

# MoMo SMS Analytics API Documentation

This API manages MoMo SMS transactions which are classified in two parts (income & expenses). This supports CRUD operations and extra endpoints for analytics.

**Explore the full project on GitHub:**[https://github.com/BodeMurairi2/momo\\_sms\\_analytics.git](https://github.com/BodeMurairi2/momo_sms_analytics.git)

[https://github.com/BodeMurairi2/momo\\_sms\\_analytics.git](https://github.com/BodeMurairi2/momo_sms_analytics.git)

## 1. GET /transactions

Retrieve all transactions.

### Request Example:

GET /transactions

### Response Example:

HTTP/1.0 200 OK

Server: BaseHTTP/0.6 Python/3.12.7

Date: Wed, 01 Oct 2025 17:20:57 GMT

Content-Type: application/json

```
[
  {
    "status": "success"
  },
  {
    "id": 1,
    "transaction_timestamp": "2025-10-01 16:32",
    "transaction": "Income",
    "details": {
      "transaction_type": "receive",
      "amount": 2000.0,
      "currency": "RWF",
      "transaction_date": "On 2024-05-10 at 16:30"
    }
  },
  {
    "id": 2,
    "transaction_timestamp": "2025-10-01 16:32",
```

```

    "transaction": "Expense",
    "details": {
      "transaction_type": "payment",
      "amount": 1000.0,
      "currency": "RWF",
      "transaction_date": "On 2024-05-10 at 16:31"
    }
  }
]

```

⚠️⚠️⚠️ [screenshot of test]

```

[{"status": "success"}, {"id": 1, "transaction_timestamp": "2025-10-01 16:32", "transaction": "Income", "details": {"transaction_type": "receive", "amount": 2000.0, "currency": "RWF", "transaction_date": "On 2024-05-10 at 16:30"}}, {"id": 2, "transaction_timestamp": "2025-10-01 16:32", "transaction": "Expense", "details": {"transaction_type": "payment", "amount": 1000.0, "currency": "RWF", "transaction_date": "On 2024-05-10 at 16:31"}}, {"id": 3, "transaction_timestamp": "2025-10-01 16:32", "transaction": "Expense", "details": {"transaction_type": "payment", "amount": 600.0, "currency": "RWF", "transaction_date": "On 2024-05-10 at 21:32"}}, {"id": 4, "transaction_timestamp": "2025-10-01 16:32", "transaction": "Income", "details": {"transaction_type": "deposit", "amount": 40000.0, "currency": "RWF", "transaction_date": "On 2024-05-11 at 18:43"}}, {"id": 5, "transaction_timestamp": "2025-10-01 16:32", "transaction": "Expense", "details": {"transaction_type": "payment", "amount": 2000.0, "currency": "RWF", "transaction_date": "On 2024-05-11 at 18:48"}}, {"id": 6, "transaction_timestamp": "2025-10-01 16:32", "transaction": "Expense", "details": {"transaction_type": "transfer", "amount": 10000.0, "currency": "RWF", "transaction_date": "On 2024-05-11 at 20:34"}}, {"id": 7, "transaction_timestamp": "2025-10-01 16:32", "transaction": "Expense", "details": {"transaction_type": "transfer", "amount": 1000.0, "currency": "RWF", "transaction_date": "On 2024-05-12 at 03:47"}}, {"id": 8, "transaction_timestamp": "2025-10-01 16:32", "transaction": "Expense", "details": {"transaction_type": "payment", "amount": 2000.0, "currency": "RWF", "transaction_date": "On 2024-05-12 at 11:41"}}, {"id": 9, "transaction_timestamp": "2025-10-01 16:32", "transaction": "Expense", "details": {"transaction_type": "payment", "amount": 10900.0, "transaction_date": "On 2024-05-12 at 11:41"}}]

```

## 2. GET /transactions/{id}

Get details of a specific transaction.

### Request Example:

GET /transactions/2

### Response Example:

HTTP/1.0 200 OK  
Server: BaseHTTP/0.6 Python/3.12.7  
Date: Wed, 01 Oct 2025 17:20:57 GMT  
Content-Type: application/json

```
{
  "id": 5,
  "transaction_timestamp": "2025-10-01 16:32",
  "transaction": "Expense",
  "details": {
    "transaction_type": "payment",
    "amount": 2000.0,
    "currency": "RWF",
    "transaction_date": "On 2024-05-11 at 18:48"
  },
  "status": "success"
}
```

!!! [screenshot of test]

```
(base) bode-murairi@bode-murairi-IdeaPad-1-15IAU7:~/Documents/programming/ALU/mo
mo_sms_analytics/api/screenshots$ curl -i http://localhost:8000/transactions/5
HTTP/1.0 200 OK
Server: BaseHTTP/0.6 Python/3.12.7
Date: Wed, 01 Oct 2025 17:20:57 GMT
Content-Type: application/json

{"id": 5, "transaction_timestamp": "2025-10-01 16:32", "transaction": "Expense",
"details": {"transaction_type": "payment", "amount": 2000.0, "currency": "RWF",
"transaction_date": "On 2024-05-11 at 18:48"}, "status": "success"}(base) bode-
```

### 3. POST /signup

Sign up to the platform.

#### Request Example:

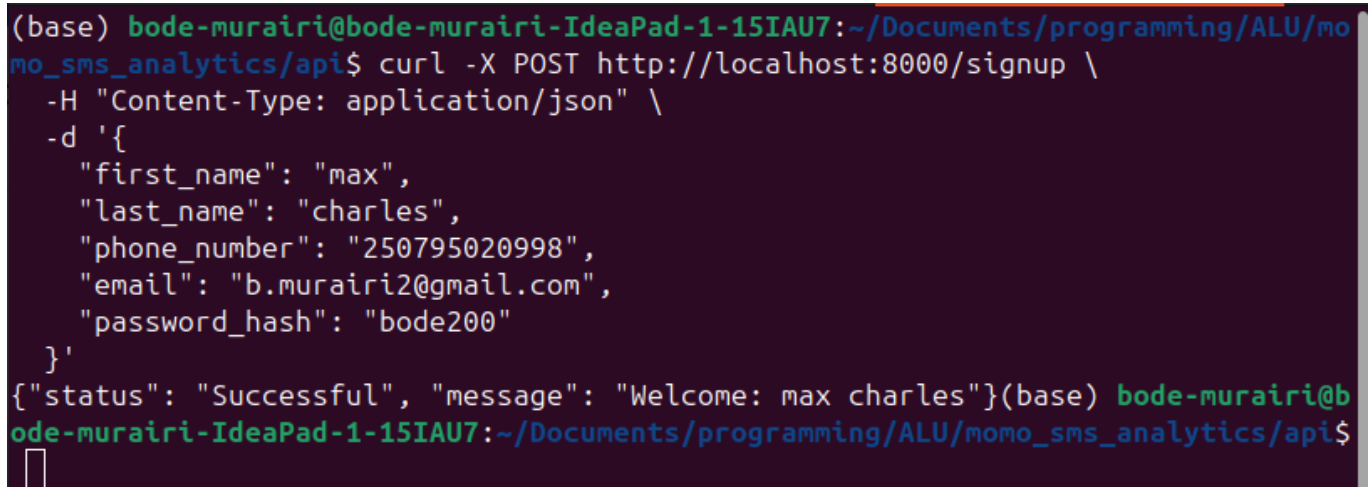
```
POST /signup
curl -X POST http://localhost:8000/signup \
-H "Content-Type: application/json" \
-d '{
  "first_name": "max",
  "last_name": "charles",
```

```
"phone_number": "250795020998",  
"email": "b.murairi2@gmail.com",  
"password_hash": "bode200"  
'
```

Response:

```
{"status": "Successful", "message": "Welcome: max charles"}
```

!!! [screenshot of test]



```
(base) bode-murairi@bode-murairi-IdeaPad-1-15IAU7:~/Documents/programming/ALU/mo  
mo_sms_analytics/api$ curl -X POST http://localhost:8000/signup \  
-H "Content-Type: application/json" \  
-d '{  
  "first_name": "max",  
  "last_name": "charles",  
  "phone_number": "250795020998",  
  "email": "b.murairi2@gmail.com",  
  "password_hash": "bode200"  
'  
{"status": "Successful", "message": "Welcome: max charles"}(base) bode-murairi@b  
ode-murairi-IdeaPad-1-15IAU7:~/Documents/programming/ALU/momo_sms_analytics/api$  
█
```

## 4. POST /transactions

Request to create a new transaction, you need first of all to be registered to our services, and log in. Follow the process in step 3 to register for our services using curl or request.

**Create a new transaction:**

**How to create a new transaction:**

```
curl -i -X POST http://localhost:8000/transactions \  

```

```
-u "b.murairi@alustudent.com:bode200" \  
-H "Content-Type: application/json" \  
-d '{  
  "type": "payment",  
  "transaction_date": "2024-06-08 15:28",  
  "amount": 2500  
'
```

Response:

HTTP/1.0 201 Created  
Server: BaseHTTP/0.6 Python/3.12.7  
Date: Wed, 01 Oct 2025 18:34:07 GMT  
Content-Type: application/json

```
{  
  "status": "Success",  
  "message": "Transaction added successfully",  
  "record": {  
    "id": 1661,  
    "transaction": "Expense",  
    "transaction_type": "payment",  
    "amount": 2500.0,  
    "currency": "RWF",  
    "transaction_date": "2024-06-08 at 15:28"  
  }  
}
```

### Error Codes:

- 422 – Missing required fields
- 400-Bad requests
- 401-Unauthorized

⚠️⚠️⚠️ [screenshot of test]

```
(base) bode-murairi@bode-murairi-IdeaPad-1-15IAU7:~/Documents/programming/ALU/momo_sms_analytics/api/screenshots$ curl -i -X POST http://localhost:8000/transactions \
-u "b.murairi@alustudent.com:bode200" \
-H "Content-Type: application/json" \
-d '{
  "type": "payment",
  "transaction_date": "2024-06-08 15:28",
  "amount": 2500
}'

HTTP/1.0 201 Created
Server: BaseHTTP/0.6 Python/3.12.7
Date: Wed, 01 Oct 2025 18:34:07 GMT
Content-Type: application/json

{"status": "Success", "message": "Transaction added successfully", "record": {"id": 1661, "transaction": "Expense", "transaction_type": "payment", "amount": 2500.0, "currency": "RWF", "transaction_date": "2024-06-08 at 15:28"}}(base) bode-murairi@bode-murairi-IdeaPad-1-15IAU7:~/Documents/programming/ALU/momo_sms_analytics/api/curl -i -X PUT http://localhost:8000/transactions \
```

## 4. PUT /transactions/{id}

Request to Update an existing transaction.

Similar to creating a transaction, put replaces completely an existing transaction. The user needs to be registered in order to perform this operation

### Request example

```
curl -i -X PUT http://localhost:8000/transactions/5 \
-u "bodemurairi2@gmail.com:Bode200" \
-H "Content-Type: application/json" \
-d '{
  "type": "payment",
  "transaction_date": "2024-06-08 15:28",
  "amount": 2500
}'
```

### Response Example:

HTTP/1.0 200 OK

Server: BaseHTTP/0.6 Python/3.12.7

Date: Wed, 01 Oct 2025 18:50:18 GMT

Content-Type: application/json

```
{
  "status": "Success",
  "message": "Transaction 5 replaced successfully",
  "record": {
    "id": 5,
    "transaction": "Expense",
    "transaction_type": "payment",
    "amount": 2500.0,
    "currency": "RWF",
    "transaction_date": "2024-06-08 at 15:28"
  }
}
```

### Error Codes:

- 422 – Missing required fields
- 400-Bad requests
- 401-Unauthorized

 [screenshot of test]

```
(base) bode-murairi@bode-murairi-IdeaPad-1-15IAU7:~/Documents/programming/ALU/mo
mo_sms_analytics/api/screenshots$ curl -i -X PUT http://localhost:8000/transacti
ons/5 \
-u "bodemurairi2@gmail.com:Bode200" \
-H "Content-Type: application/json" \
-d '{
  "type": "payment",
  "transaction_date": "2024-06-08 15:28",
  "amount": 2500
}'
HTTP/1.0 200 OK
Server: BaseHTTP/0.6 Python/3.12.7
Date: Wed, 01 Oct 2025 18:50:18 GMT
Content-Type: application/json

{"status": "Success", "message": "Transaction 5 replaced successfully", "record"
: {"id": 5, "transaction": "Expense", "transaction_type": "payment", "amount": 2
500.0, "currency": "RWF", "transaction_date": "2024-06-08 at 15:28"}}(base) bode
```

## 5. DELETE /transactions/{id}

This request deletes a transaction by ID. You need to be a registered user on the system and need to authenticate for this action.

### Request Example:

```
curl -i -X DELETE http://localhost:8000/transactions/5 \
-u "bodemurairi2@gmail.com:Bode200"
```

### Response Example:

HTTP/1.0 200 OK

Server: BaseHTTP/0.6 Python/3.12.7

Date: Wed, 01 Oct 2025 18:52:35 GMT

Content-Type: application/json

```
{"status": "Success", "message": "Record 5 deleted successfully"}
```



### Error Codes:

- 204 No Content
- 400 – Invalid ID format
- 404 – Transaction not found
- 401-Unauthorized

!!! [screenshot of test]

```
(base) bode-murairi@bode-murairi-IdeaPad-1-15IAU7:~/Documents/programming/ALU/momo_sms_analytics/api/screenshots$ curl -i -X DELETE http://localhost:8000/transactions/5 \
-u "bodemurairi2@gmail.com:Bode200"
HTTP/1.0 200 OK
Server: BaseHTTP/0.6 Python/3.12.7
Date: Wed, 01 Oct 2025 18:52:35 GMT
Content-Type: application/json

{"status": "Success", "message": "Record 5 deleted successfully"}(base) bode-murairi@bode-murairi-IdeaPad-1-15IAU7:~/Documents/programming/ALU/momo_sms_analytics/api/screenshots$
```

## 6. GET /transactions/summary

This request retrieves a summary of transactions (**count** and **total amount**).

### Request Example:

GET http://127.0.0.1:8000/transactions/summary

### Response Example (200 OK):

```
HTTP/1.0 200 OK
Server: BaseHTTP/0.6 Python/3.12.7
Date: Wed, 01 Oct 2025 19:00:18 GMT
Content-Type: application/json
```

```
{
  "total_transactions": 1660,
  "total_amount": 32904496.0,
  "currency": "RWF"
}
```

!!! [screenshot of test]

```
(venv) (base) bode-murairi@bode-murairi-IdeaPad-1-15IAU7:~/Documents/programming/ALU/momo_sms_analytics/api$ curl -i localhost:8000/transactions/summary
HTTP/1.0 200 OK
Server: BaseHTTP/0.6 Python/3.12.7
Date: Wed, 01 Oct 2025 19:59:41 GMT
Content-Type: application/json

{"total_transactions": 1660, "total_amount": 32904496.0, "currency": "RWF"}(venv)
(base) bode-murairi@bode-murairi-IdeaPad-1-15IAU7:~/Documents/programming/ALU/momo_sms_analytics/api$
```

## Error Code Summary

- 200 OK → Success
- 201 Created → New transaction added
- 204 No Content → Successful delete
- 400 Bad Request → Invalid input or ID
- 404 Not Found → Transaction not found
- 422 – Missing required fields
- 400-Bad requests
- 401-Unauthorized

## Issues with Basic Authentication

Basic Authentication is one of the simplest ways to secure an API or a web service: it just sends the username and password with every request, usually encoded in Base64. At first glance, it seems convenient, but it has several serious weaknesses:

1. Credentials are sent with every request

Every time a client makes a request, the username and password go over the network (even if encoded in Base64). If HTTPS isn't used, anyone sniffing the traffic can easily capture them.

2. Base64 is not encryption: Base64 encoding is not security. It's just a way to represent data in text. A hacker can decode it instantly and see your password in plain text.
3. Passwords can be stolen easily

Since the same credentials are sent repeatedly, if a hacker manages to intercept a request or gains access to logs, they can reuse your password forever unless it's changed.
4. No expiration or token control

Once a password is leaked, the attacker has indefinite access. There's no automatic expiration, revocation, or limited scope for the credentials.
5. No granular access control

With Basic Auth, you either have access or you don't. You can not easily restrict certain users to specific actions or endpoints without implementing complex additional checks.
6. Encourages weak password practices

Because passwords are used directly for authentication every time, there's more pressure to reuse passwords or avoid complex ones, increasing the risk of compromise.

## Why OAuth2 and JWT Are Better

1. OAuth2
  - OAuth2 uses access tokens instead of passwords. These tokens are short-lived and can be limited to specific permissions.
  - If a token is stolen, it only works for a short period and for specific actions, reducing the risk of a full account compromise.
  - Tokens can be revoked at any time without requiring the user to change their password.
2. JWT (JSON Web Tokens)
  - JWTs are self-contained tokens that carry user information and claims securely.
  - They can be signed and optionally encrypted, which ensures integrity and authenticity.
  - You can easily check permissions, expiration, and roles without querying the database for every request.

OAuth2 and JWT separate credentials from requests, allow better control, have expiration, and reduce the exposure of sensitive information.

## Other Alternatives

1. API Keys
  - Simple tokens that can be revoked or rotated without affecting the user's password.
  - Usually sent in headers, not credentials.
2. SAML (Security Assertion Markup Language)
  - Used for Single Sign-On (SSO) in enterprises.
  - Secure, federated authentication without sending passwords repeatedly.
3. Passwordless Authentication
  - Methods like magic links or one-time codes sent via email or SMS.
  - Reduces the risk of password leaks entirely.

## Conclusion:

Basic Auth is easy but insecure because it exposes passwords on every request, has no expiration, and cannot enforce fine-grained access control. Modern approaches like OAuth2 and JWT are far more secure because they use temporary, revocable tokens, can carry permissions, and reduce the risk of credential theft. Other alternatives like API keys, mTLS, SAML, or passwordless logins provide additional security based on the use case.

## Data Structures & Algorithms (DSA Integration)

Linear search constitutes a technique for locating a record by examining each element within a list sequentially until the specified ID is identified. While this methodology proves effective for small datasets, its efficiency diminishes as the volume of records increases, as it may necessitate evaluating every item present. The time complexity associated with linear search is characterized as  $O(n)$ , which indicates that the duration required for searching escalates linearly in correspondence with the number of records.

Dictionary lookup, on the other hand, employs a hash table to keep transactions with transaction ID keys, and any record can be accessed directly without having to scan the whole set of data. Thus, dictionary lookup is very fast with an average time complexity of  $O(1)$ . Although with a data set of 20 records, dictionary lookup will be faster than linear search, its difference grows very large with bigger data sets.

Other data structures can make search more efficient. For instance, by using a sorted list and binary search, we can access records within  $O(\log n)$  time, and tree-based data structures, i.e., binary search trees, can make insertions and lookup faster. In general, selecting a data structure that allows direct or logarithmic access to records is preferable to sequentially searching through a list, particularly with large data sets.

!!! [screenshot of test]

Path to the test script: api/services/test\_transactions.py

```
louis@louis-Latitude-7490:~/Documents/momo_sms_analytics/api/services$ source ../venv/bin/activate
(venv) louis@louis-Latitude-7490:~/Documents/momo_sms_analytics/api/services$ python test_transaction.py
Linear search for ID 1005: {'status': 'Success', 'message': 'Transaction added successfully', 'record': {'id': 1005, 'transaction': 'Income', 'transaction_type': 'receive', 'amount': 5000.0, 'currency': 'RWF', 'transaction_date': '2024-08-18 at 18:55'}}
Linear search time: 0.00000620 seconds

Dictionary lookup for ID 1005: {'status': 'Success', 'message': 'Transaction added successfully', 'record': {'id': 1005, 'transaction': 'Income', 'transaction_type': 'receive', 'amount': 5000.0, 'currency': 'RWF', 'transaction_date': '2024-08-18 at 18:55'}}
Dictionary lookup time: 0.00000525 seconds

Linear search for ID 1020: {'status': 'Success', 'message': 'Transaction added successfully', 'record': {'id': 1020, 'transaction': 'Income', 'transaction_type': 'deposit', 'amount': 14250.0, 'currency': 'RWF', 'transaction_date': '2024-09-09 at 08:50'}}
Linear search time: 0.00000930 seconds

Dictionary lookup for ID 1020: {'status': 'Success', 'message': 'Transaction added successfully', 'record': {'id': 1020, 'transaction': 'Income', 'transaction_type': 'deposit', 'amount': 14250.0, 'currency': 'RWF', 'transaction_date': '2024-09-09 at 08:50'}}
Dictionary lookup time: 0.00000334 seconds

Linear search for ID 1035: {'status': 'Success', 'message': 'Transaction added successfully', 'record': {'id': 1035, 'transaction': 'Expense', 'transaction_type': 'withdrawal', 'amount': 3900.0, 'currency': 'RWF', 'transaction_date': '2023-10-02 at 16:30'}}
Linear search time: 0.00000882 seconds
```

### Contributors:

- Louis Pascal Nsigo <[p.nsigo@alustudent.com](mailto:p.nsigo@alustudent.com)>
- Maurice Nshimyumukiza <[m.nshimyumu@alustudent.com](mailto:m.nshimyumu@alustudent.com)>
- Bode Murairi Murai <[b.murairi@alustudent.com](mailto:b.murairi@alustudent.com)>

**Team Name:** Summus Scriptors

**Date:** 2025-10-01