**REST API Documentation - Movie App**

**Overview**

This API allows users to interact with a movie app backend, including managing movies, user accounts, groups, and watchlists.

Base URL: http://localhost:3001

Authentication: Bearer Token (JWT)

Content-Type: application/json

**Endpoints**

**User Router:**

**1. Register a User**

Endpoint: POST /register

Description: Register a new user.

Request Body:

{

"username": "string",

"email": "string",

"password": "string (minimum 8 characters)"

}

Response:

201 Created

{

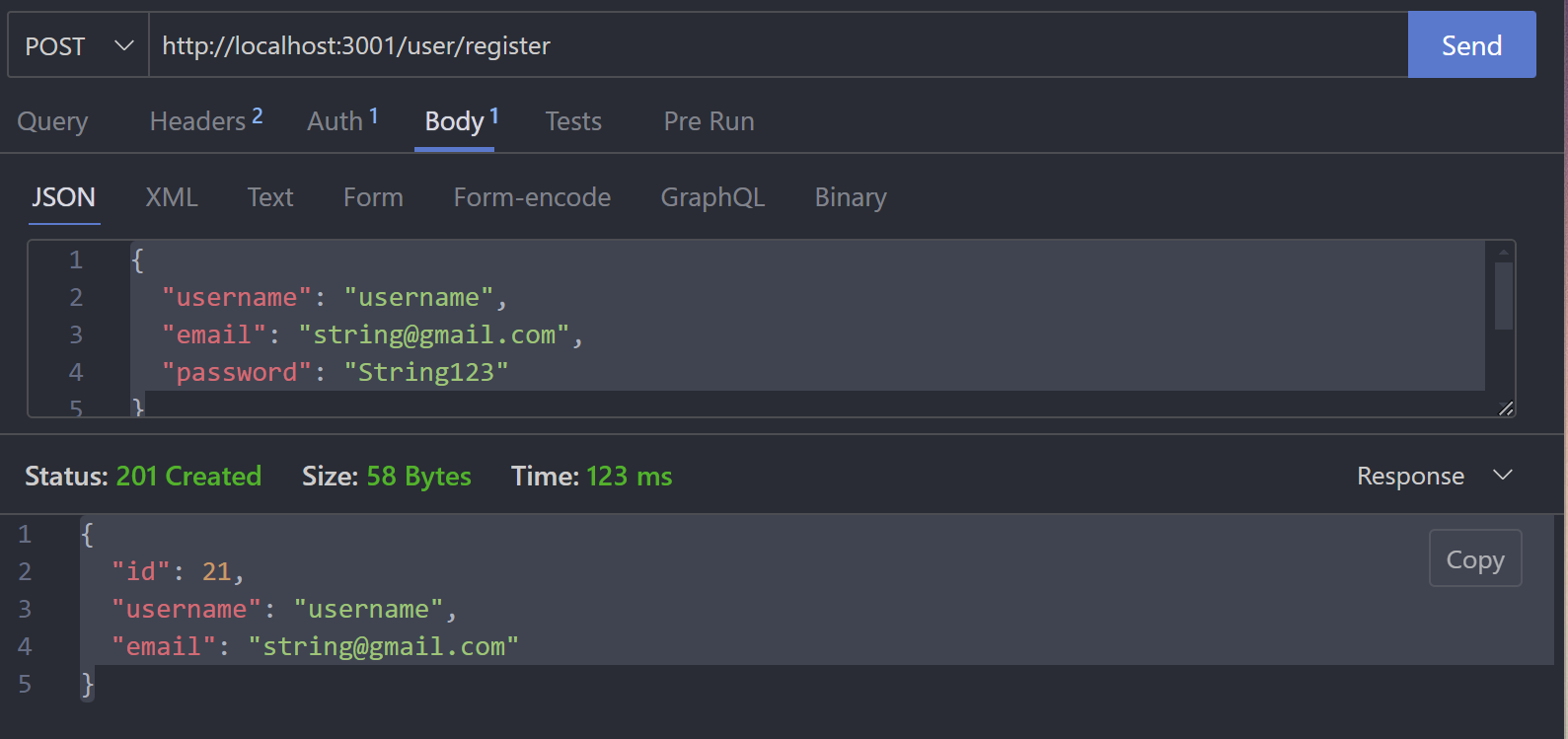
"id": number,

"username": "string",

"email": "string"

}

400 Bad Request: Invalid input.



**2. Login**

Endpoint: POST /login

Description: Authenticate a user and return a token.

Request Body:

{

"email": "string",

"password": "string"

}

Response:

200 OK:

{

"id": "integer",

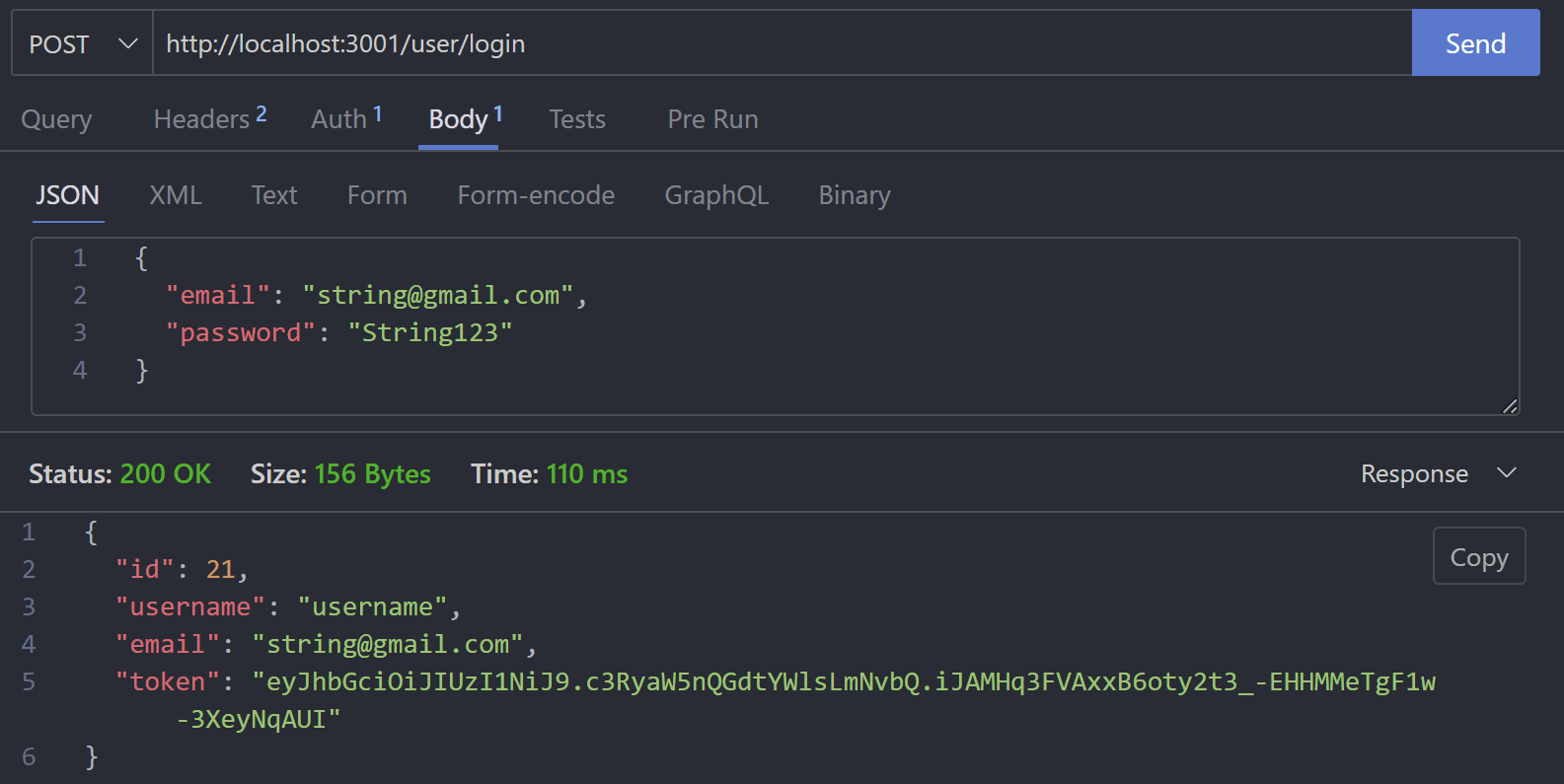
"username": "string",

"email": "string",

"token": "string"

}

401 Unauthorized: Invalid credentials.



**3. Logout**

Endpoint: POST /logout

Description: Logs out a user.

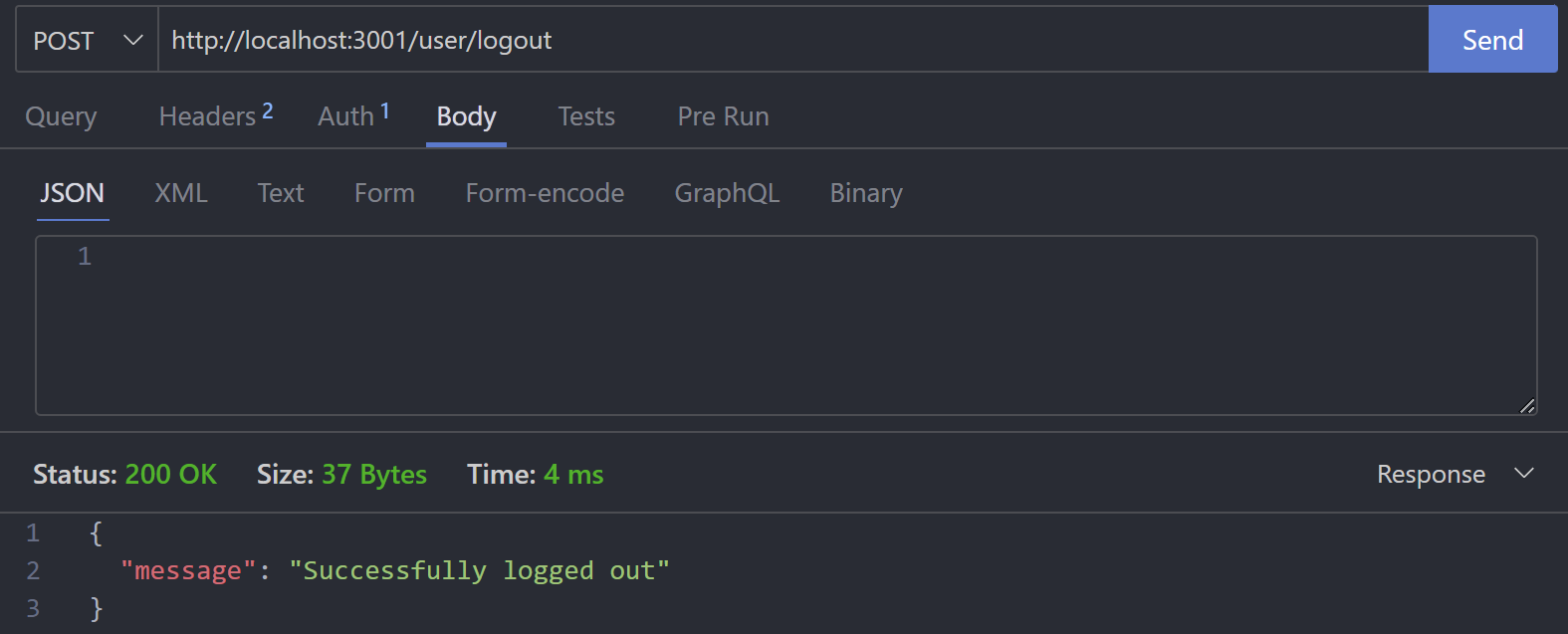
Response:

200 OK：

{

"message": "Successfully logged out"

}



1. **Delete a User**

Endpoint: DELETE /delete

Description: Delete a user account.

Request Body:

{

"id": "number"

}

Response:

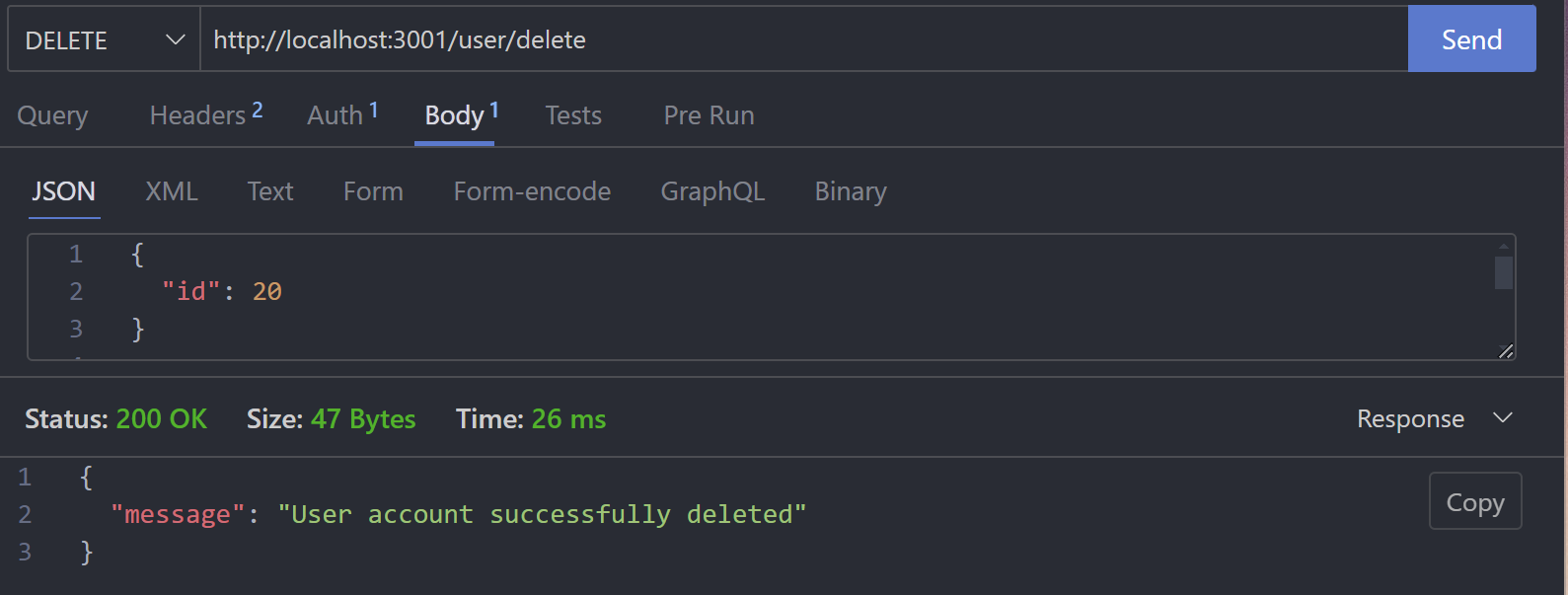
200 OK:

{

"message": "User account successfully deleted"

}

404 Not Found: User not found.



**5. Get Account Information**

Endpoint: GET /:accountId/account

Description: Retrieve account information.

Response:

200 OK:

[

{

"id": "integer",

"username": "string",

"email": "string",

"password": "string",

"share\_url": "string",

"is\_public": true,

"avatar": {

"type": "Buffer",

"data":"bytea"

},

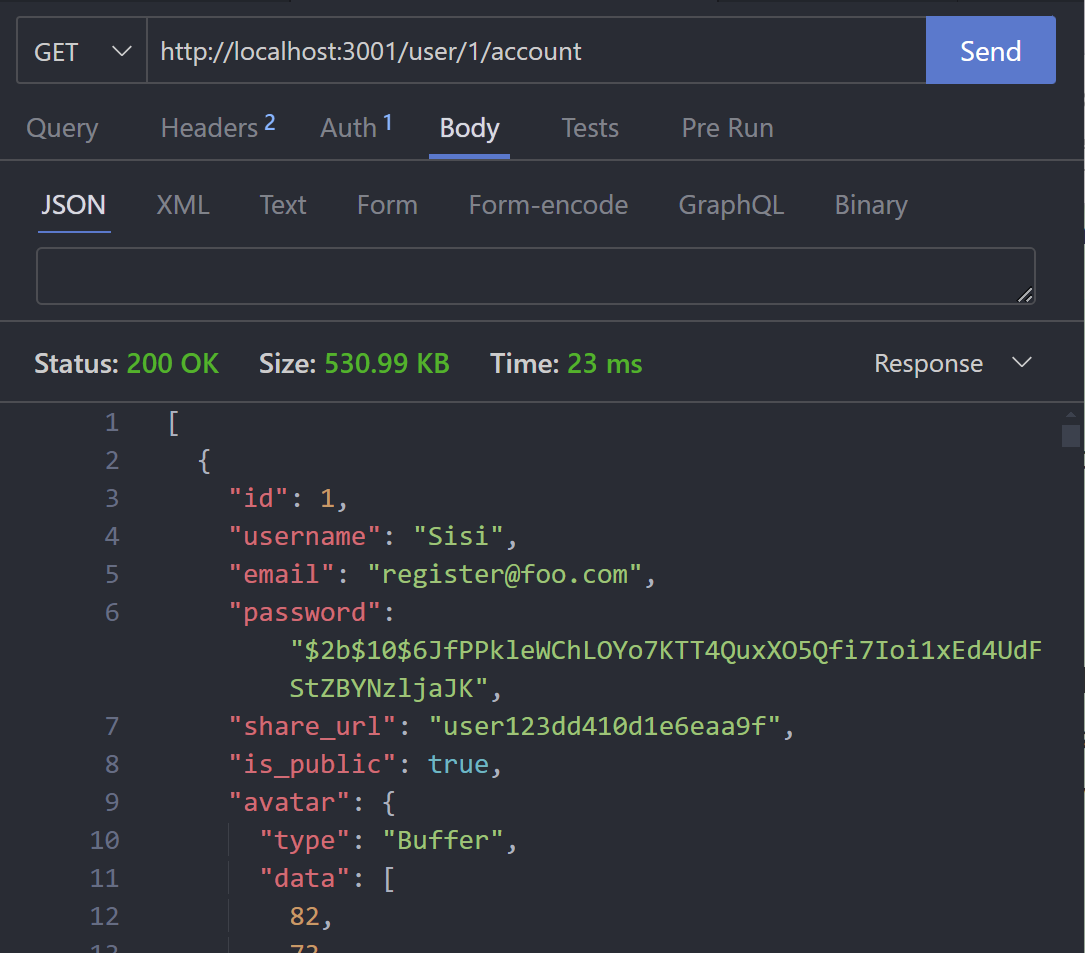
"country": "string",

"gender": "string",

"birthday": "1988-08-23T21:00:00.000Z"

}

]



**6. Edit Account Information**

Endpoint: PATCH /:accountId/editaccount

Description: Update account information.

Request Body

{

"username": "string",

"country": "string",

"gender": "string",

"birthday": "string (ISO format)"

}

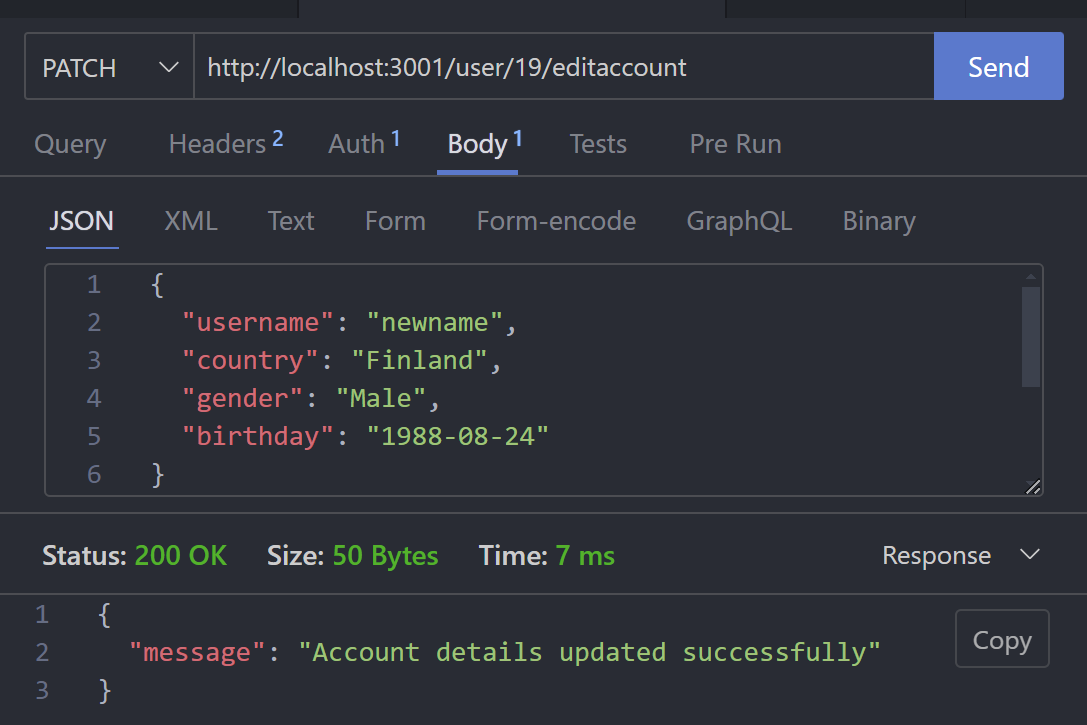
Response:

200 OK：

{

"message": "Account details updated successfully"

}



**7. Upload User Avatar**

Endpoint: POST /:accountId/uploadavatar

Description: Upload or update the user's avatar.

Request Body: Form-data with a key image for the file.

Response:

200 OK: Returns the avatar image in Base64 format.

{

"base64Image":"string"

}

**8. Get User Avatar**

Endpoint: GET /:accountId/avatar

Description: Retrieve the user's avatar.

Response:

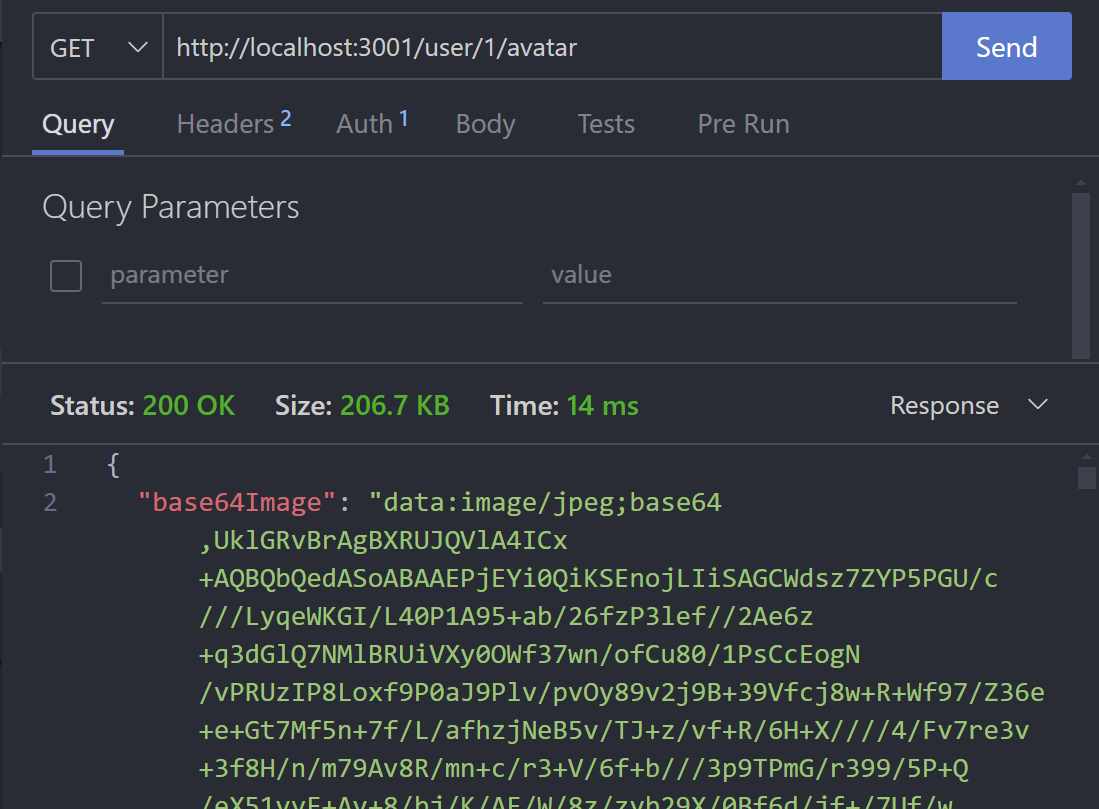
200 OK: Returns the avatar image in Base64 format.

{

"base64Image":"string"

}

404：Bad Request

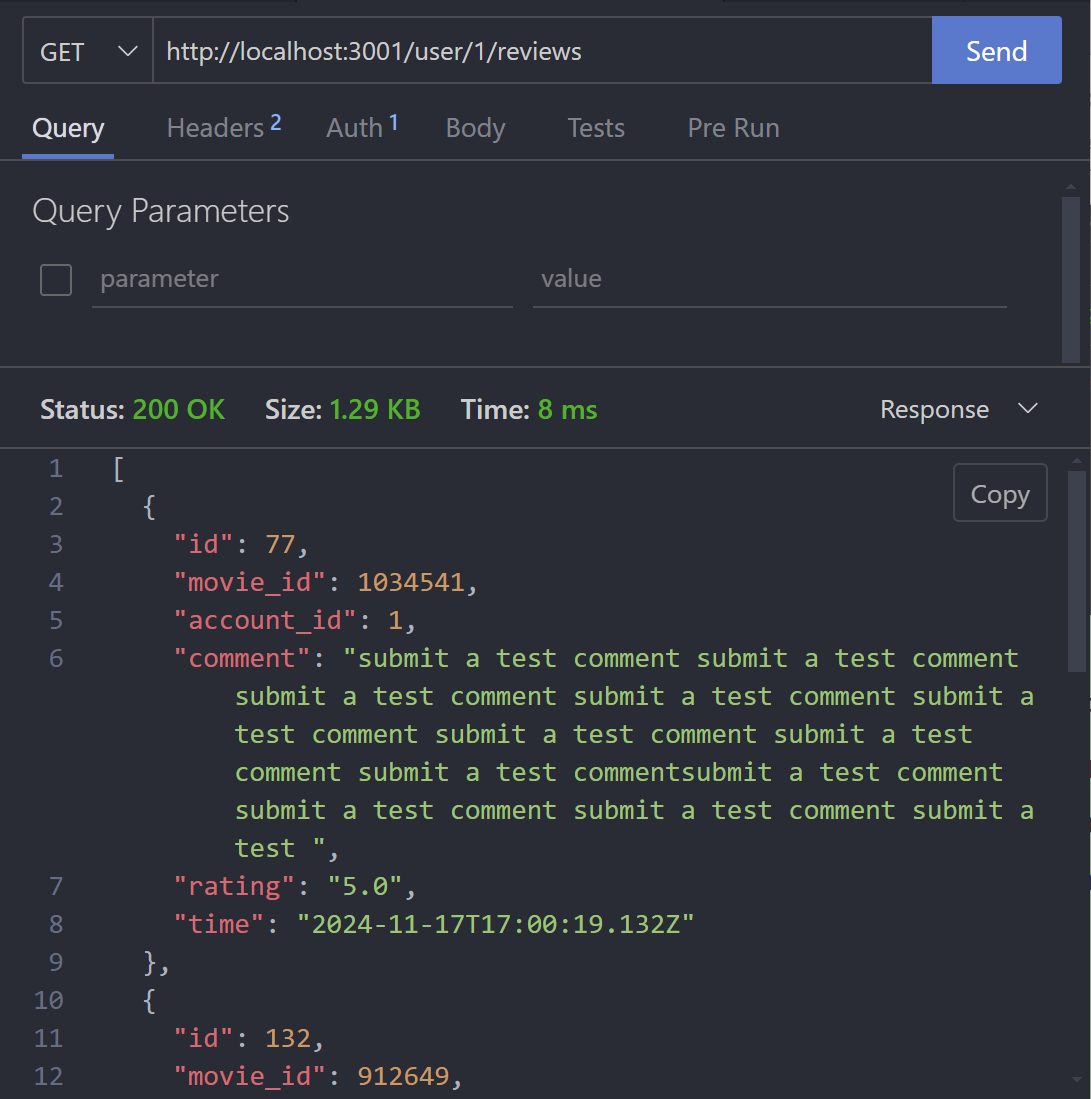


**9. Get Reviews by User**

Endpoint: GET /:accountId/reviews

Description: Get all reviews written by a user.

Response:200 OK: Returns a list of reviews.



**10. Get Share Information**

Endpoint: GET /share/:accountId

Description: Retrieve share information for a user's profile.

Response:

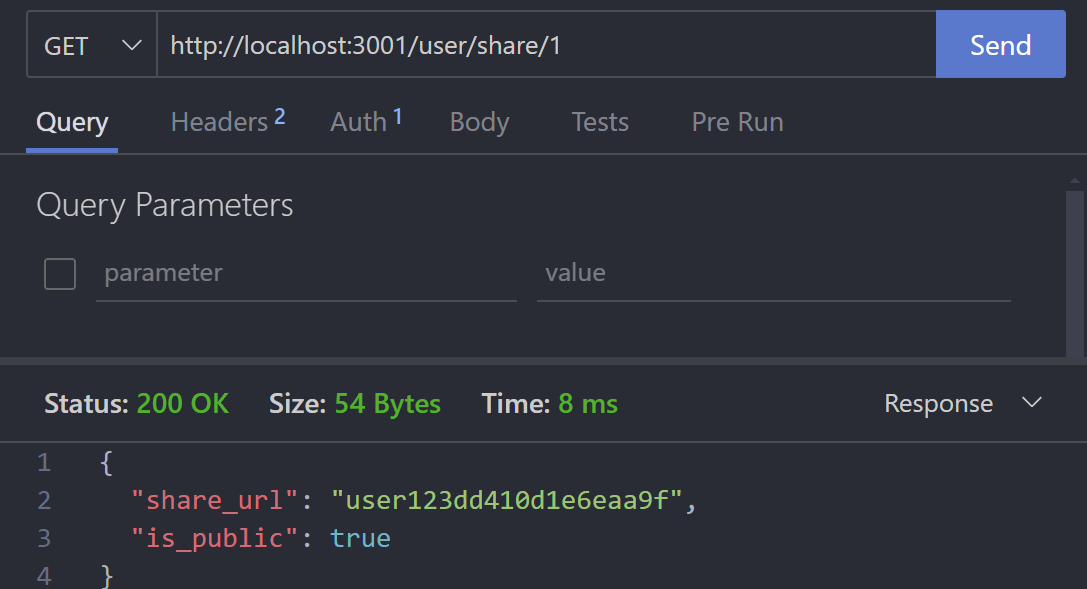
200 OK:

{

"share\_url": "string",

"is\_public":"boolean"

}



**11. Update Share Visibility**

Endpoint: PUT /share/:accountId/visibility

Description: Update the visibility of a user's shared profile.

Request Body

{

"isPublic": "boolean"

}

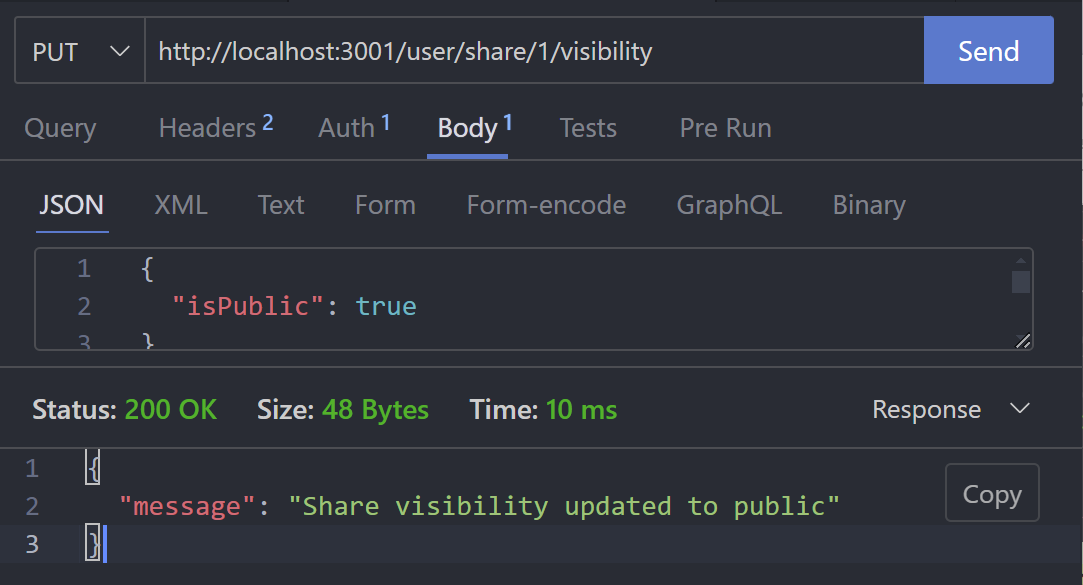
Response:

200 OK:

{

"message": "Share visibility updated to public/private"

}



**12. Get Favorites by Share URL**

Endpoint: GET /share/public/:shareUrl

Description: Retrieve a user's public favorites using their share URL.

Response:

200 OK:

{

"accountId": "integer",

"email": "string",

"favorites": [

{

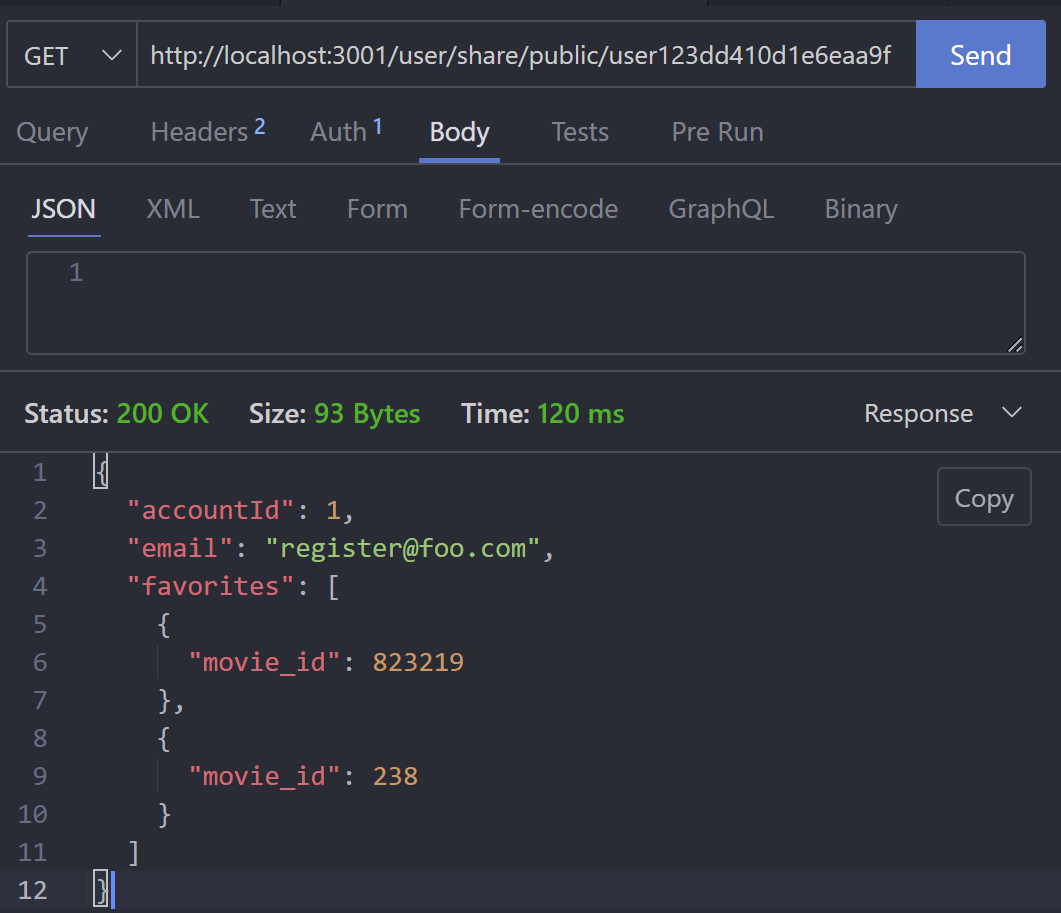
"movie\_id": "integer"

}

]

}

403 Forbidden



**13. Get All Public Shares**

Endpoint: GET /shares

Description: Retrieve a list of all publicly shared profiles.

Response:

200 OK: Returns a list of public shares.

[

{

"email": "string",

"share\_url": "string"

},

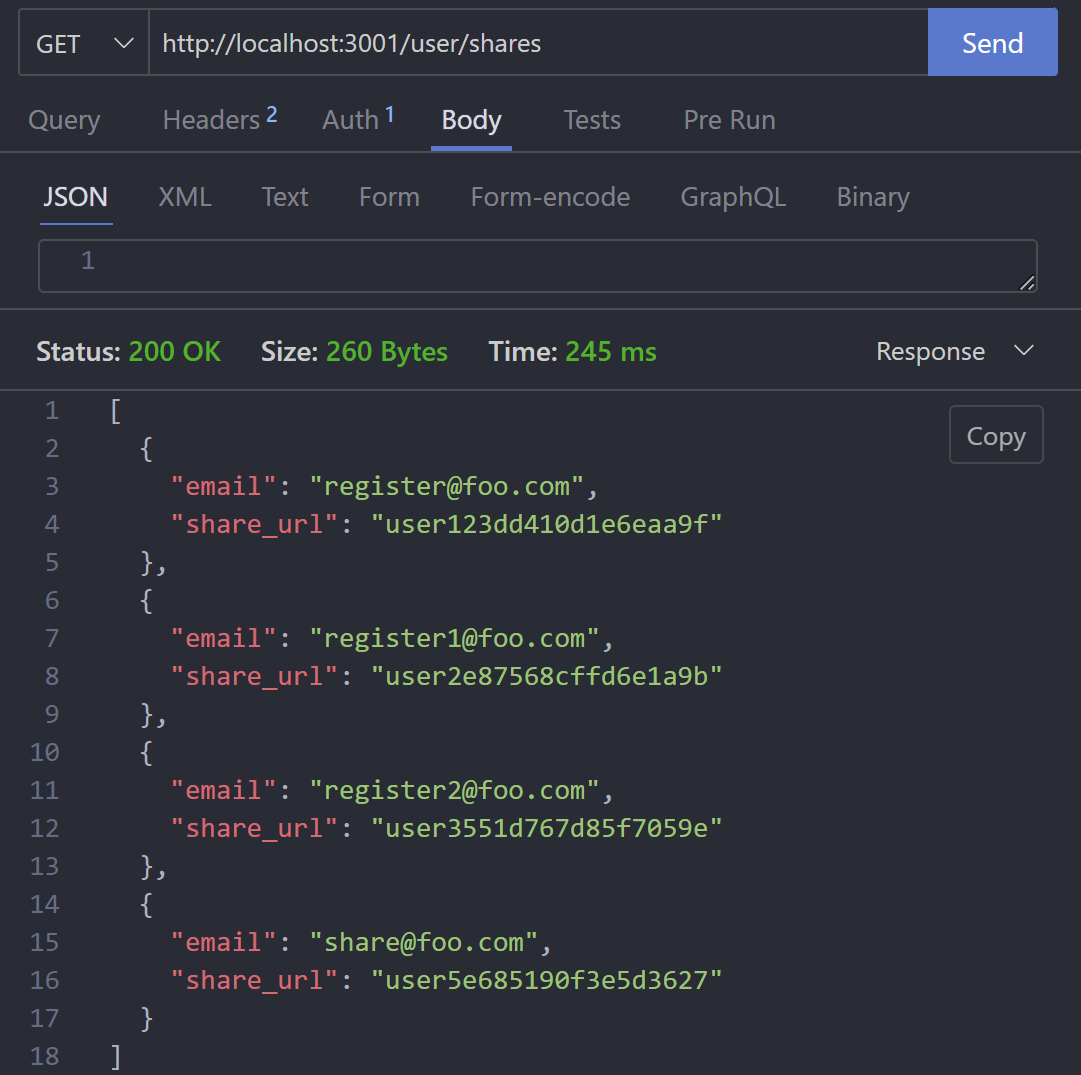
{

"email": "string",

"share\_url": "string"

}

]



**Movie Router:**

**1. Get Reviews by Movie**

Endpoint: GET /:movieId/reviews

Description: Retrieve all reviews for a specific movie.

Request Parameters:movieId (path): ID of the movie.

Response:

200 OK: Returns a list of reviews.

[

{

"id": "number",

"movie\_id": "number",

"account\_id": "number",

"comment": "string",

"rating": "number",

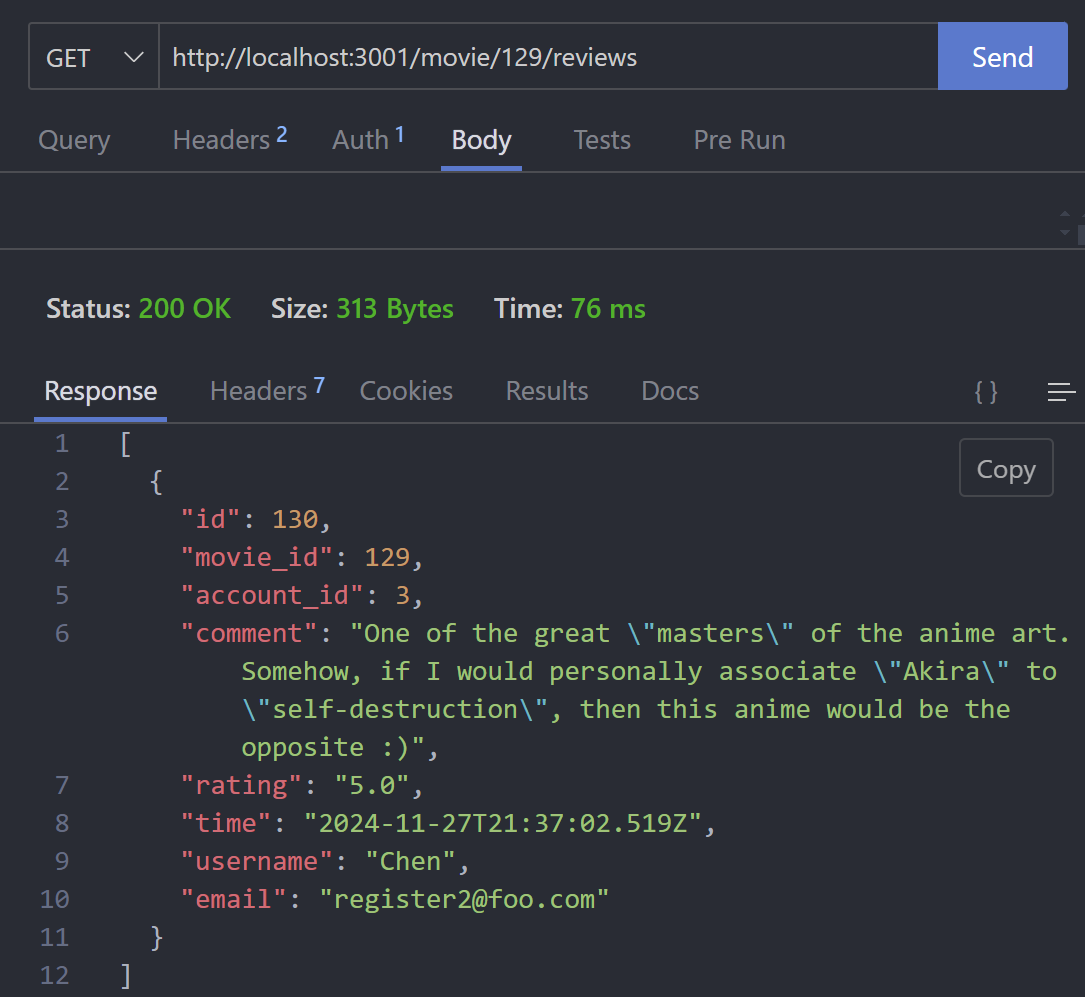
"time": "string (ISO format)",

"username": "string"

}

]

500 Internal Server Error: An error occurred.



**2. Post a New Review**

Endpoint: POST /newreview

Description: Add a new review for a movie.

Request Body:

{

"movieId": "number",

"accountId": "number",

"rating": "number",

"comment": "string"

}

Response:

201 Created: Successfully created the review.

{

"id": "number",

"movie\_id": "number",

"account\_id": "number",

"comment": "string",

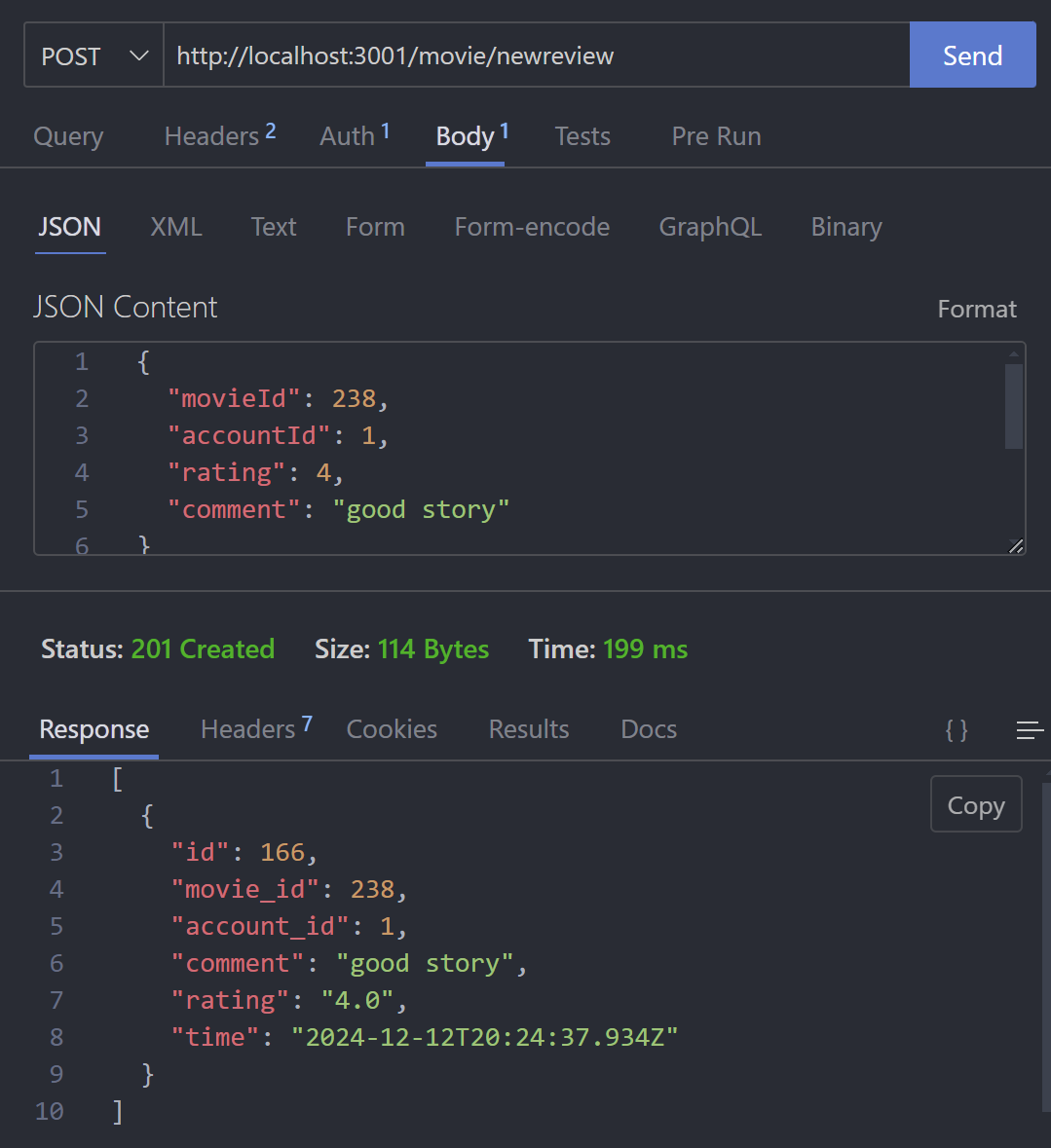
"rating": "number",

"time": "string (ISO format)"

}

400 Bad Request: Missing required fields.

500 Internal Server Error: An error occurred.



**3. Get Average Rating**

Endpoint: GET /:movieId/rating

Description: Retrieve the average rating for a specific movie.

Request Parameters:movieId (path): ID of the movie.

Response:

200 OK: Returns the average rating

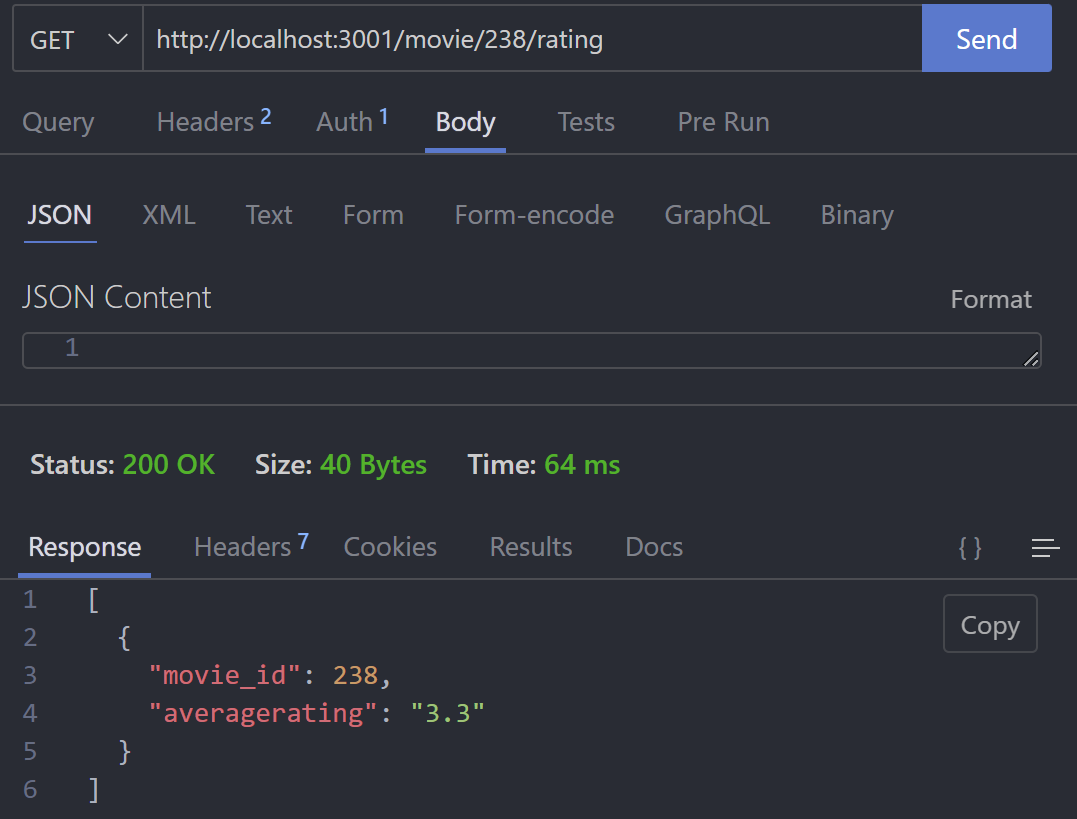
{

"movie\_id": "number",

"averagerating": "number"

}

500 Internal Server Error: An error occurred.



**4. Add or Remove a Movie from Favorites**

Endpoint: POST /favorites

Description: Add a movie to or remove it from the user's favorites list.

Request Body:

{

"accountId": "number",

"movieId": "number",

"favorite": "boolean" // true to add, false to remove

}

Response:

201 Created: Successfully added to favorites

{

"success": true,

"message": "Movie added to favorites",

"data": {

"id": "number",

"accountId": "number",

"movieId": "number"

}

}

200 OK: Successfully removed from favorites

{

"success": true,

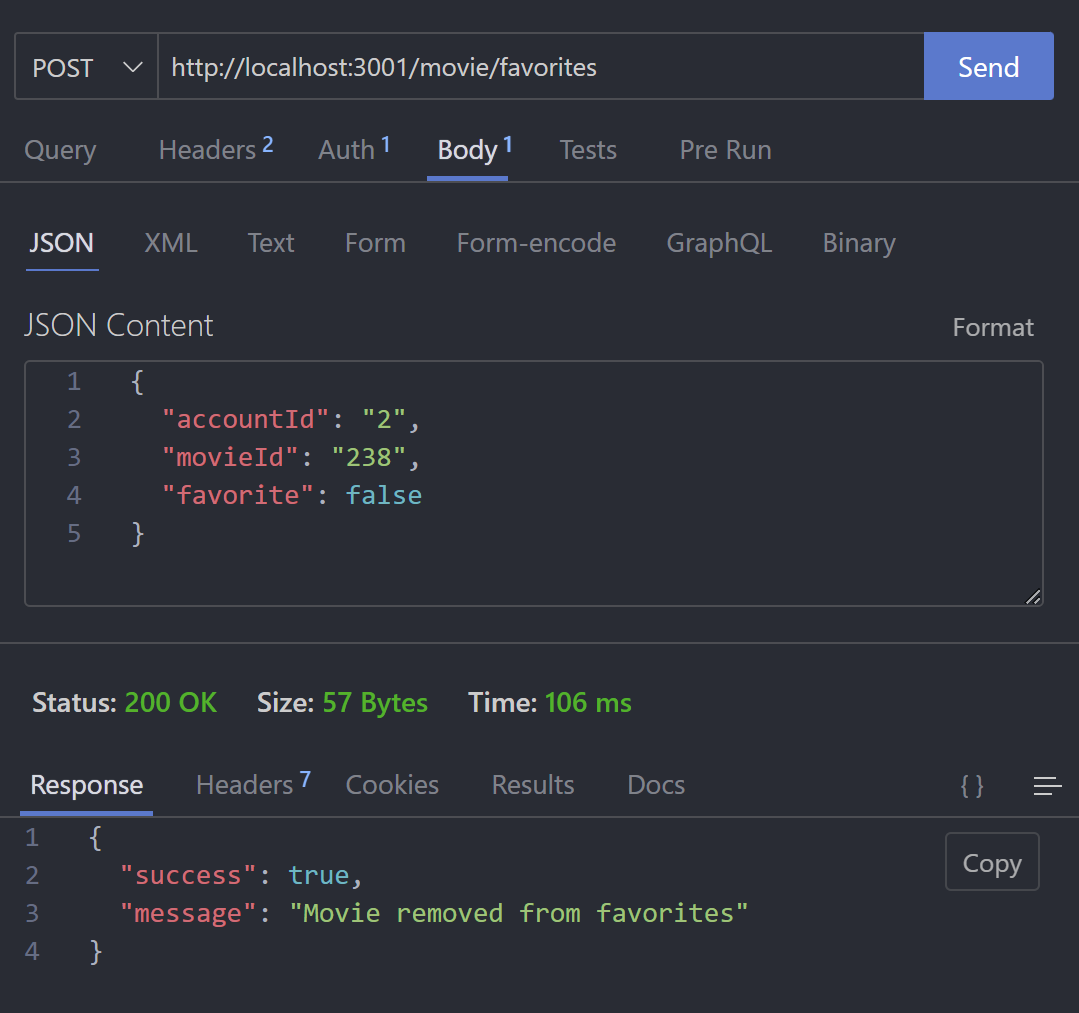
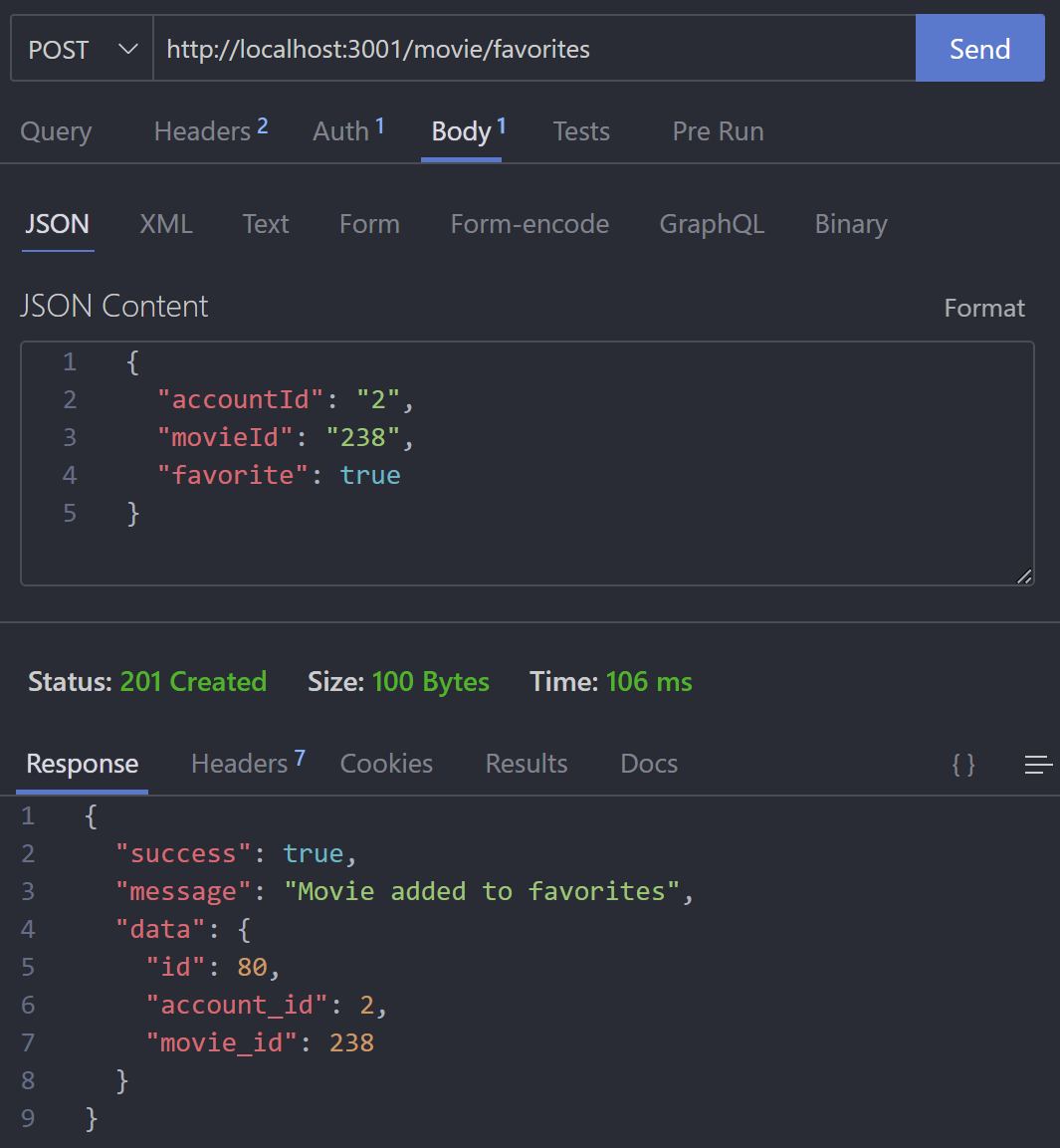
"message": "Movie removed from favorites"

}

409 Conflict: Movie is already in favorites.

404 Not Found: Movie not found in favorites.

500 Internal Server Error: An error occurred.



**5. Get Favorite Status**

Endpoint: GET /favorites/:accountId/:movieId

Description: Check if a specific movie is in the user's favorites list.

Request Parameters:accountId (path): ID of the account.

movieId (path): ID of the movie.

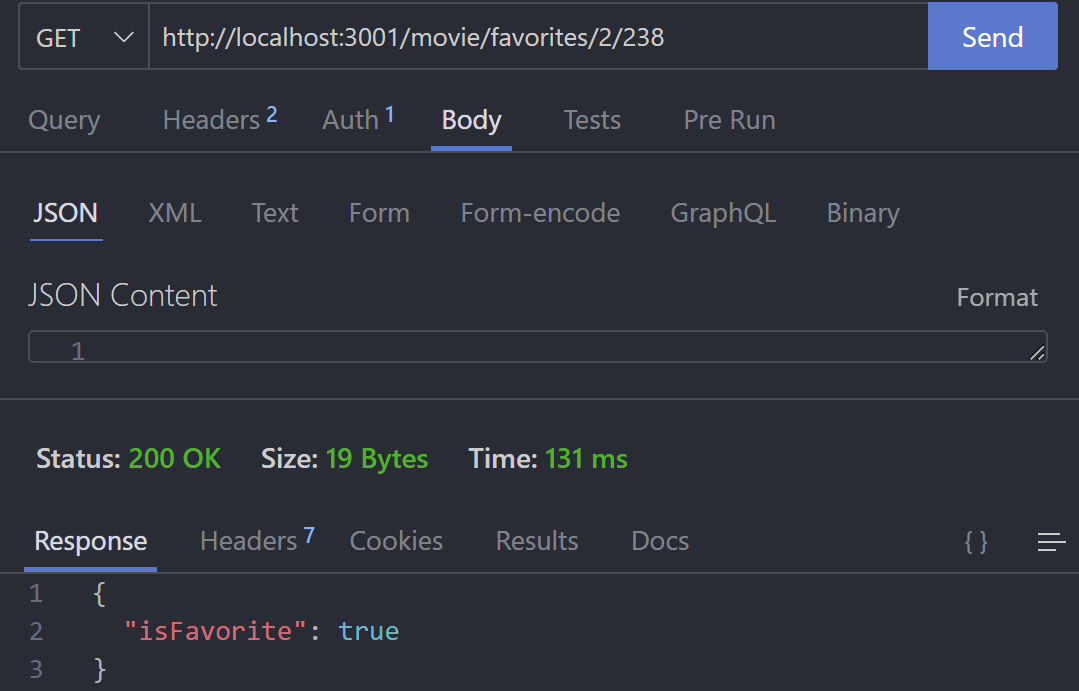
Response:200 OK: Returns the favorite status

{

"isFavorite": "boolean"

}

500 Internal Server Error: An error occurred



**6. Get All Favorites by User**

Endpoint: GET /favorites/:accountId

Description: Retrieve all favorite movies for a user.

Request Parameters:accountId (path): ID of the account.

Response:200 OK: Returns a list of favorite movies

[

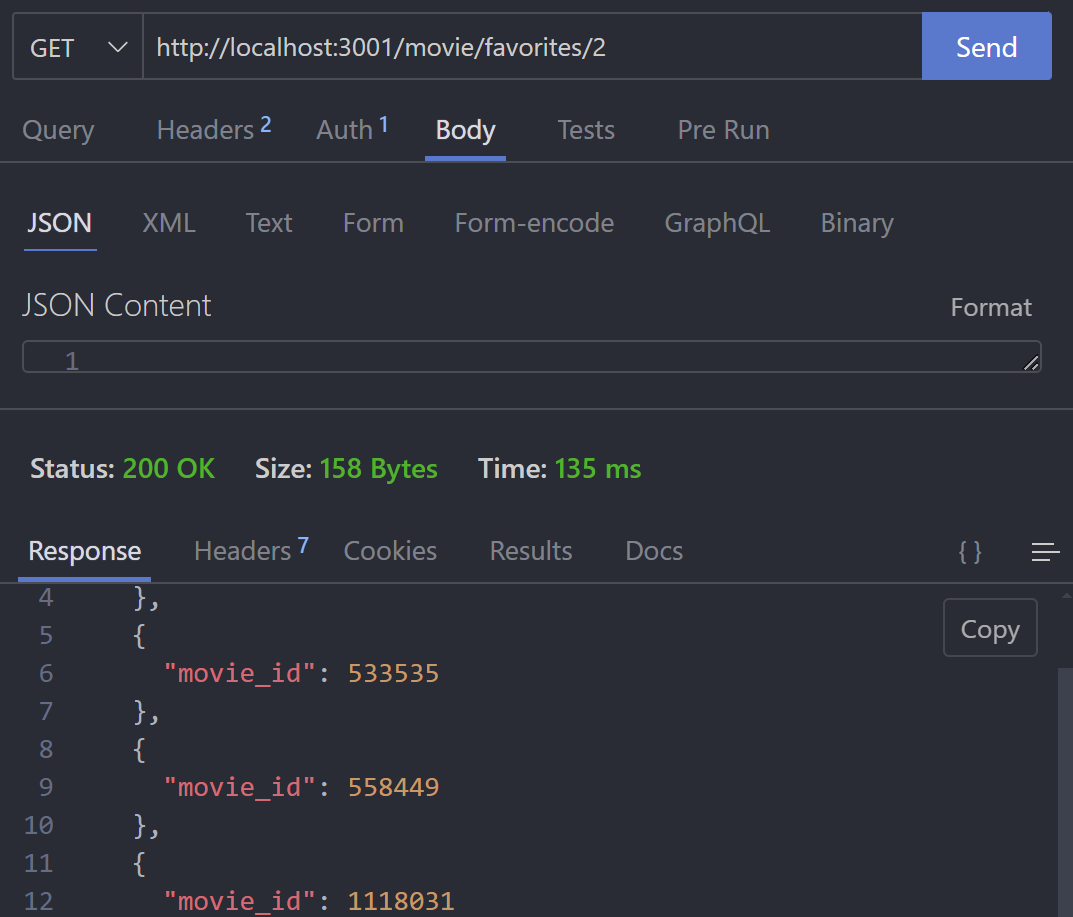
{

"movie\_id": "number"

}

]

500 Internal Server Error: An error occurred.



**Group Router**

**1. Get All Groups**

Endpoint: GET /

Description: Retrieve all groups.

Response:200 OK: Returns a list of groups

[

{

"group\_id": "number",

"group\_name": "string",

"owner\_id": "number",

"owner\_name": "string"

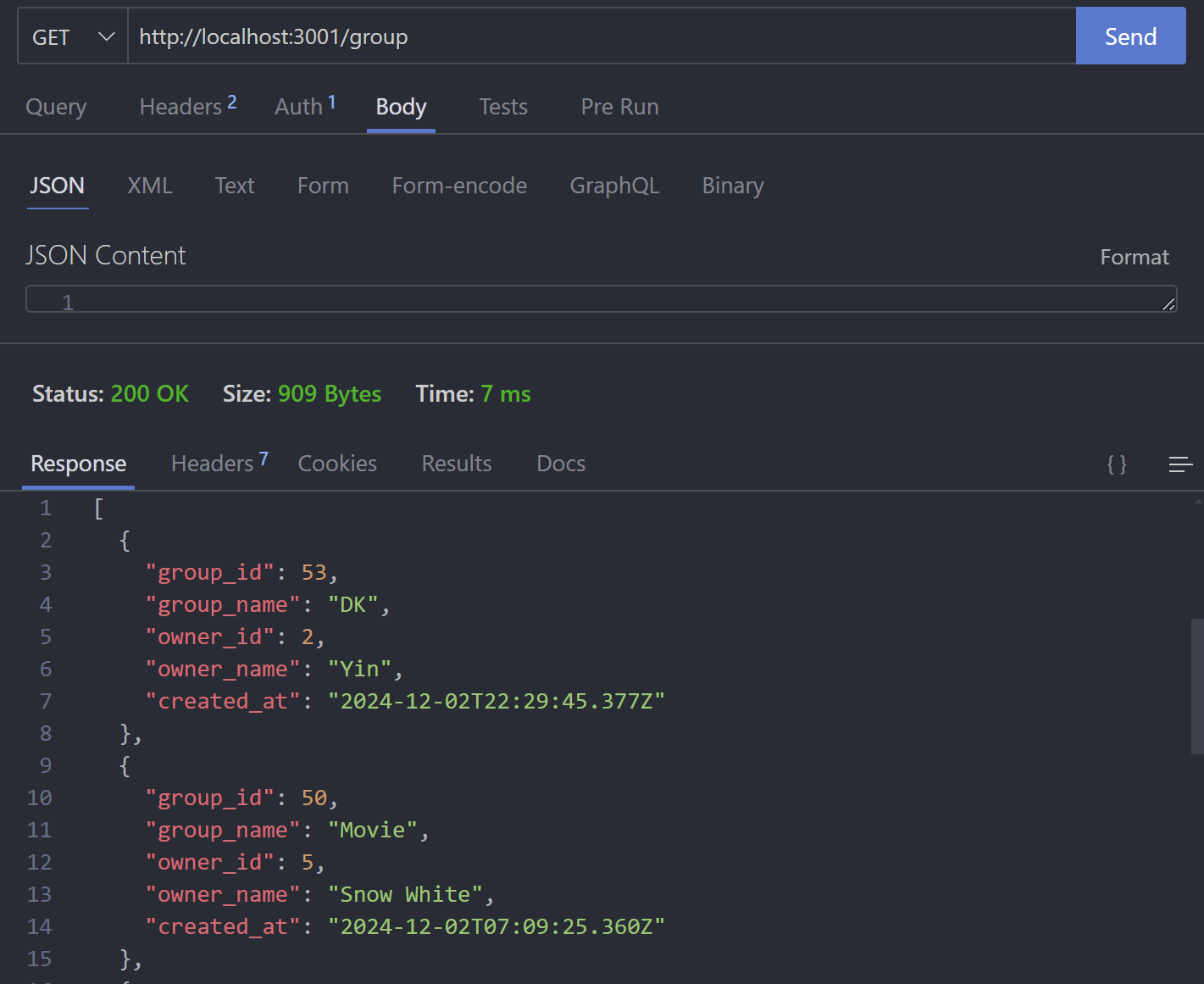
"created\_at":"string (ISO format)"

}

]

404 Not Found

500 Internal Server Error: Failed to fetch groups.



**2. Create a New Group**

Endpoint: POST /newgroup

Description: Create a new group.

Request Body:json

Copy code

{

"name": "string",

"ownerId": "number"

}

Response:201 Created: Group successfully created.json

Copy code

{

"message": "Group created successfully",

"group": {

"id": "number",

"name": "string",

"owner\_id": "number"

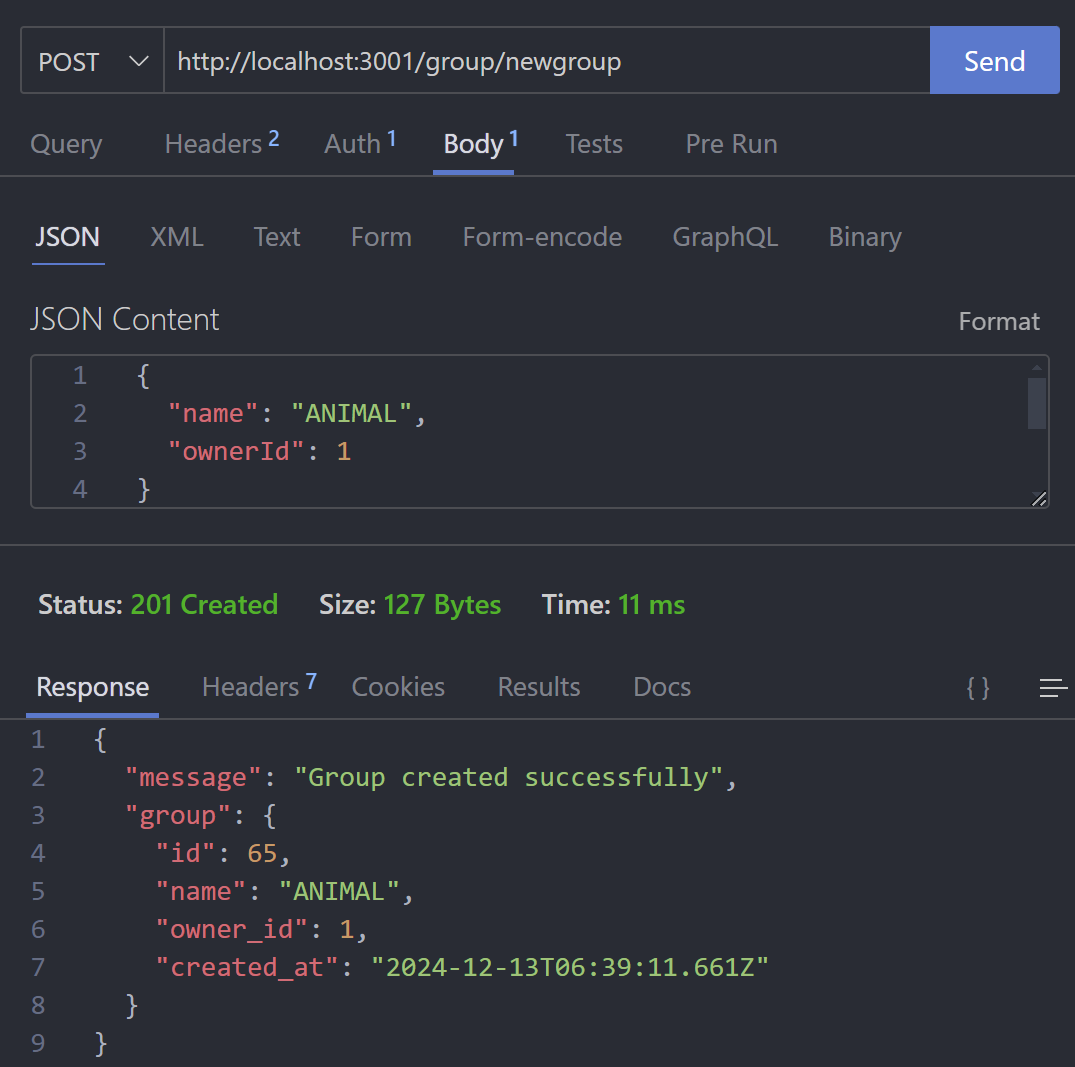
"created\_at":"string (ISO format)"

}

}

400 Bad Request: Missing required fields or group name already exists.

500 Internal Server Error: Failed to create group.



**3. Get Group Details**

Endpoint: GET /:groupId

Description: Retrieve details of a specific group.

Request Parameters:groupId (path): ID of the group.

Response:200 OK: Returns group details and members.json

Copy code

{

"groupDetails": {

"id": "number",

"name": "string",

"owner\_id": "number",

"created\_at":"string (ISO format)"

"description": "string"

},

"image": {

"type": "Buffer",

"data": "bytea",

},

"owner\_name": "string"

},

"members": [

{

"id": "number",

"username": "string"

"email": "string",

"share\_url": "string",

"is\_public": "boo",

"avatar": {

"type": "Buffer",

"data": "bytea",

},

"country": "string",

"gender": "string",

"birthday": "string (ISO format)"

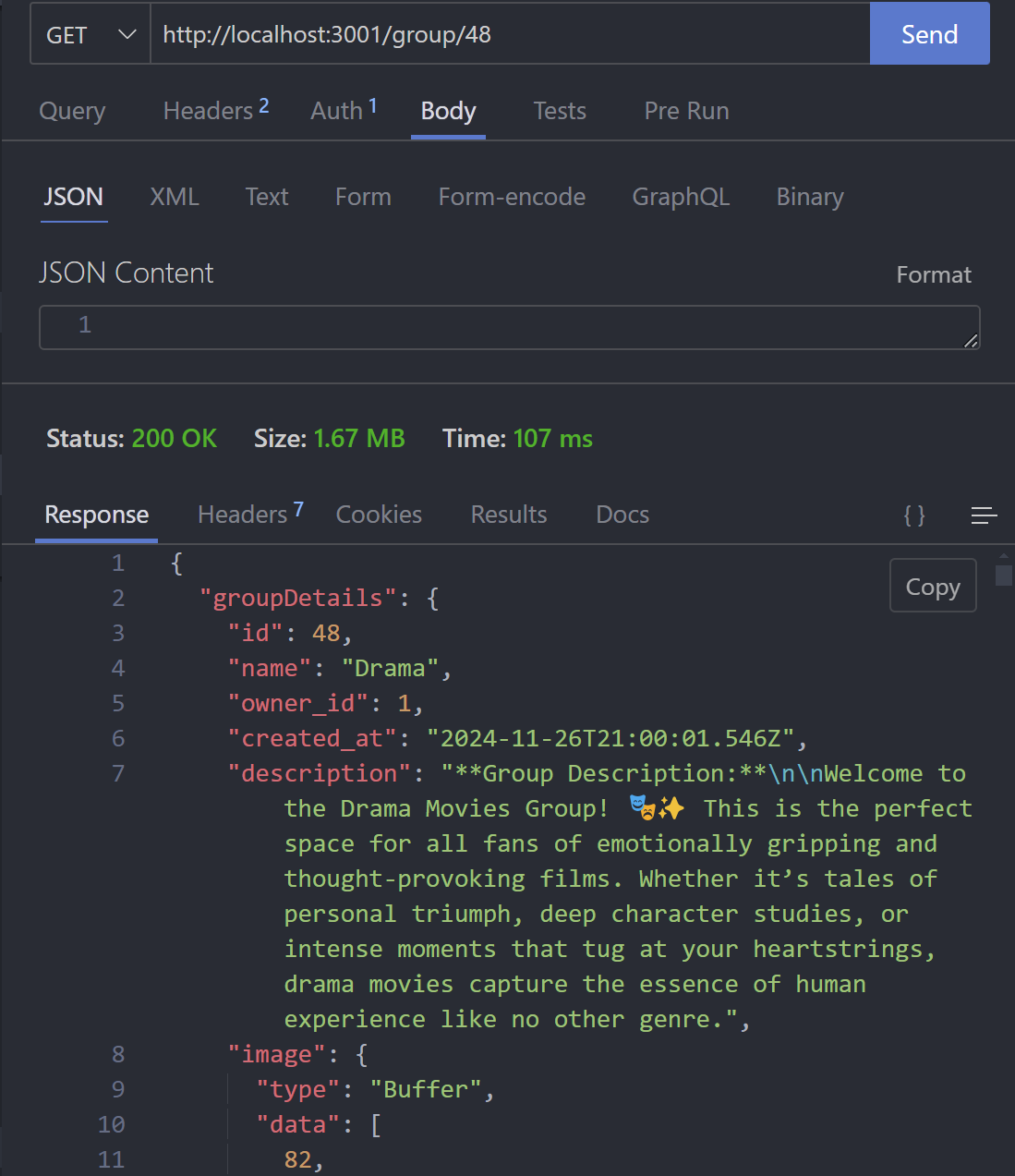
}

]

}

404 Not Found: Group not found.

500 Internal Server Error: Failed to fetch group details.



**4. Delete a Group**

Endpoint: DELETE /:groupId/delete

Description: Delete a specific group.

Request Parameters:groupId (path): ID of the group.

Response:200 OK: Group successfully deleted.json

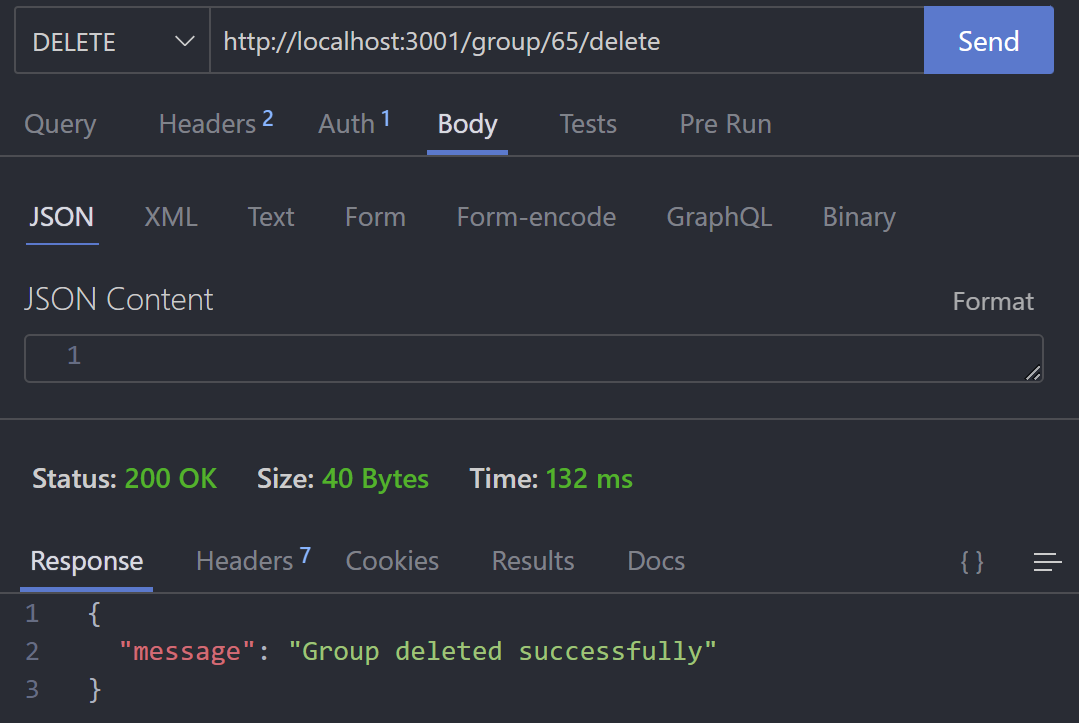
Copy code

{

"message": "Group deleted successfully"

}

500 Internal Server Error: Failed to delete group.



5. Join a Group

Endpoint: POST /:groupId/join

Description: Request to join a group.

Request Body:

{

"groupId": "number",

"accountId": "number"

}

Response:201 Created: Join request sent.json

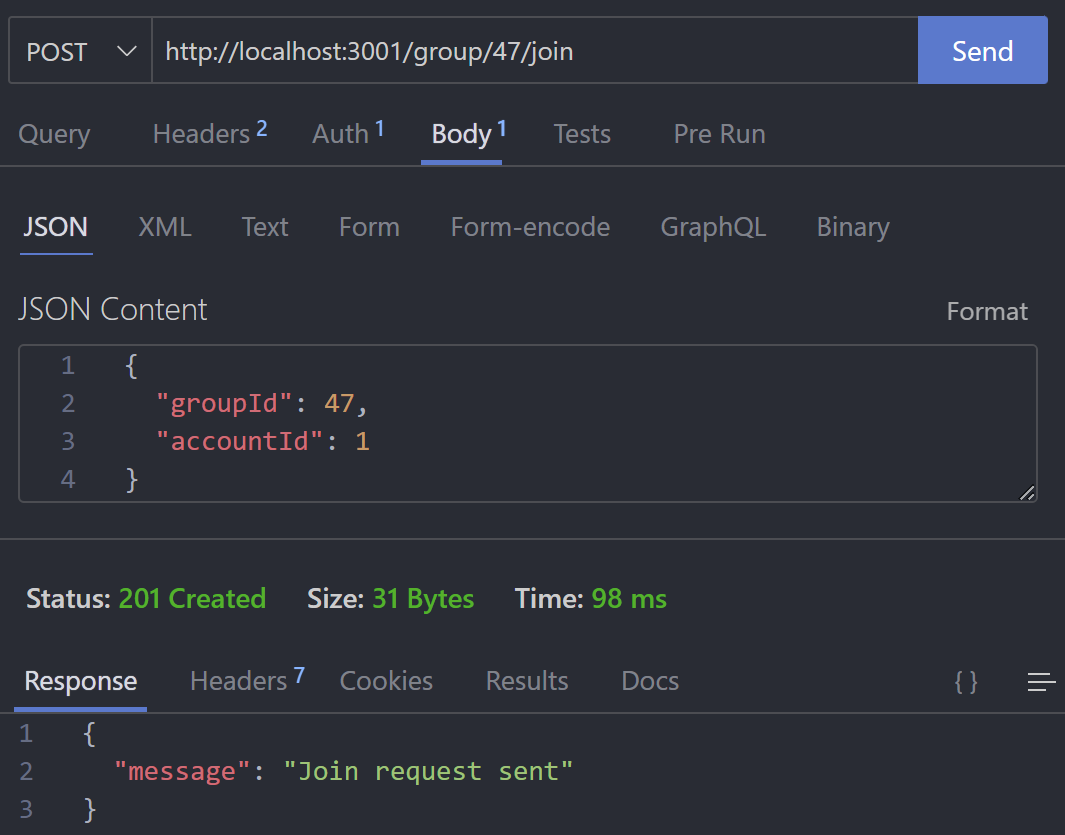
Copy code

{

"message": "Join request sent"

}

500 Internal Server Error: Failed to send join request.



**6. Get Join Requests**

Endpoint: GET /:groupId/requests

Description: Retrieve all join requests for a group.

Request Parameters:groupId (path): ID of the group.

Response:200 OK: Returns a list of join requests

{

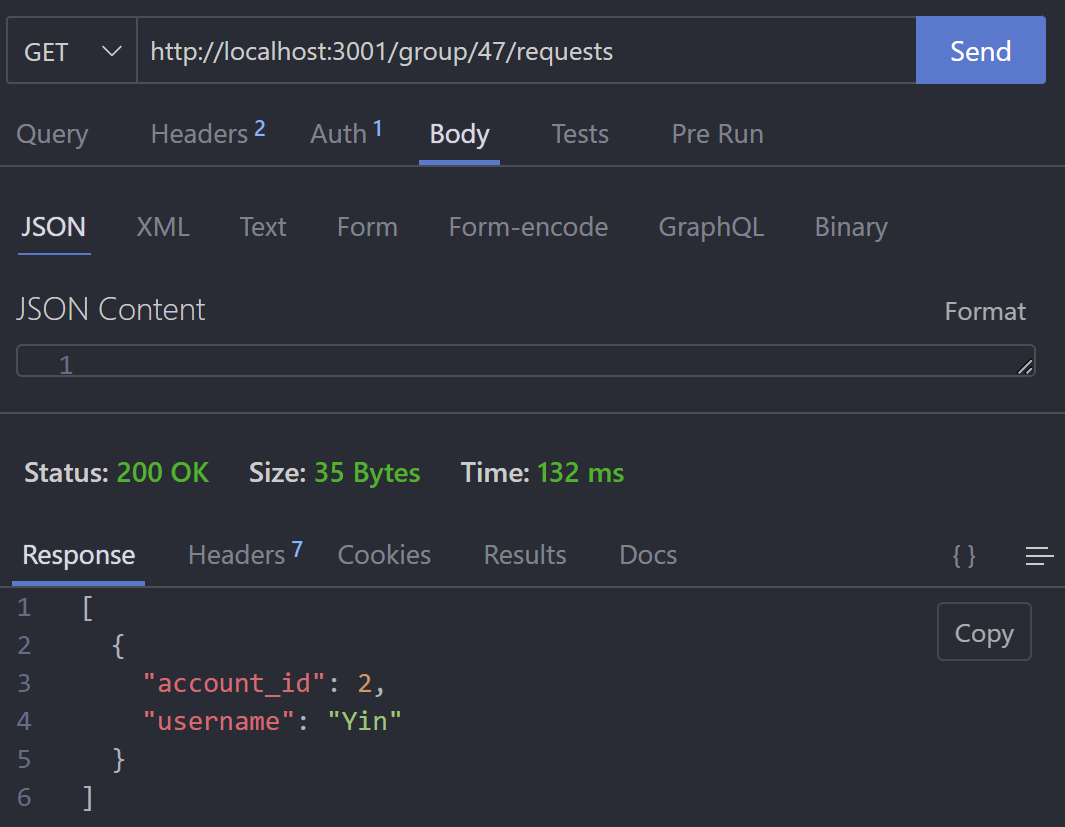
"accountId": "number",

"username": "string",

}

]

500 Internal Server Error: Failed to fetch join requests.



**7. Manage Join Requests**

Endpoint: PATCH /:groupId/members/:accountId

Description: Approve or reject a join request.

Request Body

{

"action": "string (approve/reject)"

}

Response:200 OK: Request processed successfully

{

"message": "Member approved successfully"

}

or

{

"message": "Member rejected and removed"

}

400 Bad Request: Invalid action.

500 Internal Server Error: Failed to manage request.

**8. Remove a Member**

Endpoint: DELETE /:groupId/members/:accountId/delete

Description: Remove a member from the group.

Request Parameters:groupId (path): ID of the group.

accountId (path): ID of the member.

Response:200 OK: Member removed successfully.json

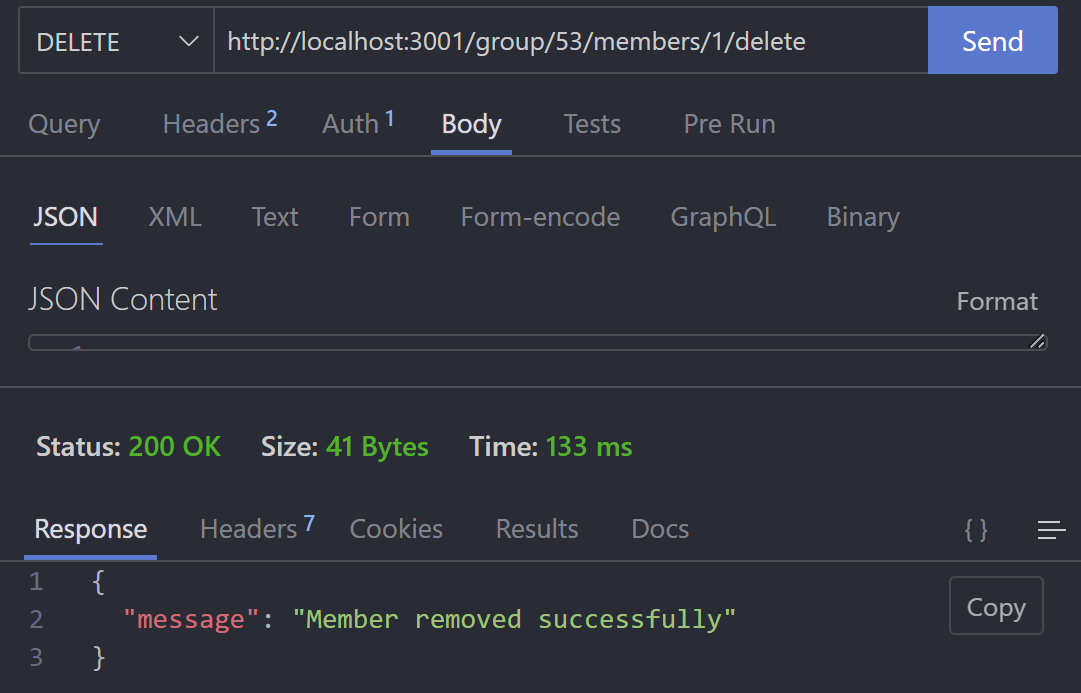
Copy code

{

"message": "Member removed successfully"

}

500 Internal Server Error: Failed to remove member.



**9. Edit Group Details**

Endpoint: PATCH /:groupId

Description: Update group details.

Request Body:json

Copy code

{

"name": "string",

"description": "string"

}

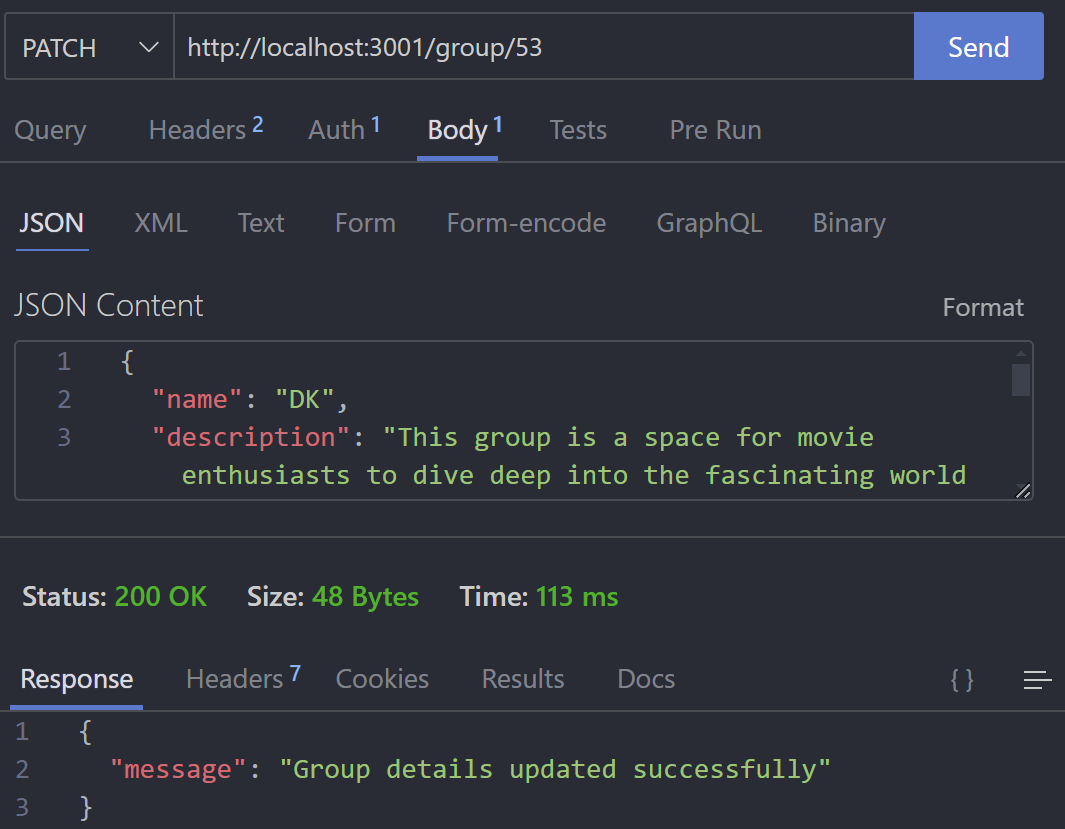
Response:200 OK: Group details updated successfully

{

"message": "Group details updated successfully"

}

500 Internal Server Error: Failed to update group details.



**10. Upload Group Image**

Endpoint: POST /:groupId/uploadimage

Description: Upload or update the group's image.

Request Body: Form-data with a key image for the file.

Response:200 OK: Returns the image in Base64 format

{

"base64Image": "string"

}

400 Bad Request: No file uploaded.

500 Internal Server Error: Failed to update group image.

**11. Get Group Image**

Endpoint: GET /:groupId/image

Description: Retrieve the group's image.

Request Parameters:groupId (path): ID of the group.

Response:200 OK: Returns the image in Base64 format.

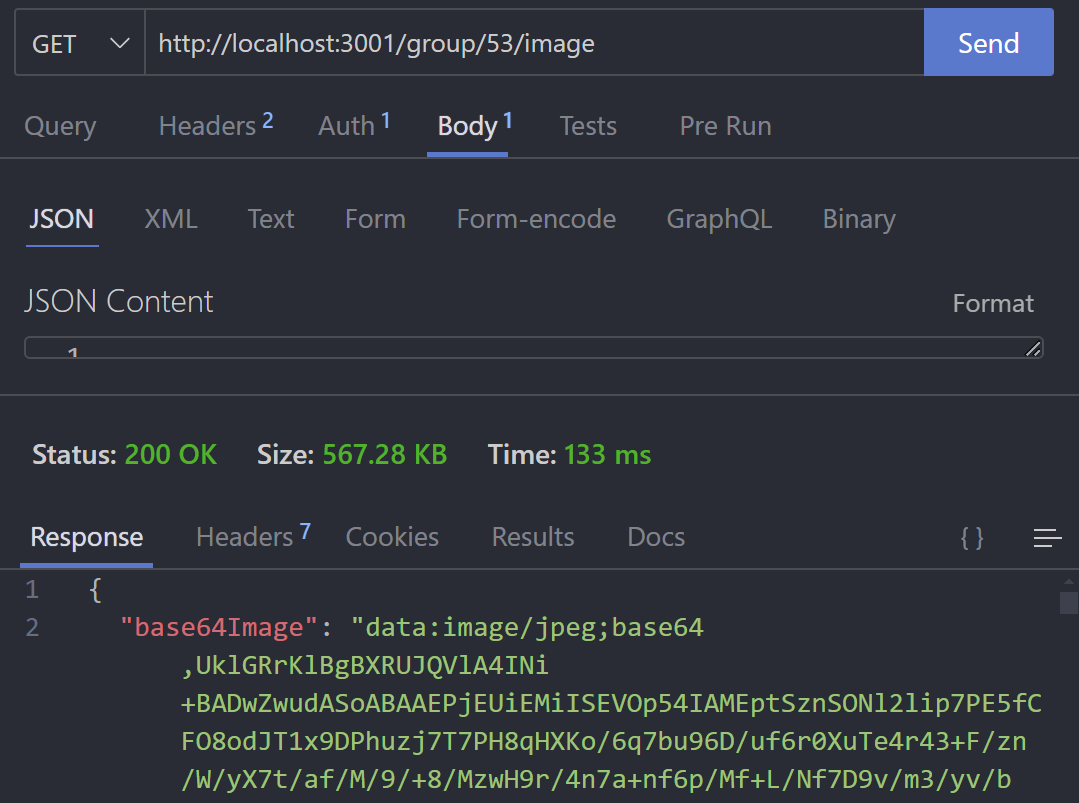
{

"base64Image": "string"

}

404 Not Found: Image not found.

500 Internal Server Error



**12. Leave a Group**

Endpoint: DELETE /:groupId/members/:userId/leave

Description: Allow a member to leave a group.

Request Parameters:groupId (path): ID of the group.

userId (path): ID of the user.

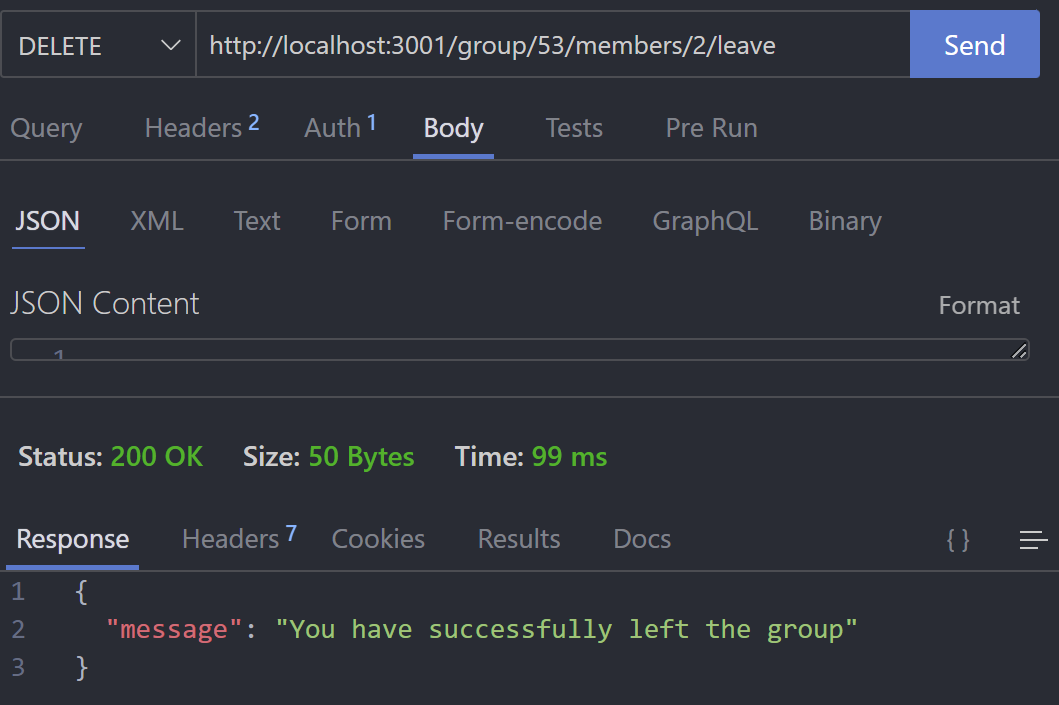
Response:200 OK: Successfully left the group.

{

"message": "You have successfully left the group"

}

500 Internal Server Error: Failed to leave group.



**13. Get User Groups**

Endpoint: GET /user/:userId

Description: Retrieve all groups a user has joined.

Request Parameters:userId (path): ID of the user.

Response:200 OK: Returns a list of groups the user is part of.json

Copy code

[

{

"id": "number",

"name": "string",

"owner\_id": "number",

"created\_at": "string (ISO format)"

"description": "string"

"avatar": {

"type": "Buffer",

"data": "bytea",

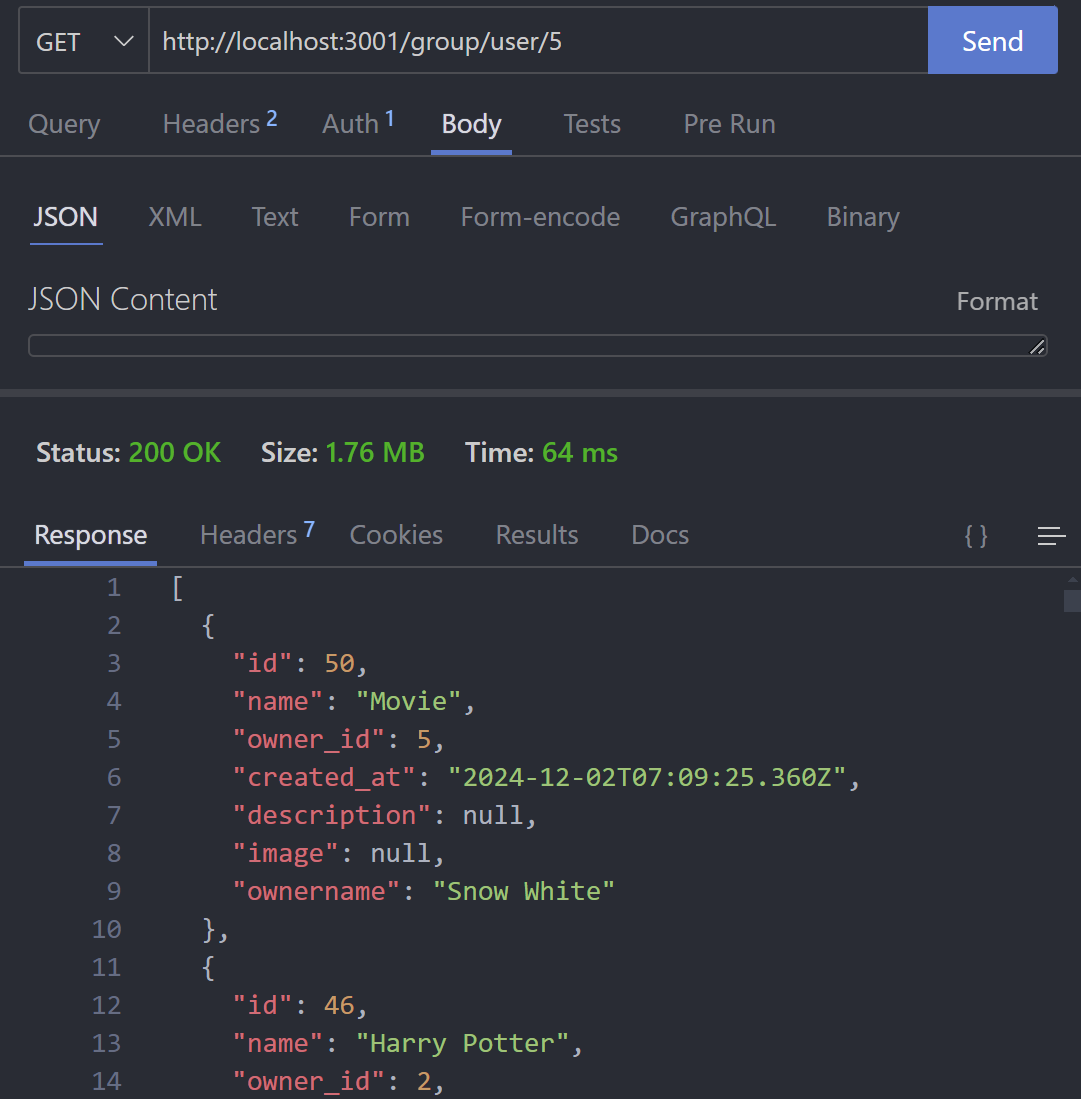
},

"ownername": "string"

}

]

500 Internal Server Error: Failed to fetch user groups.



**14. Add Content to Group**

Endpoint: POST /:groupId/addmovie

Description: Add a movie or content to the group.

Request Body

{

"userId": "number",

"contentType": "string",

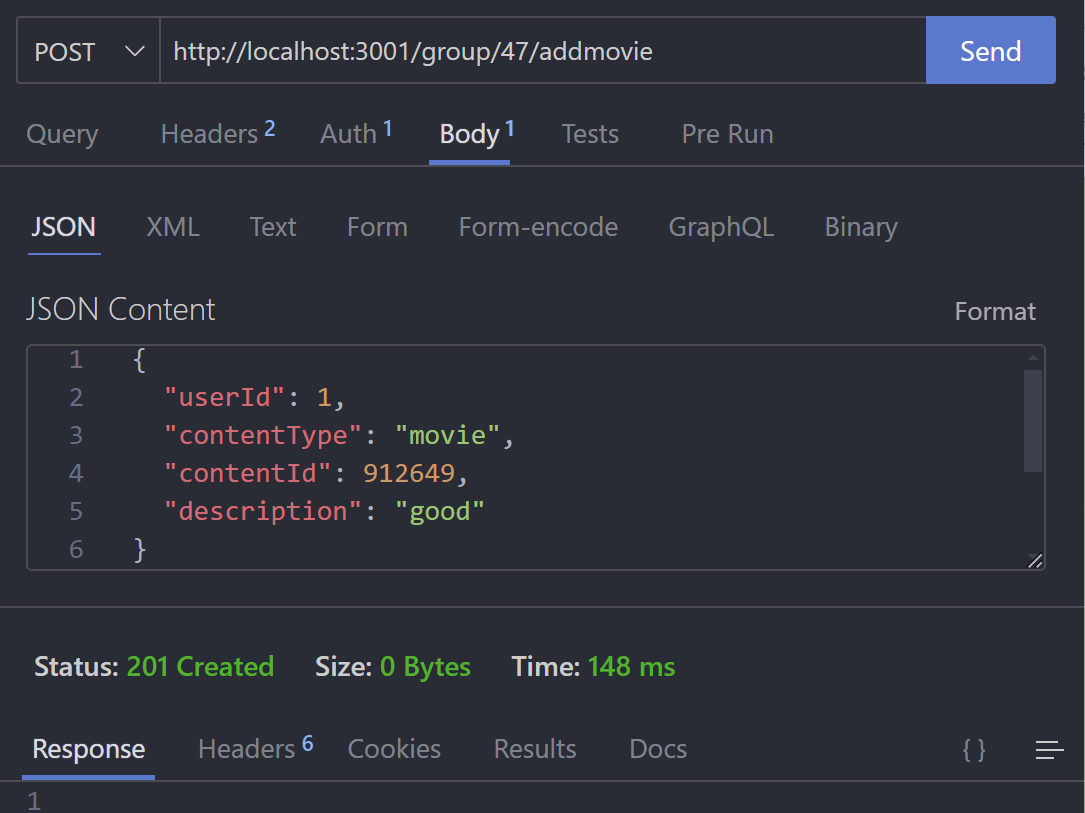
"contentId": "number",

"description": "string"

}

Response:201 Created: Content added to the group

500 Internal Server Error: Failed to add content.



**15. Get Group Content**

Endpoint: GET /:groupId/movies

Description: Retrieve content added to the group.

Request Parameters:groupId (path): ID of the group.

Response:200 OK: Returns a list of content.json

Copy code

[

{

"id": "number",

"group\_id": "number",

"added\_by": "number",

"content\_type": "string",

"content\_id": "number",

"description": "string",

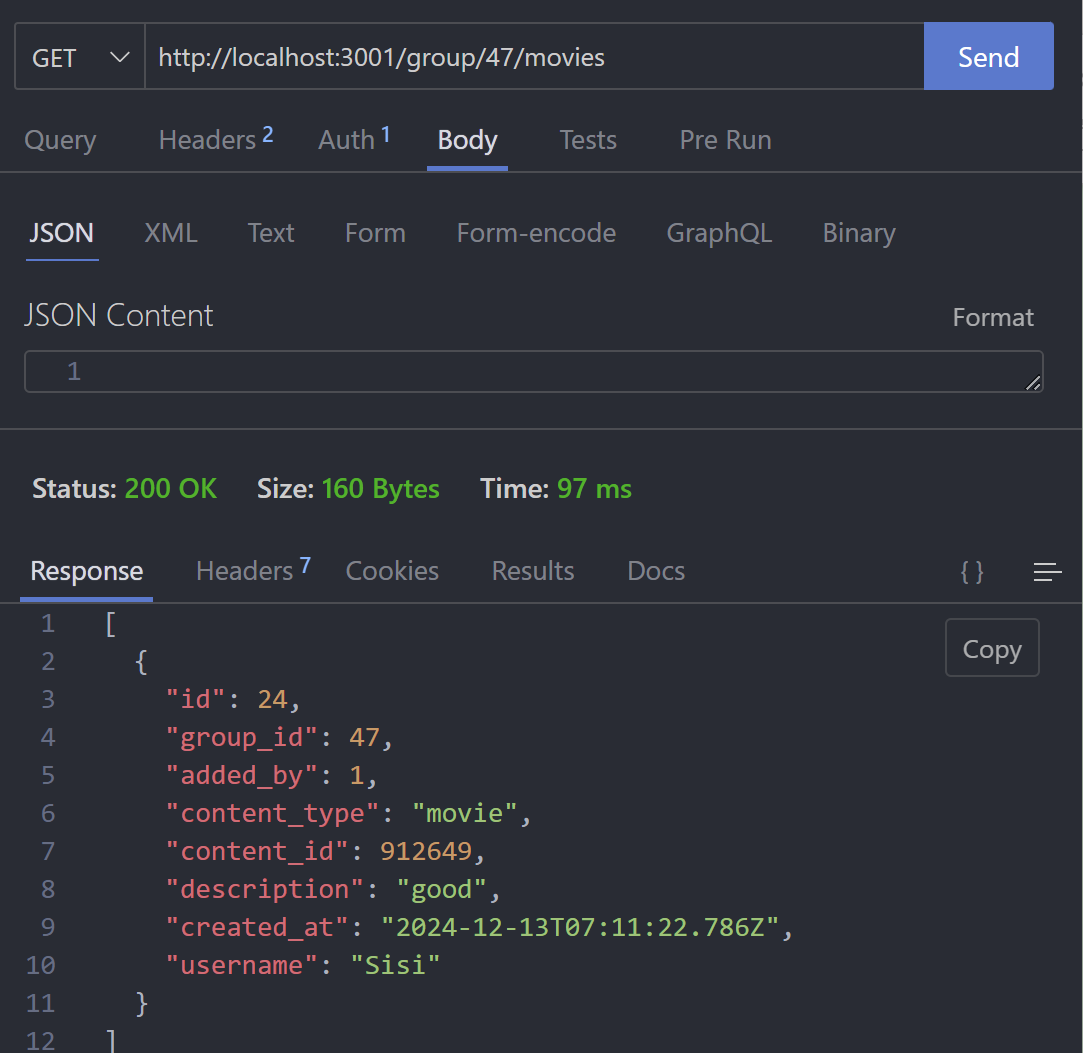
"created\_at": "string (ISO format)",

"username": "string"

}

]

500 Internal Server Error: Failed to fetch group content.



**16. Add Showtime to Group**

Endpoint: POST /:groupId/addshowtime

Description: Add a showtime to the group.

Request Body:json

Copy code

{

"userId": "number",

"movieTitle": "string",

"showTime": "string (ISO format)",

"theatre": "string",

"description": "string"

}

Response:201 Created: Showtime added to the group.json

Copy code

{

"id": "number",

"group\_id": "number",

"added\_by": "number",

"movie\_title": "string",

"show\_time": "string (ISO format)",

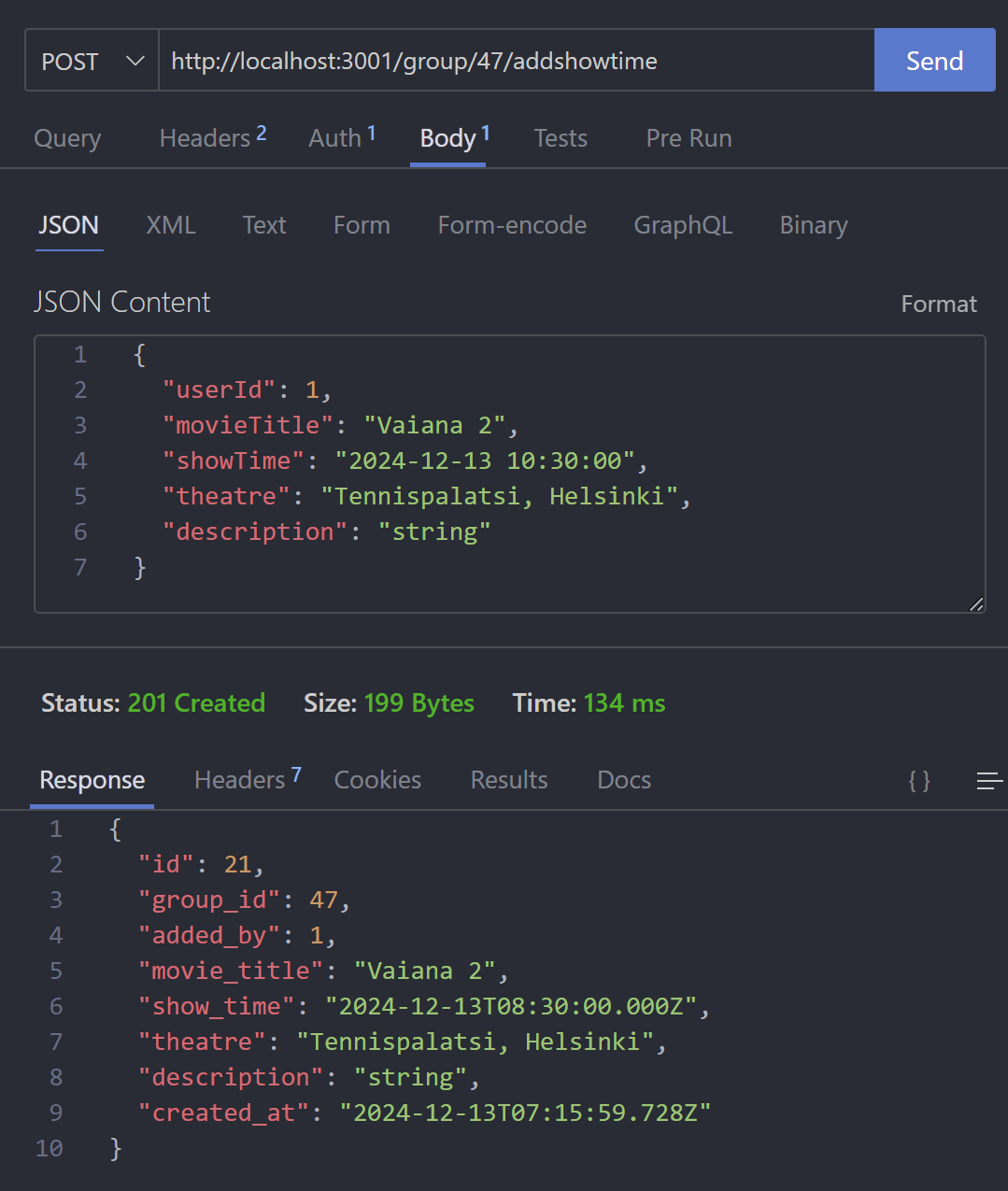
"theatre": "string",

"description": "string",

"created\_at": "string (ISO format)"

}

500 Internal Server Error: Failed to add showtime.



**17. Get Group Showtimes**

Endpoint: GET /:groupId/showtimes

Description: Retrieve all showtimes for a group.

Request Parameters:groupId (path): ID of the group.

Response:200 OK: Returns a list of showtimes.json

Copy code

[

{

"id": "number",

"group\_id": "number",

"added\_by": "number",

"movie\_title": "string",

"show\_time": "string (ISO format)",

"theatre": "string",

"description": "string",

"created\_at": "string (ISO format)",

"username": "string"

}

]

500 Internal Server Error: Failed

