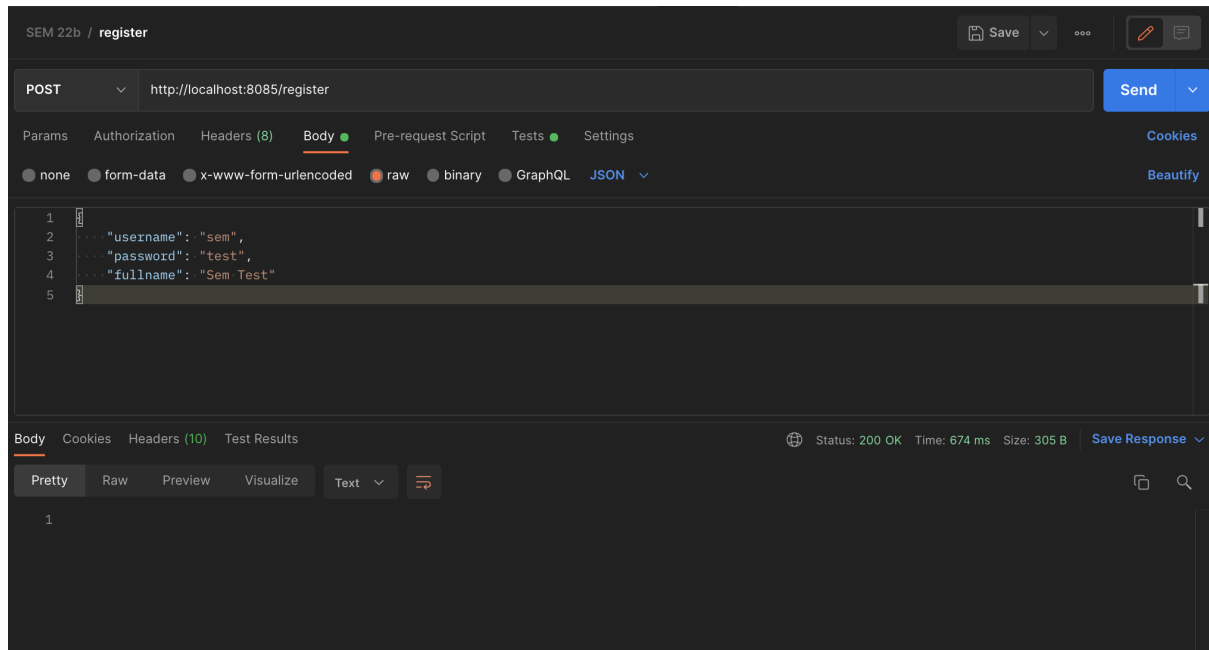


Functional Testing of our HOA application

Registering, Participating and more	2
Fetching future/past activities	6
Getting membership information	7
Creating new elections	9
Adding new options to the elections	12
Adding new votes to the elections	16
Getting results and history of elections	20
Displaying, adding, editing and deleting rules from an HOA	25
Joining an HOA	29
Leaving an HOA	33
Changing user credentials	34

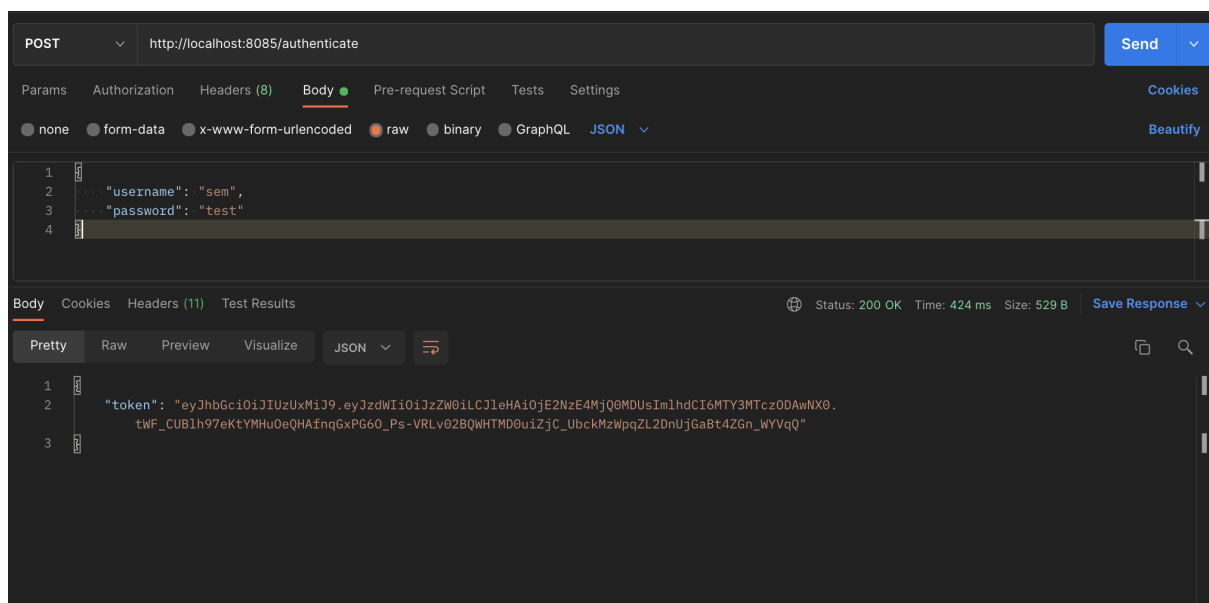
Registering, Participating and more

User Story: An user registers, creates an HOA, creates an Activity within that HOA and then participates in that Activity. The user then decides to remove participation and then remove the Activity.



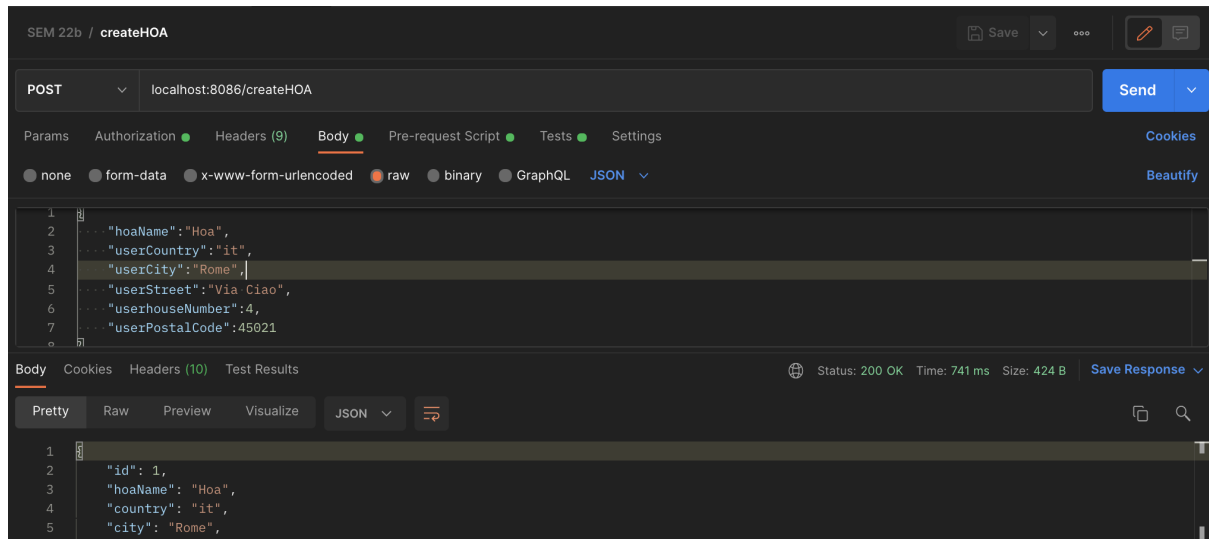
A user first registers via <http://localhost:8085/register> and in the request body, the user must include *username*, *password* and *fullname*.

If registration is successful, the user will get a **200 OK Response**.

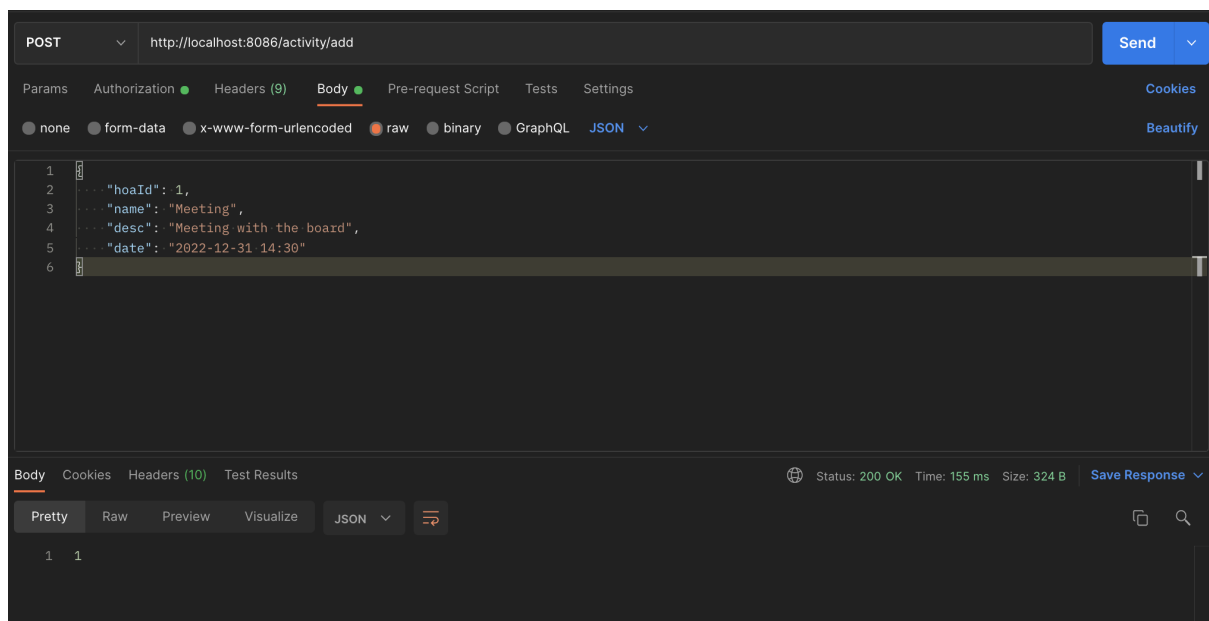


Once the user registers, the user must get the bearer token for further communication with the system. For that, the user can use <http://localhost:8085/authenticate>. In the request body, the user must submit the *username* and the *password*. If the authentication is

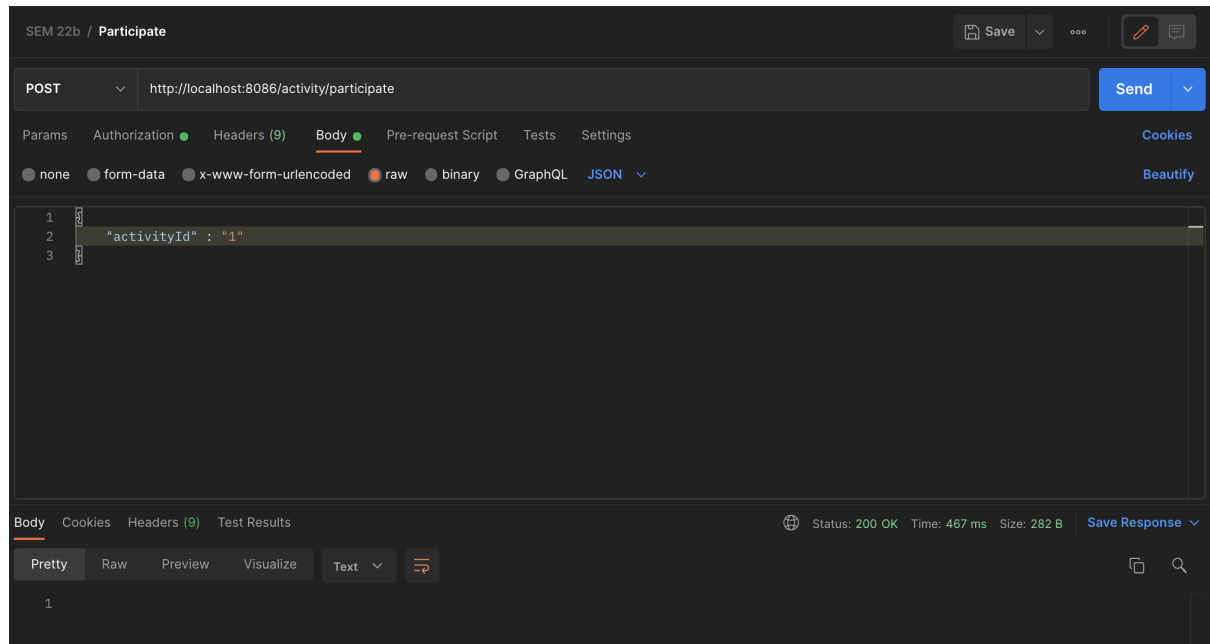
successful, you will get a **200 OK Response** with the token in the response body. This token can then be used in the authorization header of further communications.



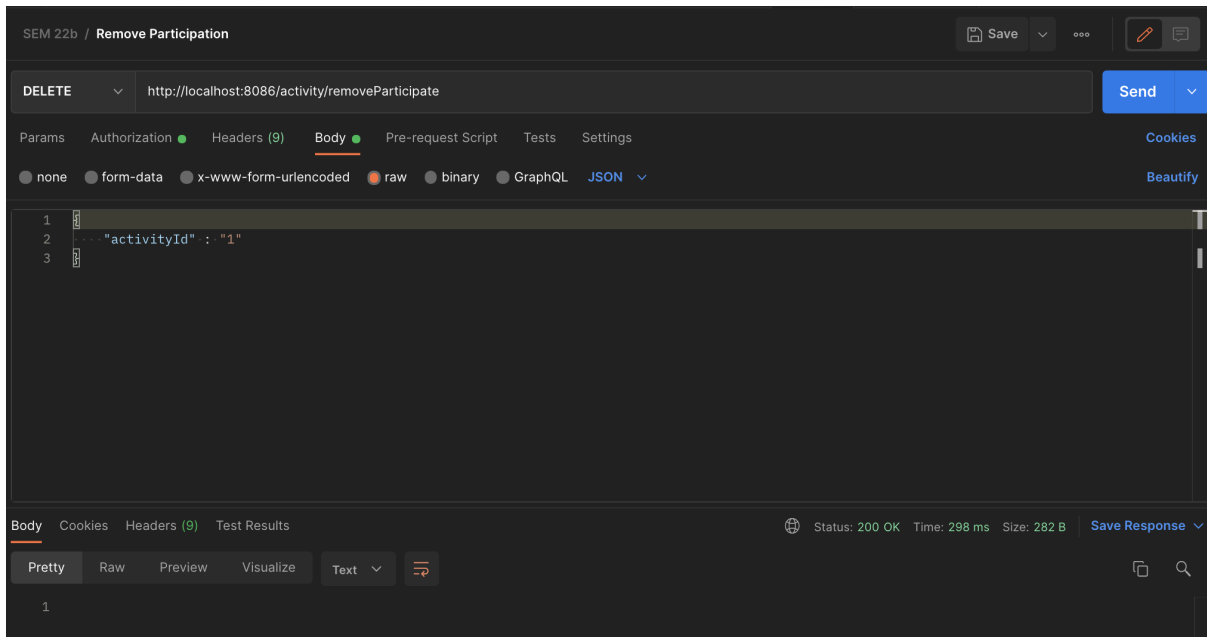
Now that the user is registered and authenticated, it is time to create our first HOA. For that, the user can use the <http://localhost:8086/createHOA>. In the request body, the user has to provide all the required details in Json format: *hoaName*, *userCountry*, *userCity*, *userStreet*, *userhouseNumber*, *userPostalCode*. Then the request can be made. If the request is successful, you will get a **200 OK Response** with all the details about the HOA in the body.



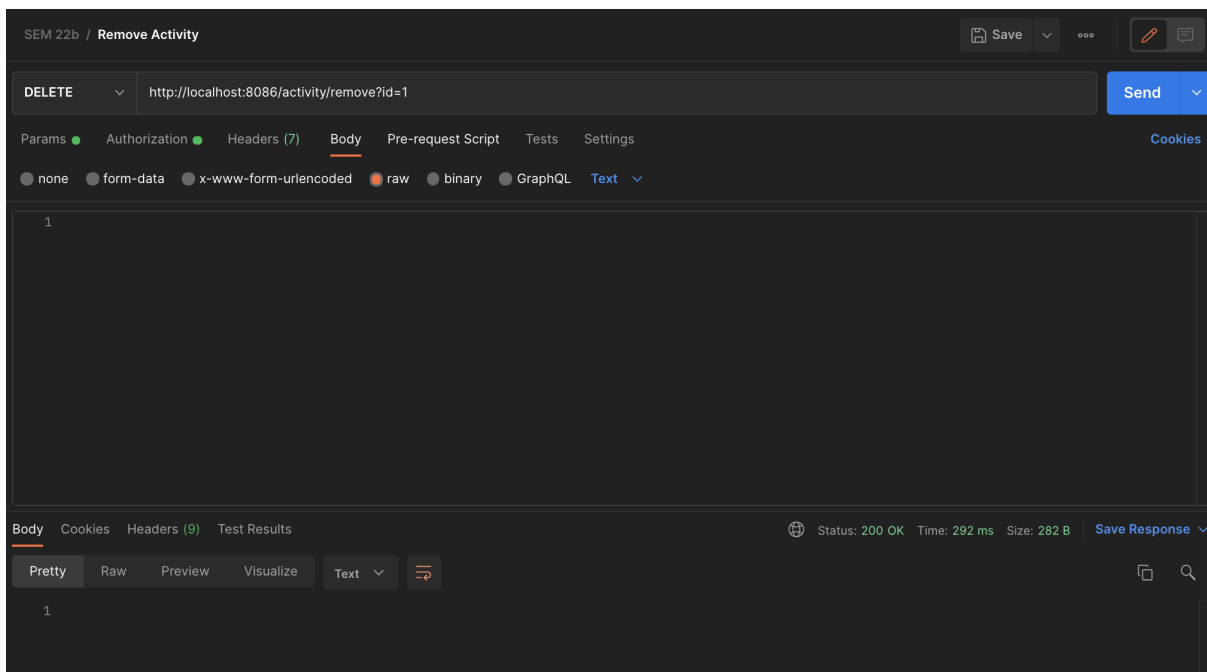
Now it is time to create an Activity. You can use the <http://localhost:8086/activity/add>. In the request body, you have to specify the *hoald*, *name*, *desc* and the *date*. If added successfully, you'll get a **200 OK Response** with the *activityId* in the response body.



Now, as a user, if you want to participate in the activity, you can use <http://localhost:8086/activity/participate>. You have to specify the `activityId` of the activity you want to participate in. It has to be in the HOA you are part of. If your participation has been noted successfully, you'll get a **200 OK Response**.



Now suddenly you decided that you can't make it and want to remove your participation? You can do that using <http://localhost:8086/activity/removeParticipate>. Just specify the activityId in the request body. If your participation has been removed successfully, then you'll get a **200 OK Response**.



For some reason, as a creator of the Activity, you decide to remove the Activity? You can do that at <http://localhost:8086/activity/remove?id=1>. You'll have to provide the activityId as a query param. If successful, you'll get a **200 OK Response**.

Fetching future/past activities

User Story: Say you already have a bunch of Activities in the database. As a user, you want to see all the future activities or all the past activities.

SELECT * FROM ACTIVITIES;

ACTIVITY_ID	CREATED_BY	DATE	DESCRIPTION	HOA_ID	NAME
1	sem	2022-12-31 15:30:00	Play with the board	1	Games
2	sem	2023-12-31 15:30:00	Meeting with the board	1	Meeting with the board
3	sem	2023-12-31 15:30:00	Meeting with the board	2	Meeting with the team
4	sem	2024-12-31 15:30:00	No	1	Discussion
5	sem	2024-05-02 16:30:00	No	1	Idk

(5 rows, 12 ms)

The current activities in the database

The screenshot shows a REST client interface. The top bar indicates a GET request to `http://localhost:8086/activity/getAllFutureActivities`. The 'Body' tab is selected, showing a JSON object: `{ "hoaId": 1 }`. The response status is 200 OK, with a time of 89 ms and a size of 773 B. The response body is displayed in a JSON array format, containing four activity objects:

```
[{"activityId": 1, "hoaId": 1, "name": "Games", "desc": "Play with the board", "date": "2022-12-31 14:30", "createdBy": "sem"}, {"activityId": 2, "hoaId": 1, "name": "Meeting with the board", "desc": "Meeting with the board", "date": "2023-12-31 14:30", "createdBy": "sem"}, {"activityId": 4, "hoaId": 1, "name": "Discussion", "desc": "No", "date": "2024-12-31 14:30", "createdBy": "sem"}, {"activityId": 5, "hoaId": 1, "name": "Idk", "desc": "No", "date": "2024-05-02 14:30", "createdBy": "sem"}]
```

Now the client wants to display all the future activities on the “abstract” notice board. The client can use the <http://localhost:8086/activity/getAllFutureActivities>. You have to mention the hoald in the request body. If successful, you’ll get a **200 OK Response** with all the future activities in that HOA in the response body.

The user can also see the past activities using <http://localhost:8086/activity/getAllPastActivities>. This can be used to access the history of an HOA.

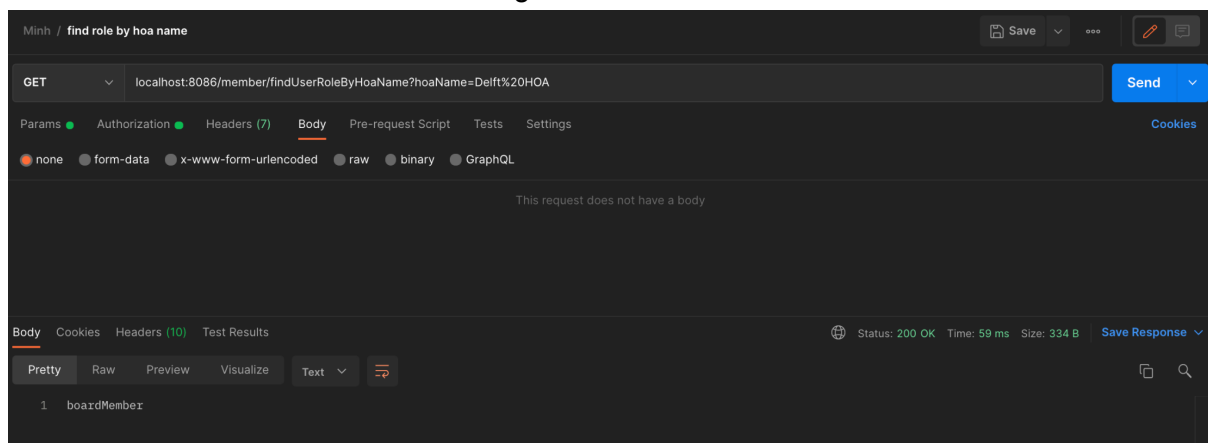
Getting membership information

User Story: *Users can get information about their memberships*

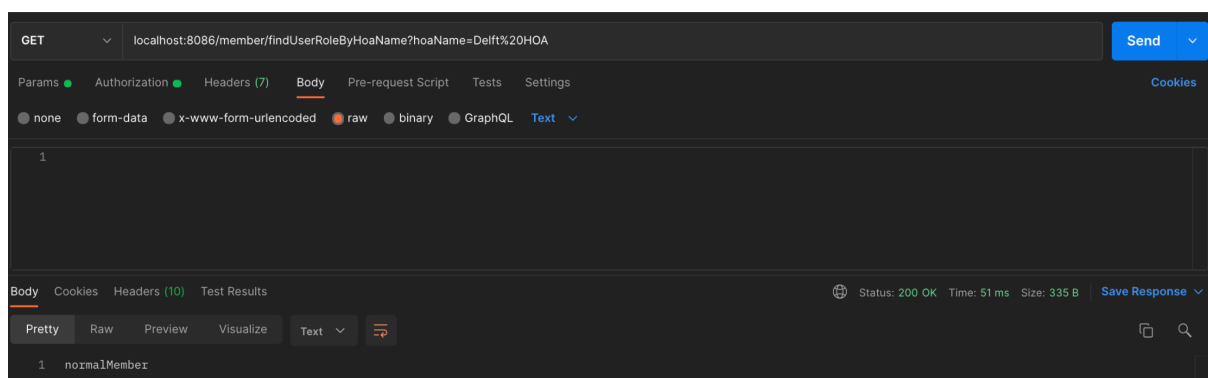
Related requirements:

- A user can check if (s)he is a board member of a specific HOA
- A user can check if (s)he is a board member of any HOAs
- A user can check if (s)he is a member of a specific HOA
- A user can see his/her joining date for a specific HOA
- A user can see his/her date of joining the board, in case that member is a board member of the HOA
- A user can see the number of board members in a HOA
- A board member can update the role of another user

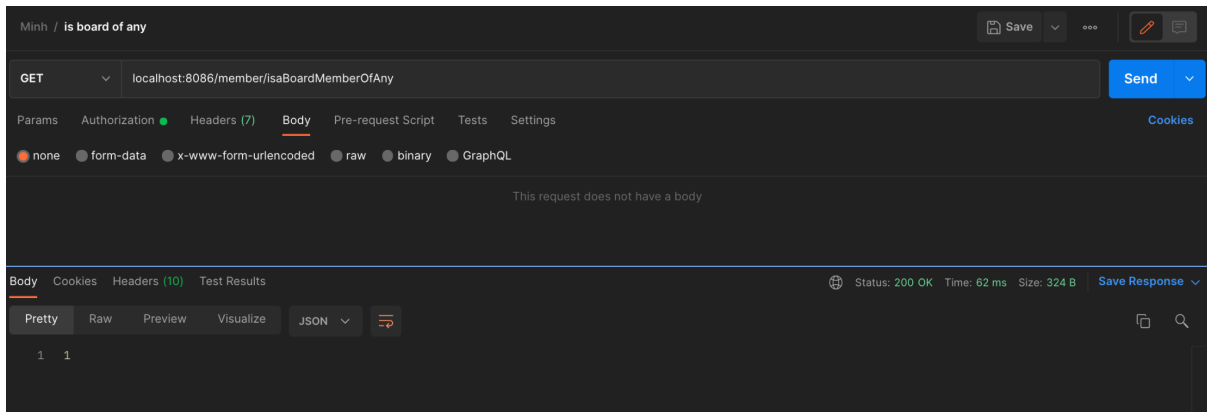
Say the user has created/joined an HOA. Now the client wants to check if the user is a boardMember? He can use the following API call to do that.



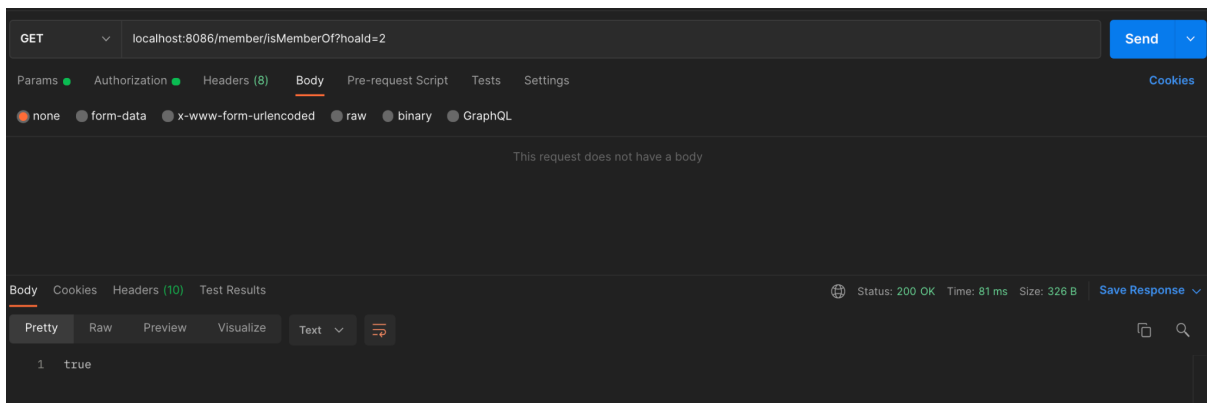
If successful, you'll get a **200 OK Response** with a string in the response body. If it was boardMember, you'll get a response with "boardMember". If not, then you'll get a response with "normalMember".



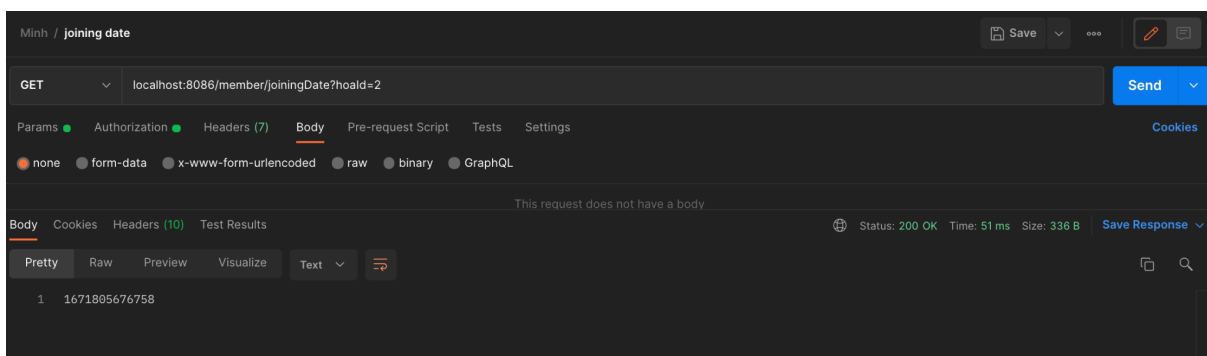
You can also use hoald to do the same. For that use this API call, <http://localhost:8086/member/findUserRoleByHoard?hoald=1>.



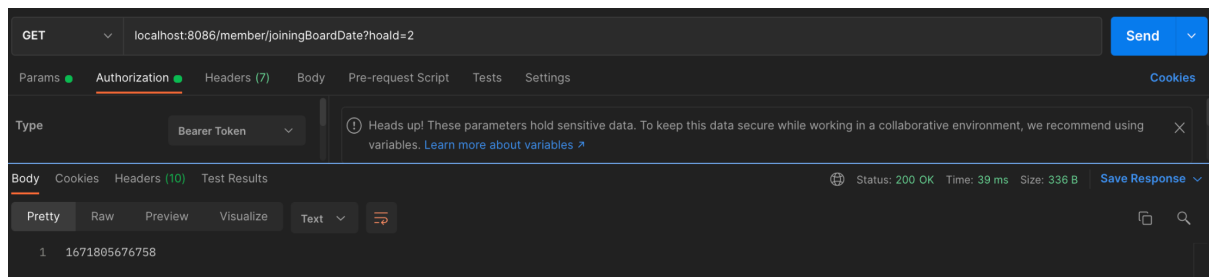
As a user, you wonder if you are a board member of any of the HOAs you are part of? Then you can use the above API call. If successful, you'll get a **200 OK Response** with the hoald of any of the HOA you are boardMember in the response body. If you are not, you'll get -1 in the response body.



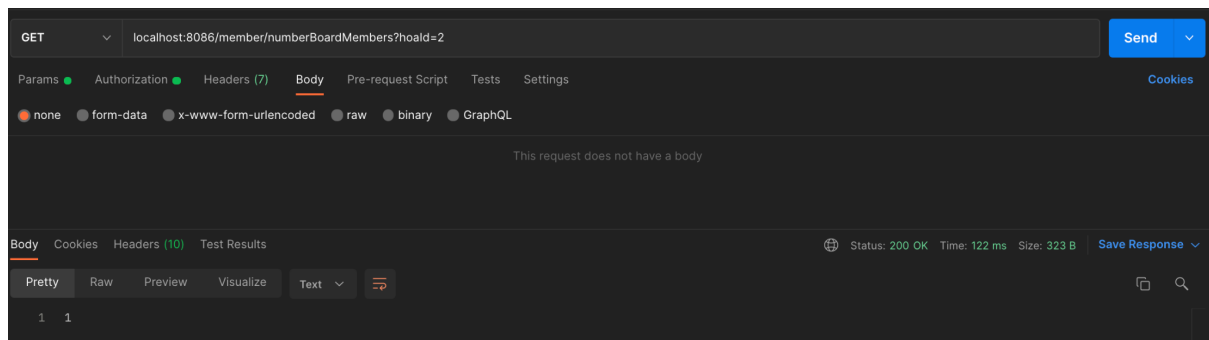
As a user, you wonder whether you are a member of a specific HOA (you have the id). You can do that using the above API call. If you are a member, you'll get a **200 OK Response** with "true" in the response body. "false" otherwise.



As a user, you wonder when did you join a specific HOA (you have the id)? You can do that using the above API call. If successful, you'll get a **200 OK Response** with the Unix Time in the response body. Unix time is a date and time representation widely used in computing. It measures time by the number of seconds that have elapsed since 00:00:00 UTC on 1 January 1970, the beginning of the Unix epoch, less adjustments made due to leap seconds.



As a user, you wonder when did you join the board of a specific HOA (you have the id)? You can do that using the above API call. If successful, you'll get a **200 OK Response** with the Unix Time in the response body.



As a user, you wonder how many board members are there for a specific HOA (you have the id)? You can do that using the above API call. If successful, you'll get a **200 OK Response** with the number of board members in the response body.

Creating new elections

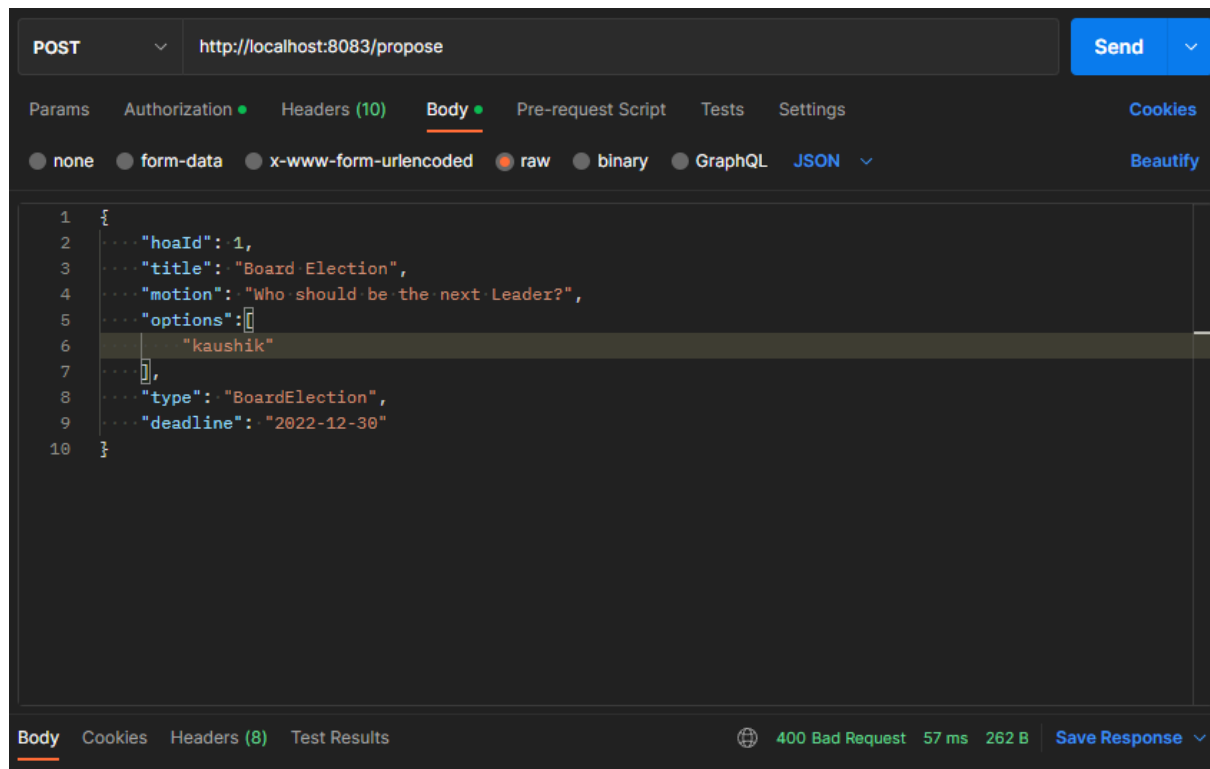
User Story: *A board member can initiate a new election (either a board or a rule changes election).*

Related requirements:

- Only board members can create new elections
- If there's a board election going on for a specific HOA, no other elections can be created

In the tests below, a few Users and HOAs have already been added along with their Memberships to the HOAs. User Minh, Kaushik, Mate and Atanas are in "Delft HOA". User Kaushik and Mate are also in "Rot HOA", together with Giacomo. Minh is a board member of "Delft HOA" and Mate is a board member of "Rot HOA".

Firstly, Kaushik tries to create a new board election of Delft HOA. He receives a Bad Request Response since he is not a board member of Delft HOA..

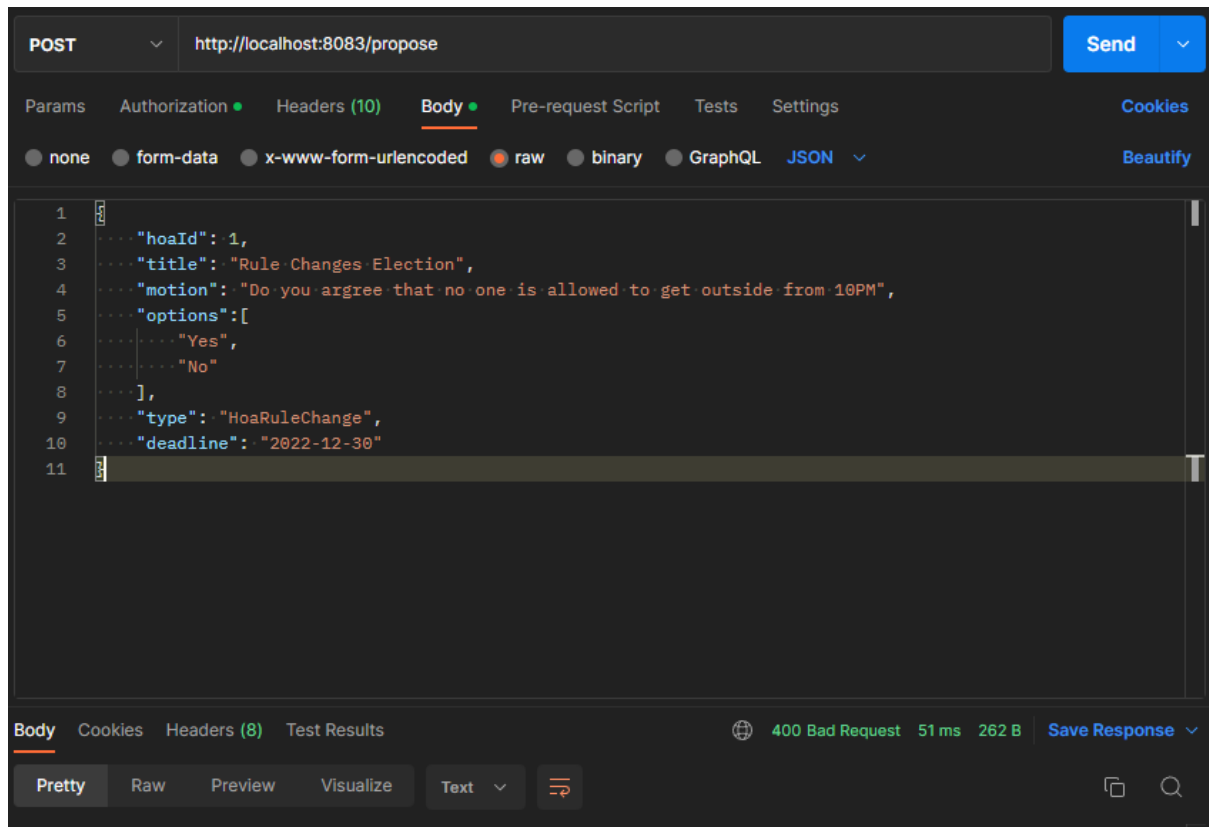


Message on the server:

```
400 : [User is not a board member of this Hoa]
Option is not valid
```

Minh tries to create a new board election. He receives a 200 response with the ID of the proposal. The list of Options only contains Minh since he cannot nominate anyone other than himself as a candidate (this is also explained late)

Minh then tries to create another rule change election, but this is not allowed since there's already a board election going on.



Message on the server:

```
minh does not have the rights to start a new proposal:  
A board election is ongoing
```

Note that all requests have to contain a authentication token associated with the user who makes the request. This token is used for checking whether the user is a board member

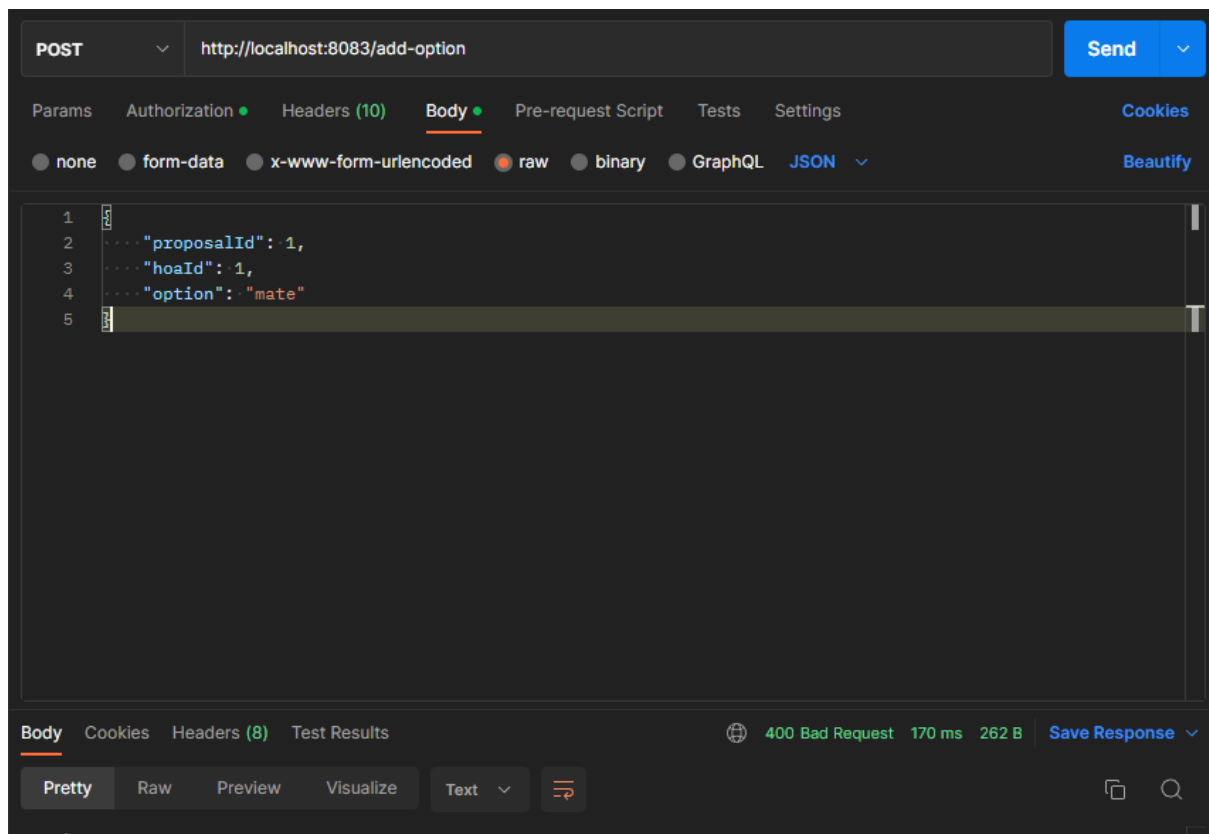
Adding new options to the elections

User Story: For board elections, members of a HOA can nominate themselves as candidates. The system performs multiple checks on whether a user is allowed to be a candidate. For rule changes, the only options available are Yes or No.

Related requirements:

- A user can only apply if (s)he is not a board member of any other HOAs
- Users can only nominate themselves, not for another person
- A user can only apply if (s)he has been in the HOA for at least 3 years

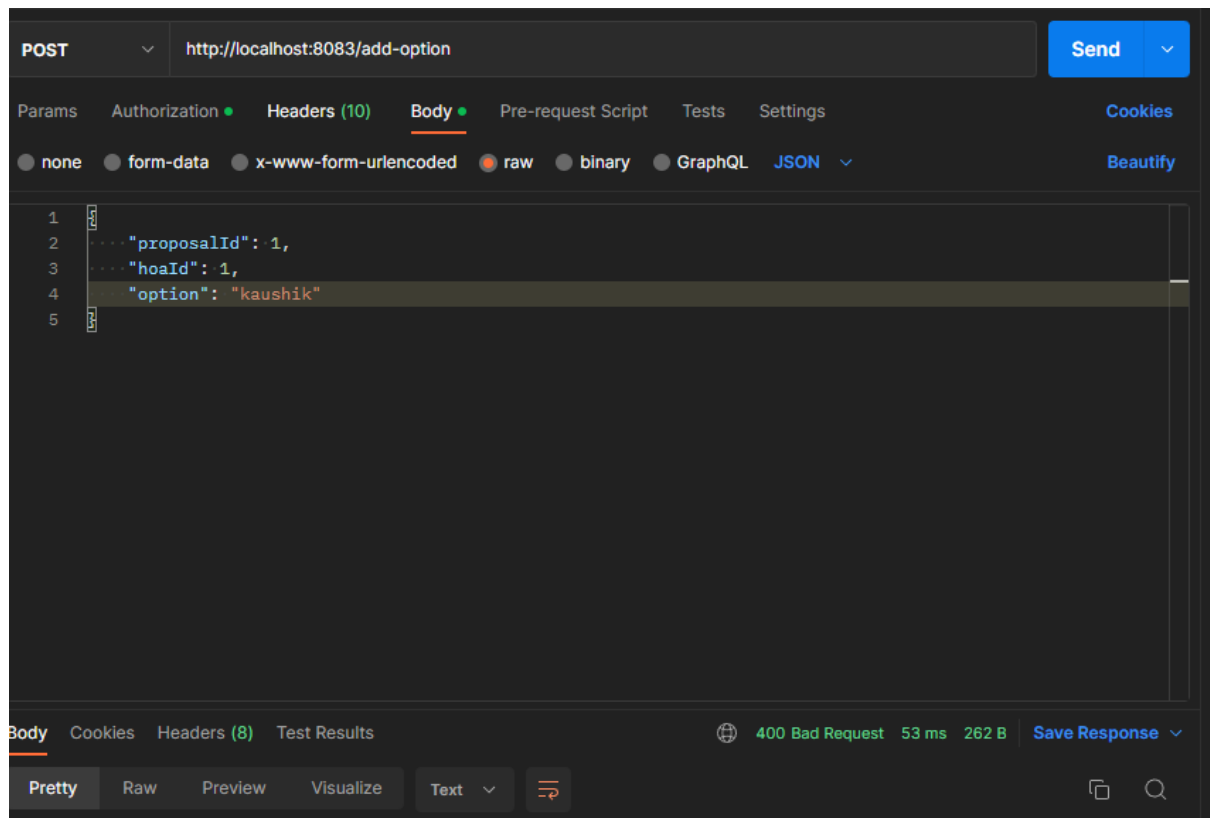
Firstly, Mate tries to nominate himself, but this is not allowed since he has already been a board member of Rot HOA



Message on the server:

```
User is already a board member of another HOA
Option is not valid
```

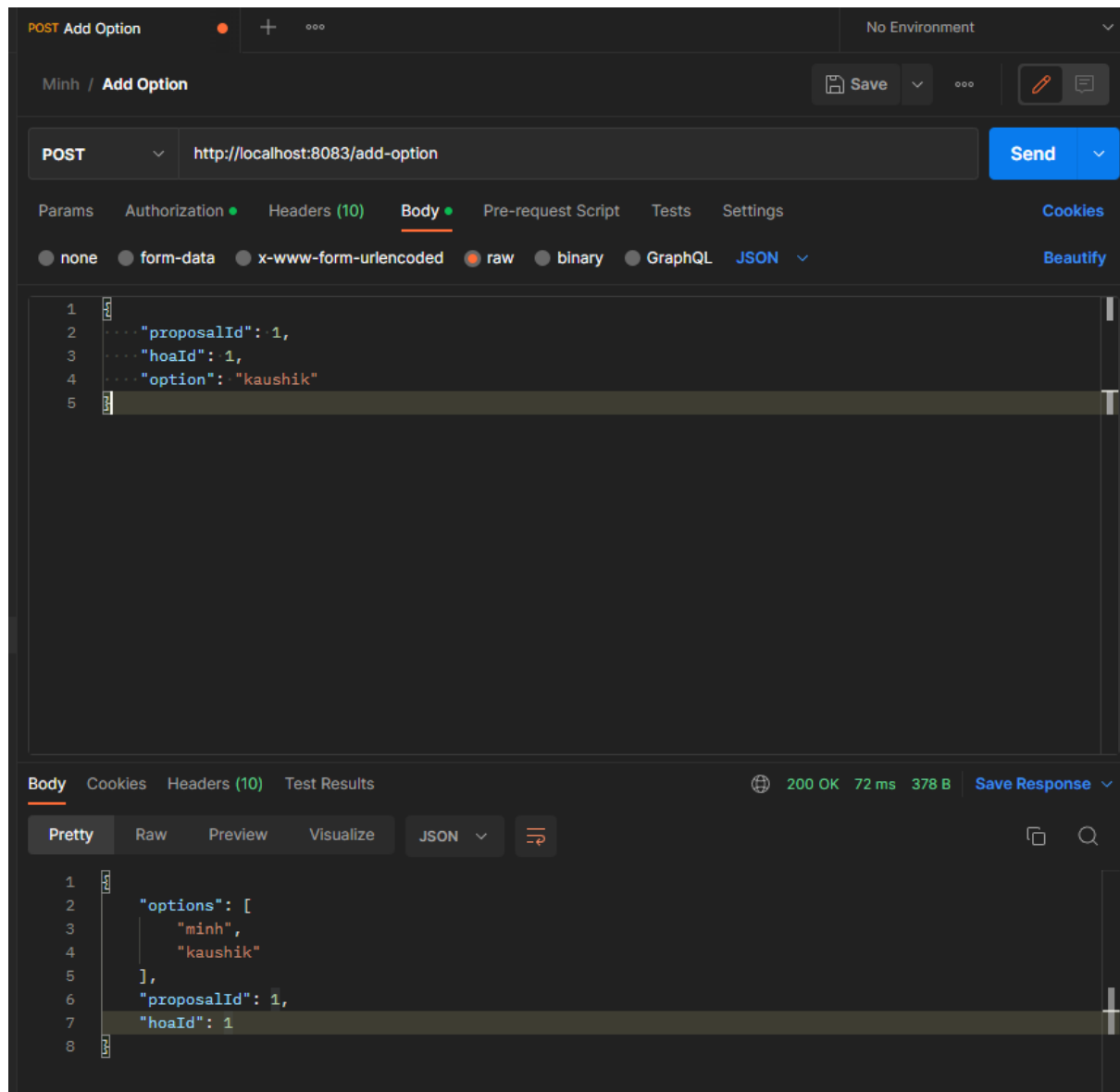
Minh tries to nominate Kaushik, this is also not allowed as user can only nominate themselves.



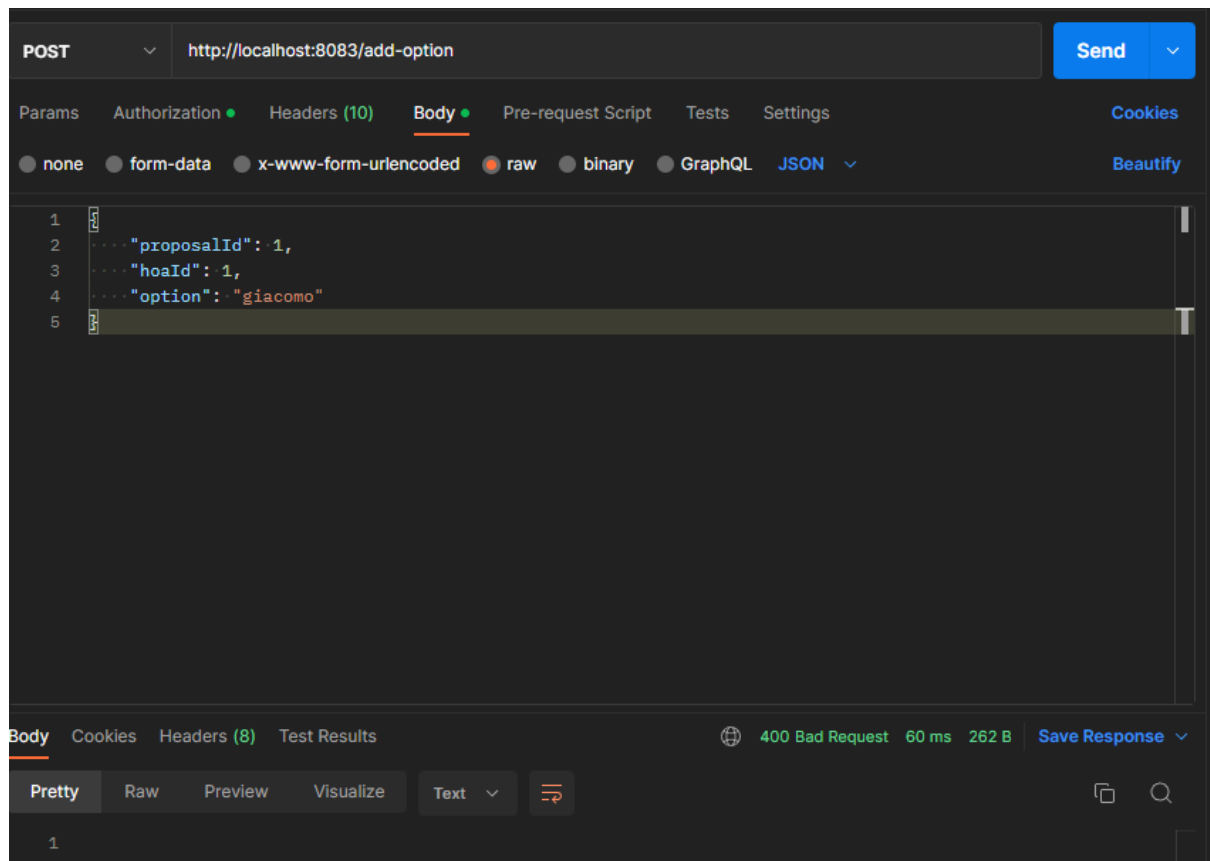
Message on the server:

```
A user can only candidate themselves
Option is not valid
```

In order to test the 3rd requirement (a user can only apply if (s)he has been in the HOA for at least 3 years), we need to manually modify the database since all of these users are recently added at the time of testing. We modified the joining date of Kaushik to be 4 years in the past. This allows him to apply.



Giacomo also tries to apply, but since he is not registered in Delft HOA, his request is rejected



Message on the server:

```
inside handle User is not a member of this HOA
Option is not valid
```

Note that all requests have to contain a authentication token associated with the user who makes the request. This token can be used to identify users, which allows us to do multiple eligibility checks on the requests

Adding new votes to the elections

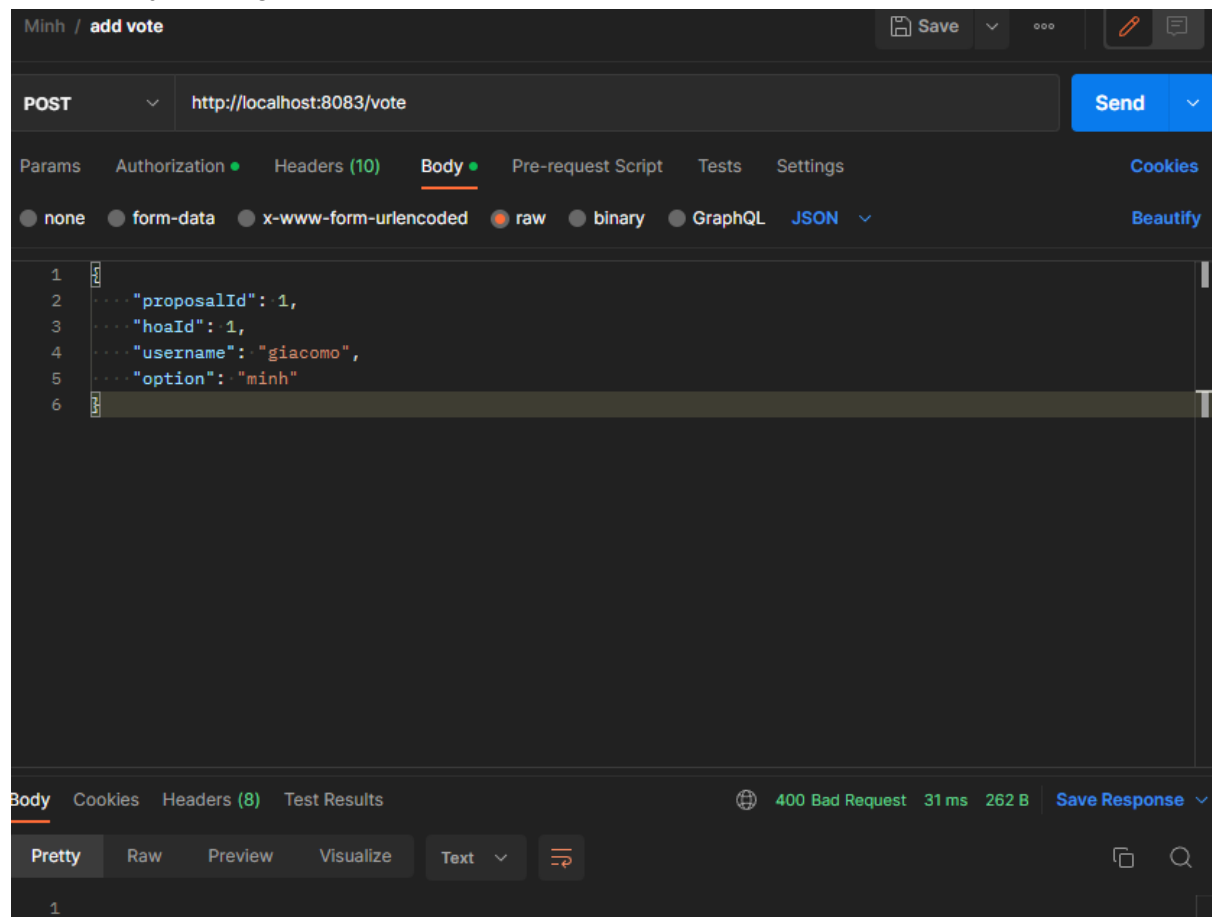
User Story: *Members of an HOA can participate in elections*

Related requirements:

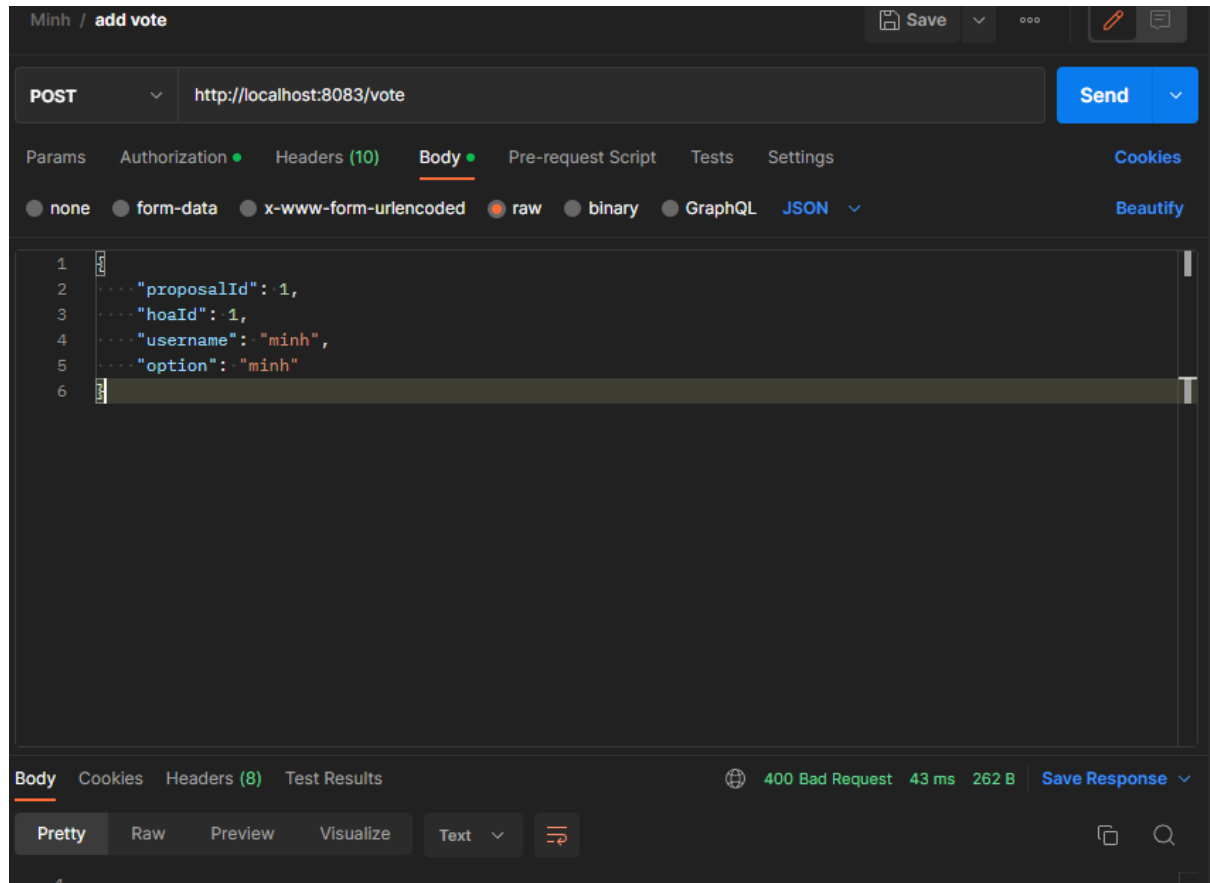
- A user can only vote if (s)he is a member of the HOA
- For board elections, a member cannot vote for himself
- A user can change or remove vote at any time during the Voting phase of an election

Note that all requests have to contain an authentication token associated with the user who makes the request. This token can be used to identify users, which allows us to do multiple eligibility checks on the requests

Continue with the previous board election, after making a call to /start, members can start to vote. At the moment, there are 2 available options: Minh and Kaushik. Giacomo tries to vote, but he is rejected again because he is not a member of Delft HOA



Minh tries to vote for himself, but also gets rejected



```
A user can't vote for himself
Vote is not valid
```

Although Minh cannot vote for himself, he can still vote for Kaushik

The screenshot shows a REST client interface with a POST request to `http://localhost:8083/vote`. The request body is a JSON object:

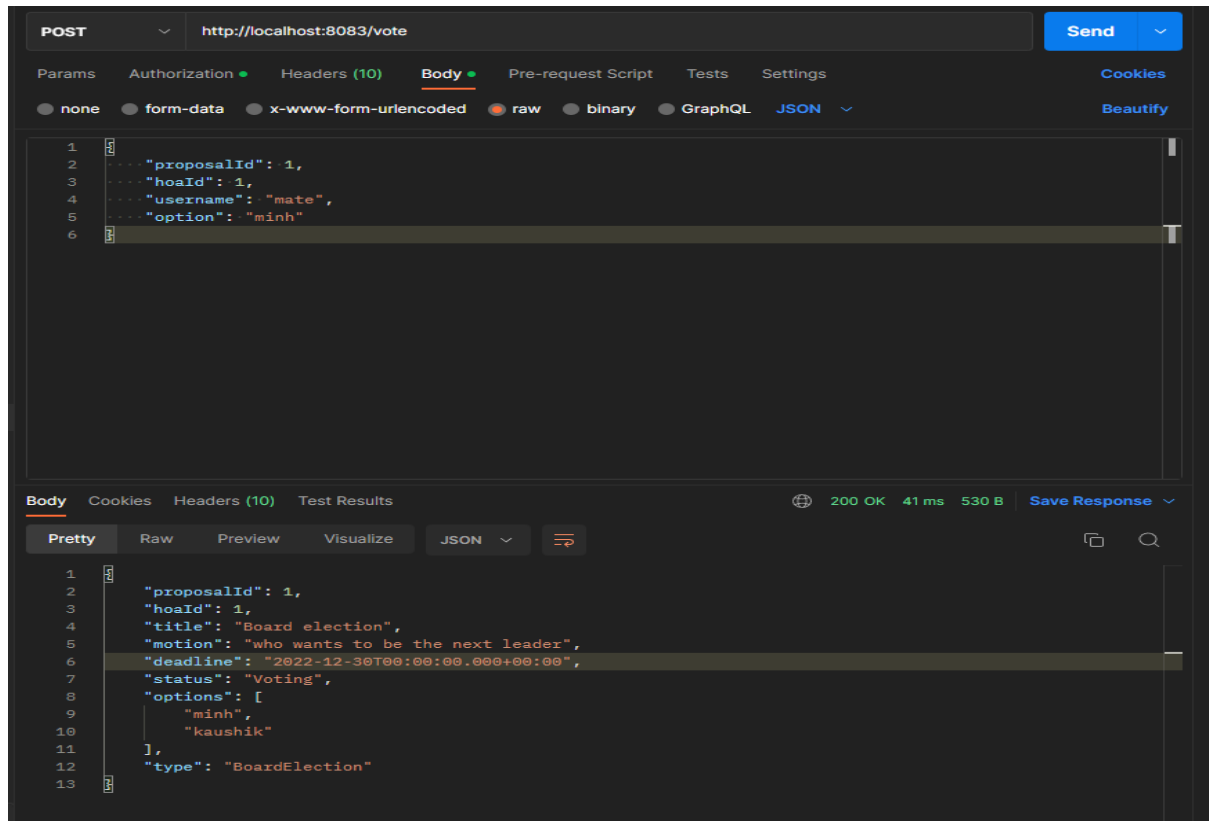
```
{  "proposalId": 1,  "hoaId": 1,  "username": "minh",  "option": "kaushik"}
```

The response is a 200 OK status with a response time of 38 ms and a size of 530 B. The response body is a JSON object:

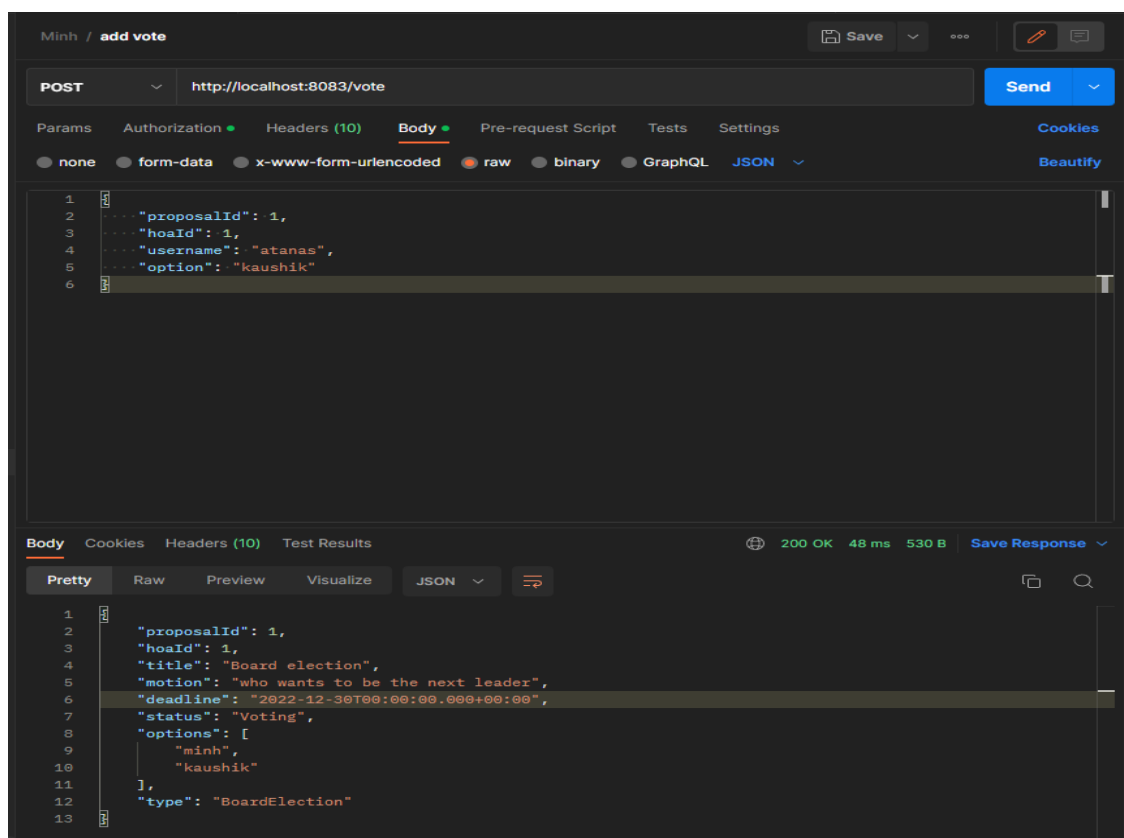
```
{  "proposalId": 1,  "hoaId": 1,  "title": "Board election",  "motion": "who wants to be the next leader",  "deadline": "2022-12-30T00:00:00.000+00:00",  "status": "Voting",  "options": [    "minh",    "kaushik"  ],  "type": "BoardElection"}
```

Meanwhile, Mate, Giacomo and Atanas are all able to vote for their preferred candidate. In this case, Mate votes for Minh while Atanas, Giacomo and Minh votes for Kaushik.

A user can also change vote. Initially, Giacomo



votes for Kaushik, he can still change his vote to Minh as long as the election is still in Voting phase



Getting results and history of elections

User Story: *Members can see the results of all elections of their HOAs*

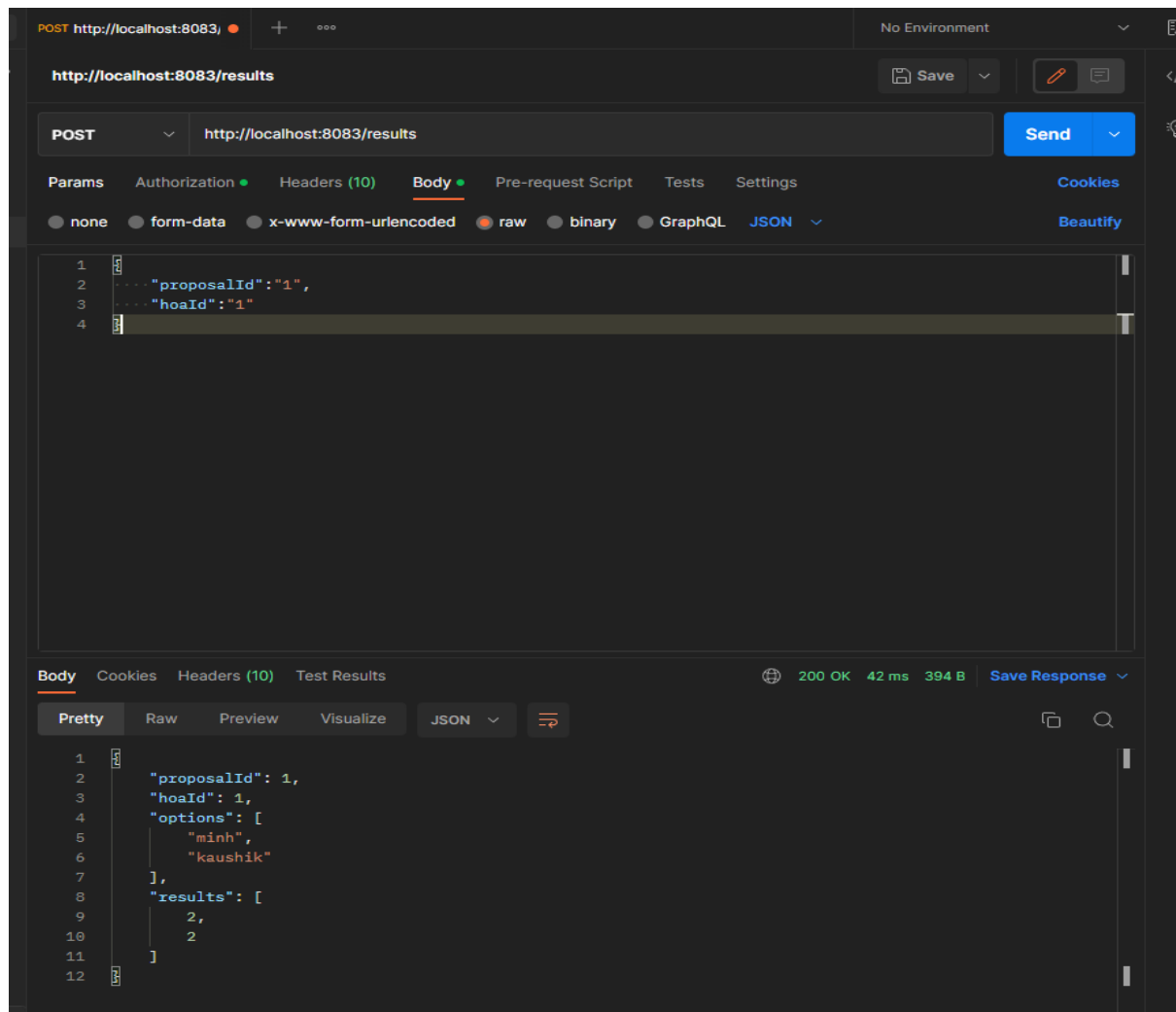
Related requirements:

- Full history of all elections (proposal voting and board elections) should be visible to all members
- Users can see all active elections of a HOA

When an election is created, a voting deadline needs to be defined. After this deadline, the system automatically closes for new votes and calculates the total votes for each option. Note that the voting service is only responsible for calculating number of votes, the decision made by the election (whether to pass a new rule or who are selected for the board) is handled by HOA microservice.

Users can access the results of an election by proposalId and hoald via /results

In the previous board election, both Minh and Kaushik get a total of 2 votes for each person. We manually changed the deadline of the election in the database for testing purpose.



A user can also find active elections of a HOA via `/active`. We create another Rule changes election.

Create a new proposal:

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

- URL:** `http://localhost:8083/propose`
- Method:** `POST`
- Body:** A JSON object representing a proposal:

```
1 {
2   "hoaId": 1,
3   "title": "New Rule election",
4   "motion": "Do you agree that no one can play loud music after 8PM",
5   "options": [
6     "Yes",
7     "No"
8   ],
9   "type": "HoaRuleChange",
10  "deadline": "2022-12-30"
11 }
```
- Response:** A successful `200 OK` response with a status of `130 ms` and a size of `339 B`. The response body is a JSON object:

```
1 {
2   "proposalId": 2
3 }
```

Getting active proposals of HOA with ID 1

The screenshot shows a REST client interface with a POST request to `http://localhost:8083/active`. The request body is a JSON object with `"hoaId": 1`. The response is a 200 OK status with a JSON body containing details about a proposal.

Request:

```
POST http://localhost:8083/active
{
  "hoaId": 1
}
```

Response:

```
200 OK 22 ms 563 B
{
  "proposalId": 2,
  "hoaId": 1,
  "title": "New Rule election",
  "motion": "Do you agree that no one can play loud music after 8PM",
  "deadline": "2022-12-30T00:00:00.000+00:00",
  "status": "UnderConstruction",
  "options": [
    "Yes",
    "No"
  ],
  "type": "HoaRuleChange"
}
```

To view the history of all elections associated with a HOA, user can make a request to /history. We manually ended the previous Rule change election.

POST history

No Environment

Minh / history

Save

Send

Params

Authorization

Headers (10)

Body

Pre-request Script

Tests

Settings

Cookies

Query Params

KEY	VALUE	DESCRIPTION	Bulk Edit
Key	Value	Description	

Body

Cookies

Headers (10)

Test Results

200 OK 36 ms 712 B Save Response

Pretty

Raw

Preview

Visualize

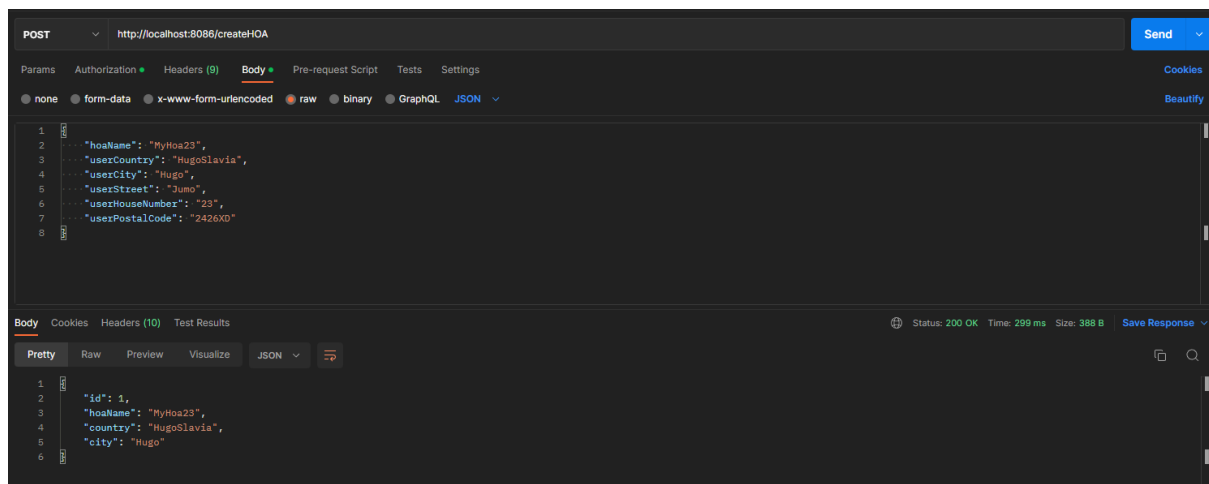
JSON

```
1  [
2    {
3      "proposalId": 1,
4      "hoaId": 1,
5      "title": "Board election",
6      "motion": "who wants to be the next leader",
7      "deadline": "2022-12-01T00:00:00.000+00:00",
8      "status": "Ended",
9      "options": [],
10     "results": []
11   },
12   {
13     "proposalId": 2,
14     "hoaId": 1,
15     "title": "New Rule election",
16     "motion": "Do you agree that no one can play loud music after 8PM",
17     "deadline": "2022-12-01T00:00:00.000+00:00",
18     "status": "Ended",
19     "options": [],
20     "results": []
21   }
22 ]
```

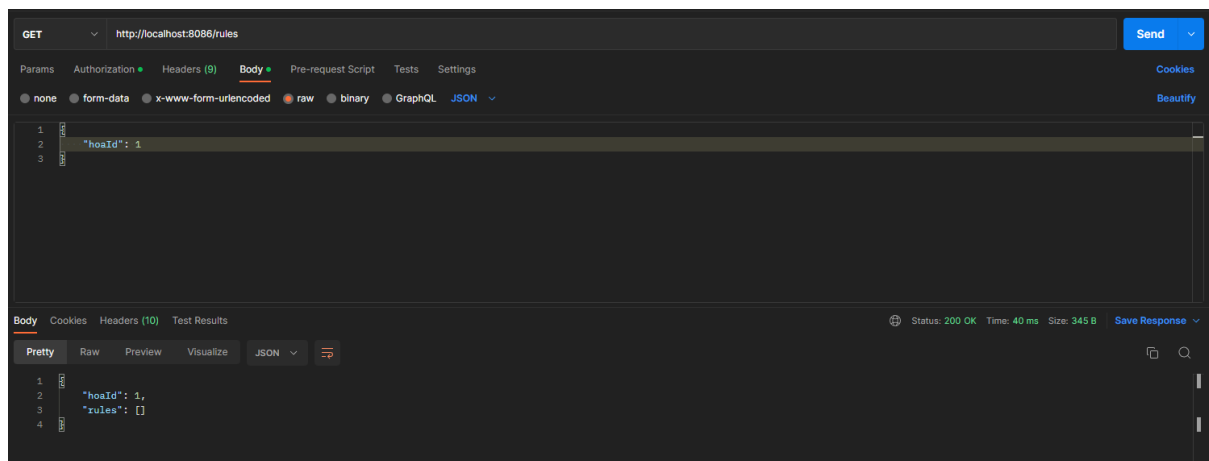

Displaying, adding, editing and deleting rules from an HOA

User Story: A user must be able to see the rules, add rules, edit rules and delete rules from the list of rules in the Hoa

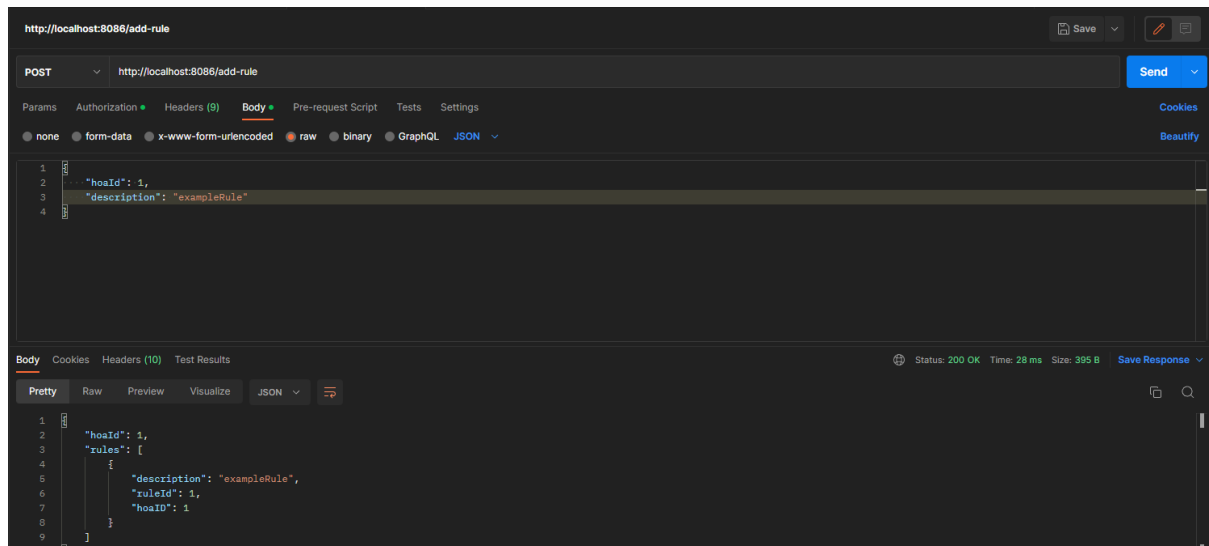
Each HOA has a List of rules. A user should be able to add, edit and delete rules (if eligible for that). To do that we have implemented a number of endpoints that help us do that. Let's assume that the user is logged <http://localhost:8085/register> and <http://localhost:8085/authenticate>. To start modifying rules, we need to create a Hoa. For this purpose we have <http://localhost:8086/createHOA> as shown in the picture below:



Without an Hoa, it's impossible to do anything with the rules. Now let's try to display the rules of our newly created Hoa. To access the rules we need to use the <http://localhost:8086/rules> url. The request contains the id of the desired Hoa. We can see the request in the picture below:



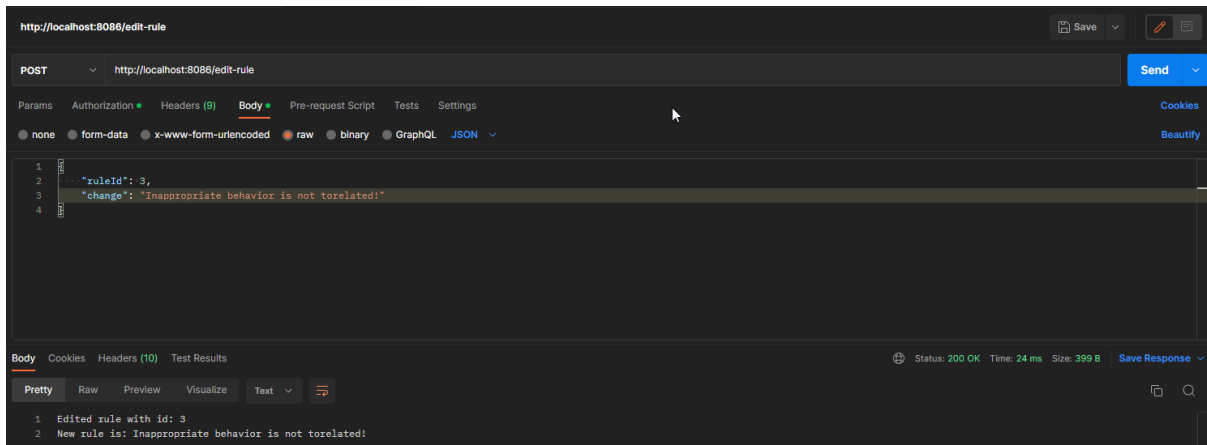
As we can see currently the HOA doesn't contain any rules. Let's try to add some. To do such, we need to use the <http://localhost:8086/add-rule>. Its requested body contains the id of the Hoa and the description of the rule. The request is depicted below:



The response after an addition is the id of the Hoa and the list of the rules (same as displaying). Let's try to add a few more rules. Now the list of the rules look like this:

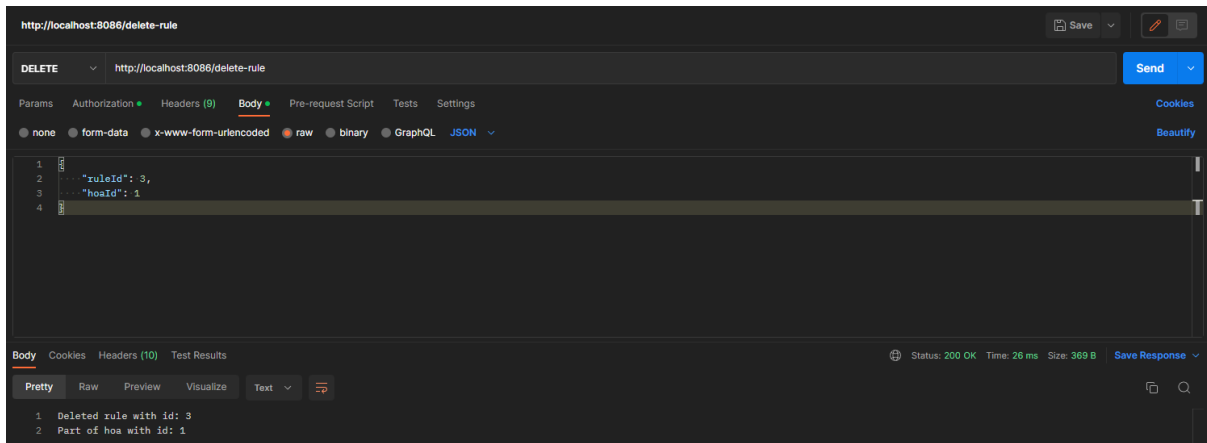


Let's assume that we don't like the description of rule 3. So after a voting process the board decides to change it. It can be changed using the <http://localhost:8086/edit-rule>. Its body contains the id of the rule that is to be changed and the new description of the rule. If an invalid id is inserted, the response would be 404 NOT FOUND error. We can see that the rule with id 3 is now changed using the request:



Now the rule list look list look like this:

And finally, if a rule is voted to be removed we can use the <http://localhost:8086/delete-rule> url to remove it from the list. Its body contains the Hoa id and the id of the rule that is to be deleted. If either one of the id of the hoa or the id of the rule is invalid then the response is 404 NOT FOUND error The request is depicted below:



And finally the list of rules looks like this:

GET

http://localhost:8086/rules

Params

Authorization

Headers (9)

Body

Pre-request Script

Tests

Settings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

1

2

3

1

2

3

Body

Cookies

Headers (10)

Test Results

Pretty

Raw

Preview

Visualize

JSON

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

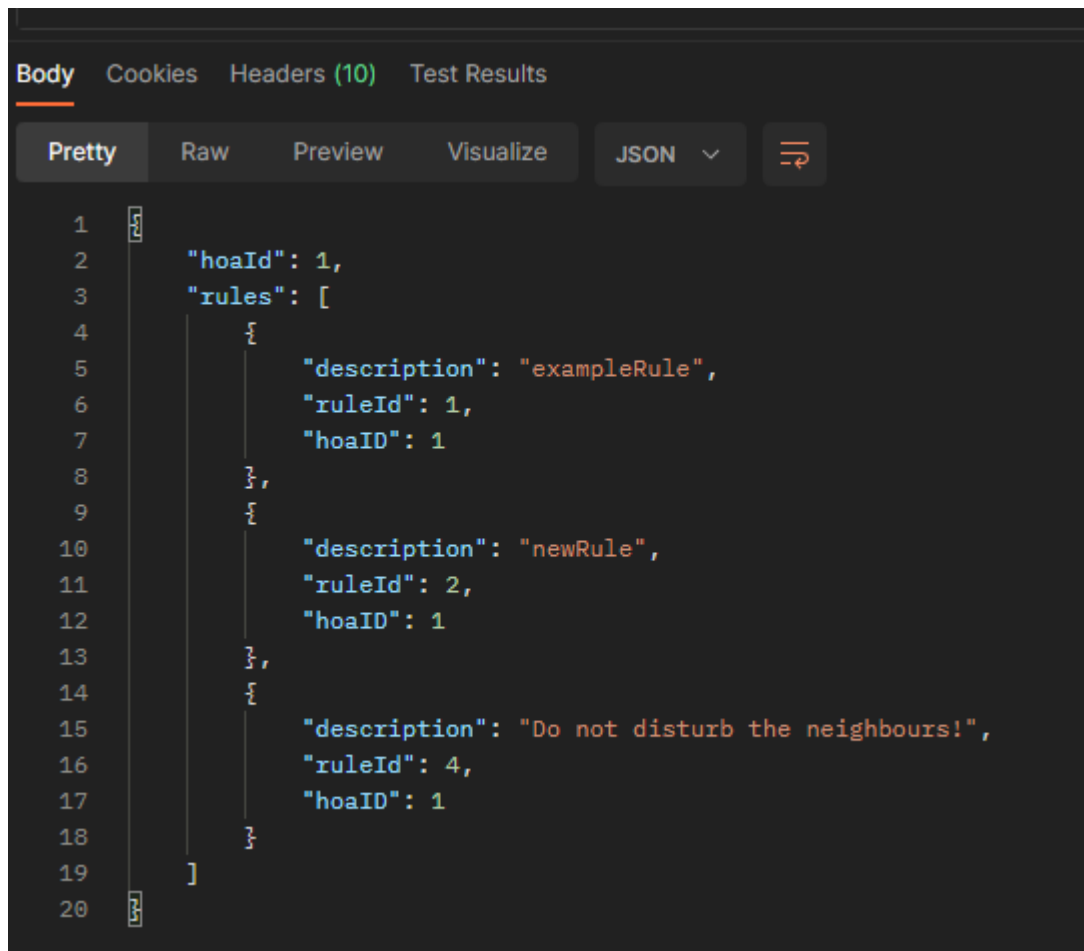
22

23

24

25

```
"hoaId": 1,  
"rules": [  
  {  
    "description": "exampleRule",  
    "ruleId": 1,  
    "hoaID": 1  
  },  
  {  
    "description": "newRule",  
    "ruleId": 2,  
    "hoaID": 1  
  },  
  {  
    "description": "Inappropriate behavior is not tolerated!",  
    "ruleId": 3,  
    "hoaID": 1  
  },  
  {  
    "description": "Do not disturb the neighbors!",  
    "ruleId": 4,  
    "hoaID": 1  
  }  
]
```



```
1 {
2   "hoaId": 1,
3   "rules": [
4     {
5       "description": "exampleRule",
6       "ruleId": 1,
7       "hoaID": 1
8     },
9     {
10      "description": "newRule",
11      "ruleId": 2,
12      "hoaID": 1
13    },
14    {
15      "description": "Do not disturb the neighbours!",
16      "ruleId": 4,
17      "hoaID": 1
18    }
19  ]
20 }
```

Joining an HOA

User Story: *A user must be able to join an existing HOA as a normal member*

In order for a user to be able to join an HOA he first needs to be logged in and authenticated, after which the user will receive a token.

The path for joining is: <http://localhost:8086/joining>

POST localhost:8086/joining

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

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

```

1 {
2   ... "hoaName": "MyHoa23",
3   ... "userCountry": "HugoSlavia",
4   ... "userCity": "Hugo",
5   ... "userStreet": "Jumo",
6   ... "userHouseNumber": "23",
7   ... "userPostalCode": "2426XD"
8 }

```

For body the user must specify the name of the HOA they wish to join, as well as the user's address, namely: country, city, street, house number(as an integer) and a postal code. In order for the request to go through the user must also specify the authentication token which was assigned during the authentication of the user

POST localhost:8086/joining

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

Type Bearer Token

The authorization header will be automatically generated when you send the request. [Learn more about authorization](#)

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborator [Learn more about variables](#)

Token

```

eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJkYSIsImV4cCI6MTY3MTcxOTY0MCwiaWF0IjoxNjc4NjMzMjQwfQ.z9rrg4G86FYD0WqT_gGHuoQTHmbPhcg5Fg_OOwf3FIBci7By5F-2hCvF2nBJdhFQG4Xcv2XyddEjKYr5ryi76d

```

This is how the method recognises which user wants to join the specific HOA

If successful the user will receive a 200 OK response

If unsuccessful user will receive one of the following error messages depending on circumstances:

- A user is already a member:

```

{
  "timestamp": "2022-12-23T15:13:18.509+00:00",
  "status": 400,
  "error": "Bad Request",
  "message": "User is already in this HOA",
  "path": "/joining"
}

```

- User's given city/country doesn't match the city/country this user is trying to join

```
{
  "timestamp": "2022-12-23T15:15:12.488+00:00",
  "status": 400,
  "error": "Bad Request",
  "message": "Address not compatible with HOA area",
  "path": "/joining"
}
```

- Field was empty/blank:

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

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

```
1 {
2   ... "hoaName": "MyHoa23",
3   ... "userCountry": "",
4   ... "userCity": "Hugo",
5   ... "userStreet": "Jumo",
6   ... "userHouseNumber": "23",
7   ... "userPostalCode": "2426XD"
8 }
```

Body Cookies Headers (9) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "timestamp": "2022-12-23T15:16:23.565+00:00",
3   "status": 400,
4   "error": "Bad Request",
5   "message": "Fields can not be Empty",
6   "path": "/joining"
7 }
```

- Address field was null:

● none ● form-data ● x-www-form-urlencoded ● raw ● binary ● GraphQL JSON ▾

```
1 {
2   ... "hoaName": "MyHoa23",
3   ... "userCountry": null,
4   ... "userCity": "Hugo",
5   ... "userStreet": "Jumo",
6   ... "userHouseNumber": "23",
7   ... "userPostalCode": "2426XD"
8 }
```

Body Cookies Headers (9) Test Results

Pretty

Raw

Preview

Visualize

JSON ▾



```
1 {
2   "timestamp": "2022-12-23T15:18:15.453+00:00",
3   "status": 400,
4   "error": "Bad Request",
5   "message": "Fields can not be Invalid(null)",
6   "path": "/joining"
7 }
```

- House number was negative:

Body Cookies Headers (9) Test Results

Pretty

Raw

Preview

Visualize

JSON ▾



```
1 {
2   "timestamp": "2022-12-23T15:19:30.337+00:00",
3   "status": 400,
4   "error": "Bad Request",
5   "message": "House Number must be a positive integer",
6   "path": "/joining"
7 }
```

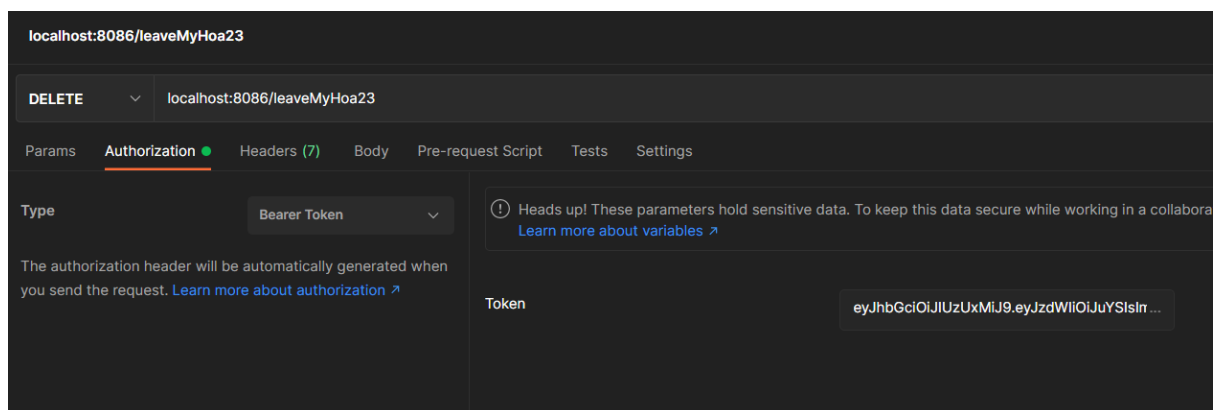

Leaving an HOA

User Story: A user must be able to leave an existing HOA that they are a part of

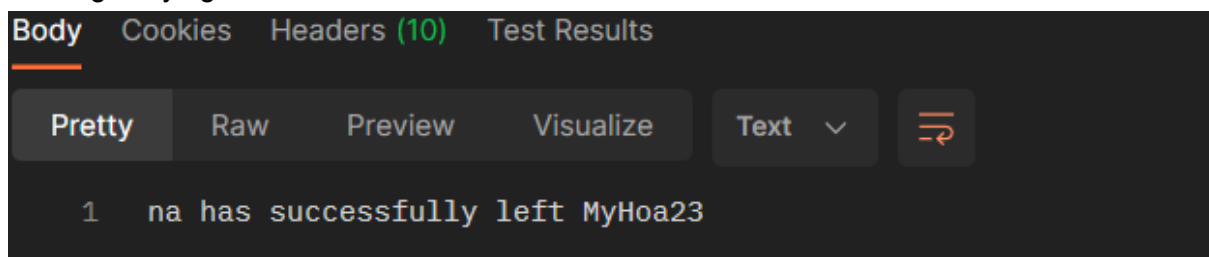
If a users wishes to leave a specific HOA he first needs to be logged in and authenticated, after which he is a able to do so by entering the link shown below:

<http://localhost:8086/leaveMyHoa23>

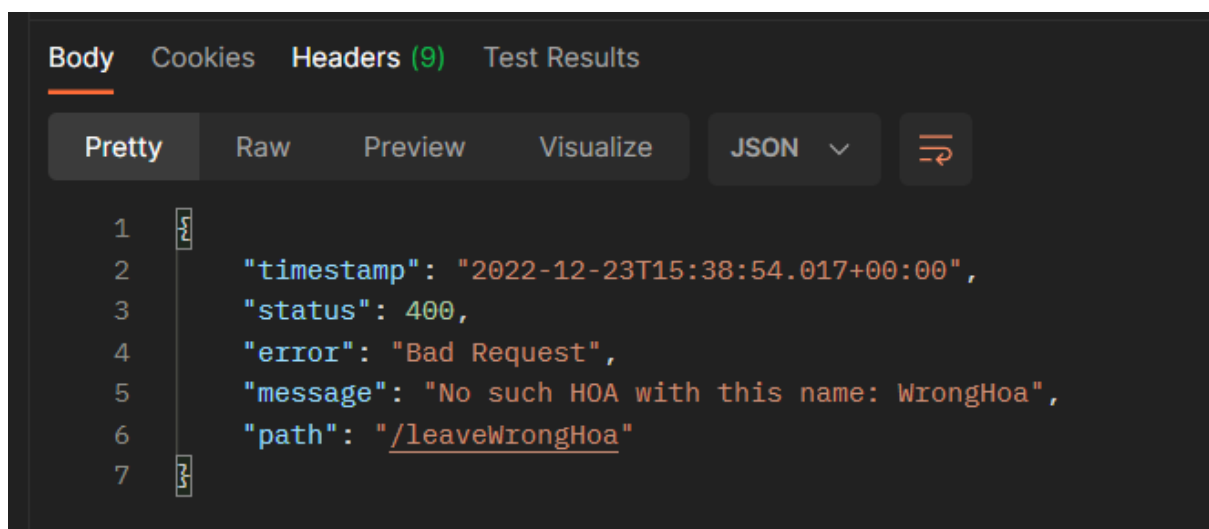
Which consists of the path /leave , followed by the name of the HOA the user wishes to leave. In order to do so the user must enter the url and authentication token as shown below:



If the request was successful the sender should receive a 200 OK as well as a custom message saying which user left which HOA



If the name entered was incorrect(a HOA with this name does not exist) the sender will receive:



If user is not a member of the specified HOA the sender will receive:

Changing user credentials

User Story: *A user should be able to change their password or full name.*

The user first registers in the system. This is done by sending a HTTP POST request through the URL: <http://localhost:8085/register>. The user enters a username, a password, and their full name to be able to make an account.

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

- Method:** POST
- URL:** `http://localhost:8085/register`
- Body:** JSON format, containing:

```
1 {  
2   "username": "JDoe",  
3   "password": "password123",  
4   "fullname": "John Doe"  
5 }
```
- Status:** 200 OK
- Time:** 411 ms
- Size:** 305 B

As all parameters entered were within the allowed limits, and there doesn't exist a user with the same username, the server sends a response confirming the registration was successful.

The user now realises that their password is not ideal, and wants to change it to something else. They can do that by sending a POST request to the URL: <http://localhost:8085/changepassword>. They need to include their username, current password, and their new password.

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

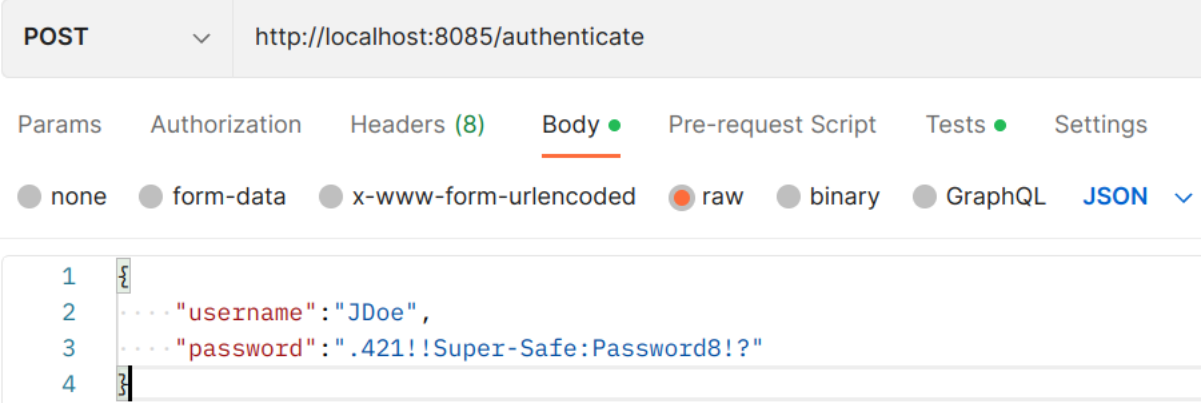
- Method:** POST
- URL:** `http://localhost:8085/changepassword`
- Body:** JSON format, containing:

```
1 {  
2   "username": "JDoe",  
3   "password": "password123",  
4   "newAttribute": ".421!!Super-Safe:Password8!?"  
5 }
```

As all credentials are correct and the new password is within the length limit, the server responds with 200 OK, to let the user know their password was successfully changed.

 Status: 200 OK Time: 323 ms Size: 305 B

The user is then able to log into the system with their new password, through the <http://localhost:8085/authenticate> URL.



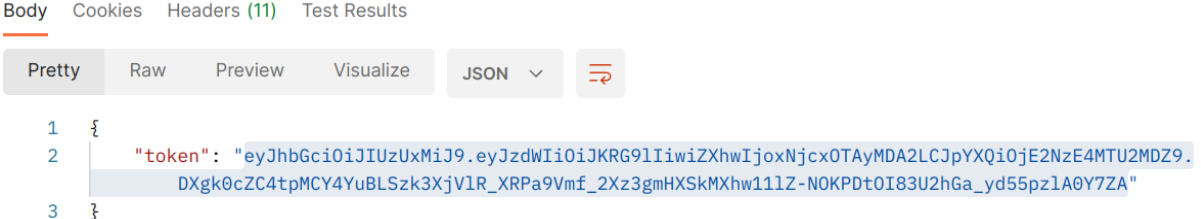
POST ▼ http://localhost:8085/authenticate

Params Authorization Headers (8) **Body** ● Pre-request Script Tests ● Settings


☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON** ▼

```
1 {
2   ... "username": "JDoe",
3   ... "password": ".421!!Super-Safe:Password8!?"
4 }
```

The server then responds with their authentication token and 200 OK, as the user successfully authenticated with their new password.



Body Cookies Headers (11) Test Results

Pretty Raw Preview Visualize JSON ▼ 

```
1 {
2   "token": "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJKRG9lIiwiaXhwIjoxNjc0TAyMDA2LCJpYXQiOiE2NzE4MTU2MDZ9.DXgk0cZC4tpMCY4YuBLSzk3XjVlR_XRPa9VmF_2Xz3gmHXSkmXhw111Z-NOKPDt0I83U2hGa_yd55pz1A0Y7ZA"
3 }
```