MoMo SMS Transaction API Project

MOMO SMS REST API

This is a REST API designed to access parsed mobile money sms transactions made with the different details that constitute those transactions like: their type, amount, receiver, or the sender.
Our API uses the data format of json where we are able to access the files originally in xml format.
http://localhost:8000 this is our Base URL to access our REST API.

AUTHENTICATION

This API uses HTTP Basic authentication to restrict access to protected endpoints
Clients must send an authorization header containing Base64-encoded username and password with each request.
For the basic authentication these are the valid Credentials:
. Username : admin
. Password : password
What happens when there is failure in authentication:
. Request without valid authentications return 401 Unauthorized

ENDPOINT LIST

In our REST API we used the standard HTTP methods known as:
. GET
. POST
. PUT
. DELETE
Here are the endpoints URL for our MOMO SMS REST API :

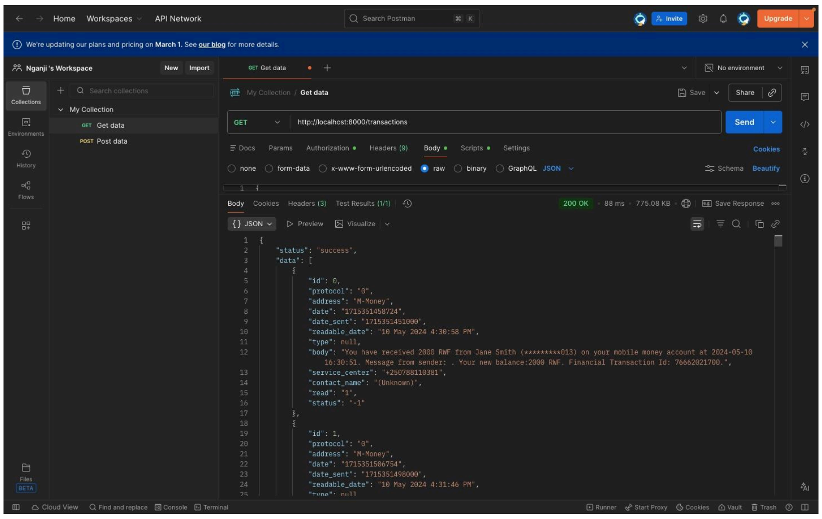
GET

URL : http://localhost:8000/transactions

EndPoint: GET/transactions

With this particular endpoint we are able to get all the transactions in the MOMO SMS database.
To access this particular endpoint the user/client first has to pass through the basic authentication process to be able to make the request.

As seen in the image below the status code returned is 200 means OK.



GET http://localhost:8000/transactions

200 OK · 88 ms · 775.08 KB

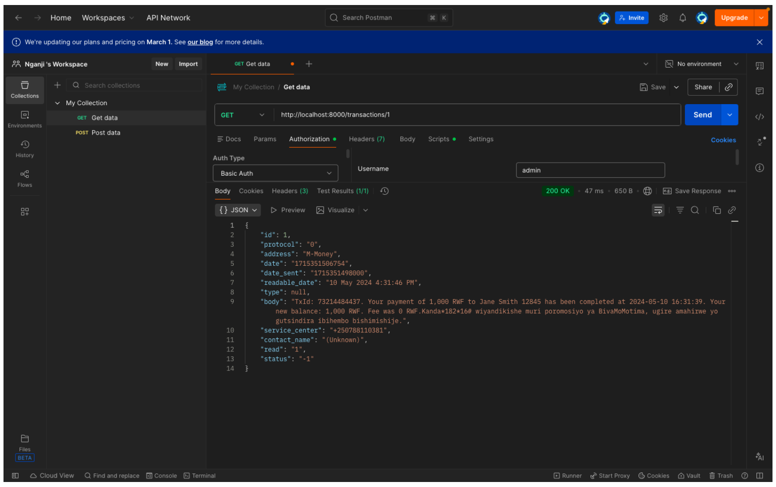
```
{
    "status": "success",
    "data": [
        {
            "id": 0,
            "protocol": "0",
            "address": "M-Money",
            "date": "1715351458724",
            "date_sent": "1715351451000",
            "readable_date": "10 May 2024 4:30:58 PM",
            "type": null,
            "body": "You have received 2000 RWF from Jane Smith (*********013) on your mobile money account at 2024-05-10
               16:30:51. Message from sender: . Your new balance:2000 RWF. Financial Transaction Id: 76662021700.",
            "service_center": "+250788110381",
            "contact_name": "(Unknown)",
            "read": "1",
            "status": "-1"
        },
        {
            "id": 1,
            "protocol": "0",
            "address": "M-Money",
            "date": "1715351506754",
            "date_sent": "1715351498000",
            "readable_date": "10 May 2024 4:31:46 PM",
            "type": null
```

GET/ID

URL: http://localhost:8000/transactions/1

Endpoint: GET/transactions/id

This endpoint allows us to access only one transaction depending on their id.
Authentication required is the basic authentication where the USERNAME and Password are required.
The status code with this particular endpoint is : 200 OK



GET http://localhost:8000/transactions/1

Auth Type: Basic Auth

Username: admin

200 OK · 47 ms · 650 B

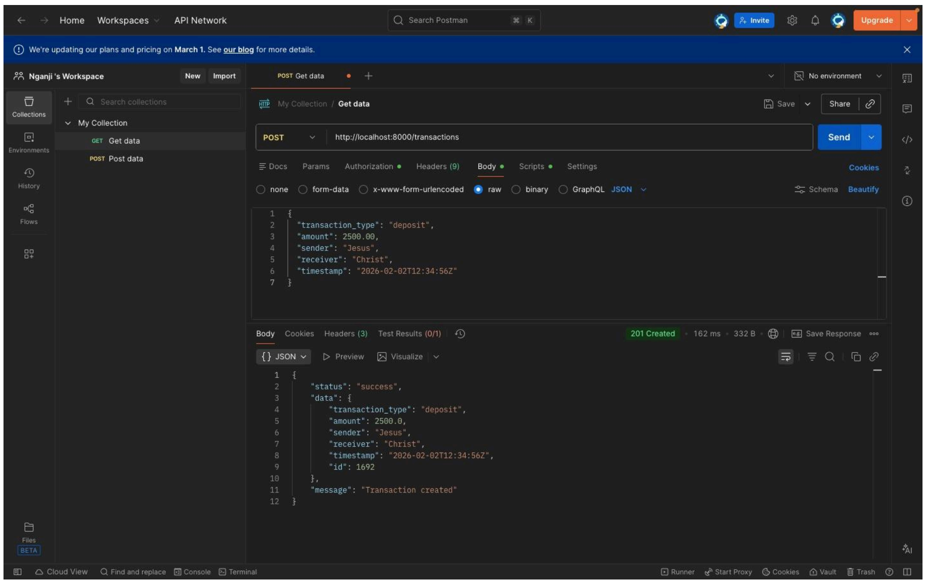
```
{
    "id": 1,
    "protocol": "0",
    "address": "M-Money",
    "date": "1715351506754",
    "date_sent": "1715351498000",
    "readable_date": "10 May 2024 4:31:46 PM",
    "type": null,
    "body": "TxId: 73214484437. Your payment of 1,000 RWF to Jane Smith 12845 has been completed at 2024-05-10 16:31:39. Your
        new balance: 1,000 RWF. Fee was 0 RWF.Kanda*182*16# wiyandikishe muri poromosiyo ya BivaMoMotima, ugire amahirwe yo
        gutsindira ibihembo bishimishije.",
    "service_center": "+250788110381",
    "contact_name": "(Unknown)",
    "read": "1",
    "status": "-1"
}
```

POST

URL: http://localhost:8000/transactions

EndPoint : POST /transactions

With this particular endpoint, the client is able to add their own transactions to the json file in the MOMO SMS database. This is done in the body and with specific fields already predefined.
To access this http request the user must first go through the basic authentication process.
The status code response got from this request is : 200 OK as seen below.



POST http://localhost:8000/transactions

```
{
  "transaction_type": "deposit",
  "amount": 2500.00,
  "sender": "Jesus",
  "receiver": "Christ",
  "timestamp": "2026-02-02T12:34:56Z"
}
```

201 Created · 162 ms · 332 B

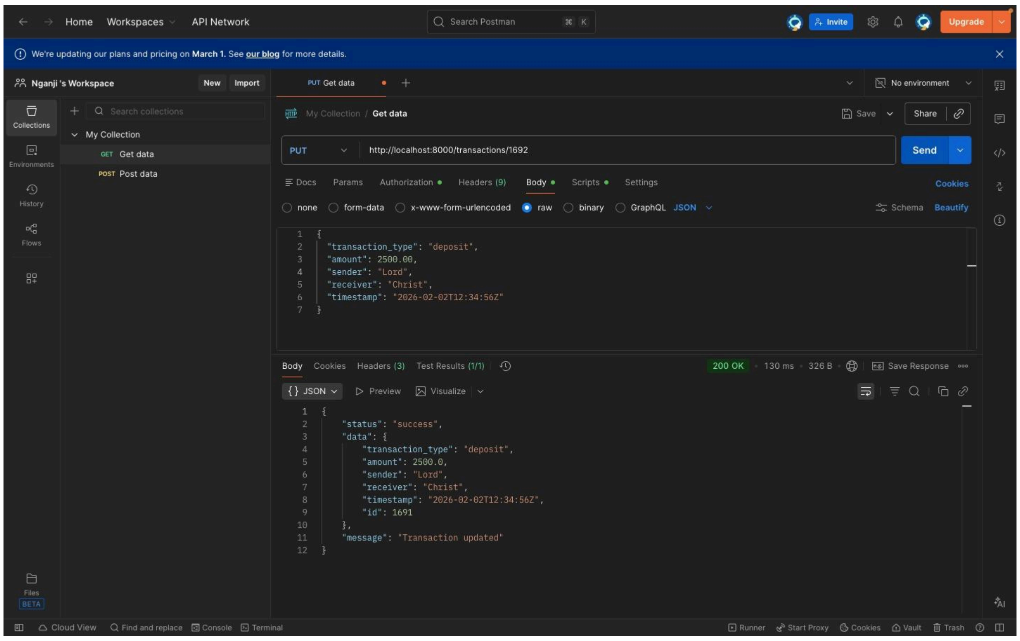
```
{
    "status": "success",
    "data": {
        "transaction_type": "deposit",
        "amount": 2500.0,
        "sender": "Jesus",
        "receiver": "Christ",
        "timestamp": "2026-02-02T12:34:56Z",
        "id": 1692
    },
    "message": "Transaction created"
}
```

PUT

URL : http://localhost:8000/transactions/1692

EndPoint: PUT /transactions/id

What PUT does is to help us modify the json file for a particular transactions using their id as reference.
This type of access is also under auth whereby the user needs to get past the authentication part before continuing.
Once the transaction has been updated the status code becomes : 200 OK.



We're updating our plans and pricing on **March 1**. See **our blog** for more details.

My Collection / **Get data**

PUT http://localhost:8000/transactions/1692

none · form-data · x-www-form-urlencoded · [✓ raw] · binary · GraphQL · JSON

```
{
  "transaction_type": "deposit",
  "amount": 2500.00,
  "sender": "Lord",
  "receiver": "Christ",
  "timestamp": "2026-02-02T12:34:56Z"
}
```

Body · Cookies · Headers (3) · Test Results (1/1) — 200 OK · 130 ms · 326 B

{ } JSON

```
{
    "status": "success",
    "data": {
        "transaction_type": "deposit",
        "amount": 2500.0,
        "sender": "Lord",
        "receiver": "Christ",
        "timestamp": "2026-02-02T12:34:56Z",
        "id": 1691
    },
    "message": "Transaction updated"
}
```

DELETE

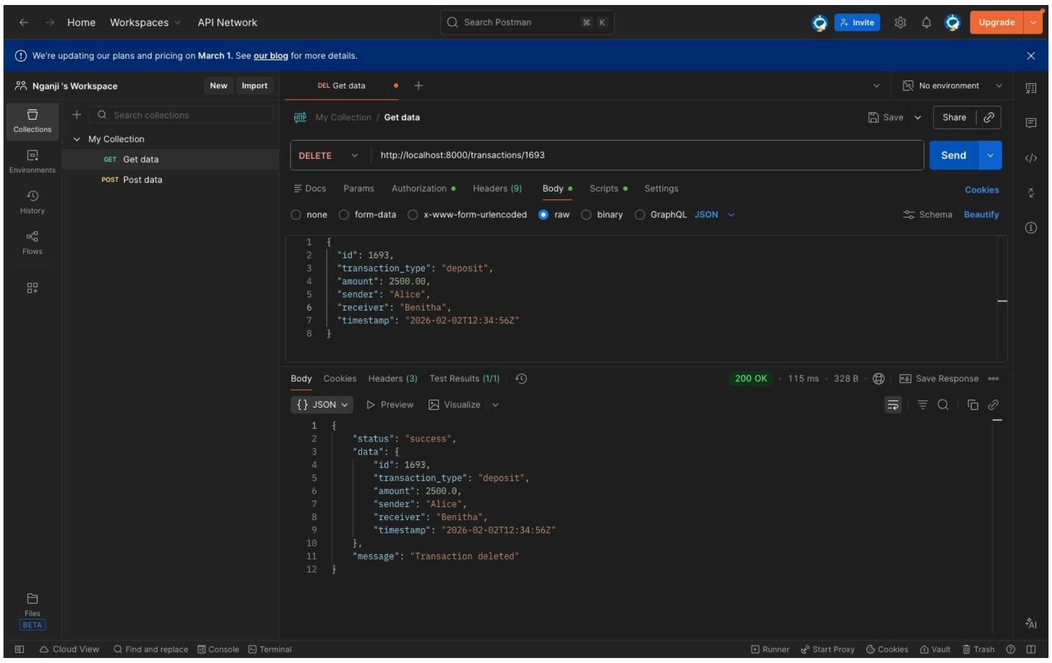
URL: http://localhost:8000/transactions/1693

EndPoint: DELETE /transactions/id

What this particular endpoint helps us achieve is that we are able to remove a particular transaction based on its id .

Basic authentication : Username and Password are required to access this field.

Once the transaction is deleted the status code of 200 OK is given as response.



DELETE http://localhost:8000/transactions/1693

```
{
  "id": 1693,
  "transaction_type": "deposit",
  "amount": 2500.00,
  "sender": "Alice",
  "receiver": "Benitha",
  "timestamp": "2026-02-02T12:34:56Z"
}
```

200 OK · 115 ms · 328 B

```
{
    "status": "success",
    "data": {
        "id": 1693,
        "transaction_type": "deposit",
        "amount": 2500.0,
        "sender": "Alice",
        "receiver": "Benitha",
        "timestamp": "2026-02-02T12:34:56Z"
    },
    "message": "Transaction deleted"
}
```

ERROR RESPONSE

401 Unauthorized means the server received your request but refused to authenticate it.

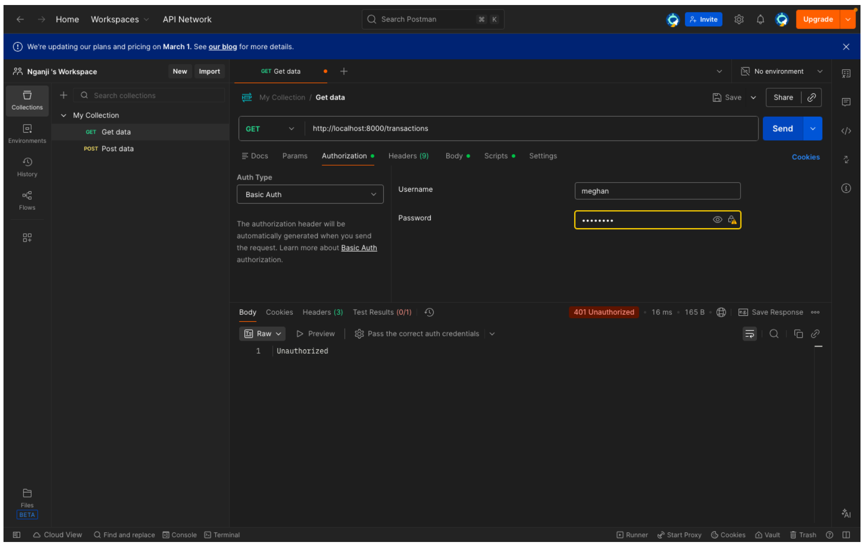
From the image:

- **We sending a GET request to** http://localhost:8000/transactions
- **Basic Auth is enabled in Postman**
- **A username** (meghan) **and password are provided**
- **The server responds with 401 Unauthorized and the message *"Unauthorized"***

In Other Words;

- **The username or password is incorrect, or**
- **The backend doesn't recognize Basic Auth for this endpoint, or**

- **The user exists but doesn’t have permission to access /transactions, or**
- **The API expects a different auth method (e.g., Bearer token / JWT instead of Basic Auth)**



GET http://localhost:8000/transactions

Auth Type: Basic Auth

The authorization header will be automatically generated when you send the request. Learn more about Basic Auth authorization.

Username: meghan

Password: ••••••••

401 Unauthorized · 16 ms · 165 B

```
Unauthorized
```

HTTPS STATUS CODES

These are some of the status codes that we encountered;

1. 200 OK

- **Meaning: The request was successful, and the server is returning the requested data.**
- **In MoMo SMS transaction context:**
  - **You requested the status of a transaction.**
  - **The server found the transaction and is returning the details.**

2. 401 Unauthorized

- **Meaning: The request requires authentication or the provided credentials are invalid.**

- **In MoMo SMS transaction context:**
    - **You did not include your API key or token, or it has expired.**
    - **The server refuses to process your transaction without proper authentication.**

3. 201 Created

- **Meaning: A new resource has been successfully created on the server.**
- **In MoMo SMS transaction context:**
    - **You initiated a new payment request.**
    - **The transaction was successfully recorded.**

4. 404 Not Found

- **Meaning: The requested resource could not be found on the server.**
- **In MoMo SMS transaction context:**
    - **You requested the status of a transaction with an ID that doesn't exist.**
    - **Or trying to fetch a phone number that's not registered.**

5. 400 Bad Request

- **Meaning: The request could not be processed because it was malformed or missing required data.**
- **In MoMo SMS transaction context:**
    - **Missing** receiver **or** amount **in your request.**
    - **Sending invalid characters in the phone number.**

Results Of DSA Comparison

This code shows how using the right data structures makes searching transactions faster and easier. Instead of checking every transaction one by one, it stores extra information that helps the program find results quickly. Searching by sender, receiver, or type becomes almost instant, and searching by amount or time is quicker because the data is already organized. Overall, this approach saves time and works much better when the number of transactions becomes large.

Limitations Of Basic Authentication

Hard-coded credentials

The username and password are written directly in the code. This is unsafe because anyone who can see the code can know the login details, and changing them requires editing the code.

Weak security

Basic Authentication only uses Base64 encoding, which is not encryption. If the request is intercepted (especially without HTTPS), the username and password can be easily decoded.

No user management

This code supports only one fixed user (admin). It cannot handle multiple users, roles, or permissions.

No session or token handling

The username and password must be sent with every request, increasing the risk of exposure and making it less secure than token-based methods.

Poor scalability

As the system grows, managing users and security using this approach becomes difficult and impractical.

Limited error feedback

The function only returns True or False, giving no clear reason why authentication failed