhost:8080/api/users/3
```

A PUT request with ID to update a particular user's details and their posts.



id * required
integer($int64)
(path)
3

Responses

Curl

```
curl -X 'DELETE' \
  'http://localhost:8080/api/users/3' \
  -H 'accept: */*'
```

Request URL

```
http://localhost:8080/api/users/3
```

Server response

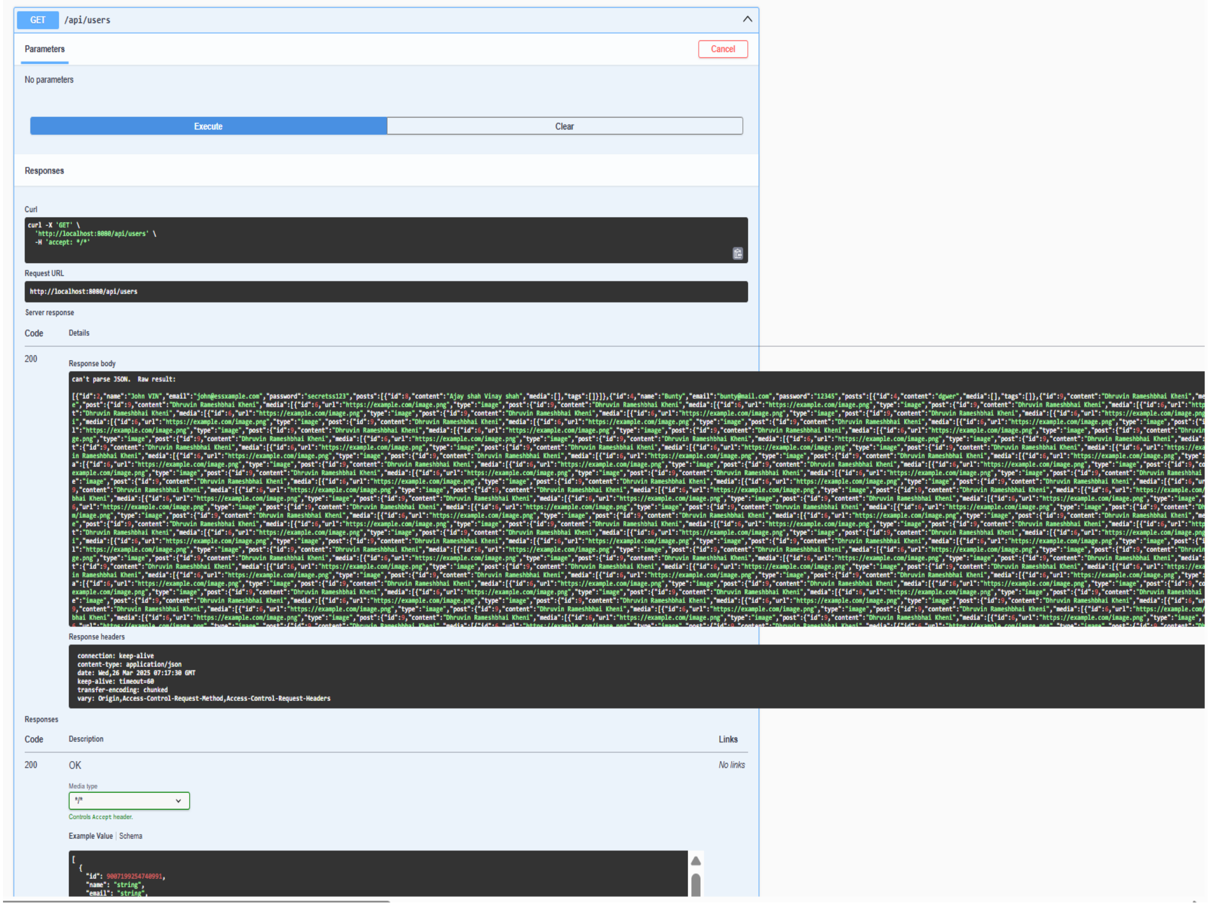
Code	Details
200	Response headers

```
connection: keep-alive
content-length: 0
date: Wed,26 Mar 2025 07:15:59 GMT
keep-alive: timeout=60
vary: Origin,Access-Control-Request-Method,Access-Control-Request-Headers
```

Responses

Code	Description	Links
200	OK	*No links*

A DELETE request with Id to remove a particular user after handling foreign key constraints



Responses

Curl

```
curl -X 'GET' \
  'http://localhost:8080/api/users' \
  -H 'accept: */*'
```

Request URL

```
http://localhost:8080/api/users
```

Server response

Code	Details
200	Response body

```
can't parse JSON.  Raw result:
```

[{"id":2,"name":"John VIN","email":"john@essxample.com","password":"secretss123","posts":[{"id":8,"content":"Ajay shah Vinay shah","media":[],"tags":[]}]},{"id":4,"name":"Bunty","email":"bunty@mail.com","password":"12345","posts":[{"id":4,"content":"dgwer","media":[],"tags":[]},{"id":9,"content":"Dhruvin Rameshbhai Kheni","me
e","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"http
t":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png

Response headers

```
connection: keep-alive
content-type: application/json
date: Wed,26 Mar 2025 07:17:30 GMT
keep-alive: timeout=60
transfer-encoding: chunked
vary: Origin,Access-Control-Request-Method,Access-Control-Request-Headers
```

Responses

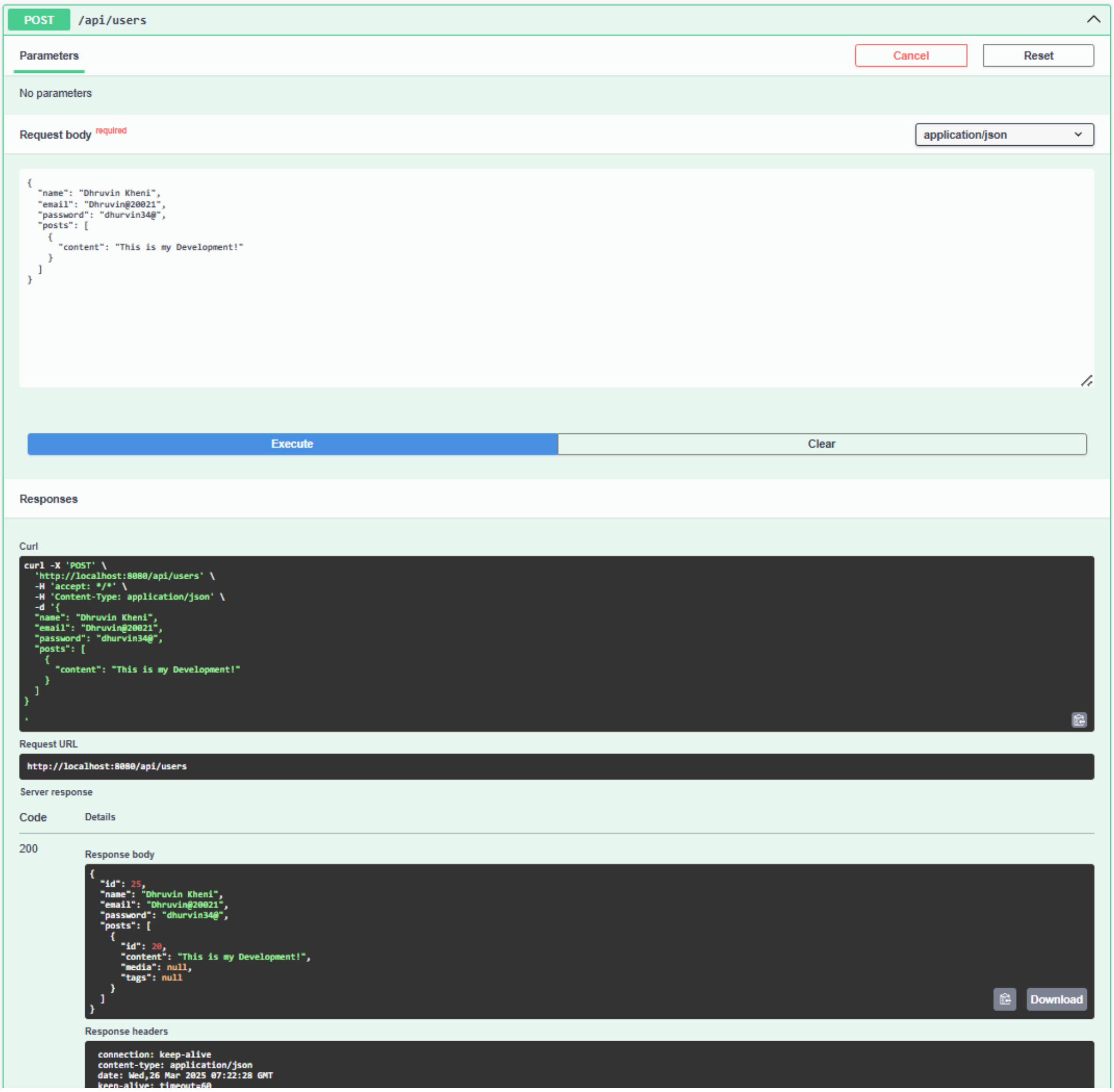
Code	Description	Links
200	OK	*No links*

Media type
/
Controls Accept header.

Example Value | Schema

```
[
  {
    "id": 9007199254740991,
    "name": "string",
    "email": "string",
```

A GET request to fetch all users along with their associated posts, media, and tags



```
{
  "name": "Dhruvin Kheni",
  "email": "Dhruvin@20021",
  "password": "dhurvin34@",
  "posts": [
    {
      "content": "This is my Development!"
    }
  ]
}
```

Curl

```
curl -X 'POST' \
  'http://localhost:8080/api/users' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "name": "Dhruvin Kheni",
  "email": "Dhruvin@20021",
  "password": "dhurvin34@",
  "posts": [
    {
      "content": "This is my Development!"
    }
  ]
}
'
```

Request URL

```
http://localhost:8080/api/users
```

Server response

Code	Details
200	Response body

```
{
  "id": 25,
  "name": "Dhruvin Kheni",
  "email": "Dhruvin@20021",
  "password": "dhurvin34@",
  "posts": [
    {
      "id": 20,
      "content": "This is my Development!",
      "media": null,
      "tags": null
    }
  ]
}
```

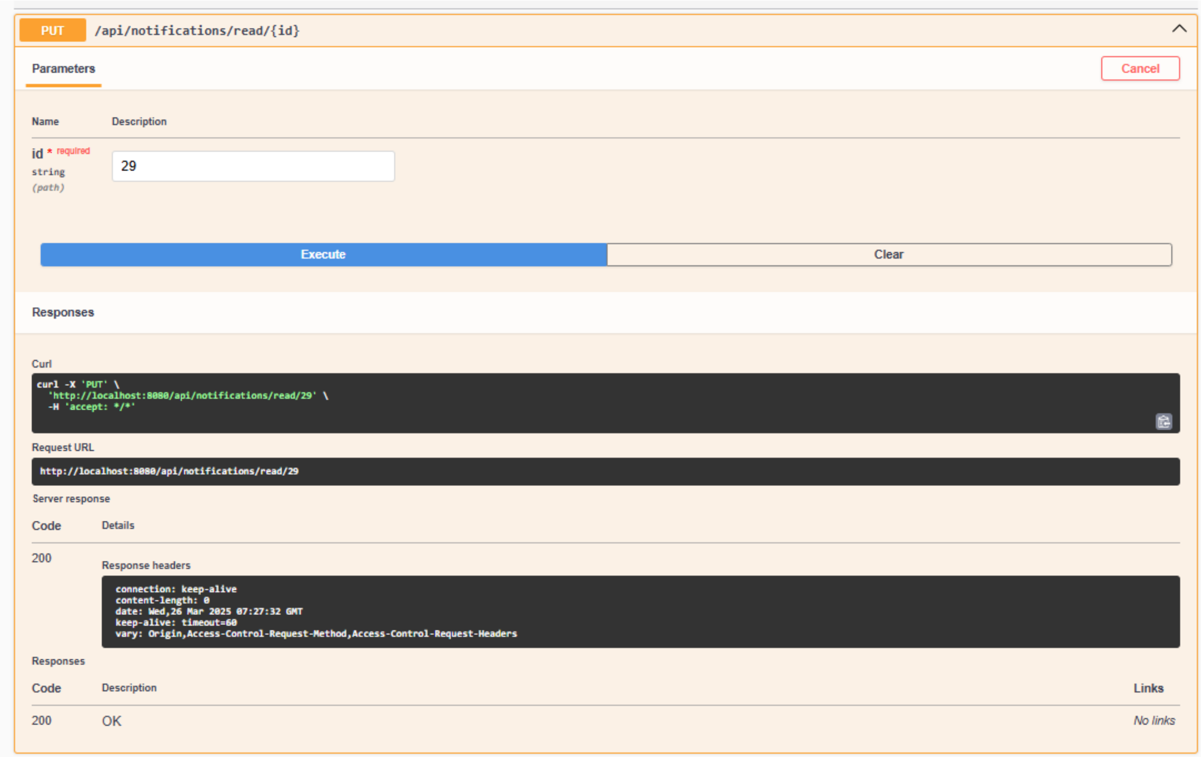
Response headers

```
connection: keep-alive
content-type: application/json
date: Wed,26 Mar 2025 07:22:28 GMT
keep-alive: timeout=60
```

A POST request to create a new user along with an associated post.

2. Notification Management

- Users can mark notifications as read via PUT /api/notifications/read/{id}.
- Notifications are created using POST /api/notifications.
- CRUD operations for notifications were tested and worked as expected.
- Proper validation ensures that only valid notification IDs are processed



Name	Description
id * required string *(path)*	29

Responses

Curl

```
curl -X 'PUT' \
  'http://localhost:8080/api/notifications/read/29' \
  -H 'accept: */*'
```

Request URL

```
http://localhost:8080/api/notifications/read/29
```

Server response

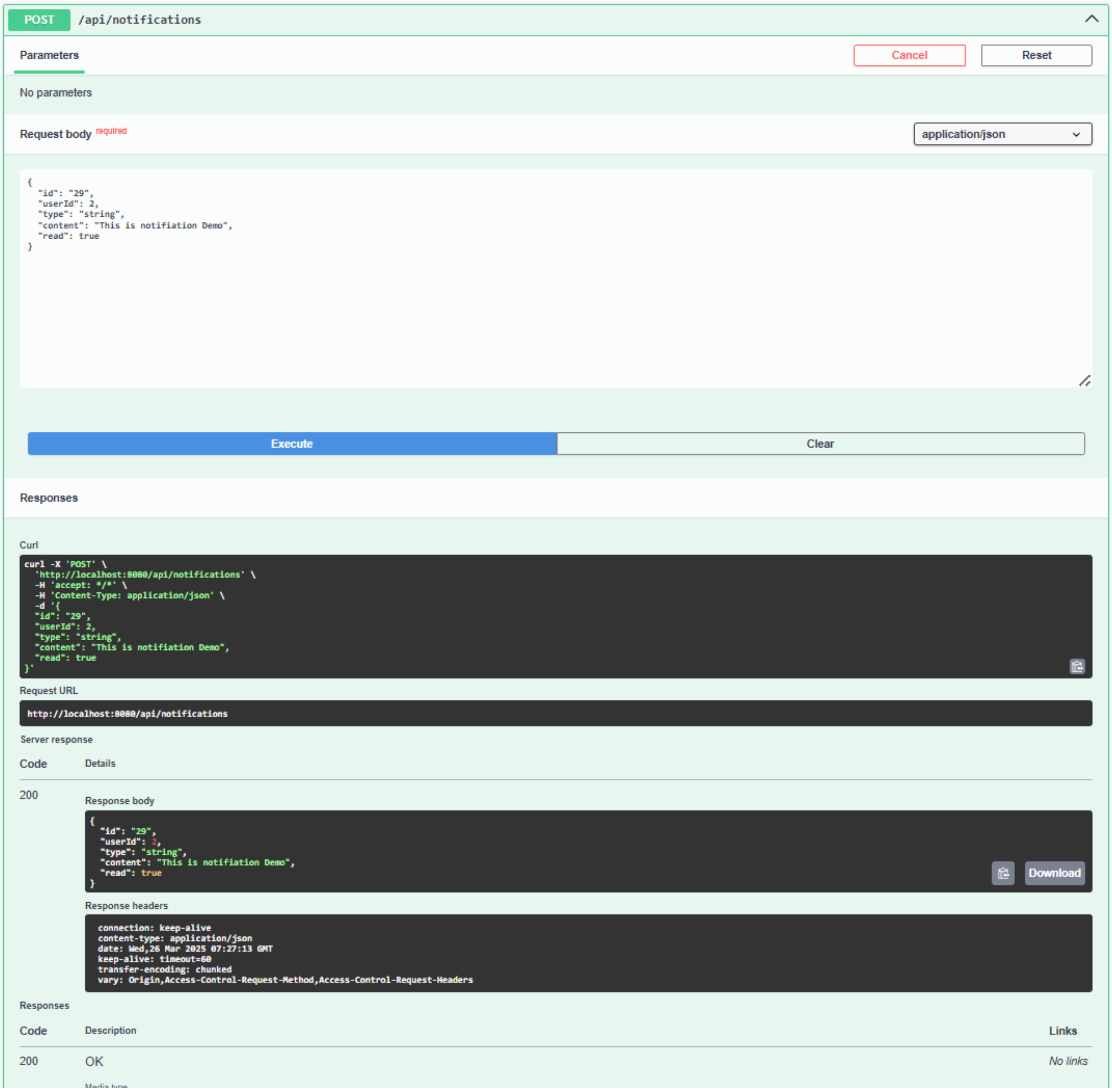
Code	Details
200	Response headers

```
connection: keep-alive
content-length: 0
date: Wed,26 Mar 2025 07:27:32 GMT
keep-alive: timeout=60
vary: Origin,Access-Control-Request-Method,Access-Control-Request-Headers
```

Responses

Code	Description	Links
200	OK	*No links*

A PUT request to mark a specific notification (ID: 29) as read.



```
{
  "id": "29",
  "userId": 2,
  "type": "string",
  "content": "This is notifiation Demo",
  "read": true
}
```

Curl

```
curl -X 'POST' \
  'http://localhost:8080/api/notifications' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "id": "29",
  "userId": 2,
  "type": "string",
  "content": "This is notifiation Demo",
  "read": true
}'
```

Request URL

```
http://localhost:8080/api/notifications
```

Server response

Code Details

200 Response body

```
{
  "id": "29",
  "userId": 2,
  "type": "string",
  "content": "This is notifiation Demo",
  "read": true
}
```

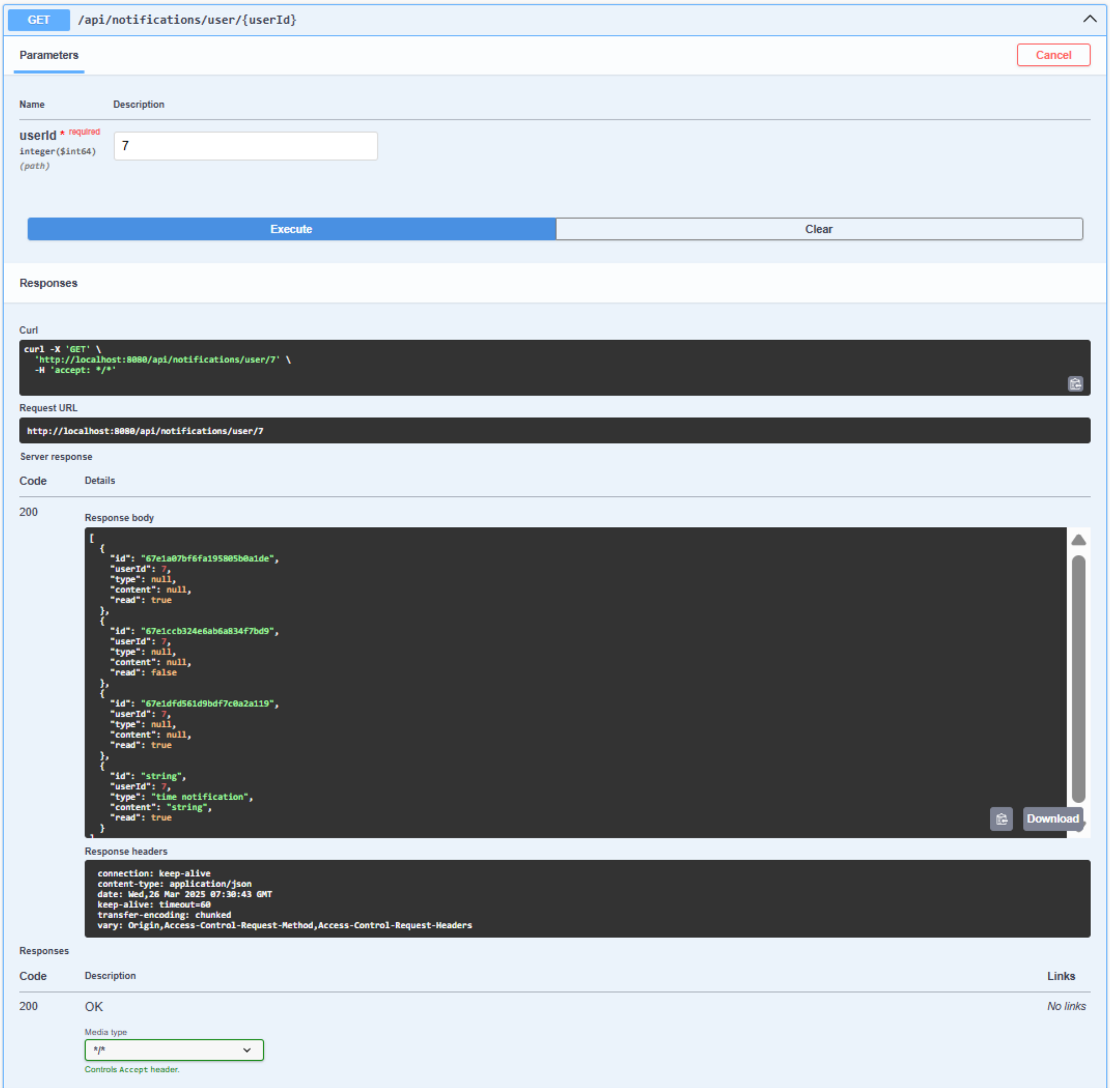
Response headers

```
 connection: keep-alive
 content-type: application/json
 date: Wed,26 Mar 2025 07:27:13 GMT
 keep-alive: timeout=60
 transfer-encoding: chunked
 vary: Origin,Access-Control-Request-Method,Access-Control-Request-Headers
```

Responses

Code	Description	Links
200	OK	*No links*

A POST request to create a new notification for user ID 2 with the content "This is notification Demo".



Name	Description
userId * required integer($int64) *(path)*	7

Curl

```
curl -X 'GET' \
  'http://localhost:8080/api/notifications/user/7' \
  -H 'accept: */*'
```

Request URL

```
http://localhost:8080/api/notifications/user/7
```

Server response

Code Details

200

Response body

```
[
  {
    "id": "67e1a07bf6fa195805b0a1de",
    "userId": 7,
    "type": null,
    "content": null,
    "read": true
  },
  {
    "id": "67e1ccb324e6ab6a834f7bd9",
    "userId": 7,
    "type": null,
    "content": null,
    "read": false
  },
  {
    "id": "67e1dfd561d9bdf7c0a2a119",
    "userId": 7,
    "type": null,
    "content": null,
    "read": true
  },
  {
    "id": "string",
    "userId": 7,
    "type": "time notification",
    "content": "string",
    "read": true
  }
]
```

Response headers

```
connection: keep-alive
content-type: application/json
date: Wed,26 Mar 2025 07:30:43 GMT
keep-alive: timeout=60
transfer-encoding: chunked
vary: Origin,Access-Control-Request-Method,Access-Control-Request-Headers
```

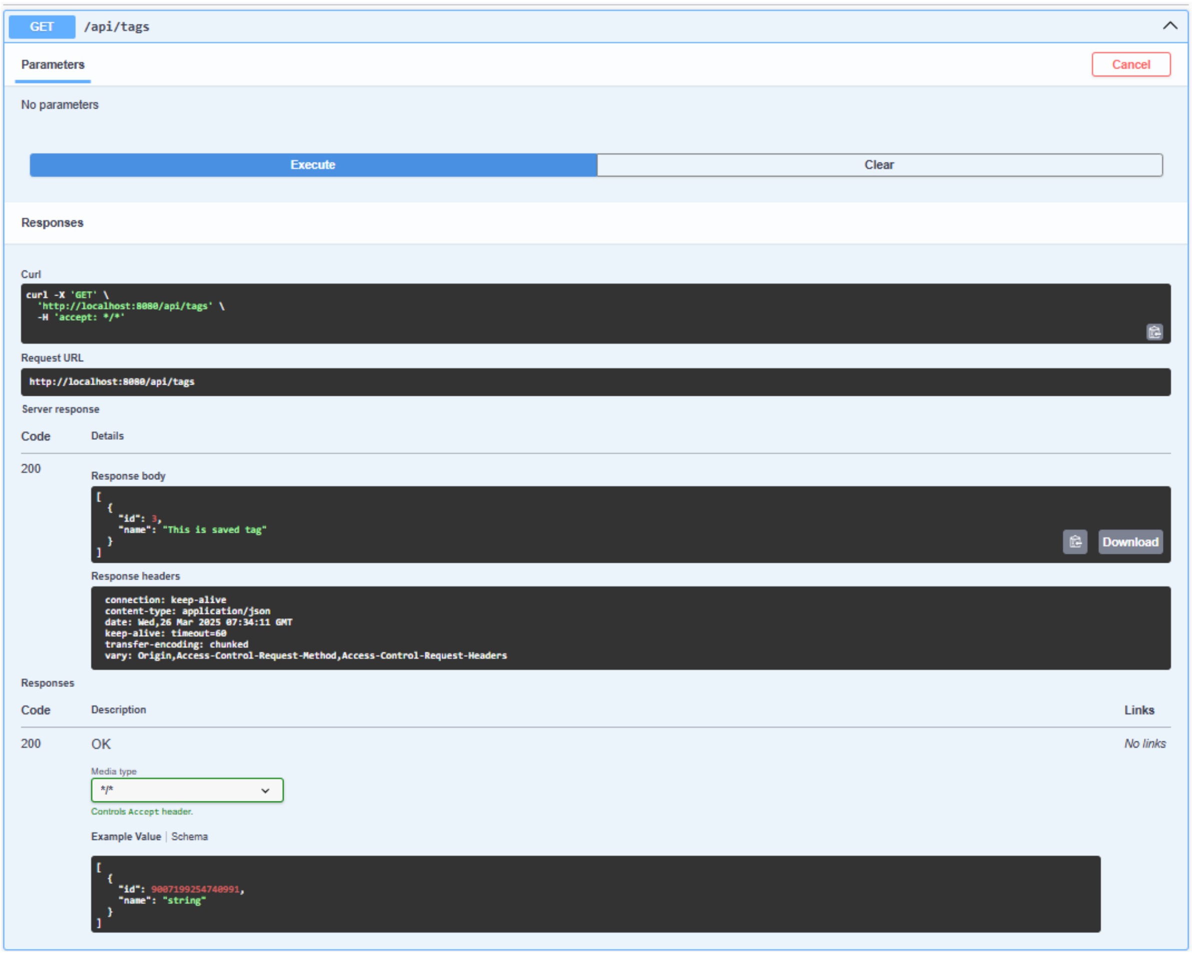
Responses

Code	Description	Links
200	OK Media type */* Controls Accept header.	*No links*

A GET request to fetch all notifications for a specific user by userId (here, userId = 7).

3. Tag Management

- Tags can be created and assigned to posts.
- All tag-related operations like Create and Fetch tested successfully



No parameters

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:8080/api/tags' \
  -H 'accept: */*'
```

Request URL

```
http://localhost:8080/api/tags
```

Server response

Code | Details

200

Response body

```
[
  {
    "id": 3,
    "name": "This is saved tag"
  }
]
```

Response headers

```
 connection: keep-alive
 content-type: application/json
 date: Wed,26 Mar 2025 07:34:11 GMT
 keep-alive: timeout=60
 transfer-encoding: chunked
 vary: Origin,Access-Control-Request-Method,Access-Control-Request-Headers
```

Responses

Code	Description	Links
200	OK	*No links*

Media type

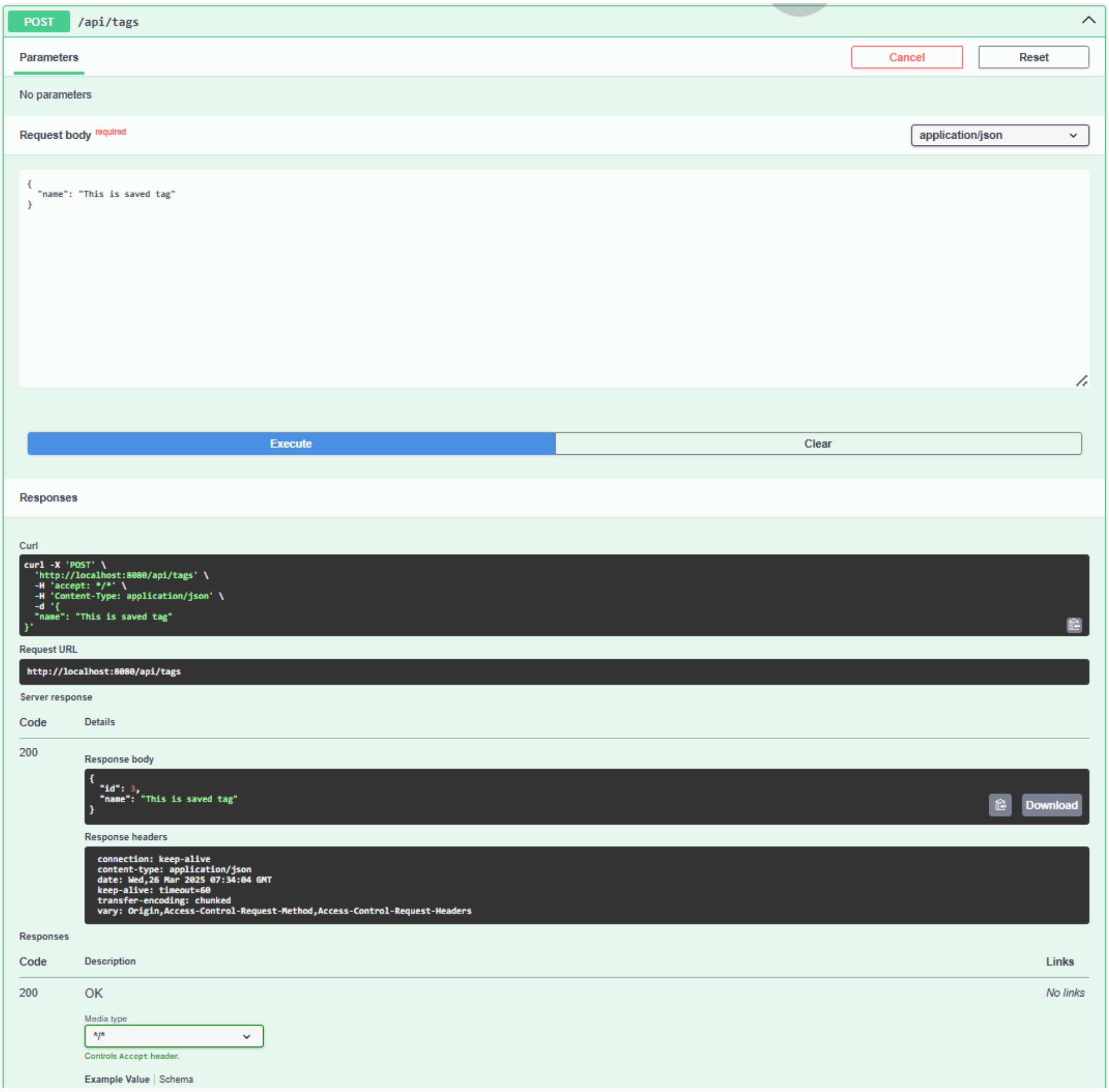
/

Controls Accept header.

Example Value | Schema

```
[
  {
    "id": 9007199254740991,
    "name": "string"
  }
]
```

A GET request to fetch all tags saved in the database.



No parameters

Request body required

application/json

```
{
  "name": "This is saved tag"
}
```

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:8080/api/tags' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "name": "This is saved tag"
}'
```

Request URL

```
http://localhost:8080/api/tags
```

Server response

Code | Details

200

Response body

```
{
  "id": 3,
  "name": "This is saved tag"
}
```

Response headers

```
connection: keep-alive
content-type: application/json
date: Wed,26 Mar 2025 07:34:04 GMT
keep-alive: timeout=60
transfer-encoding: chunked
vary: Origin,Access-Control-Request-Method,Access-Control-Request-Headers
```

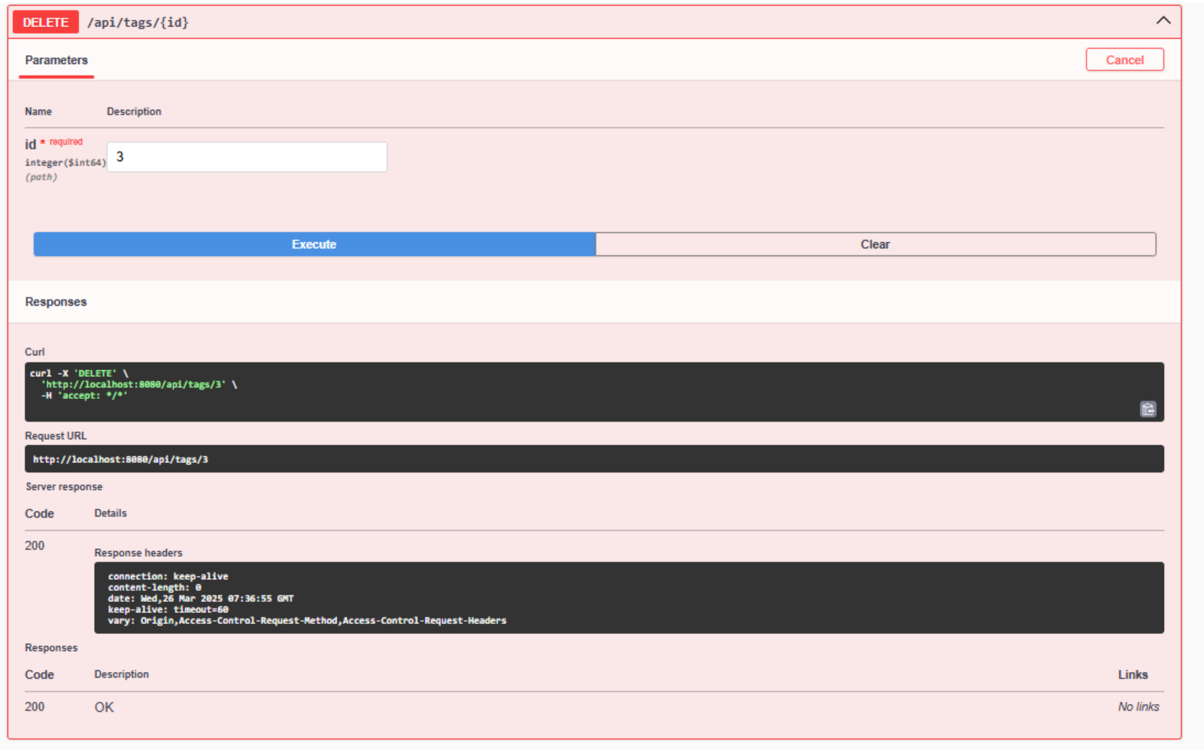
Responses

Code	Description	Links
200	OK	No links

Media type: */*

Controls Accept header.

A POST request to create a new tag with the name "This is saved tag".



Name	Description
id * required integer($int64) *(path)*	3

Responses

Curl

```
curl -X 'DELETE' \
  'http://localhost:8080/api/tags/3' \
  -H 'accept: */*'
```

Request URL

```
http://localhost:8080/api/tags/3
```

Server response

Code	Details
200	Response headers

```
connection: keep-alive
content-length: 0
date: Wed,26 Mar 2025 07:36:55 GMT
keep-alive: timeout=60
vary: Origin,Access-Control-Request-Method,Access-Control-Request-Headers
```

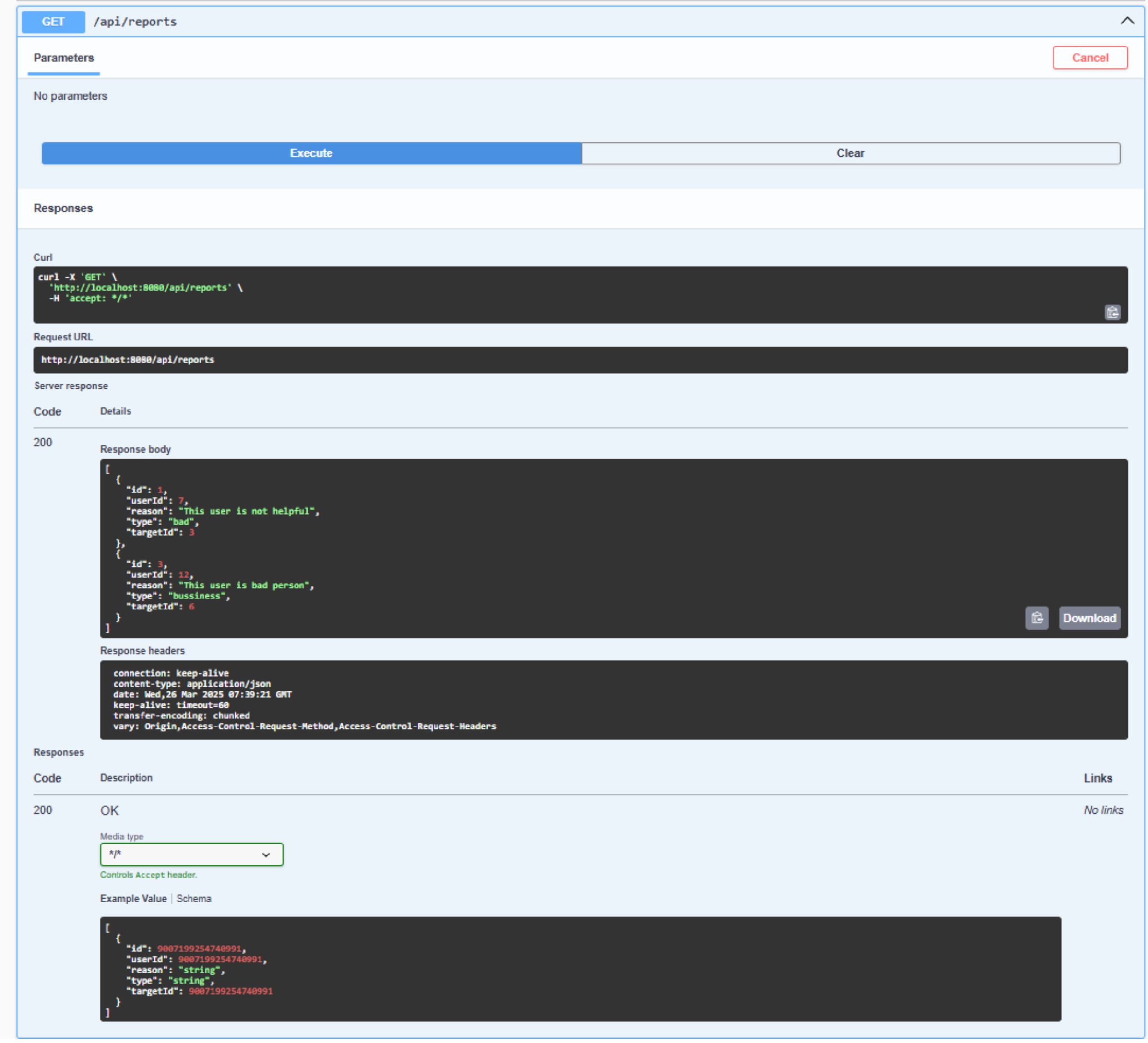
Responses

Code	Description	Links
200	OK	*No links*

A DELETE request to remove a tag by its ID (here, tag ID = 3).

4. Report Management

- Reports can be created to flag users or content.
- Validations ensure only existing users and valid targets can be reported.
- All report-related operations like Create tested successfully.



No parameters

Curl

```
curl -X 'GET' \
  'http://localhost:8080/api/reports' \
  -H 'accept: */*'
```

Request URL

```
http://localhost:8080/api/reports
```

Server response

Code	Details
200	Response body

```
[
  {
    "id": 1,
    "userId": 7,
    "reason": "This user is not helpful",
    "type": "bad",
    "targetId": 3
  },
  {
    "id": 3,
    "userId": 12,
    "reason": "This user is bad person",
    "type": "bussiness",
    "targetId": 6
  }
]
```

Response headers

```
 connection: keep-alive
 content-type: application/json
 date: Wed,26 Mar 2025 07:39:21 GMT
 keep-alive: timeout=60
 transfer-encoding: chunked
 vary: Origin,Access-Control-Request-Method,Access-Control-Request-Headers
```

Responses

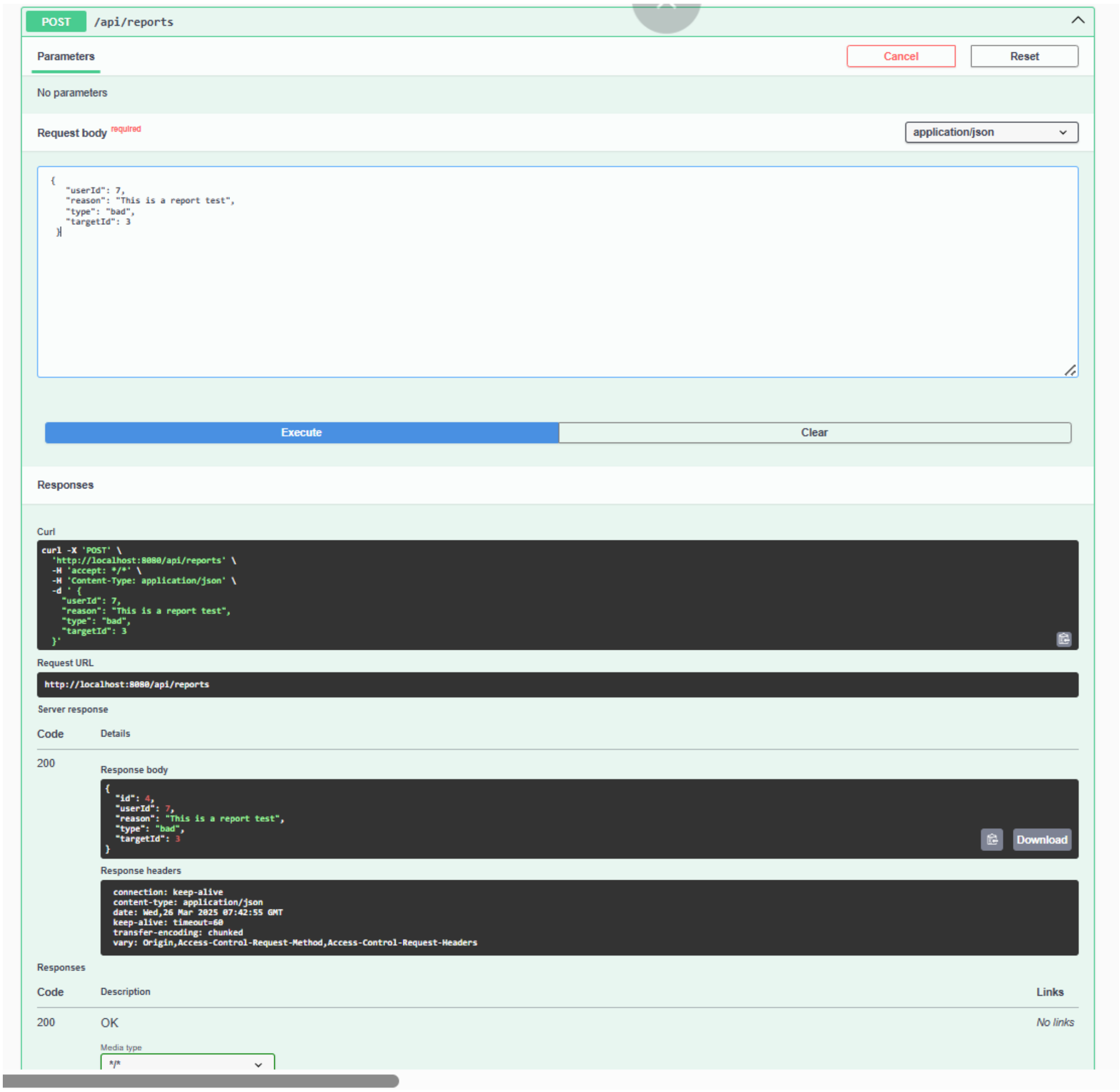
Code	Description	Links
200	OK	*No links*

Media type: */*

Controls Accept header.

```
[
  {
    "id": 9007199254740991,
    "userId": 9007199254740991,
    "reason": "string",
    "type": "string",
    "targetId": 9007199254740991
  }
]
```

A **GET** request to fetch all reports submitted in the system.



No parameters

```
{
   "userId": 7,
   "reason": "This is a report test",
   "type": "bad",
   "targetId": 3
 }
```

Curl

```
curl -X 'POST' \
  'http://localhost:8080/api/reports' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d ' {
    "userId": 7,
    "reason": "This is a report test",
    "type": "bad",
    "targetId": 3
  }'
```

Request URL

```
http://localhost:8080/api/reports
```

Server response

Code	Details
200	Response body

```
{
  "id": 4,
  "userId": 7,
  "reason": "This is a report test",
  "type": "bad",
  "targetId": 3
}
```

Response headers

```
 connection: keep-alive
 content-type: application/json
 date: Wed,26 Mar 2025 07:42:55 GMT
 keep-alive: timeout=60
 transfer-encoding: chunked
 vary: Origin,Access-Control-Request-Method,Access-Control-Request-Headers
```

Responses

Code	Description	Links
200	OK Media type */*	*No links*

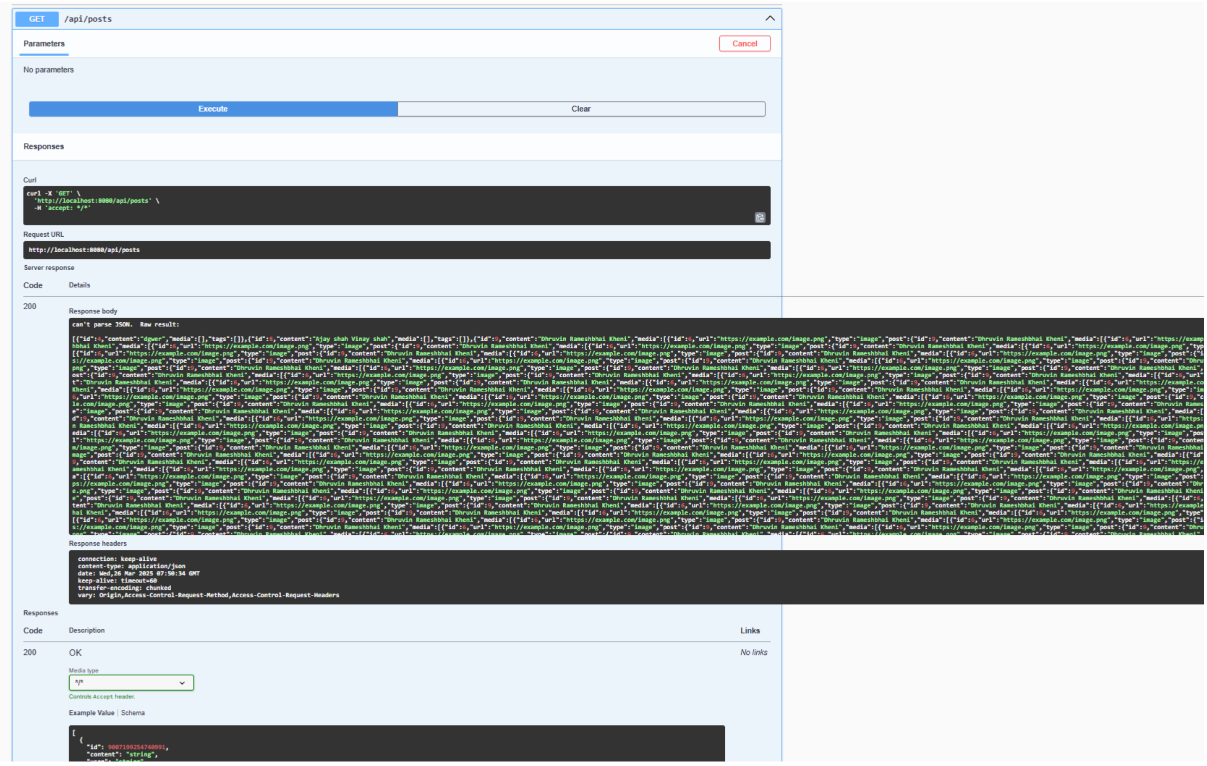
A **POST** request to create a new report with details such as userId, type, reason, and targetId.

Delete Report ID

- Report deletion functionality tested successfully.
- A DELETE request is made by specifying the report ID (e.g., /api/reports/4).
- On successful deletion, the server returns **200 OK**.
- Works similarly to tag deletion—just pass the ID and the report is removed.

5. Post Management

- Posts can be created with optional media and tags.
- Each post is linked to a user (userId required).
- Validations ensure only registered users can create posts.
- All post-related operations like Create and Fetch were tested successfully.



No parameters

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:8080/api/posts' \
  -H 'accept: */*'
```

Request URL

```
http://localhost:8080/api/posts
```

Server response

Code Details

200 Response body

```
can't parse JSON.  Raw result:

[{"id":4,"content":"dgwer","media":[],"tags":[]},{"id":8,"content":"Ajay shah Vinay shah","media":[],"tags":[]},{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://examp
hbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","typ
[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"i
s://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhru
png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni",
ost":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url"
t":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.co
Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"im
6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"c
le.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rames
e":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[
d":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https:
n Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.pn
edia":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","pos
l":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"conten
m/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai
mage","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id"
9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://e
ameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png",
a":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":
ps://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhr
e.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni
e","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,
tent":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example
hai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type"
[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"i
s://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhruvin Rameshbhai Kheni","media":[{"id":6,"url":"https://example.com/image.png","type":"image","post":{"id":9,"content":"Dhru
```

Response headers

```
connection: keep-alive
content-type: application/json
date: Wed,26 Mar 2025 07:50:34 GMT
keep-alive: timeout=60
transfer-encoding: chunked
vary: Origin,Access-Control-Request-Method,Access-Control-Request-Headers
```

Responses

Code	Description	Links
200	OK	*No links*

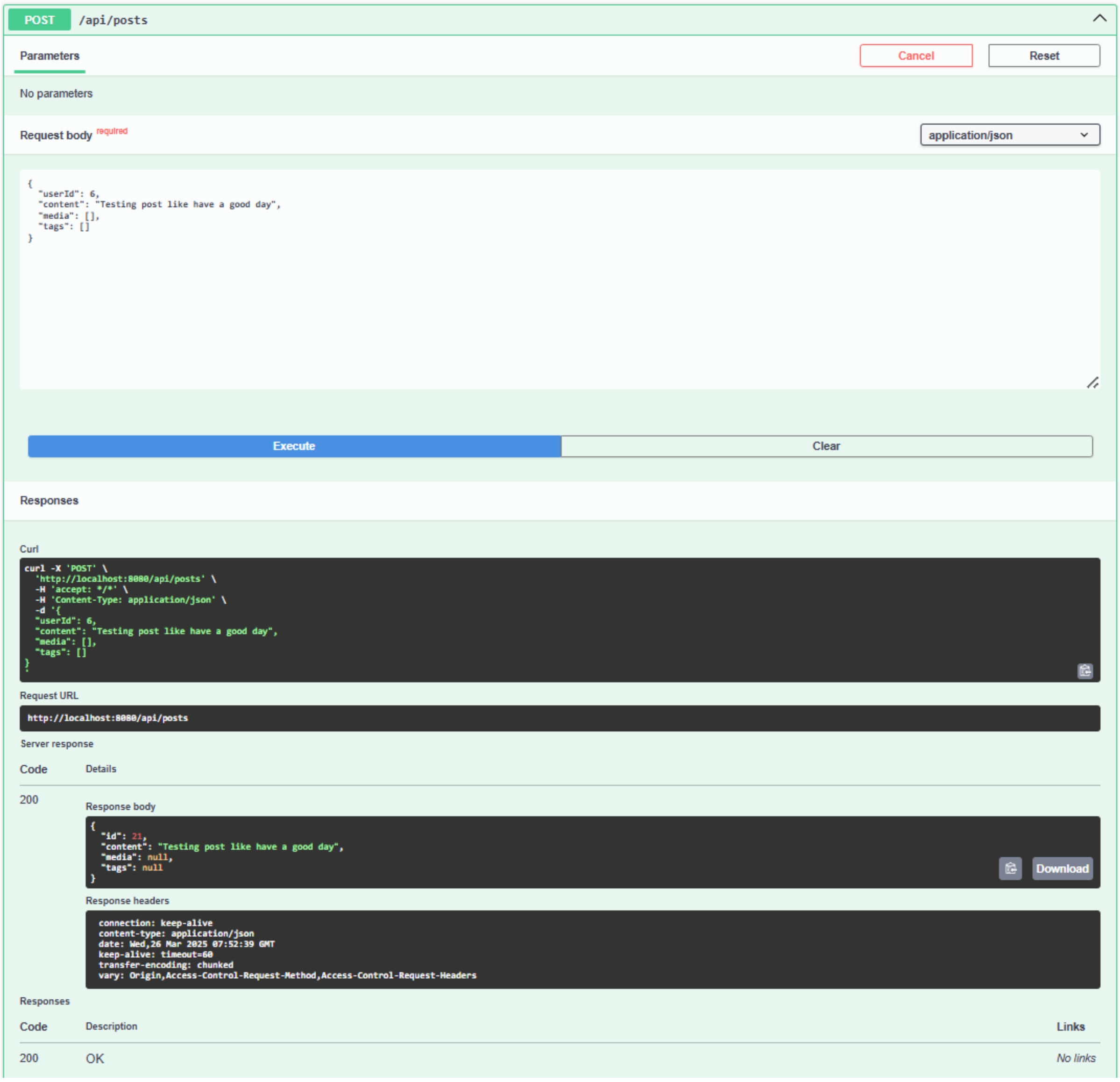
Media type: */*

Controls Accept header.

Example Value | Schema

```
[
  {
    "id": 9007199254740991,
    "content": "string",
```

A **GET** request to fetch all posts with their associated media and tags.



```
{
  "userId": 6,
  "content": "Testing post like have a good day",
  "media": [],
  "tags": []
}
```

Curl

```
curl -X 'POST' \
  'http://localhost:8080/api/posts' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "userId": 6,
  "content": "Testing post like have a good day",
  "media": [],
  "tags": []
}
'
```

Request URL

```
http://localhost:8080/api/posts
```

Server response

Code Details

200

Response body

```
{
  "id": 21,
  "content": "Testing post like have a good day",
  "media": null,
  "tags": null
}
```

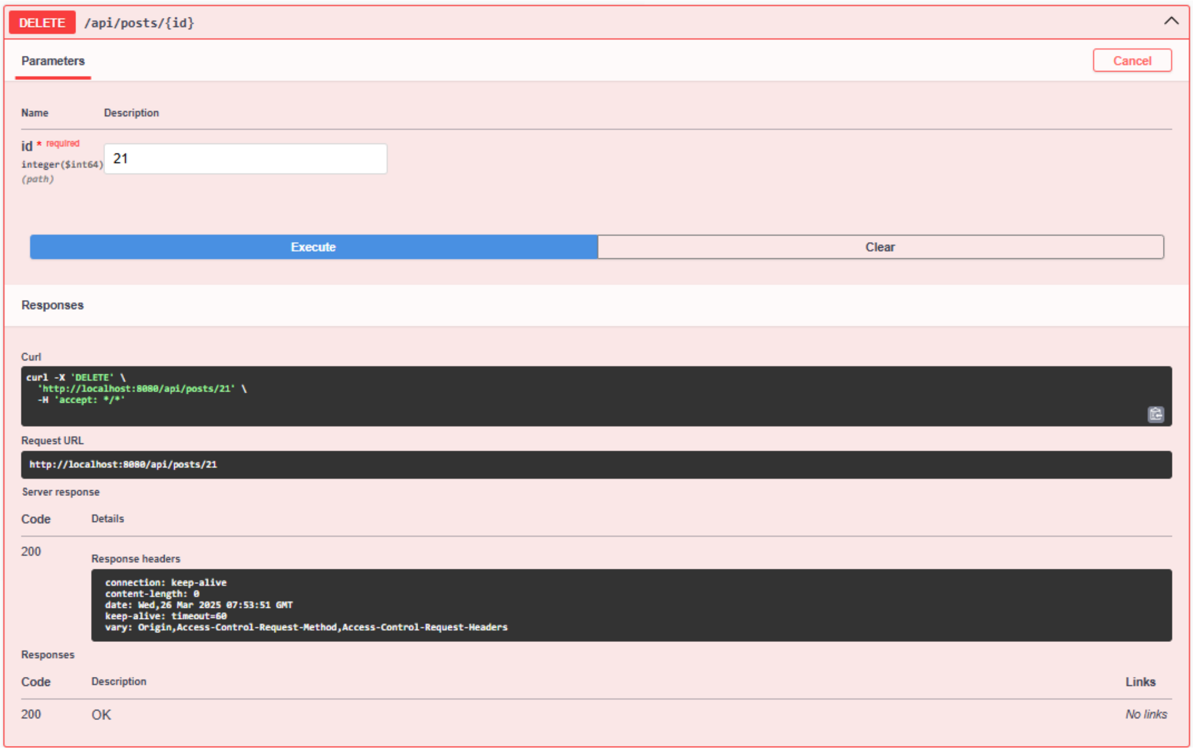
Response headers

```
 connection: keep-alive
 content-type: application/json
 date: Wed,26 Mar 2025 07:52:39 GMT
 keep-alive: timeout=60
 transfer-encoding: chunked
 vary: Origin,Access-Control-Request-Method,Access-Control-Request-Headers
```

Responses

Code	Description	Links
200	OK	*No links*

A **POST** request to create a new post with userId, content, and empty arrays for media and tags.



Name	Description
id * required integer($int64) *(path)*	21

Curl

```
curl -X 'DELETE' \
  'http://localhost:8080/api/posts/21' \
  -H 'accept: */*'
```

Request URL

```
http://localhost:8080/api/posts/21
```

Server response

Code	Details
200	Response headers

```
connection: keep-alive
content-length: 0
date: Wed,26 Mar 2025 07:53:51 GMT
keep-alive: timeout=60
vary: Origin,Access-Control-Request-Method,Access-Control-Request-Headers
```

Responses

Code	Description	Links
200	OK	*No links*

A **DELETE** request to remove a specific post using its ID (here, post ID = 21), resulting in a successful 200 OK response.

6. Post-Tag Management

- Post-Tag relations can be created, fetched, and deleted using appropriate endpoints.
- Ensures correct association of tags with posts using valid post and tag IDs.
- All operations like Create, Fetch, and Delete tested successfully.
- This module works similarly to the Report Controller in terms of request structure and response format — using standard JSON payloads and returning 200 OK on success.

7. Message Management

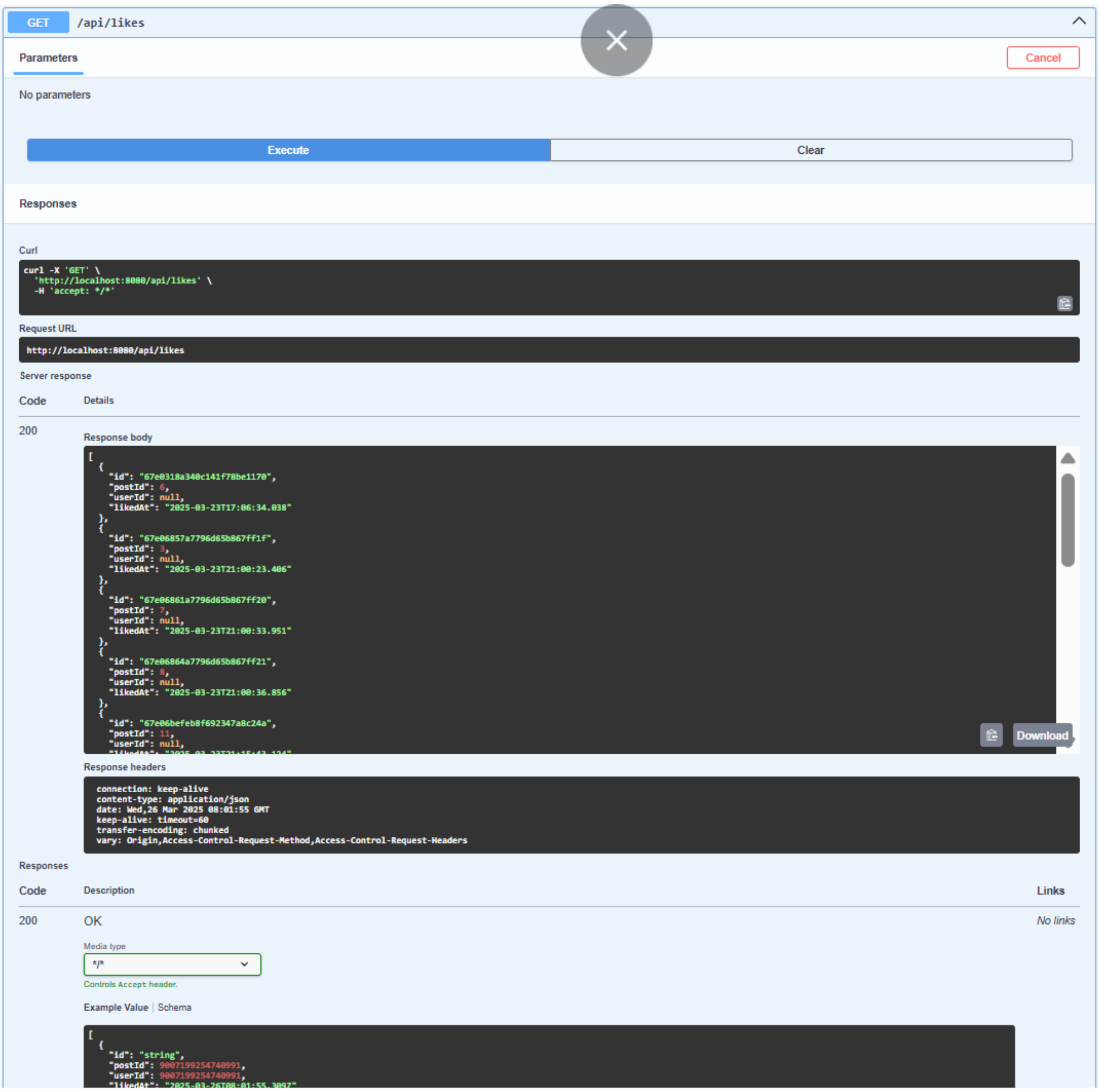
- Messages can be created and retrieved for specific users using userId.
- Validations ensure messages are linked to valid users.
- All operations like Create and Fetch tested successfully.
- Similar to the Report Controller, all requests and responses follow a consistent JSON structure, with clear validation and status messages.

8. Media Management

- Media files (such as images or links) can be associated with posts.
- Supports creating, retrieving, and deleting media using valid post IDs.
- All media-related operations tested and verified for correct linkage with posts.
- Like the Report Controller, this module uses the same request-response format, with structured JSON data and standard HTTP response codes.

9. Like Management

- Likes can be added to posts by users and retrieved using post or user IDs.
- Ensures a post cannot be liked multiple times by the same user (validation logic).
- All operations such as Create, Fetch, and Delete tested successfully.
- This controller is similar in structure and behavior to the **Media Management** — using standardized JSON request/response formats with appropriate success/error handling and 200 OK confirmations



No parameters

```
curl -X 'GET' \
  'http://localhost:8080/api/likes' \
  -H 'accept: */*'
```

```
http://localhost:8080/api/likes
```

200

```
[
  {
    "id": "67e0318a340c141f78be1170",
    "postId": 6,
    "userId": null,
    "likedAt": "2025-03-23T17:06:34.038"
  },
  {
    "id": "67e06857a7796d65b867ff1f",
    "postId": 3,
    "userId": null,
    "likedAt": "2025-03-23T21:00:23.406"
  },
  {
    "id": "67e06861a7796d65b867ff20",
    "postId": 7,
    "userId": null,
    "likedAt": "2025-03-23T21:00:33.951"
  },
  {
    "id": "67e06864a7796d65b867ff21",
    "postId": 8,
    "userId": null,
    "likedAt": "2025-03-23T21:00:36.856"
  },
  {
    "id": "67e06befeb8f692347a8c24a",
    "postId": 11,
    "userId": null,
    "likedAt": "2025-03-23T21:15:43.124"
```

```
connection: keep-alive
content-type: application/json
date: Wed,26 Mar 2025 08:01:55 GMT
keep-alive: timeout=60
transfer-encoding: chunked
vary: Origin,Access-Control-Request-Method,Access-Control-Request-Headers
```

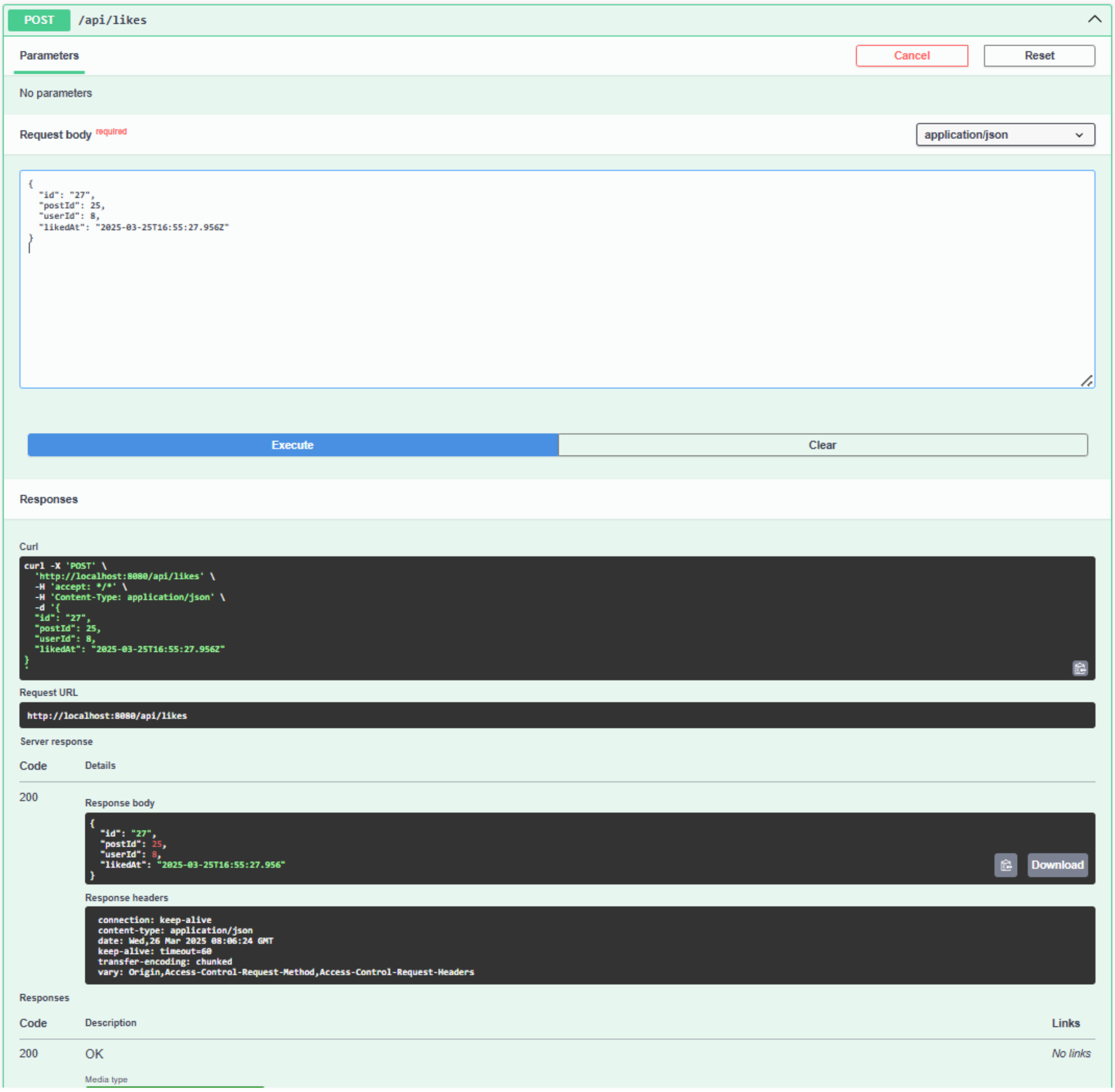
200 OK — No links

/

Controls Accept header.

```
[
  {
    "id": "string",
    "postId": 9007199254740991,
    "userId": 9007199254740991,
    "likedAt": "2025-03-26T08:01:55.309Z"
```

A GET request to retrieve all likes from the system with details including postId, userId, and timestamp.



```
{
  "id": "27",
  "postId": 25,
  "userId": 8,
  "likedAt": "2025-03-25T16:55:27.956Z"
}
```

```
curl -X 'POST' \
  'http://localhost:8080/api/likes' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "id": "27",
  "postId": 25,
  "userId": 8,
  "likedAt": "2025-03-25T16:55:27.956Z"
}
'
```

Request URL

```
http://localhost:8080/api/likes
```

Server response

Code	Details
200	Response body

```
{
  "id": "27",
  "postId": 25,
  "userId": 8,
  "likedAt": "2025-03-25T16:55:27.956"
}
```

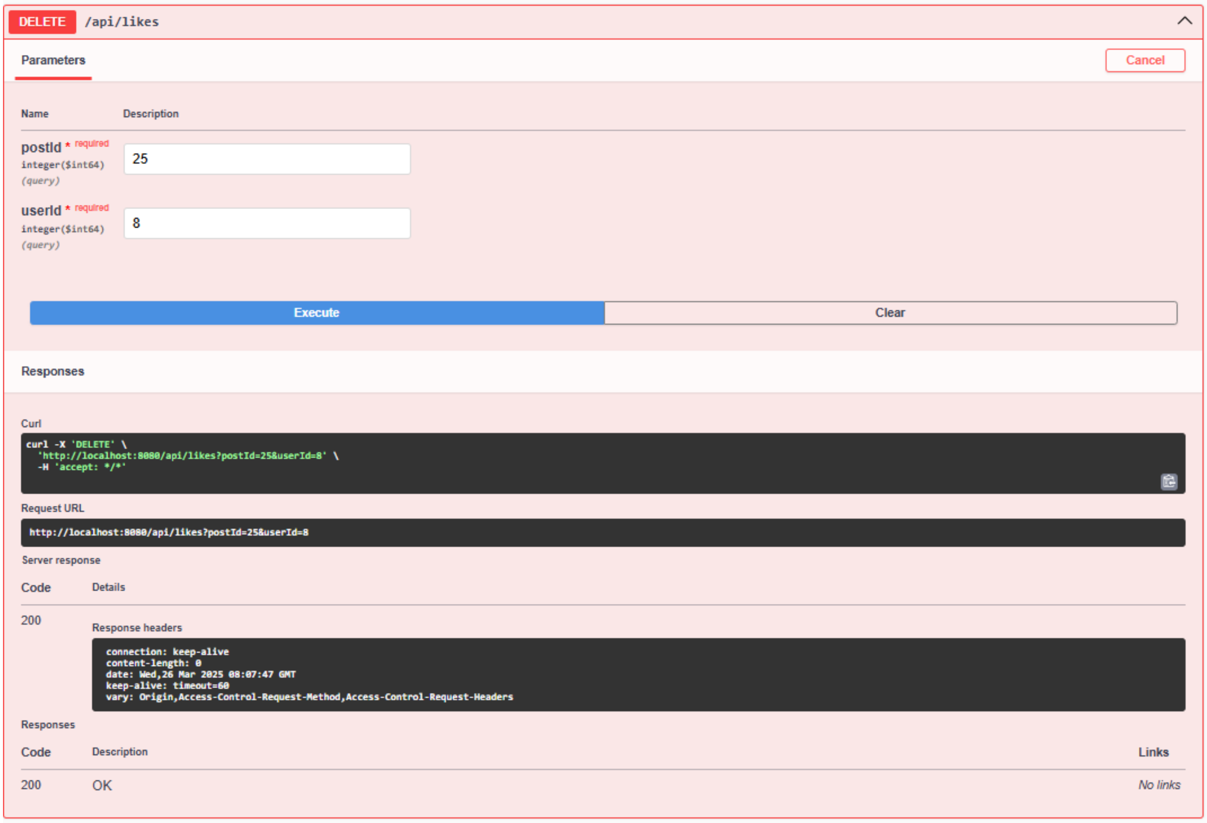
Response headers

```
 connection: keep-alive
 content-type: application/json
 date: Wed,26 Mar 2025 08:06:24 GMT
 keep-alive: timeout=60
 transfer-encoding: chunked
 vary: Origin,Access-Control-Request-Method,Access-Control-Request-Headers
```

Responses

Code	Description	Links
200	OK Media type	*No links*

A POST request to add a like to a specific post by a user with a timestamp of when the like was made.



Name	Description
postId * required integer($int64) *(query)*	25
userId * required integer($int64) *(query)*	8

Responses

Curl

```
curl -X 'DELETE' \
  'http://localhost:8080/api/likes?postId=25&userId=8' \
  -H 'accept: */*'
```

Request URL

```
http://localhost:8080/api/likes?postId=25&userId=8
```

Server response

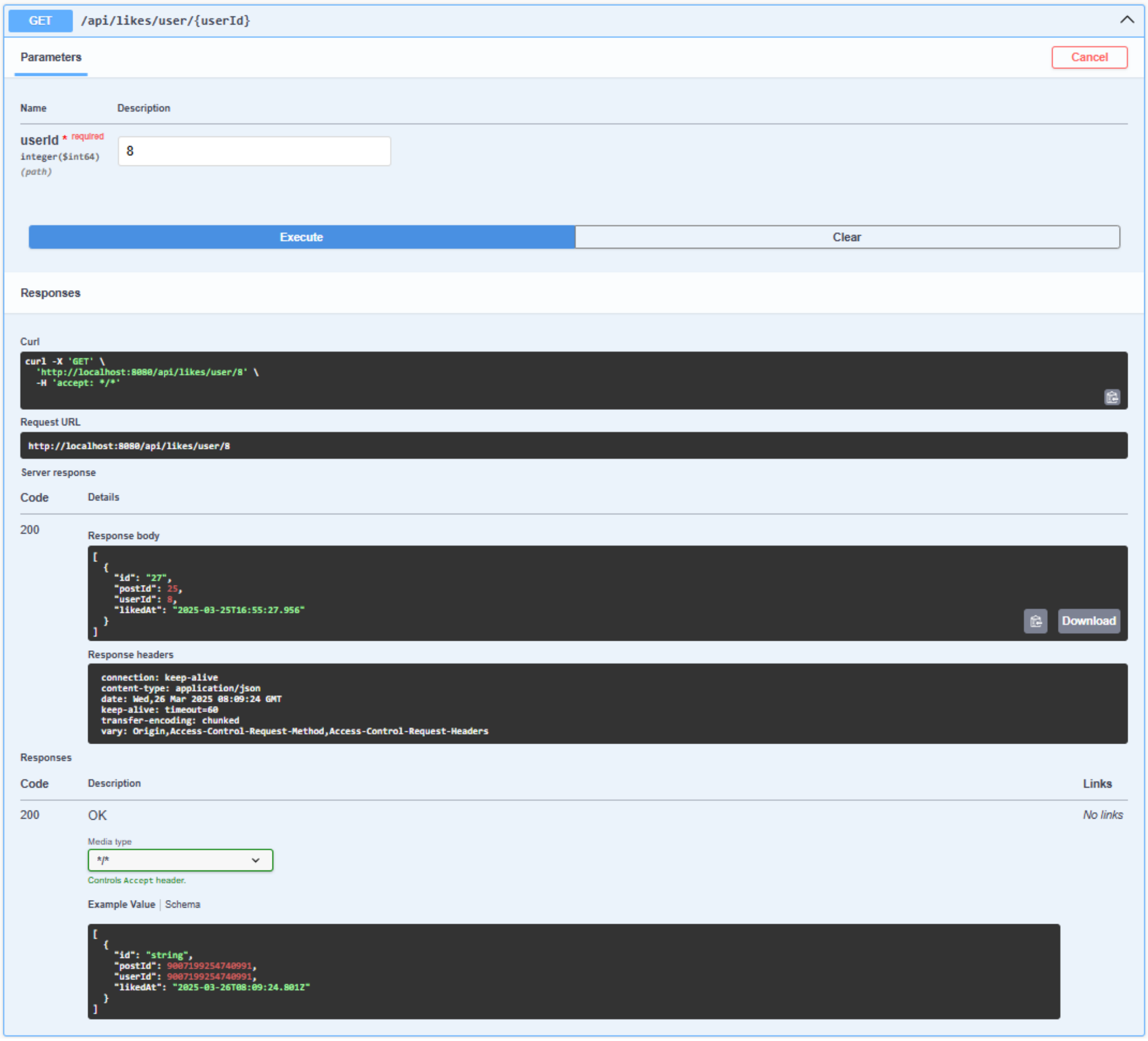
Code	Details
200	Response headers

```
connection: keep-alive
content-length: 0
date: Wed,26 Mar 2025 08:07:47 GMT
keep-alive: timeout=60
vary: Origin,Access-Control-Request-Method,Access-Control-Request-Headers
```

Responses

Code	Description	Links
200	OK	*No links*

A DELETE request to remove a like based on a combination of postId and userId, confirming with a 200 OK response.



Name	Description
userId * required integer($int64) (path)	8

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:8080/api/likes/user/8' \
  -H 'accept: */*'
```

Request URL

```
http://localhost:8080/api/likes/user/8
```

Server response

Code Details

200

Response body

```
[
  {
    "id": "27",
    "postId": 25,
    "userId": 8,
    "likedAt": "2025-03-25T16:55:27.956"
  }
]
```

Response headers

```
 connection: keep-alive
 content-type: application/json
 date: Wed,26 Mar 2025 08:09:24 GMT
 keep-alive: timeout=60
 transfer-encoding: chunked
 vary: Origin,Access-Control-Request-Method,Access-Control-Request-Headers
```

Responses

Code Description Links

200 OK No links

Media type: */*

Controls Accept header.

Example Value | Schema

```
[
  {
    "id": "string",
    "postId": 9007199254740991,
    "userId": 9007199254740991,
    "likedAt": "2025-03-26T08:09:24.801Z"
  }
]
```

A GET request to retrieve all likes made by a specific user using their userId (here, userId = 8).

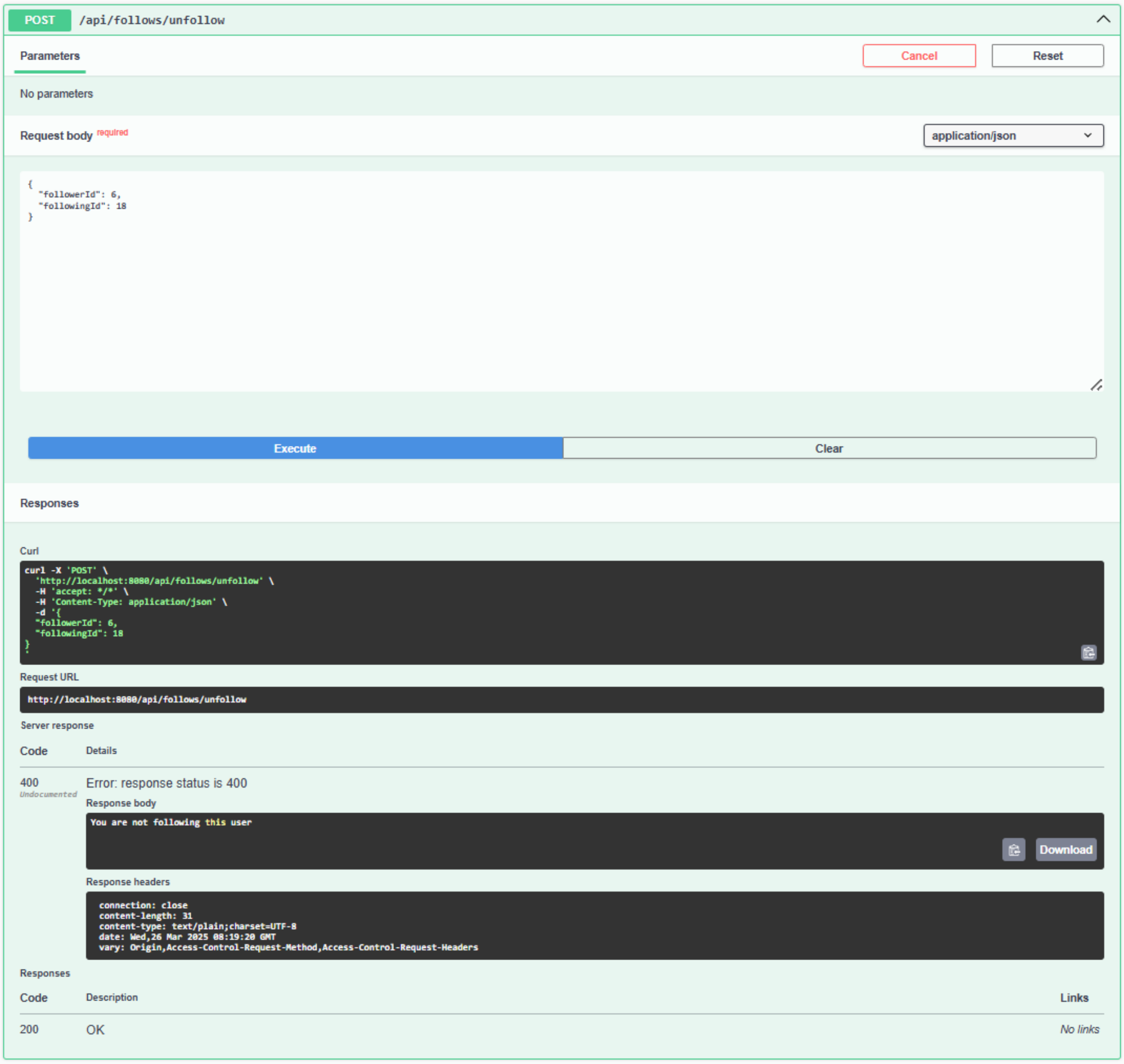
Get like using PostID

- Likes can be retrieved based on both **User ID** and **Post ID**.
- The GET request to /api/likes/post/{postId} works successfully.
- You only need to input the postId, and the system returns all likes associated with that post.
- This endpoint functions similarly to the GET /api/likes/user/{userId} endpoint.
- The response includes user and timestamp details of likes on that post.

10. Follow Management

- The application supports follow and unfollow functionality between users.

- Endpoints include creating a follow, unfollowing a user, and retrieving follow data.
- The **POST** request to /api/follows/follow and /api/follows/unfollow works by submitting followerId and followingId.
- You can retrieve all followings of a user using GET /api/follows/user/{userId} — simply input the userId and receive the list of users they follow.
- These endpoints follow a consistent format, similar to other modules like Likes and Reports.



```
{
  "followerId": 6,
  "followingId": 18
}
```

```
curl -X 'POST' \
  'http://localhost:8080/api/follows/unfollow' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "followerId": 6,
  "followingId": 18
}'
```

```
http://localhost:8080/api/follows/unfollow
```

Code	Details
400 Undocumented	Error: response status is 400

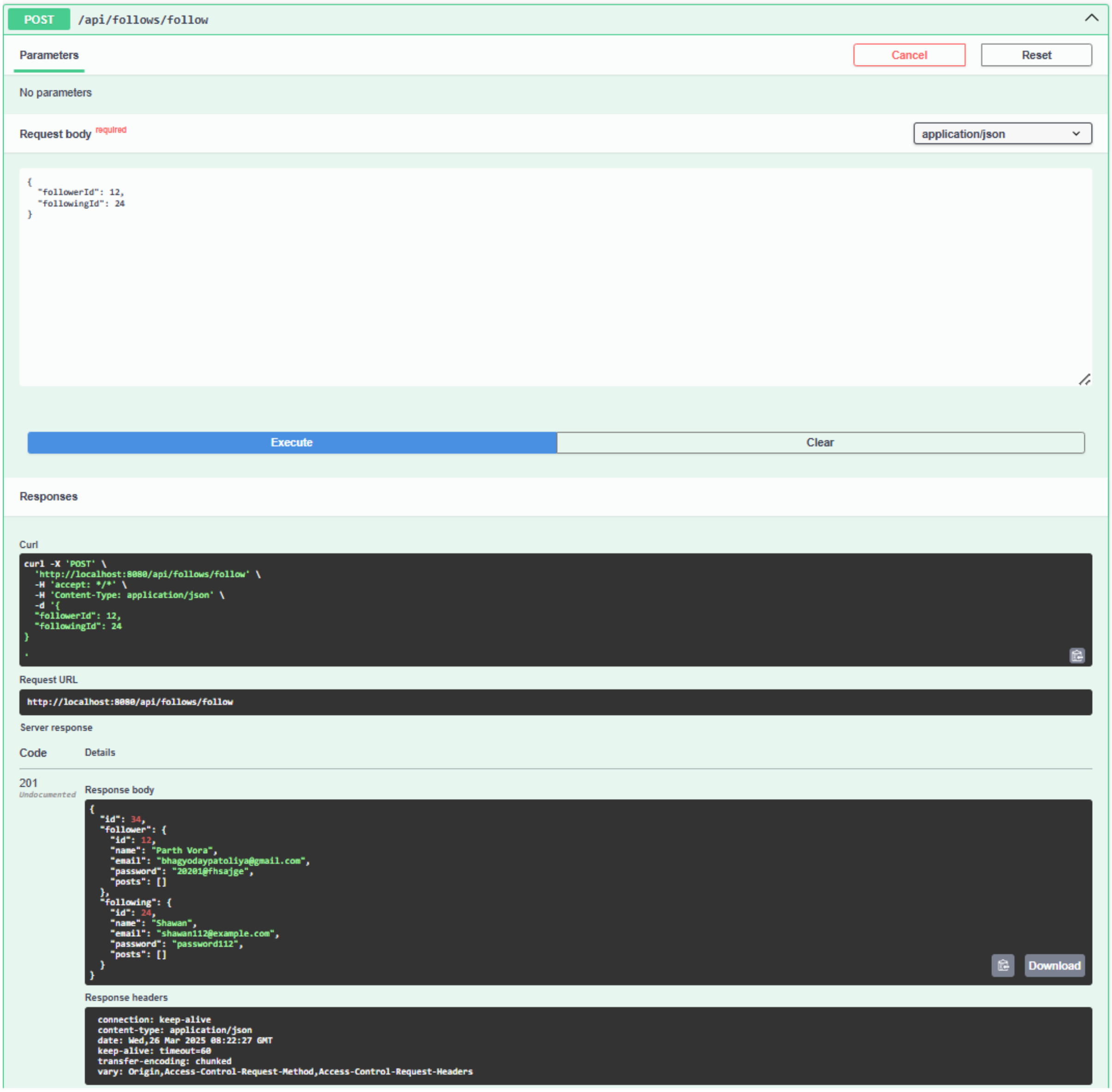
Response body

```
You are not following this user
```

```
 connection: close
 content-length: 31
 content-type: text/plain;charset=UTF-8
 date: Wed,26 Mar 2025 08:19:20 GMT
 vary: Origin,Access-Control-Request-Method,Access-Control-Request-Headers
```

Code	Description	Links
200	OK	No links

A **POST** request to unfollow a user by providing followerId and followingId. If not already following, it returns a proper validation error.



/api/follows/follow

Parameters

No parameters

Request body required: application/json

```
{
  "followerId": 12,
  "followingId": 24
}
```

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:8080/api/follows/follow' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "followerId": 12,
  "followingId": 24
}
'
```

Request URL

```
http://localhost:8080/api/follows/follow
```

Server response

Code: 201 *Undocumented*

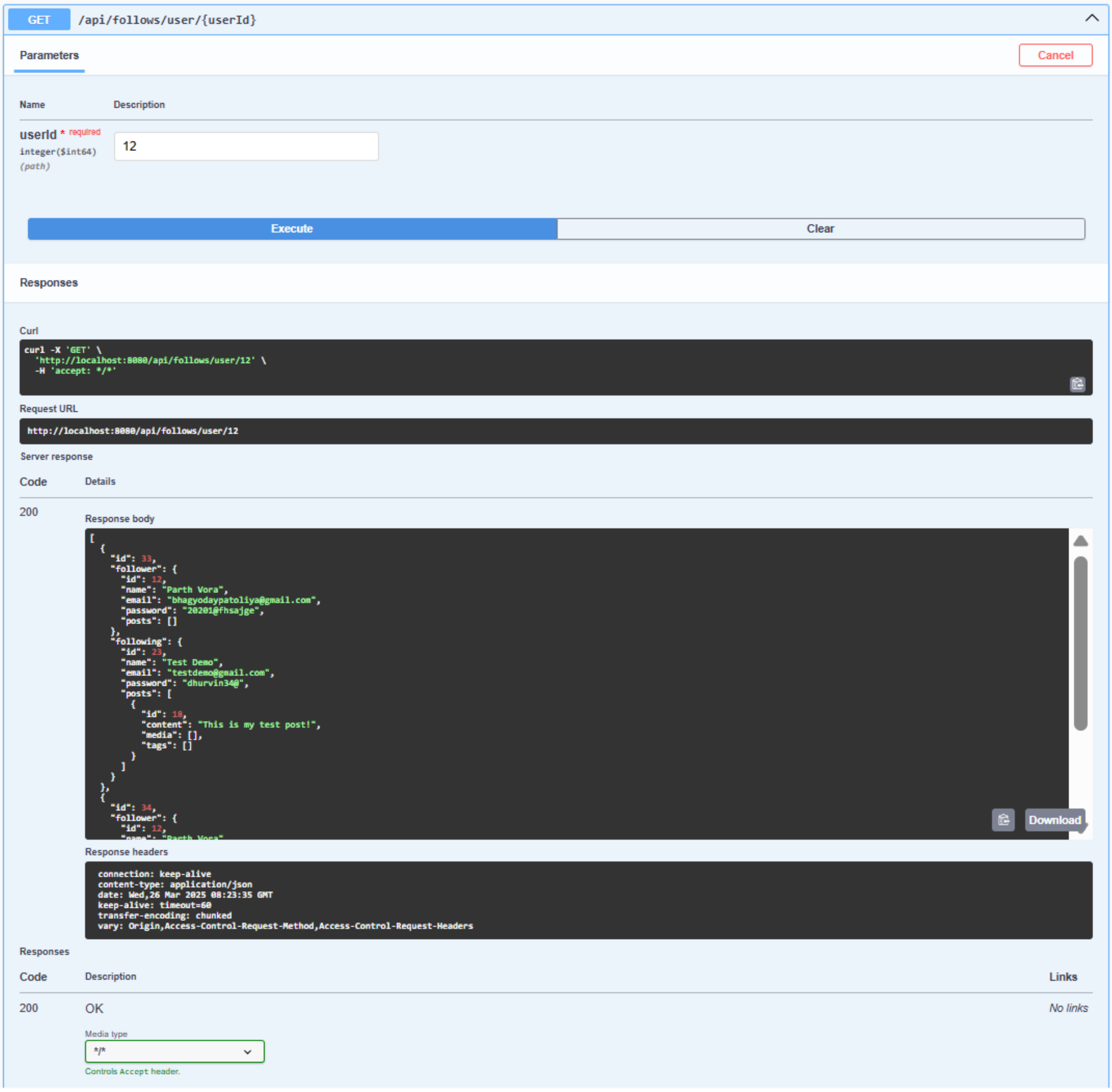
Details: Response body

```
{
  "id": 34,
  "follower": {
    "id": 12,
    "name": "Parth Vora",
    "email": "bhagyodaypatoliya@gmail.com",
    "password": "20201@fhsajge",
    "posts": []
  },
  "following": {
    "id": 24,
    "name": "Shawan",
    "email": "shawan112@example.com",
    "password": "password112",
    "posts": []
  }
}
```

Response headers

```
 connection: keep-alive
 content-type: application/json
 date: Wed,26 Mar 2025 08:22:27 GMT
 keep-alive: timeout=60
 transfer-encoding: chunked
 vary: Origin,Access-Control-Request-Method,Access-Control-Request-Headers
```

A **POST** request to follow a user by submitting both followerId and followingId. Returns the full follow relationship details upon success.



/api/follows/user/{userId}

Name | Description

userId * required
integer($int64)
(path)
12

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:8080/api/follows/user/12' \
  -H 'accept: */*'
```

Request URL

```
http://localhost:8080/api/follows/user/12
```

Server response

Code | Details

200

Response body

```
[
  {
    "id": 33,
    "follower": {
      "id": 12,
      "name": "Parth Vora",
      "email": "bhagyodaypatoliya@gmail.com",
      "password": "20201@fhsajge",
      "posts": []
    },
    "following": {
      "id": 23,
      "name": "Test Demo",
      "email": "testdemo@gmail.com",
      "password": "dhurvin34@",
      "posts": [
        {
          "id": 18,
          "content": "This is my test post!",
          "media": [],
          "tags": []
        }
      ]
    }
  },
  {
    "id": 34,
    "follower": {
      "id": 12,
      "name": "Parth Vora"
```

Response headers

```
connection: keep-alive
content-type: application/json
date: Wed,26 Mar 2025 08:23:35 GMT
keep-alive: timeout=60
transfer-encoding: chunked
vary: Origin,Access-Control-Request-Method,Access-Control-Request-Headers
```

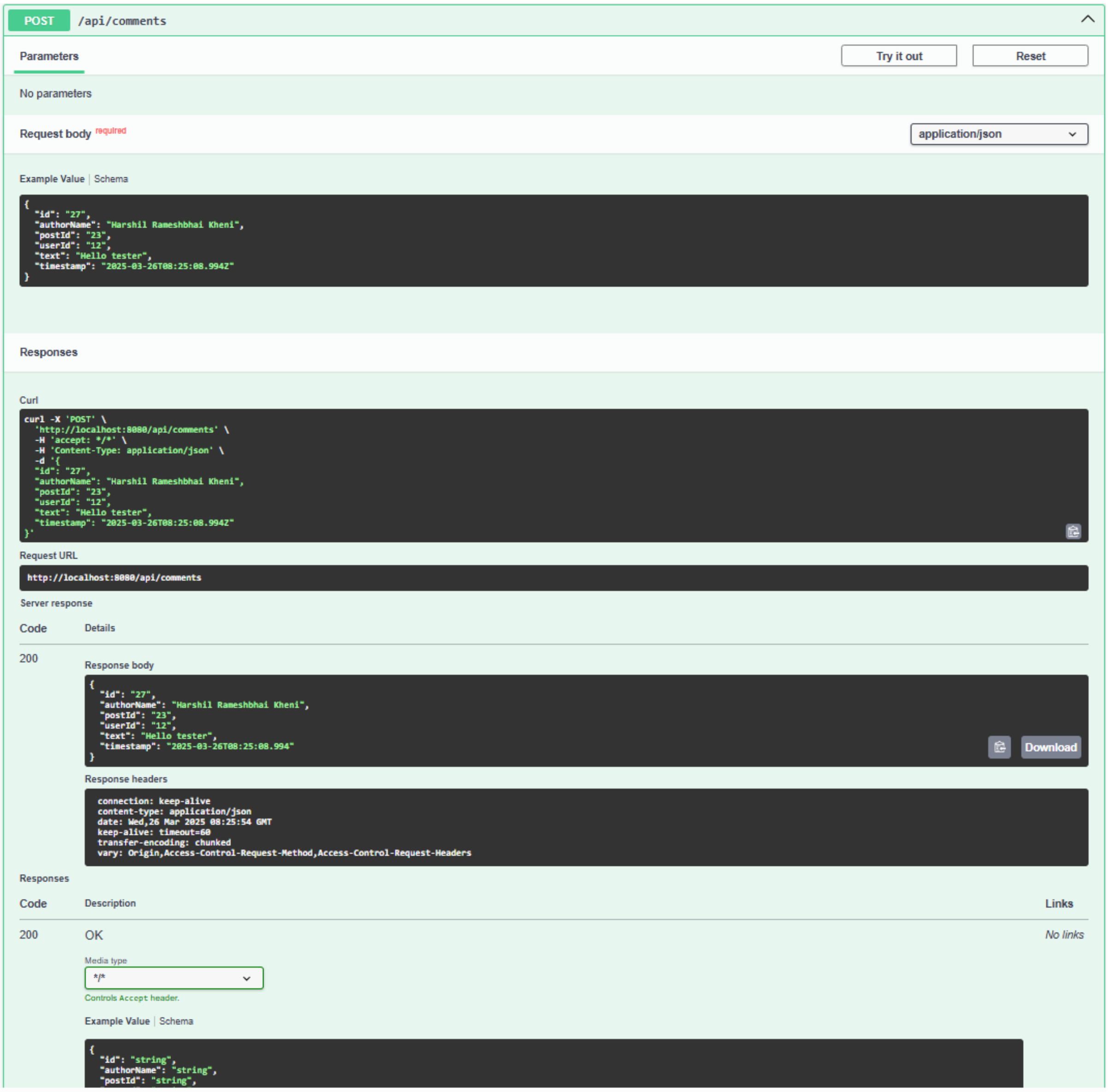
Responses

Code	Description	Links
200	OK Media type */* Controls Accept header.	*No links*

A **GET** request to retrieve all users that the specified user (here, userId = 12) is following, along with their profile and post data.

11. Comment Management

- Comments can be created, retrieved, and linked to specific posts.
- Full CRUD operations are supported for comments using MongoDB.
- Validations ensure comment content is stored and retrieved accurately.
- All comment-related endpoints have been tested successfully, similar in structure and response format to the **Post Mangement**.



No parameters

Request body required

```
{
  "id": "27",
  "authorName": "Harshil Rameshbhai Kheni",
  "postId": "23",
  "userId": "12",
  "text": "Hello tester",
  "timestamp": "2025-03-26T08:25:08.994Z"
}
```

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:8080/api/comments' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "id": "27",
  "authorName": "Harshil Rameshbhai Kheni",
  "postId": "23",
  "userId": "12",
  "text": "Hello tester",
  "timestamp": "2025-03-26T08:25:08.994Z"
}'
```

Request URL

```
http://localhost:8080/api/comments
```

Server response

Code Details

200

Response body

```
{
  "id": "27",
  "authorName": "Harshil Rameshbhai Kheni",
  "postId": "23",
  "userId": "12",
  "text": "Hello tester",
  "timestamp": "2025-03-26T08:25:08.994"
}
```

Response headers

```
connection: keep-alive
content-type: application/json
date: Wed,26 Mar 2025 08:25:54 GMT
keep-alive: timeout=60
transfer-encoding: chunked
vary: Origin,Access-Control-Request-Method,Access-Control-Request-Headers
```

Responses

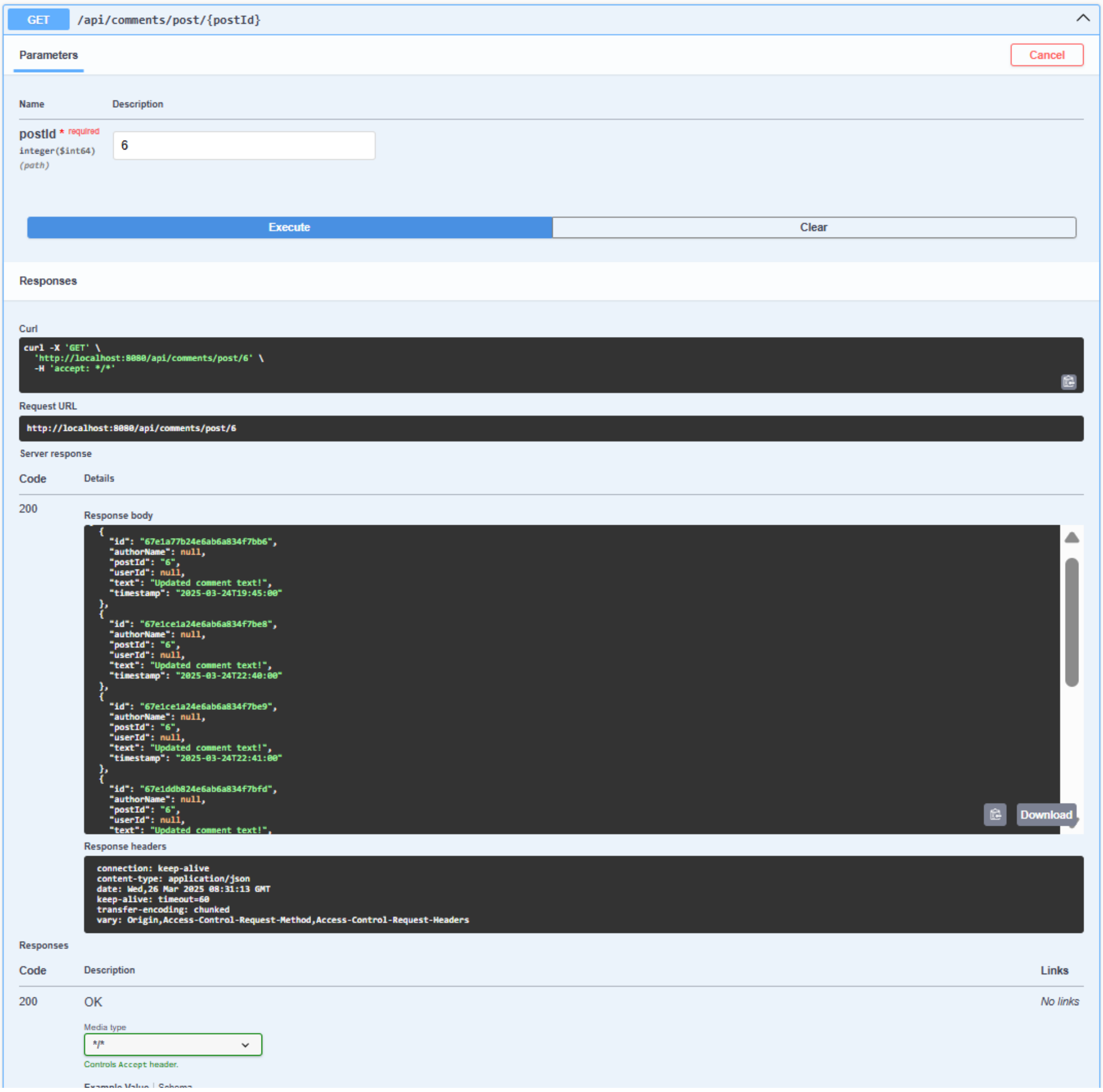
Code	Description	Links
200	OK	*No links*

Media type: */*

Controls Accept header.

```
{
  "id": "string",
  "authorName": "string",
  "postId": "string",
```

A POST request to create a new comment with fields like author, postId, text, and timestamp. The response confirms the comment was saved successfully.



```
curl -X 'GET' \
  'http://localhost:8080/api/comments/post/6' \
  -H 'accept: */*'
```

```
http://localhost:8080/api/comments/post/6
```

```
  {
    "id": "67e1a77b24e6ab6a834f7bb6",
    "authorName": null,
    "postId": "6",
    "userId": null,
    "text": "Updated comment text!",
    "timestamp": "2025-03-24T19:45:00"
  },
  {
    "id": "67e1ce1a24e6ab6a834f7be8",
    "authorName": null,
    "postId": "6",
    "userId": null,
    "text": "Updated comment text!",
    "timestamp": "2025-03-24T22:40:00"
  },
  {
    "id": "67e1ce1a24e6ab6a834f7be9",
    "authorName": null,
    "postId": "6",
    "userId": null,
    "text": "Updated comment text!",
    "timestamp": "2025-03-24T22:41:00"
  },
  {
    "id": "67e1ddb824e6ab6a834f7bfd",
    "authorName": null,
    "postId": "6",
    "userId": null,
    "text": "Updated comment text!",
```

```
connection: keep-alive
content-type: application/json
date: Wed,26 Mar 2025 08:31:13 GMT
keep-alive: timeout=60
transfer-encoding: chunked
vary: Origin,Access-Control-Request-Method,Access-Control-Request-Headers
```

A GET request to fetch all comments related to a specific post using postId. Here, comments for postId = 6 are retrieved successfully with relevant details.

7. Challenges and Solutions

Challenge 1: Exception Handling Across SQL and NoSQL

Problem: Managing consistent exception responses from both MySQL (SQL) and MongoDB (NoSQL) services, especially when operations like follow, like, comment, or fetch fail due to missing IDs or invalid relationships.

Solution: Custom exception classes with proper @ResponseStatus annotations for clear and consistent error messaging across the API.

Example:

```
@ResponseStatus(HttpStatus.NOT_FOUND)
public class UserNotFoundException extends RuntimeException {
    public UserNotFoundException(String message) {
        super(message);
    }
}

// UserService.java
public User getUserById(Long id) {
    return userRepository.findById(id)
        .orElseThrow(() -> new UserNotFoundException("User not found with id: " + id));
}
```

Challenge 2: Handling Hybrid Data Models (SQL + NoSQL)

Problem: Integrating SQL-based tables (Users, Posts, Follows) with NoSQL collections (Comments, Likes, Messages, Notifications) in a seamless manner.

Solution: Clear separation of repositories and service layers for SQL and NoSQL, using annotations like @Document for Mongo entities and @Entity for SQL ones. Ensured proper mapping and connection where necessary (e.g., referencing SQL postId in MongoDB comment document).

Challenge 3: Post–User Relationship and Recursion

Problem: Circular references between Posts and Users caused infinite JSON recursion during API response serialization.

Solution: Used Jackson annotations @JsonManagedReference and @JsonBackReference to manage bidirectional relationships and avoid serialization loops.

```
// User.java
@OneToMany(mappedBy = "user", cascade = CascadeType.ALL)
@JsonManagedReference
```

```
private List<Post> posts;

// Post.java
@ManyToOne
@JoinColumn(name = "user_id")
@JsonBackReference
private User user;
```

Challenge 4: MongoDB Query Limitations

Problem: Fetching comments, likes, and notifications efficiently from large MongoDB collections.

Solution: Indexed frequently queried fields (e.g., postId, userId), implemented pagination, and used @Query annotations or repository methods for custom filters.

Challenge 5: Follow and Like System Constraints

Problem: Foreign key violations in MySQL when trying to delete a user who is still referenced in the follow table.

Solution: Added manual checks and deletion logic in service layers to cascade delete or restrict operations with meaningful messages (e.g., “User cannot be deleted as they are being followed”).

8. Conclusion and Future Work

The **Social Media RESTful API using Spring Boot + MongoDB & MySQL** project successfully demonstrates the use of hybrid databases to handle both structured and unstructured data in a social media platform. With a clean modular design, the system supports CRUD operations, follows, likes, comments, tagging, reporting, and notifications.

Future Improvements

1. **User Authentication & Authorization**
    - Add JWT-based login and role-level access (e.g., admin, user)
2. **Enhanced UI/UX**
    - Integrate with React frontend using Bootstrap or Tailwind CSS
3. **Real-Time Features**
    - Enable WebSocket or polling for real-time notifications and messages
4. **ElasticSearch Integration**
    - Use ElasticSearch for advanced search features across posts, comments, and users
5. **Analytics & Dashboards**
    - Implement like trends, comment heatmaps, and top-followed users visualization