Title: MiniWall Blogging Service

Chaitali Desale
13821873

Cloud Computing coursework project report, Department of Computer Science and Information Systems, Birkbeck College, University of London 2022

Introduction

The goal of this project is to create an API service to serve a blogging website called MiniWall. The service can be used as a social media blogging platform. This report outlines the infrastructure, design, and technologies used to develop software for the MiniWall API service. Some of the major components include user and post management. A post is the content which a user can broadcast to other users. Only registered MiniWall users can share a post, like and comment on other users' posts. Users can see a feed of posts shared by other users. For scalability and availability, Github is used for version control and the API service is deployed on Google Cloud Platform using docker containerisation. This report also documents test cases that are used to verify the working of the service. Please note that although the software developed focuses on the backend service and doesn't involve a frontend API, the developed service makes it easier to add a frontend service.

The MiniWall API service is written in Node.js and has three main components which are user management, post management, and the database. This report delineates these components in detail in the following sections and also explains the rationale behind different decisions that structure the API service. The code is located on Github at https://github.com/cdesale/MiniWall-REST-API.

Database

A NoSQL database was chosen for MiniWall because it provides a much needed flexibility for a new project allowing for quick iterations, it is easily scalable, and provides a simple querying interface. Specifically MongoDB (https://www.mongodb.com/) was chosen as it is open source, has good community support, and has great integrations on majority cloud platforms. The database is remotely hosted on Cloud MongoDB solution provided by Atlas (cloud.mongodb.com).

The database structure for MiniWall involves two major components: a User and a Post. As the blog site allows users to comment on others posts, another object that is required is a Comment. Basically, a Post can have multiple Comments. Both Post and Comment have a User as its owner. By storing a Comment in its own collection we can have better management of Comments, similar to Posts. Furthermore, a Post can be liked by multiple Users.

The User schema has the following structure:

```
User:
    username
    email
    password
    date
```

The above schema is sufficient for a User model at this stage and more attributes such as `displayName`, `dateOfBirth`, etc. can be added when features using this attribute are added. It should be noted that the `password` attribute above is stored in hashed format. The `data` attribute is used to store the timestamp of when the user registered on the service.

The Comment schema has the following structure:

```
Comment:
    commentText
```

```
    owner
    timeStamp
```

The comment schema has a `commentText` for storing the comment, an `owner` attribute that stores the ObjectID of a User object which refers to a user in User schema, finally a `timeStamp` to store when the comment was created.

The Post schema has the following structure:

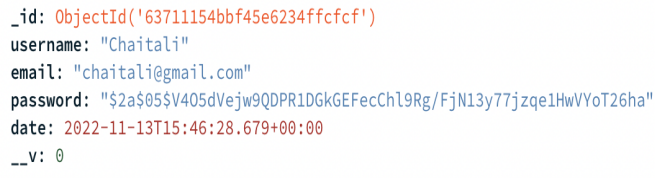
```
Post:
    title
    description
    owner
    timeStamp
    comments
    likes
    likesCount
```

A Post consists of a `title` which is a short headline. A thorough `description` attribute to that holds the actual post. An `owner` which references a User object through ObjectId that created the post. A `timeStamp` to store the time when the post was created. A `comments` attribute to store a list of ObjectIds referencing Comment objects which essentially are comments posted by users on the post. The `likes` attribute stores a list of ObjectIds referencing User objects which indicates the users who liked the post. Storing likes this way allows MiniWall to implement a way for users to unlike a liked post. An attribute `likesCount` is used to store the number of likes, this helps in efficiently querying the posts and ranking them based on number of likes.

In the current state the service doesn't offer any API that will update the User, Post, or Comment model objects but those can be added with rather ease. For example, if a user wants to update their username only one object/document will need to be updated while all the user's posts and comments will automatically get the updated user's details.

Inorder to access this data from MongoDB hosted on Cloud MongoDB, I used Mongoose library which is a MongoDB object modelling library for Node.js. Everything in Mongoose starts with a schema and each schema maps to a MongoDB collection and defines the shape of the documents within that collection. Mongoose library also offers an API to connect to the remotely hosted MongoDB. The URL of the remote database is stored in a `.env` file so that we can potentially have separate databases per hosting environment. For example, the DB URL can be easily configured to be separate for development and production environments. An environment file is also excluded from being committed to Github, this way the remote URL is never stored anywhere else other than on server or developer computers. The Dotenv library in Node.js is used for reading from .env files. The Dotenv library is a zero-dependency module that loads environment variables from a `.env` file. Storing configuration in the environment file separate from code is based on The Twelve-Factor App (12factor.net) methodology.

The following are the screenshots of a document/object in each of the collection:



User

Comment

Post

User Management

There are three functionalities under user management that are used in this project viz.; user creation, user login and API authentication. These endpoints are handled in the `users.js` routes to manage `/user` endpoints and is added as a middleware. Except for the user registration and the home page endpoints, all the other endpoints are authenticated and only allow valid registered users to use the endpoints.

The `GET /user/register`, `GET /user/login`, and `GET /user/` endpoints are used to inform users to register on the MiniWall service. When a frontend API is developed, these endpoints can be used to display user registration pages.

Registration

The `POST /user/register` endpoint implements the functionality that creates a user in MiniWall. The endpoint expects a form consisting of `username`, `email`, and `password`. The form data is parsed and passed into request's body parameter. This parsing is done in the middleware by the body-parser module in npm. Before using the form data, the fields are first validated using the Joi module from npm. This module validates fields before they can be stored against schema types in the database. If the validations don't match, the endpoint returns a `HTTP 400` error code. The endpoint then verifies if a User already exists in the users collection with the provided email and username. If it exists, the `HTTP 400` error code is returned. Finally, if a user doesn't exist, then a user with provided details is created. The user's password is hashed using Bcryptjs npm module and then stored instead of storing the password in plain text. The password encryption also includes a salt generated using the `genSalt` API of Bcryptjs module. Using salt in password encryption gives an added security from decrypting an encrypted password.

On successful creation of a user, the API responds with a json web token in the header. This web token should be used by the user (and frontend service) to authenticate subsequent MiniWall APIs. This web token essentially acts as an OAuth token for authenticating API endpoints. The Jsonwebtoken module uses a user ObjectId and a secret to generate the OAuth web token. The

secret provided is stored in the environment file to avoid it getting committed to Github and keeping it safe.

Login

The `POST /user/login` API is implemented to login a registered user into the MiniWall service. The API requires a user to provide their email and password which is then validated using the Joi module first. After validation, the endpoint first checks if a User exists in the users collection based on the provided email. If a user exists, the provided password is then compared against the stored encrypted password using Bcryptjs module. On successful validation, similar to user registration, an OAuth json web token is returned to the user which the user/frontend service can use to make authenticated API calls. In case a user does not exist if the provided password is incorrect the `HTTP 400` and `HTTP 401` error codes are returned accordingly.

Authenticating APIs

All the post management APIs should only be made by authenticated users. To ensure this, MiniWall service uses OAuth web tokens. These tokens should be added in the request header against an `auth-token` key. The generation of verification of the auth tokens is done by using the Jsonwebtoken npm module. For token verification, the Jsonwebtoken provides an API that requires the token to verify and the secret used in signing the token. This makes it easier to verify since no MongoDB query has to be made to verify the token, for example, check if a user exists with the provided token.

Post Management

The post management APIs allow users to see a feed of posts, create a new post, like, and comment on a post. These API are handled by the `posts.js` route which is added in the middle layer against `/post` endpoint.

Feed of posts

The `GET /post/feed` API shows a user a rank based ordered feed of posts. The rank is based on how many likes a post has, and when the post was created. The ranked list of posts is obtained by querying the posts collection using Mongoose npm module. The query ensures to return fields that do not risk revealing users credentials.

Creation of posts

The `POST /post/create` API allows users to create a new post. For a user to create a new post, they must send the post's title and description through a form. The form data is validated using the Joi npm module to satisfy the schema type definition. If the form validation fails an `HTTP 400` error code is thrown. After form validation, a Post model's object is created with the `title` attribute mapped to the provided title and `description` attribute mapped to the provided description. The `owner` attribute is filled internally by extracting the user's ObjectId from the `auth-token`. The post object is then saved in the posts collection.

Activities on posts

In the current state of MiniWall, users are only allowed to like and comment on posts. However, the design of the database and structure of logic allows easy addition of updating a post, editing a comment, and un-liking a post.

The `POST /post/like/:postId` API allows users to like an existing post which is not owned by them. Ideally, this API will not be exposed to a user but instead used by a frontend API service. The frontend should display a UI for the user to click on a like button which is associated with a post. When a user clicks on a like button, on a post, the frontend API service should make the API call for the user. The endpoint ensures that the post with `postId` exists and is not owned by the user liking the post. Furthermore, the endpoint also ensures that a user cannot re-like a post that the user has already liked. This is implemented by checking if the liking user's userId exists in the `likes` attribute of Post entity, which is an array. If the user is successfully able to like a post, then the user's userID is added to the `likes` attribute array and the `likesCount` attribute is incremented.

The `POST /post/comment/:postId` API allows users to comment on an existing post which is not owned by them. This endpoint will also ideally not be exposed to the user but instead but used by a frontend API service. The frontend service should show a UI for users to comment on a post. This endpoint requires the user to provide form data with commentText. This data is then validated against schema type requirements using the Joi npm module. Once validated, the endpoint then verifies if a post exists with the provided `postId`. The endpoint then verifies that the post is not owned by the commenting user. After all the verifications, if successful, an object of the Comment model is created where the `commentText` attribute is mapped with the user provided commentText and `owner` attribute is mapped with the commenter's userId which is fetched from the `auth-token` header. The comment object is then saved and its generated ObjectId is then added to the `comments` attribute array of the post object. The updated post object is then saved to update the data in MongoDB.

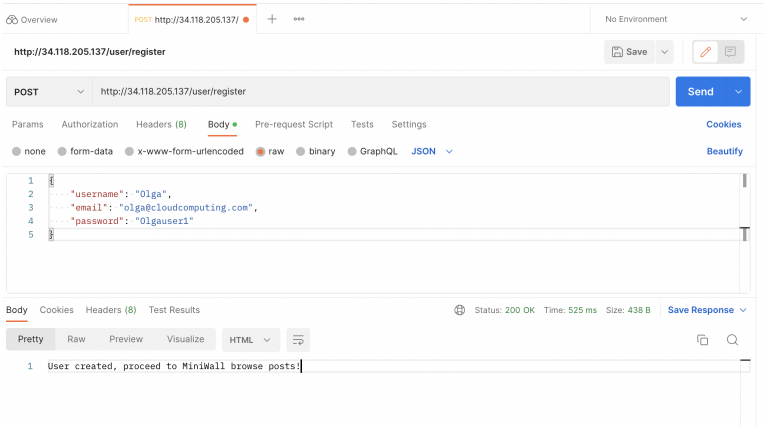
Users can also see just the comments and likes for a given post using `GET /post/comment/:postId` and `GET /post/like/:postId` API endpoints. This is shown in TC 11 and TC 14.

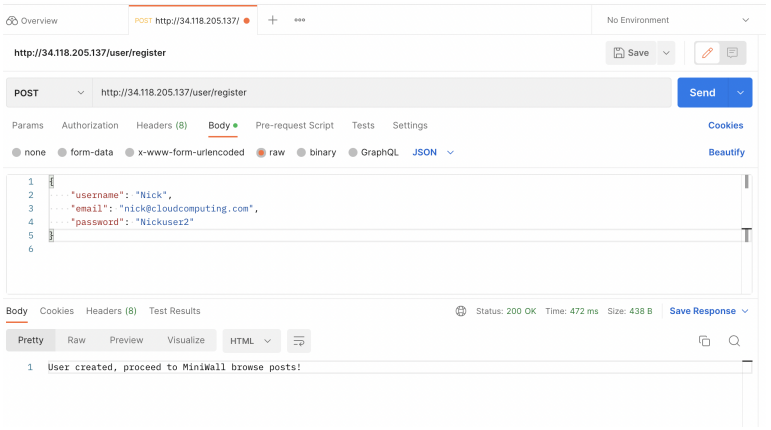
Future improvement

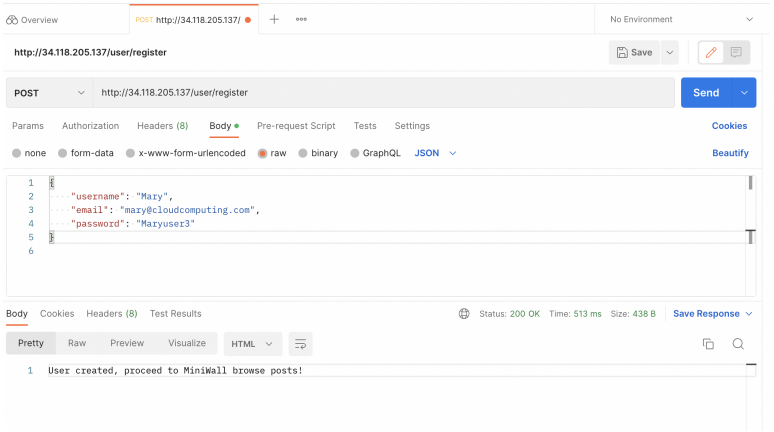
One of the future improvements could be breaking down the backend API service into two separate microservices. One microservice should be responsible for all the user management logic and the other for post management. This way I would be able to apply the principle of separation of concerns between the two components.

Test Cases

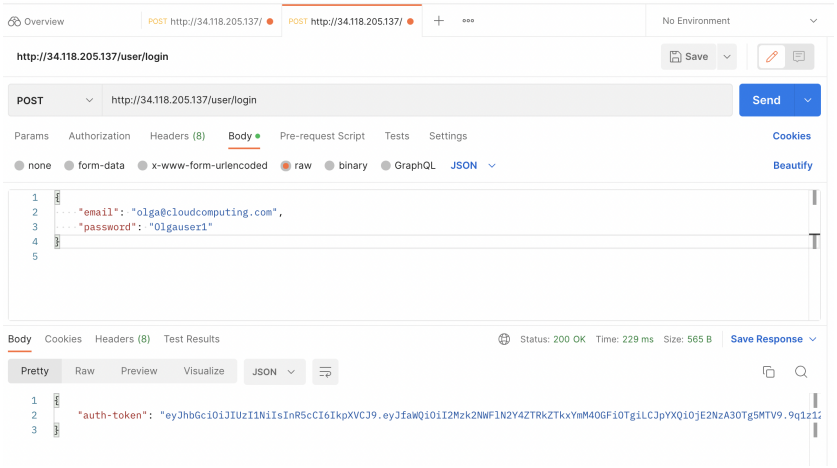
Test case 1:

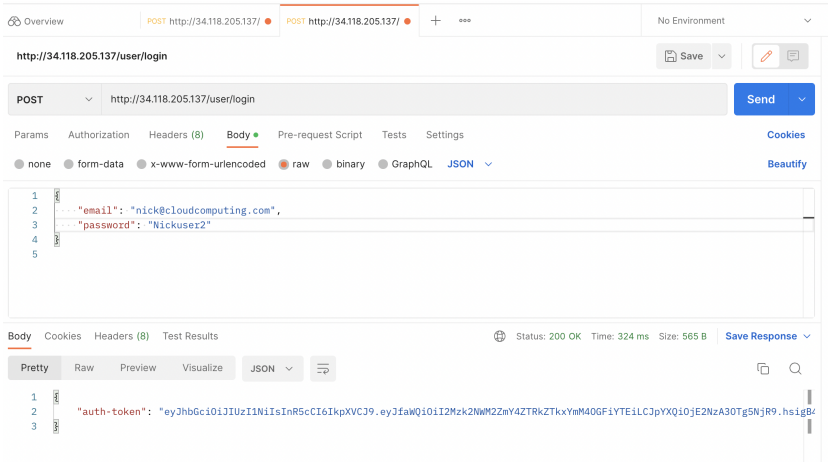


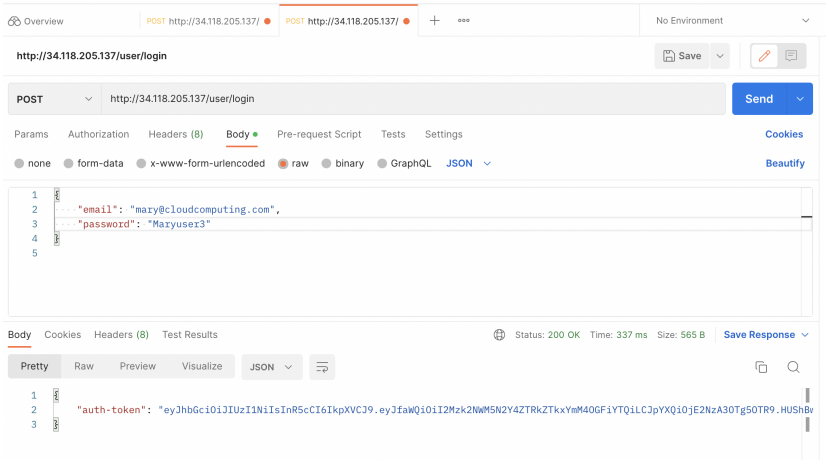




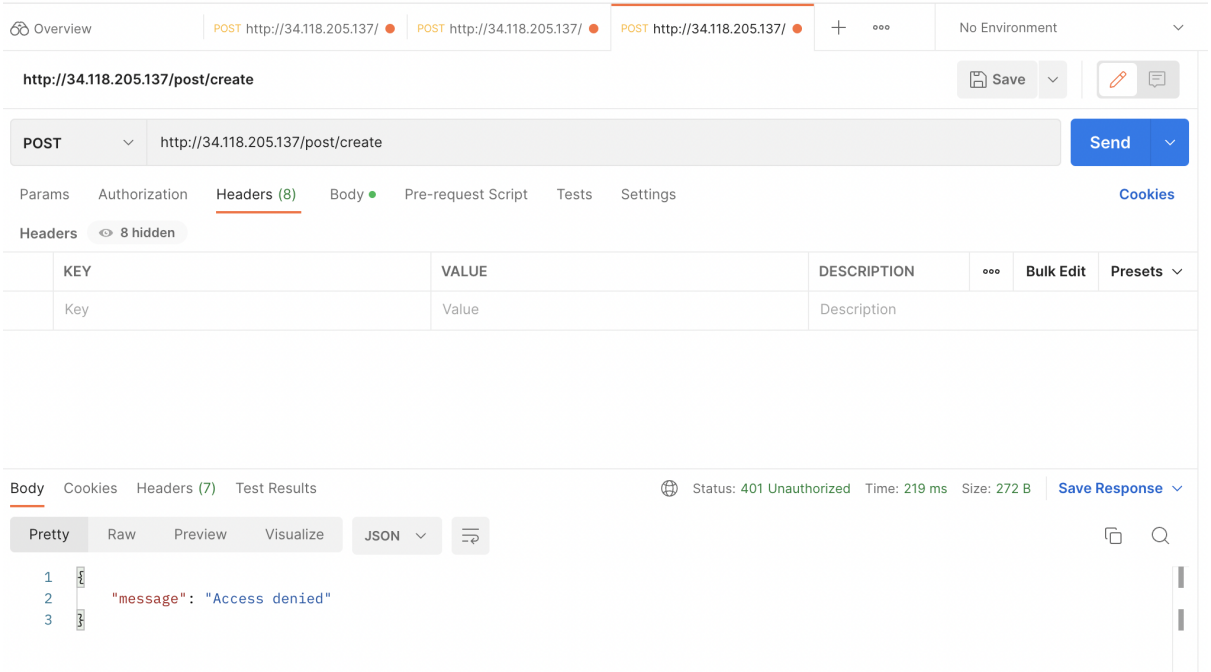
Test case 2:


Overview
POST http://34.118.205.137/
POST http://34.118.205.137/
No Environment
http://34.118.205.137/user/login
Save
POST
http://34.118.205.137/user/login
Send
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"email": "olga@cloudcomputing.com",
"password": "Olgauser1"
}
Body
Cookies
Headers (8)
Test Results
Status: 200 OK
Time: 229 ms
Size: 565 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"auth-token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI2Mzk2NWFlN2Y4ZTRkZTkxYmM4OGFiOTgiLCJpYXQiOjE2NzA3OTg5MTV9.9q1z1
}

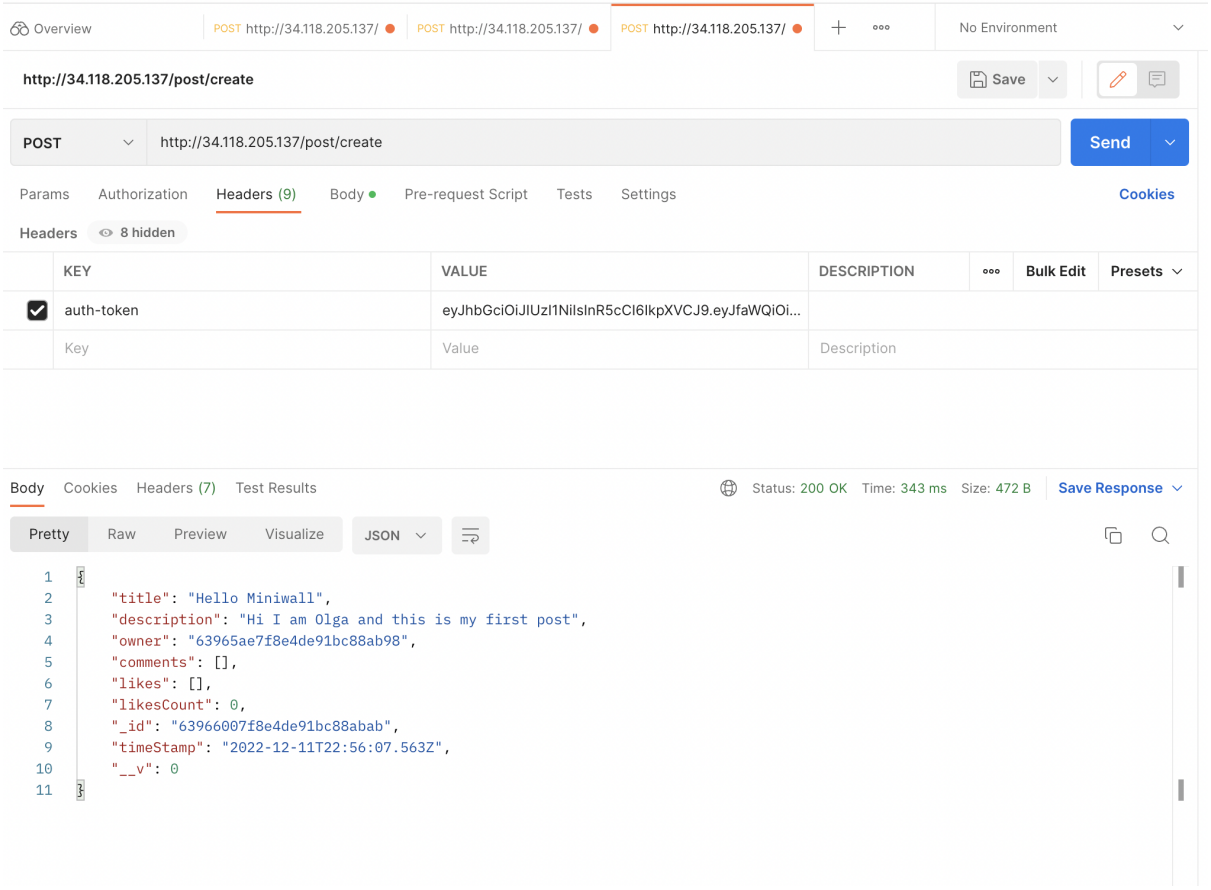

Overview
POST http://34.118.205.137/
POST http://34.118.205.137/
No Environment
http://34.118.205.137/user/login
Save
POST
http://34.118.205.137/user/login
Send
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"email": "nick@cloudcomputing.com",
"password": "Nickuser2"
}
Body
Cookies
Headers (8)
Test Results
Status: 200 OK
Time: 324 ms
Size: 565 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"auth-token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI2Mzk2NWM2ZmY4ZTRkZTkxYmM4OGFiYTEiLCJpYXQiOjE2NzA3OTg5NjR9.hsigB
}

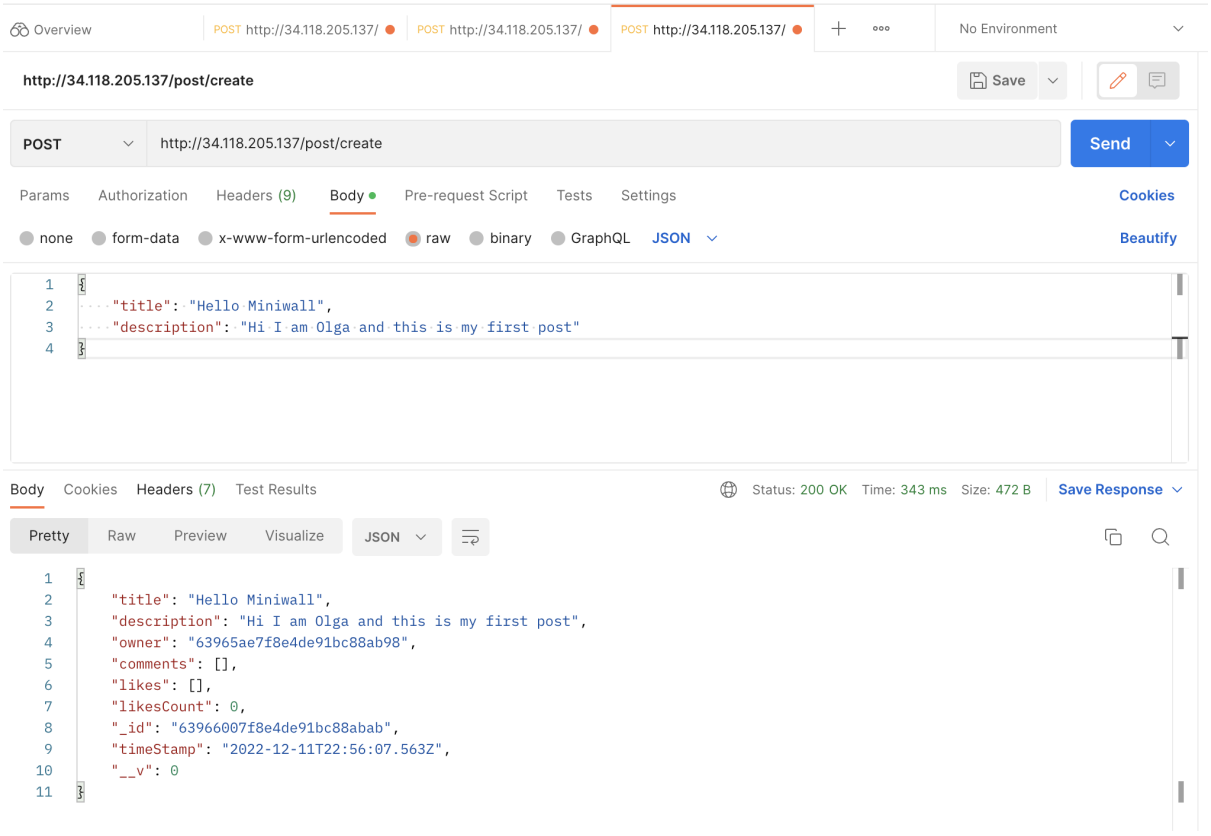

Overview
POST http://34.118.205.137/
POST http://34.118.205.137/
No Environment
http://34.118.205.137/user/login
Save
POST
http://34.118.205.137/user/login
Send
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"email": "mary@cloudcomputing.com",
"password": "Maryuser3"
}
Body
Cookies
Headers (8)
Test Results
Status: 200 OK
Time: 337 ms
Size: 565 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"auth-token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI2Mzk2NWM5N2Y4ZTRkZTkxYmM4OGFiYTQiLCJpYXQiOjE2NzA3OTg5OTR9.HUShB
}

Test 3:

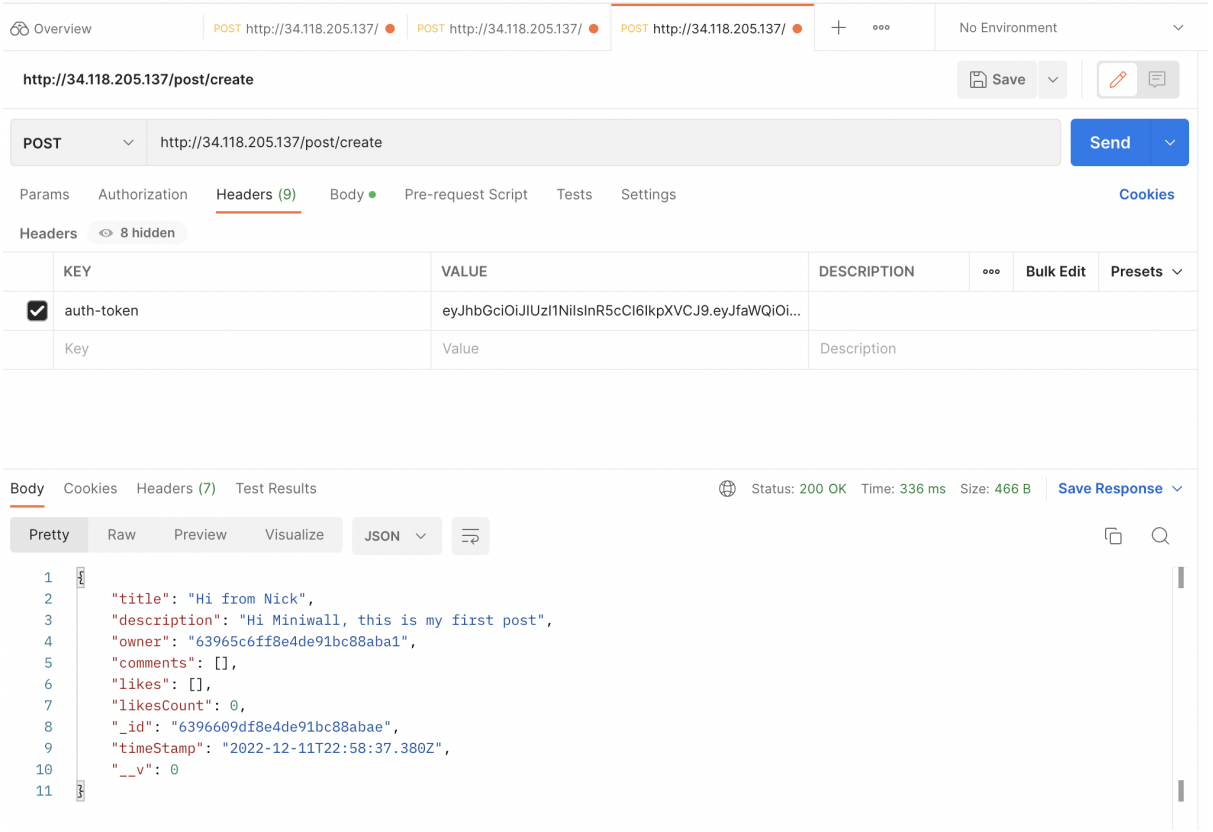

Overview
POST http://34.118.205.137/
POST http://34.118.205.137/
POST http://34.118.205.137/
No Environment
http://34.118.205.137/post/create
Save
POST
http://34.118.205.137/post/create
Send
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
Headers
8 hidden
KEY
VALUE
DESCRIPTION
Bulk Edit
Presets
Key
Value
Description
Body
Cookies
Headers (7)
Test Results
Status: 401 Unauthorized
Time: 219 ms
Size: 272 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
1 {
2 "message": "Access denied"
3 }

Test case 4

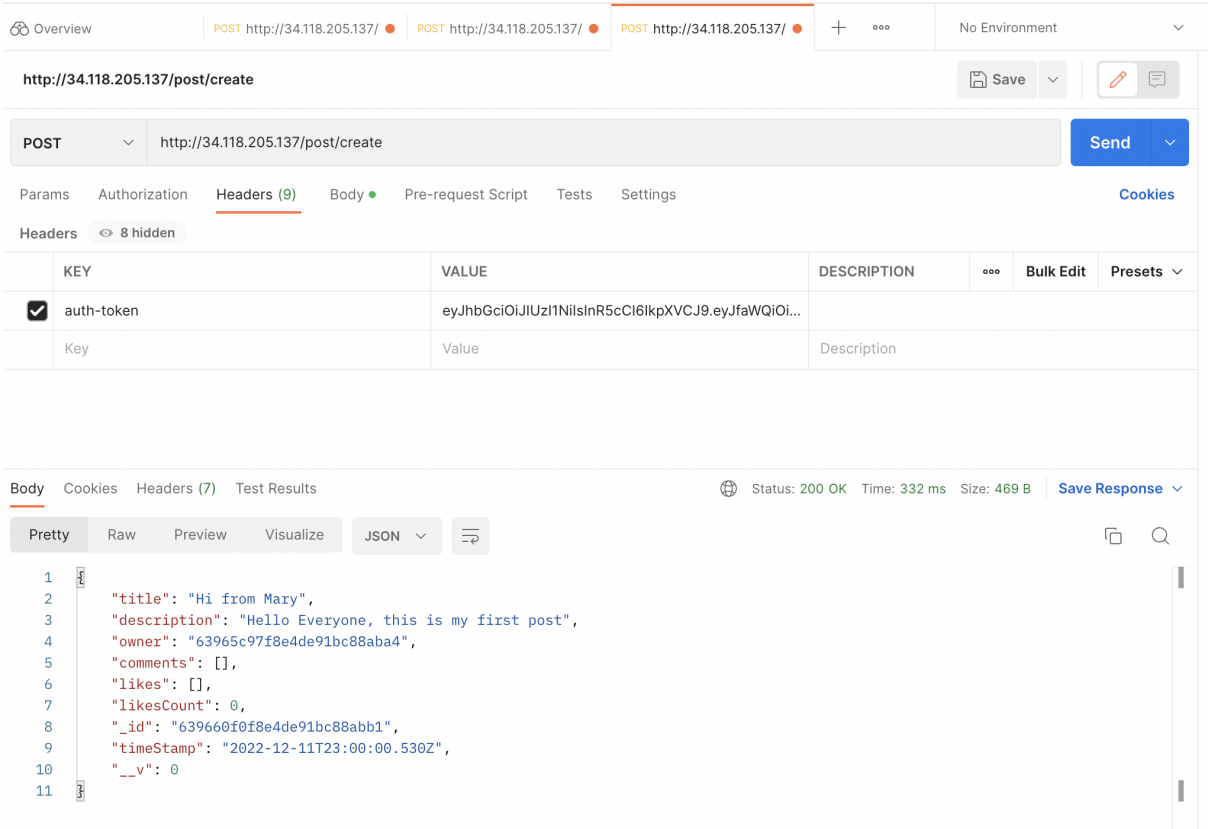

Overview
POST http://34.118.205.137/
POST http://34.118.205.137/
POST http://34.118.205.137/
No Environment
http://34.118.205.137/post/create
Save
POST
http://34.118.205.137/post/create
Send
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
Cookies
Headers
8 hidden
KEY
VALUE
DESCRIPTION
Bulk Edit
Presets
auth-token
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOi...
Key
Value
Description
Body
Cookies
Headers (7)
Test Results
Status: 200 OK
Time: 343 ms
Size: 472 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"title": "Hello Miniwall",
"description": "Hi I am Olga and this is my first post",
"owner": "63965ae7f8e4de91bc88ab98",
"comments": [],
"likes": [],
"likesCount": 0,
"_id": "63966007f8e4de91bc88abab",
"timeStamp": "2022-12-11T22:56:07.563Z",
"__v": 0
}


Overview
POST http://34.118.205.137/
POST http://34.118.205.137/
POST http://34.118.205.137/
No Environment
http://34.118.205.137/post/create
Save
POST
http://34.118.205.137/post/create
Send
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"title": "Hello Miniwall",
"description": "Hi I am Olga and this is my first post"
}
Body
Cookies
Headers (7)
Test Results
Status: 200 OK
Time: 343 ms
Size: 472 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"title": "Hello Miniwall",
"description": "Hi I am Olga and this is my first post",
"owner": "63965ae7f8e4de91bc88ab98",
"comments": [],
"likes": [],
"likesCount": 0,
"_id": "63966007f8e4de91bc88abab",
"timeStamp": "2022-12-11T22:56:07.563Z",
"__v": 0
}

Test case 5:

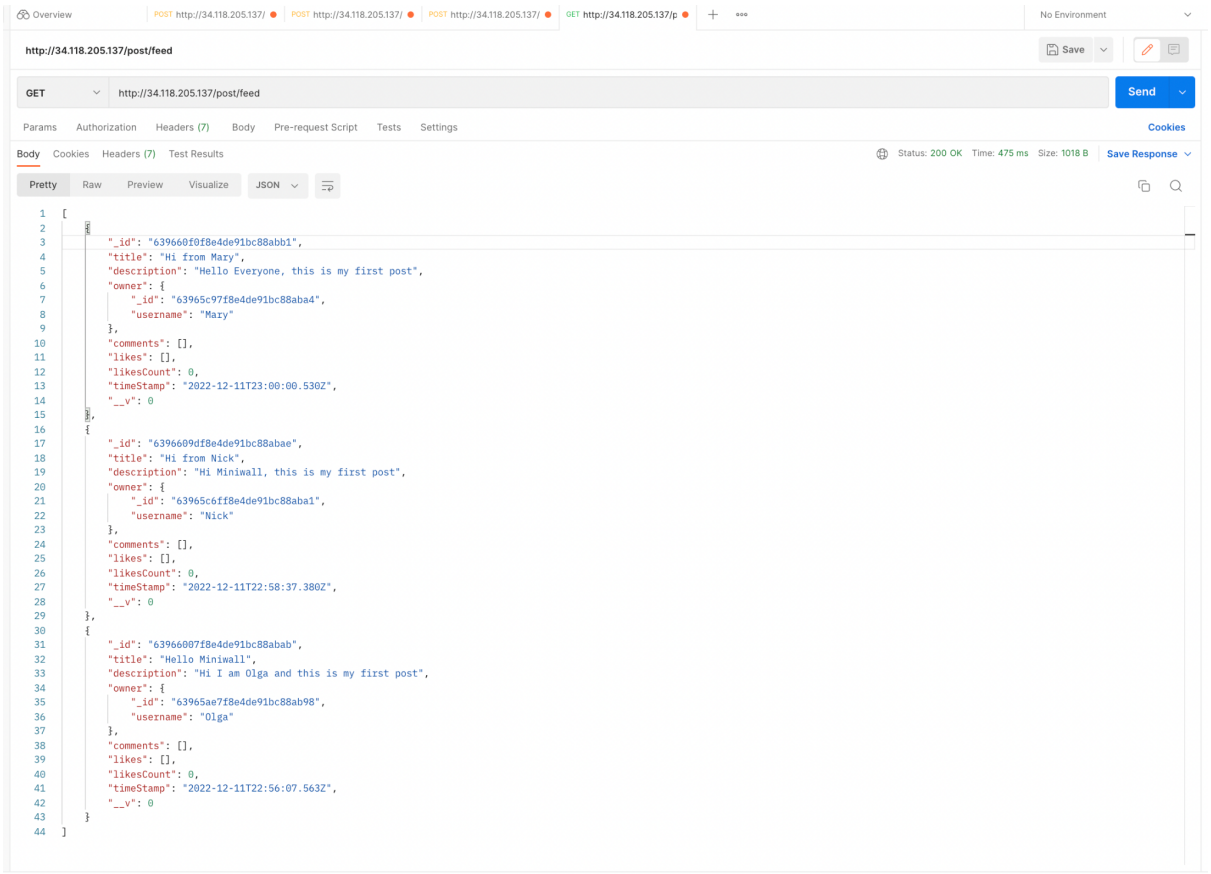


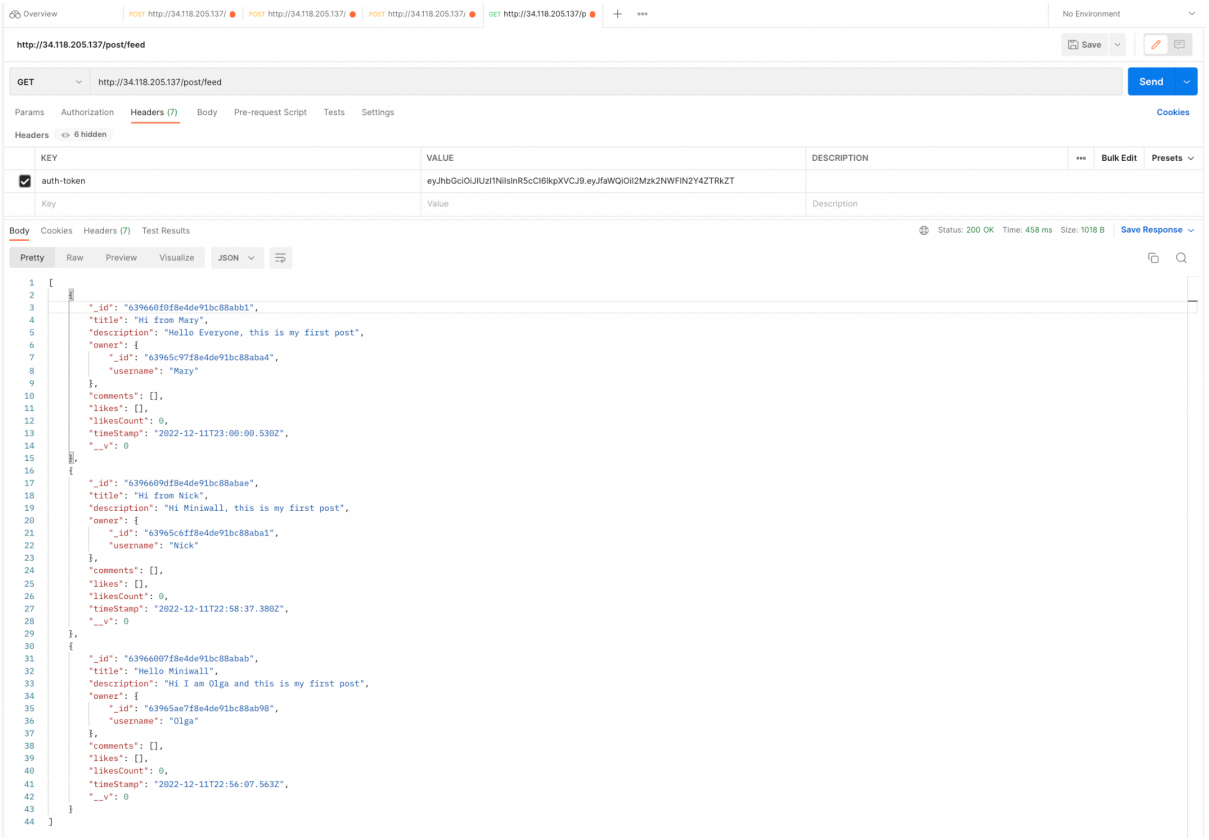
Test case 6:


Overview
POST http://34.118.205.137/
POST http://34.118.205.137/
POST http://34.118.205.137/
No Environment
http://34.118.205.137/post/create
Save
POST
http://34.118.205.137/post/create
Send
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
Cookies
Headers
8 hidden
KEY
VALUE
DESCRIPTION
Bulk Edit
Presets
auth-token
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOi...
Key
Value
Description

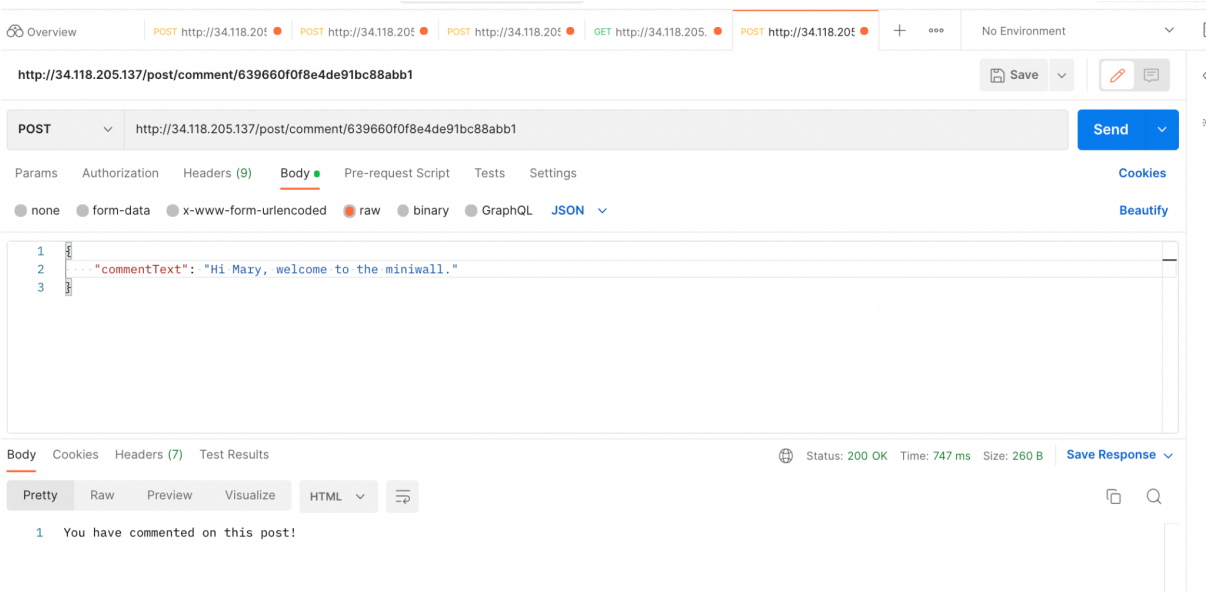
Body
Cookies
Headers (7)
Test Results
Status: 200 OK
Time: 332 ms
Size: 469 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"title": "Hi from Mary",
"description": "Hello Everyone, this is my first post",
"owner": "63965c97f8e4de91bc88aba4",
"comments": [],
"likes": [],
"likesCount": 0,
"_id": "639660f0f8e4de91bc88abb1",
"timeStamp": "2022-12-11T23:00:00.530Z",
"__v": 0
}

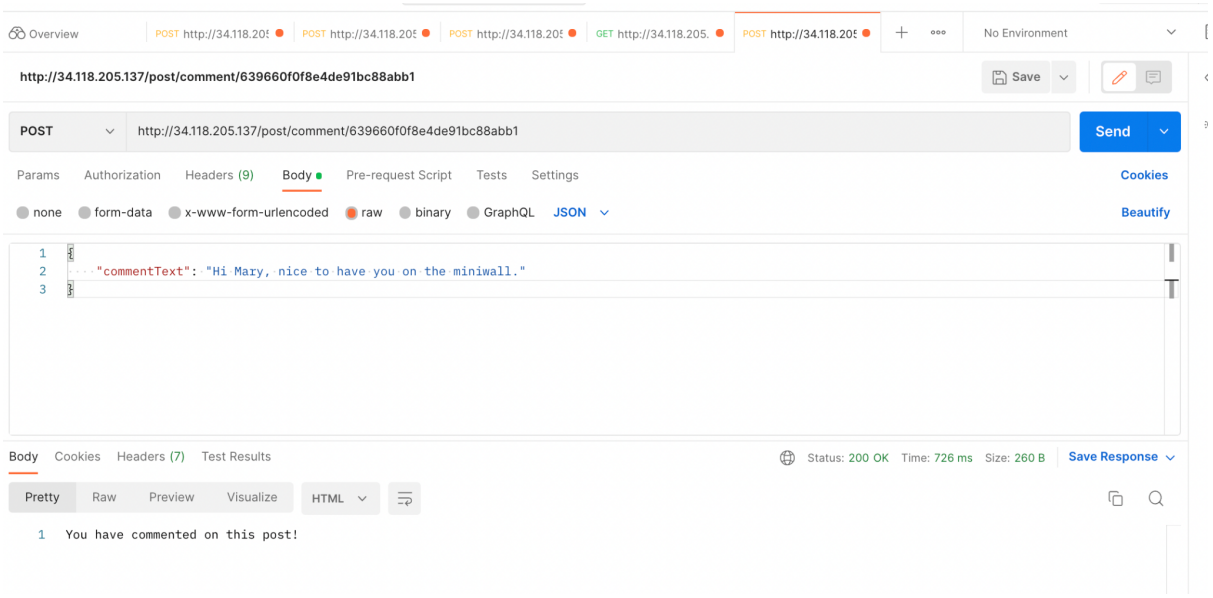
Test case 7:

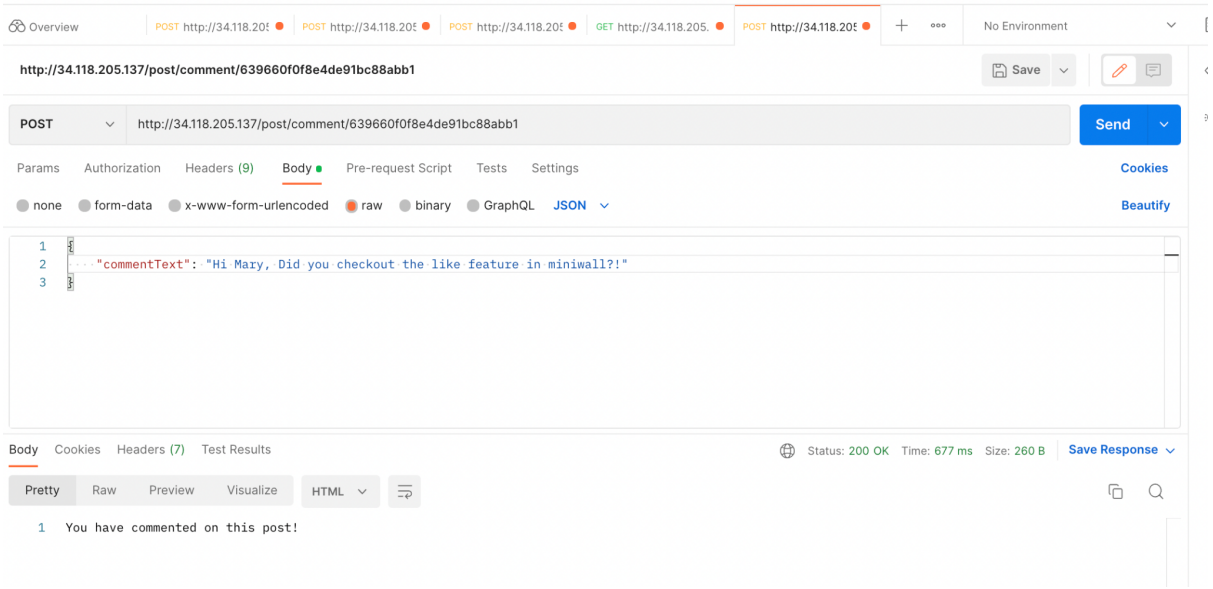

Overview
GET http://34.118.205.137/p
No Environment
http://34.118.205.137/post/feed
Save
GET
http://34.118.205.137/post/feed
Send
Params
Authorization
Headers (7)
Body
Pre-request Script
Tests
Settings
Cookies
Body
Cookies
Headers (7)
Test Results
Status: 200 OK
Time: 475 ms
Size: 1018 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
[
{
"_id": "639660f0f8e4de91bc88abb1",
"title": "Hi from Mary",
"description": "Hello Everyone, this is my first post",
"owner": {
"_id": "63965c97f8e4de91bc88aba4",
"username": "Mary"
},
"comments": [],
"likes": [],
"likesCount": 0,
"timeStamp": "2022-12-11T23:00:00.530Z",
"__v": 0
},
{
"_id": "6396609df8e4de91bc88abae",
"title": "Hi from Nick",
"description": "Hi Miniwall, this is my first post",
"owner": {
"_id": "63965c6ff8e4de91bc88aba1",
"username": "Nick"
},
"comments": [],
"likes": [],
"likesCount": 0,
"timeStamp": "2022-12-11T22:58:37.380Z",
"__v": 0
},
{
"_id": "63966007f8e4de91bc88abab",
"title": "Hello Miniwall",
"description": "Hi I am Olga and this is my first post",
"owner": {
"_id": "63965ae7f8e4de91bc88ab98",
"username": "Olga"
},
"comments": [],
"likes": [],
"likesCount": 0,
"timeStamp": "2022-12-11T22:56:07.563Z",
"__v": 0
}
]

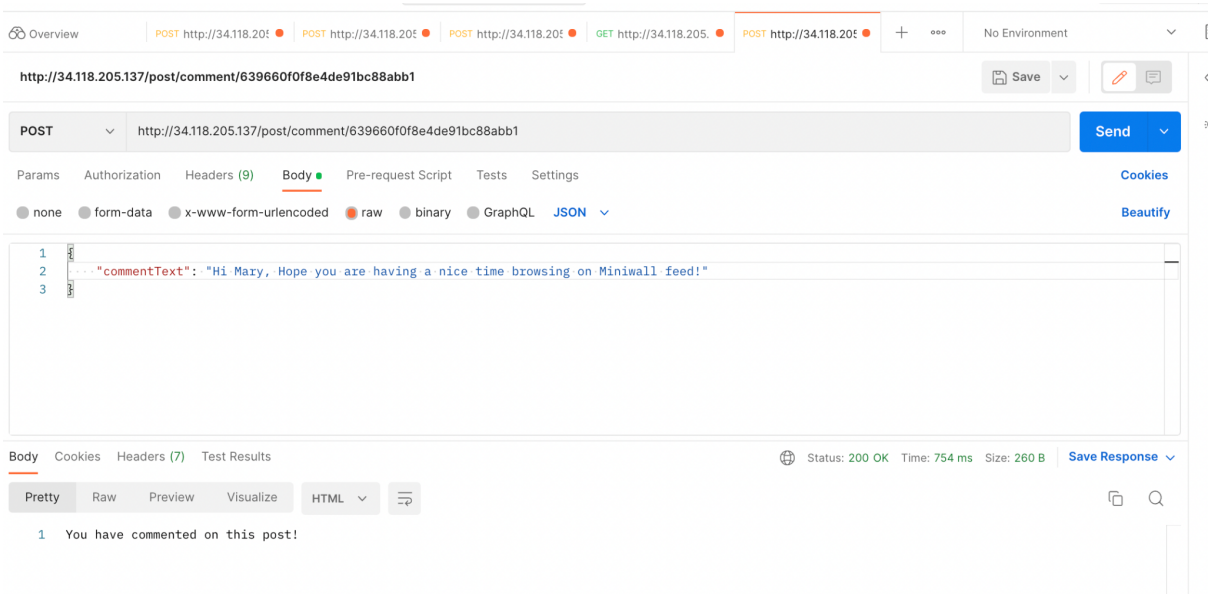

http://34.118.205.137/post/feed
GET
http://34.118.205.137/post/feed
Send
Params
Authorization
Headers (7)
Body
Pre-request Script
Tests
Settings
Cookies
KEY
VALUE
DESCRIPTION
Bulk Edit
Presets
auth-token
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI2Mzk2NWFlN2Y4ZTRkZT
Body
Cookies
Headers (7)
Test Results
Status: 200 OK
Time: 458 ms
Size: 1018 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
[
{
"_id": "639660f0f8e4de91bc88abb1",
"title": "Hi from Mary",
"description": "Hello Everyone, this is my first post",
"owner": {
"_id": "63965c97f8e4de91bc88aba4",
"username": "Mary"
},
"comments": [],
"likes": [],
"likesCount": 0,
"timeStamp": "2022-12-11T23:00:00.530Z",
"__v": 0
},
{
"_id": "6396609df8e4de91bc88abae",
"title": "Hi from Nick",
"description": "Hi Miniwall, this is my first post",
"owner": {
"_id": "63965c6ff8e4de91bc88aba1",
"username": "Nick"
},
"comments": [],
"likes": [],
"likesCount": 0,
"timeStamp": "2022-12-11T22:58:37.380Z",
"__v": 0
},
{
"_id": "63966007f8e4de91bc88abab",
"title": "Hello Miniwall",
"description": "Hi I am Olga and this is my first post",
"owner": {
"_id": "63965ae7f8e4de91bc88ab98",
"username": "Olga"
},
"comments": [],
"likes": [],
"likesCount": 0,
"timeStamp": "2022-12-11T22:56:07.563Z",
"__v": 0
}
]

Test case 8:

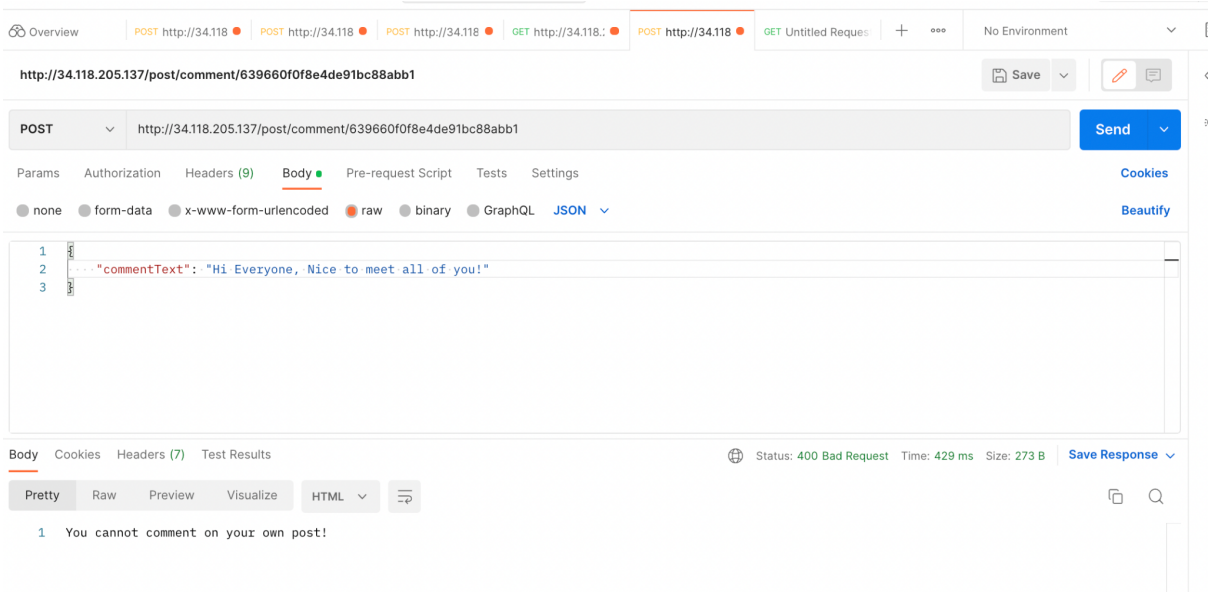

Overview
No Environment
http://34.118.205.137/post/comment/639660f0f8e4de91bc88abb1
Save
POST
http://34.118.205.137/post/comment/639660f0f8e4de91bc88abb1
Send
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"commentText": "Hi Mary, welcome to the miniwall."
}
Body
Cookies
Headers (7)
Test Results
Status: 200 OK
Time: 747 ms
Size: 260 B
Save Response
Pretty
Raw
Preview
Visualize
HTML
You have commented on this post!


Overview
POST http://34.118.205
POST http://34.118.205
POST http://34.118.205
GET http://34.118.205.
POST http://34.118.205
No Environment
http://34.118.205.137/post/comment/639660f0f8e4de91bc88abb1
Save
POST
http://34.118.205.137/post/comment/639660f0f8e4de91bc88abb1
Send
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
1
2
3
"commentText": "Hi Mary, nice to have you on the miniwall."
Body
Cookies
Headers (7)
Test Results
Status: 200 OK
Time: 726 ms
Size: 260 B
Save Response
Pretty
Raw
Preview
Visualize
HTML
1
You have commented on this post!

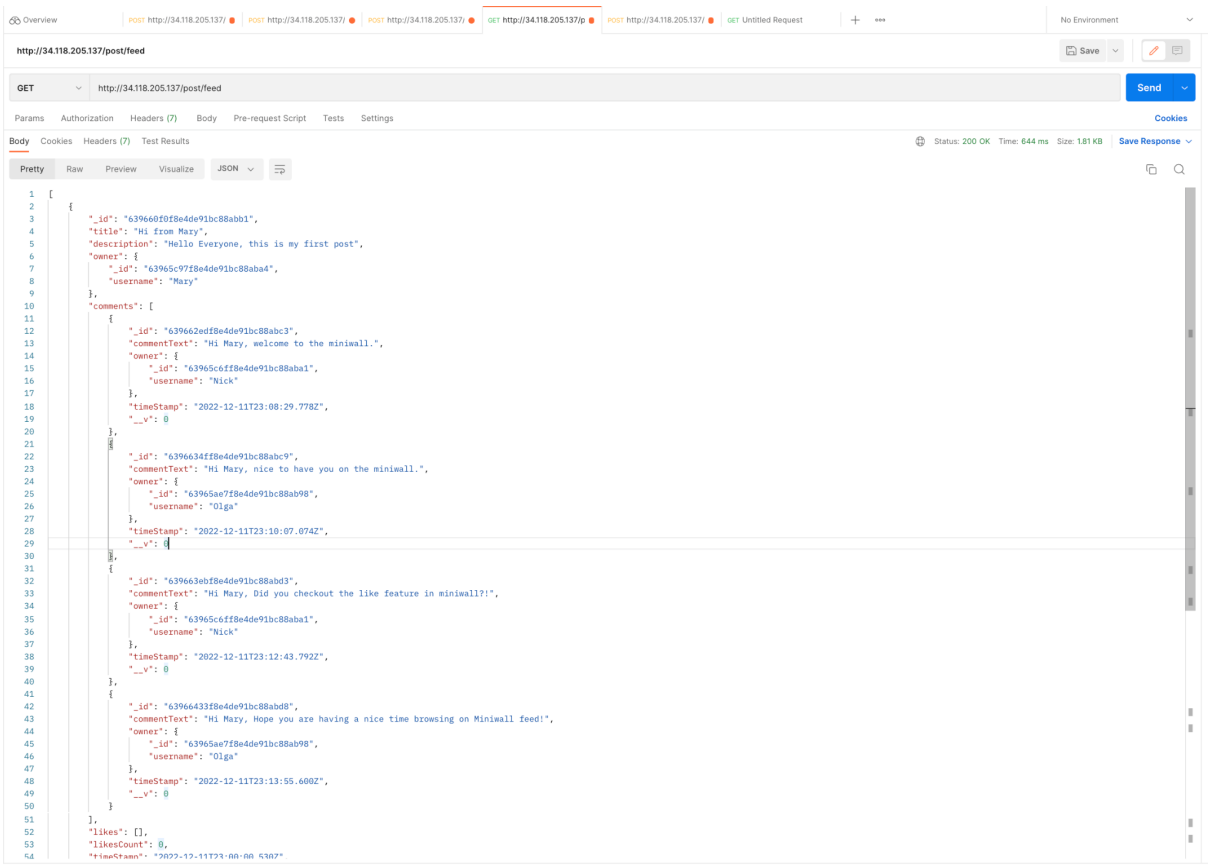

Overview
POST http://34.118.205
POST http://34.118.205
POST http://34.118.205
GET http://34.118.205.
POST http://34.118.205
No Environment
http://34.118.205.137/post/comment/639660f0f8e4de91bc88abb1
Save
POST
http://34.118.205.137/post/comment/639660f0f8e4de91bc88abb1
Send
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
1
2
3
"commentText": "Hi Mary, Did you checkout the like feature in miniwall?!"
Body
Cookies
Headers (7)
Test Results
Status: 200 OK
Time: 677 ms
Size: 260 B
Save Response
Pretty
Raw
Preview
Visualize
HTML
1
You have commented on this post!

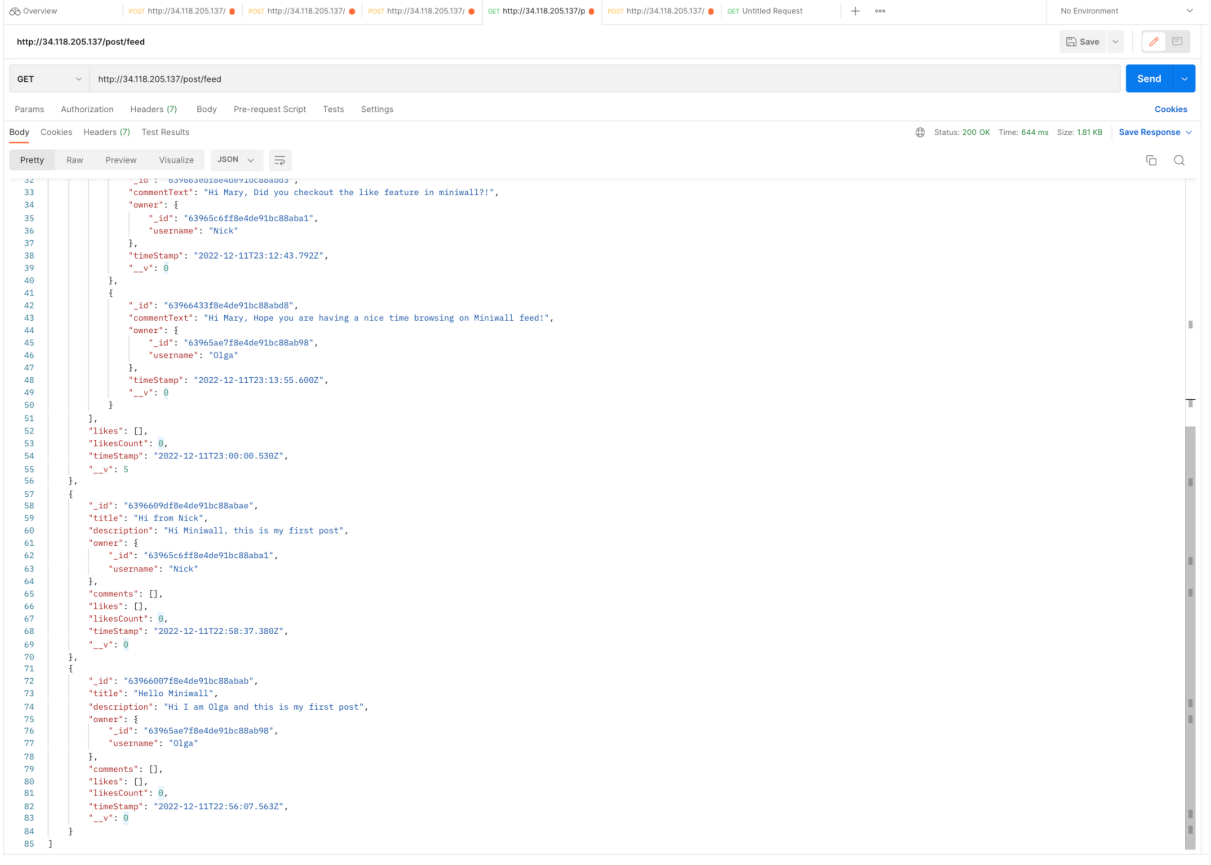


Test case 9:

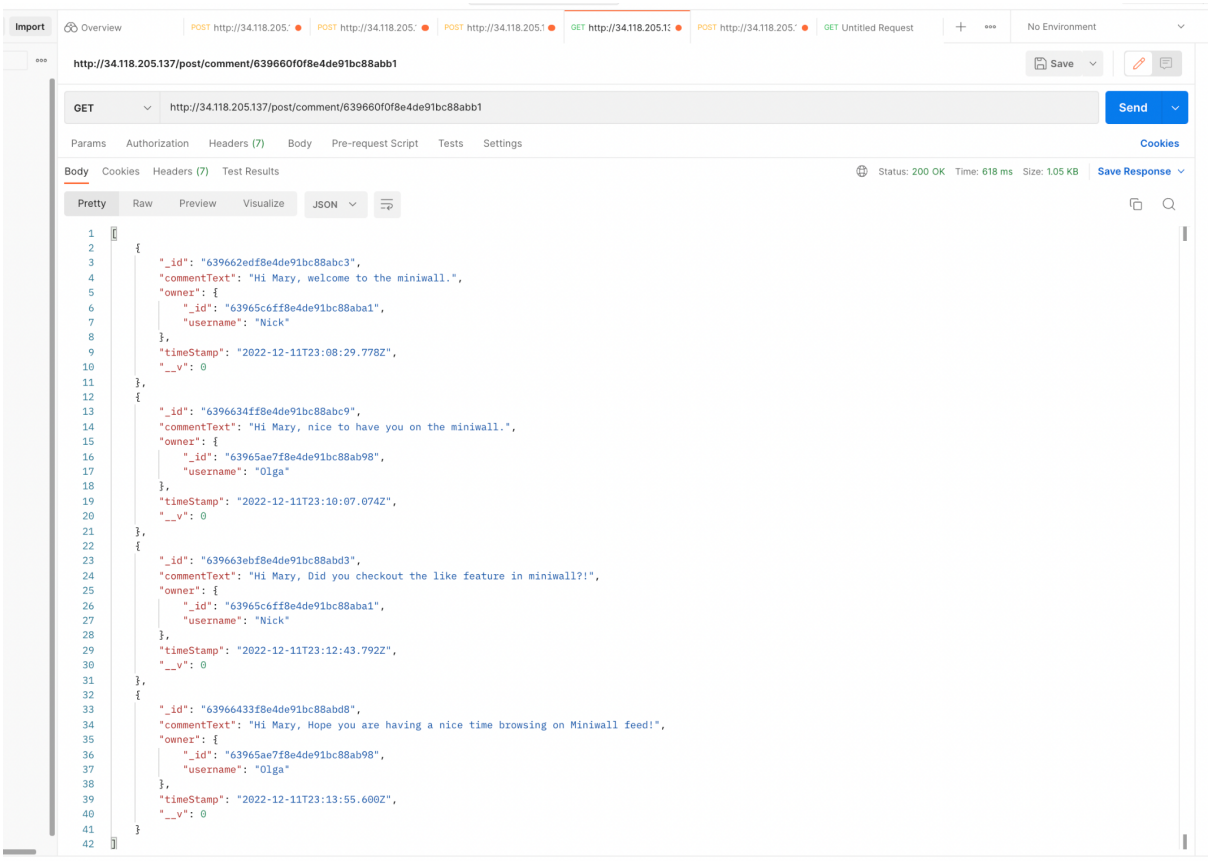


Test case 10:

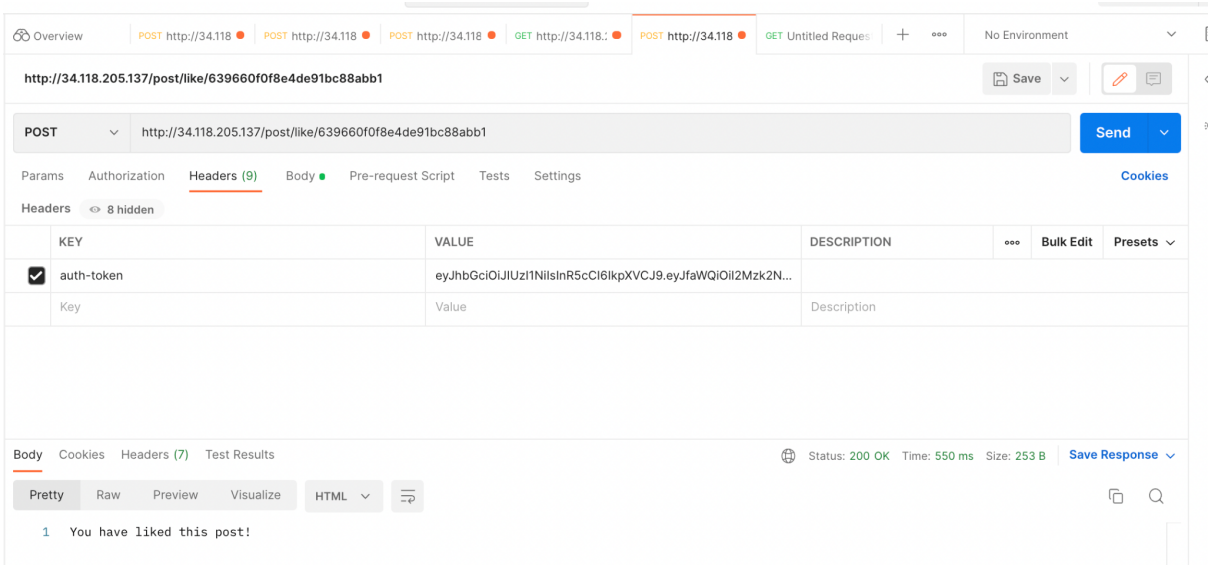

Overview
POST http://34.118.205.137/
POST http://34.118.205.137/
POST http://34.118.205.137/
GET http://34.118.205.137/p
POST http://34.118.205.137/
GET Untitled Request
No Environment
http://34.118.205.137/post/feed
Save
GET
http://34.118.205.137/post/feed
Send
Params
Authorization
Headers (7)
Body
Pre-request Script
Tests
Settings
Cookies
Body
Cookies
Headers (7)
Test Results
Status: 200 OK
Time: 644 ms
Size: 1.81 KB
Save Response
Pretty
Raw
Preview
Visualize
JSON
[
{
"_id": "639660f0f8e4de91bc88abb1",
"title": "Hi from Mary",
"description": "Hello Everyone, this is my first post",
"owner": {
"_id": "63965c97f8e4de91bc88aba4",
"username": "Mary"
},
"comments": [
{
"_id": "639662edf8e4de91bc88abc3",
"commentText": "Hi Mary, welcome to the miniwall.",
"owner": {
"_id": "63965c6ff8e4de91bc88aba1",
"username": "Nick"
},
"timeStamp": "2022-12-11T23:08:29.778Z",
"__v": 0
},
{
"_id": "6396634ff8e4de91bc88abc9",
"commentText": "Hi Mary, nice to have you on the miniwall.",
"owner": {
"_id": "63965ae7f8e4de91bc88ab98",
"username": "Olga"
},
"timeStamp": "2022-12-11T23:10:07.074Z",
"__v": 0
},
{
"_id": "639663ebf8e4de91bc88abd3",
"commentText": "Hi Mary, Did you checkout the like feature in miniwall?!",
"owner": {
"_id": "63965c6ff8e4de91bc88aba1",
"username": "Nick"
},
"timeStamp": "2022-12-11T23:12:43.792Z",
"__v": 0
},
{
"_id": "63966433f8e4de91bc88abd8",
"commentText": "Hi Mary, Hope you are having a nice time browsing on Miniwall feed!",
"owner": {
"_id": "63965ae7f8e4de91bc88ab98",
"username": "Olga"
},
"timeStamp": "2022-12-11T23:13:55.600Z",
"__v": 0
}
],
"likes": [],
"likesCount": 0,

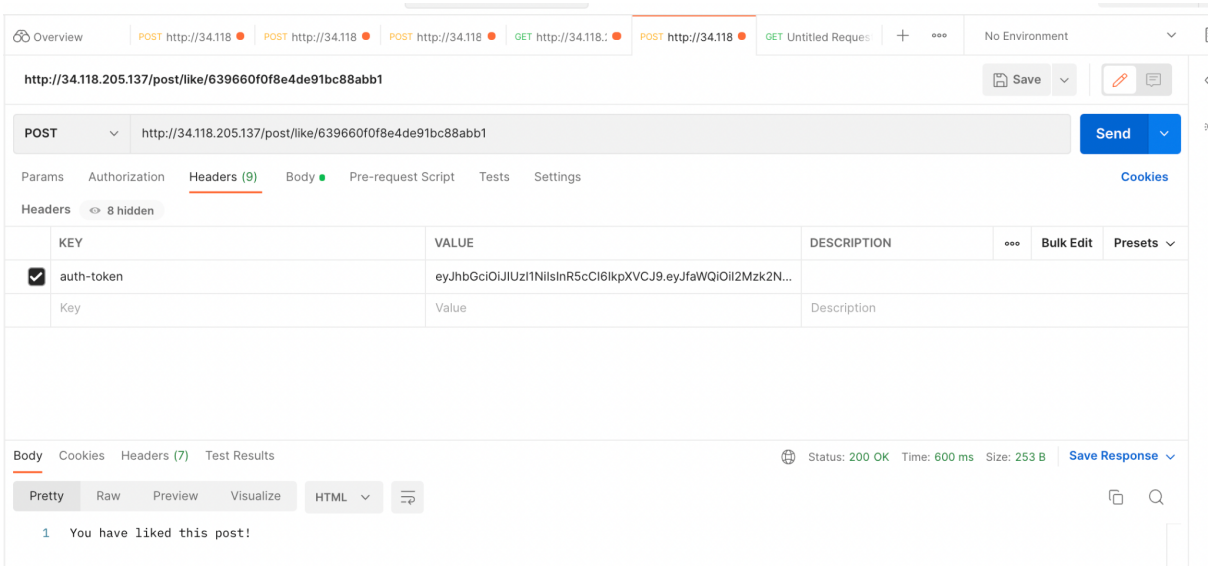

Overview
POST http://34.118.205.137/
POST http://34.118.205.137/
POST http://34.118.205.137/
GET http://34.118.205.137/p
POST http://34.118.205.137/
GET Untitled Request
No Environment
http://34.118.205.137/post/feed
Save
GET
http://34.118.205.137/post/feed
Send
Params
Authorization
Headers (7)
Body
Pre-request Script
Tests
Settings
Cookies
Body
Cookies
Headers (7)
Test Results
Status: 200 OK
Time: 644 ms
Size: 1.81 KB
Save Response
Pretty
Raw
Preview
Visualize
JSON
"commentText": "Hi Mary, Did you checkout the like feature in miniwall?!",
"owner": {
"_id": "63965c6ff8e4de91bc88aba1",
"username": "Nick"
},
"timeStamp": "2022-12-11T23:12:43.792Z",
"__v": 0
},
{
"_id": "63966433f8e4de91bc88abd8",
"commentText": "Hi Mary, Hope you are having a nice time browsing on Miniwall feed!",
"owner": {
"_id": "63965ae7f8e4de91bc88ab98",
"username": "Olga"
},
"timeStamp": "2022-12-11T23:13:55.600Z",
"__v": 0
}
],
"likes": [],
"likesCount": 0,
"timeStamp": "2022-12-11T23:00:00.530Z",
"__v": 5
},
{
"_id": "6396609df8e4de91bc88abae",
"title": "Hi from Nick",
"description": "Hi Miniwall, this is my first post",
"owner": {
"_id": "63965c6ff8e4de91bc88aba1",
"username": "Nick"
},
"comments": [],
"likes": [],
"likesCount": 0,
"timeStamp": "2022-12-11T22:58:37.380Z",
"__v": 0
},
{
"_id": "63966007f8e4de91bc88abab",
"title": "Hello Miniwall",
"description": "Hi I am Olga and this is my first post",
"owner": {
"_id": "63965ae7f8e4de91bc88ab98",
"username": "Olga"
},
"comments": [],
"likes": [],
"likesCount": 0,
"timeStamp": "2022-12-11T22:56:07.563Z",
"__v": 0
}
]

Test case 11:

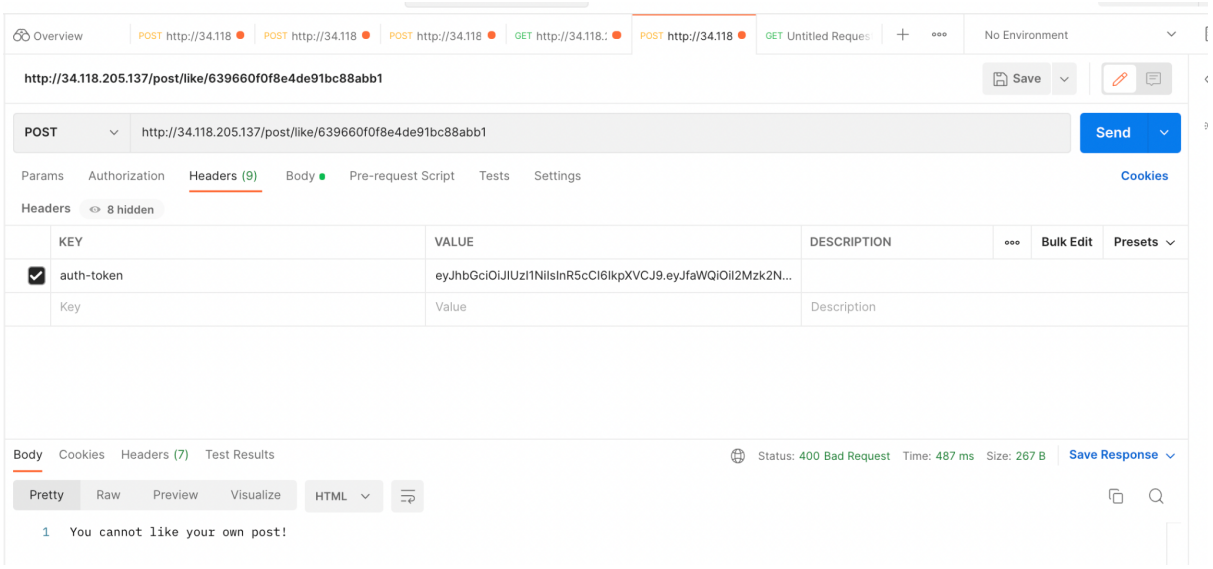


Test case 12:

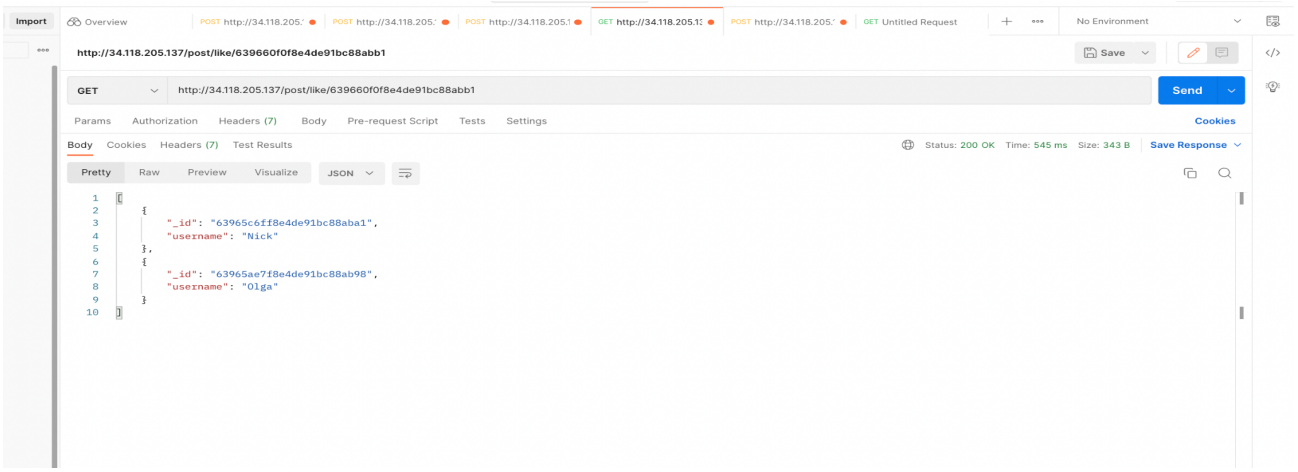




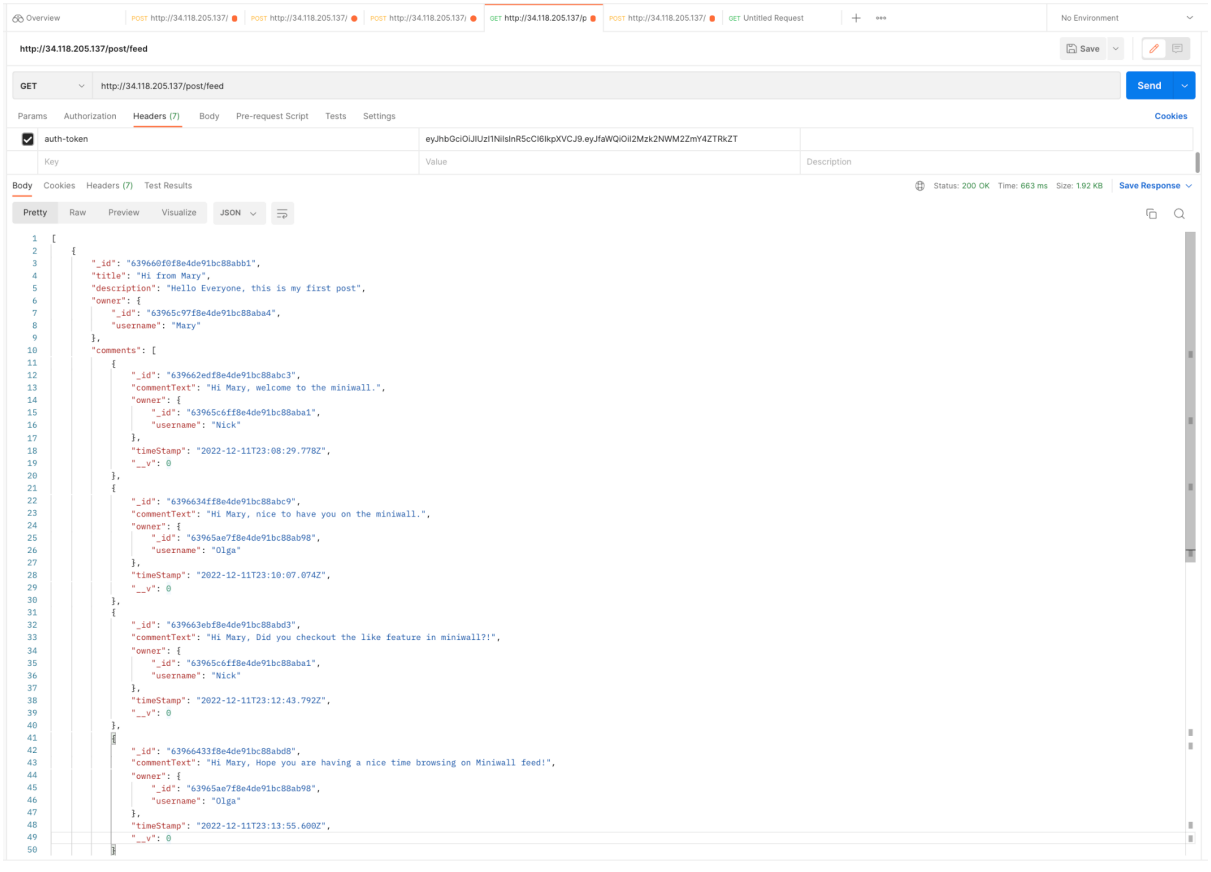
Test case 13:

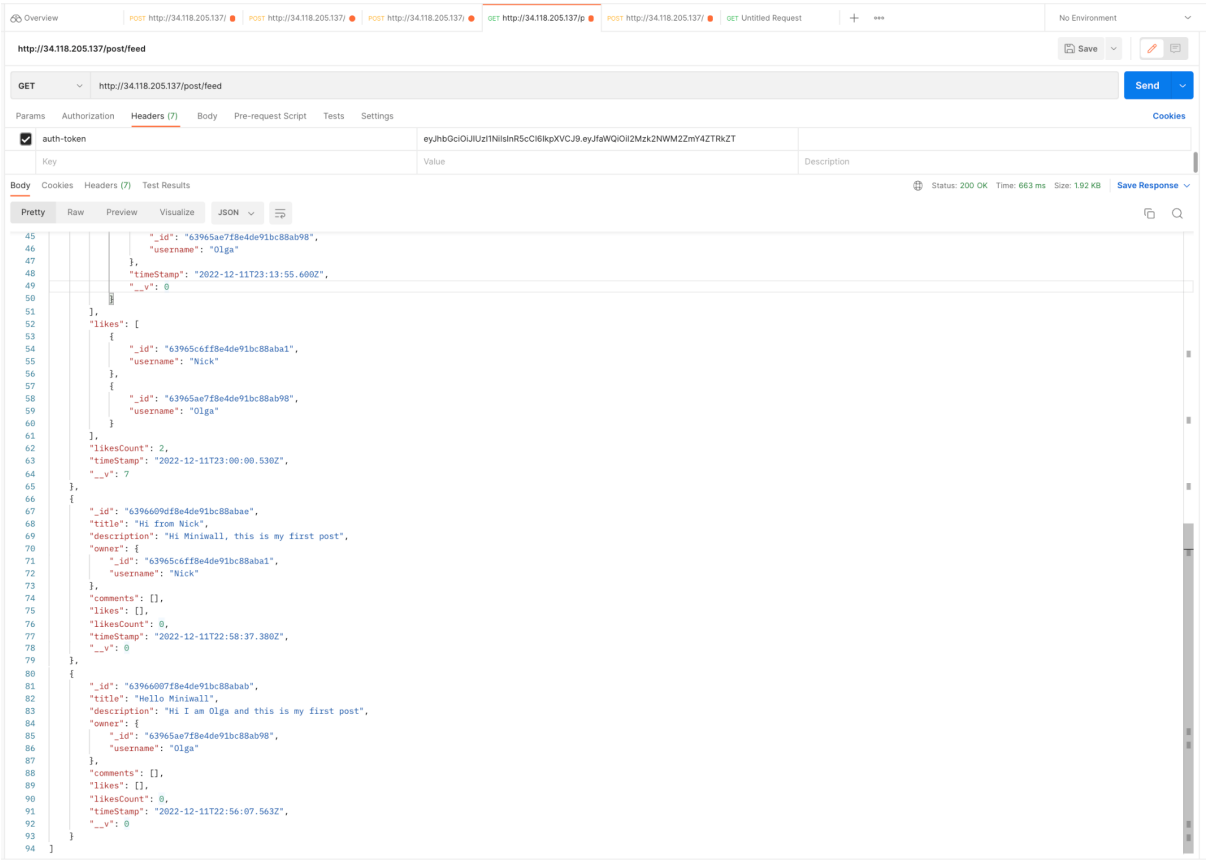


Test case 14:


http://34.118.205.137/post/like/639660f0f8e4de91bc88abb1
GET
http://34.118.205.137/post/like/639660f0f8e4de91bc88abb1
Send
Params Authorization Headers (7) Body Pre-request Script Tests Settings
Cookies
Body Cookies Headers (7) Test Results
Status: 200 OK Time: 545 ms Size: 343 B
Save Response
Pretty Raw Preview Visualize JSON
[
{
"_id": "63965c6ff8e4de91bc88aba1",
"username": "Nick"
},
{
"_id": "63965ae7f8e4de91bc88ab98",
"username": "Olga"
}
]

Test case 15:


http://34.118.205.137/post/feed
GET
http://34.118.205.137/post/feed
Send
Params Authorization Headers (7) Body Pre-request Script Tests Settings
Cookies
auth-token
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI2Mzk2NWM2ZmY4ZTRkZT
Key
Value
Description
Body Cookies Headers (7) Test Results
Status: 200 OK Time: 663 ms Size: 1.92 KB
Save Response
Pretty Raw Preview Visualize JSON
[
{
"_id": "639660f0f8e4de91bc88abb1",
"title": "Hi from Mary",
"description": "Hello Everyone, this is my first post",
"owner": {
"_id": "63965c97f8e4de91bc88aba4",
"username": "Mary"
},
"comments": [
{
"_id": "639662edf8e4de91bc88abc3",
"commentText": "Hi Mary, welcome to the miniwall.",
"owner": {
"_id": "63965c6ff8e4de91bc88aba1",
"username": "Nick"
},
"timeStamp": "2022-12-11T23:08:29.778Z",
"__v": 0
},
{
"_id": "6396634ff8e4de91bc88abc9",
"commentText": "Hi Mary, nice to have you on the miniwall.",
"owner": {
"_id": "63965ae7f8e4de91bc88ab98",
"username": "Olga"
},
"timeStamp": "2022-12-11T23:10:07.074Z",
"__v": 0
},
{
"_id": "639663ebf8e4de91bc88abd3",
"commentText": "Hi Mary, Did you checkout the like feature in miniwall?!",
"owner": {
"_id": "63965c6ff8e4de91bc88aba1",
"username": "Nick"
},
"timeStamp": "2022-12-11T23:12:43.792Z",
"__v": 0
},
{
"_id": "63966433f8e4de91bc88abd8",
"commentText": "Hi Mary, Hope you are having a nice time browsing on Miniwall feed!",
"owner": {
"_id": "63965ae7f8e4de91bc88ab98",
"username": "Olga"
},
"timeStamp": "2022-12-11T23:13:55.600Z",
"__v": 0
}



References

Google(2022) Google terms of service. Available at: https://mongoosejs.com/docs/guide.html (Accessed: 10th December 2022)

Google(2022) Google terms of service. Available at: https://www.npmjs.com/package/dotenv (Accessed: 10th December 2022)

Google(2022) Google terms of service. Available at: https://joi.dev/api/?v=17.7.0#introduction (Accessed: 10th December 2022)

Google(2022) Google terms of service. Available at: https://www.npmjs.com/package/body-parser (Accessed: 10th December 2022)

Google(2022) Google terms of service. Available at: https://www.npmjs.com/package/bcryptjs (Accessed: 10th December 2022)

Google(2022) Google terms of service. Available at: https://www.npmjs.com/package/jsonwebtoken (Accessed: 10th December 2022)