MomoTransaction REST API Documentation

APIs act as bridges between applications, enabling data sharing and communication, but this also makes them common targets for attacks. API security focuses on protecting data, endpoints, and user identities from unauthorized access, manipulation, or misuse. Without proper security, sensitive information such as personal data or financial transactions can be exposed. Core practices include authentication, authorization, encryption, and input validation to ensure that only legitimate users and systems interact with the API safely.

This API allows clients to manage **MoMo Quick** transactions across multiple tables (“Momo_credit”, “Momo_debit”, “Agent”, “Bank_Deposit”, “MTN_Bundle”, “Utilities”). Authentication is required for all endpoints.

Authentication

Scheme: HTTP Basic Authentication
Header format:
Authorization: Basic <base64(username:password)>
Example:
curl -u admin:secret http://localhost:8000/transactions

Endpoints:

1. Get All Transactions

Endpoint:
GET /transactions
Example:
curl -u admin:secret http://localhost:8000/transactions

json


```
{
  "momoquick": [
    {
      "Momo_credit": [
        {
            "Transaction_ID": "MC-001",
            "Name": "Jane Smith",
            "Amount": 2000,
            "Date": "2024-05-10 16:30:51"
        }
      ]
    },
    {
      "Momo_debit": []
    }
  ]
```

}

Error Codes:
401 Unauthorized – Missing or invalid authentication

2. Get Single Transaction

Endpoint:
GET /transactions/{table}/{transaction_id}
Request Example:
curl -u admin:secret http://localhost:8000/transactions/Momo_credit/MC-001

Response Example:
json
```
{
  "Transaction_ID": "MC-001",
  "Name": "Jane Smith",
  "amount": 2000,
  "date": "2024-05-10 16:30:51"
}
```

Error Codes:
401 Unauthorized – Invalid credentials
404 Not Found – Table or transaction not found

3. Add a New Transaction

Endpoint:
POST /transactions

Request Body:
json
```
{
  "table": "Momo_credit",
  "record": {
    "Name":"Jonny Bravo",
    "amount": 5000,
    "date": "2025-10-02 16:30:51"
  }
}
```

Request Example:

```
curl -u admin:secret -X POST http://localhost:8000/transactions \
-H "Content-Type: application/json" \
-d '{
```

```
  "table": "Momo_credit",
  "record": {
    "Name": "Jonny Bravo"
    "amount": 5000,
    "date": "2025-10-02 16:30:51"
  }
}'
```

Response Example:

```
{
  "message": "Transaction added",
  "Transaction_ID": "MC-002"
}
```

Error Codes:
400 Bad Request – Missing table or record
404 Not Found – Table not found

4. Update a Transaction

Endpoint:
PUT /transactions

Request Body:

```
{
  "table": "Momo_credit",
  "Transaction_ID": "MC-002",
  "updates": {
    "amount": 6000
  }
}
```

Request Example:

```
curl -u admin:secret -X PUT http://localhost:8000/transactions \
-H "Content-Type: application/json" \
-d '{
  "table": "Momo_credit",
  "Transaction_ID": "MC-002",
  "updates": { "amount": 6000 }
}'
```

Response Example:

```
{
  "message": "Transaction MC-002 updated"
}
```

Error Codes:
400 Bad Request – Missing required fields
404 Not Found – Transaction or table not found

---

5. Delete a Transaction

Endpoint:
DELETE /transactions/{table}/{transaction_id}

Request Example:
curl -u admin:secret -X DELETE http://localhost:8000/transactions/Momo_credit/MC-002

Response Example:
```
{
  "message": "Transaction MC-002 deleted"
}
```

Error Codes:
400 Bad Request – Wrong path format
404 Not Found – Transaction or table not found

Error Response Format

All errors return JSON in this format:
```
{
  "error": "Message describing the issue"
}
```

Results of DSA comparison

From the screenshot in /screenshots/dsatest.png we can see that Dictionary lookup is much faster than a linear search. The dictionary lookup took 0.000002 seconds to compute while linear took 0.000278 seconds to compute. The dictionary lookup shows a time complexity of O(1) which is vastly superior to the O(n) practiced in the linear search.
Feel free to run the script in /tests/dsa_test.py to test and see for yourself.

Basic Authentication limitation

Basic Authentication is weak because it sends the username and password with every request, making it easy for attackers to steal if not well protected. It also lacks features like token expiration and detailed access control. Better options are **JWT**, which uses tokens that can expire, and **OAuth2**, which is widely used for secure logins and access control.