MoMo SMS Transactions – Secure REST API Report

Team 11: Ingabe Mbayire Melyssa | Murenzi Bonheur | Teta Butera Nelly

Contents

1. Introduction to API Security 2
1.1 Why API Security Matters 2
1.2 How Our Authentication Works 2
2. Documentation of API Endpoints 2
GET /transactions 2
GET /transactions/{id} 3
POST /transactions 3
PUT /transactions/{id} 3
DELETE /transactions/{id} 3
3. Results of DSA Comparison 3
3.1 Overview 3
3.2 how the comparison works. 4
3.3 Screenshot Evidence 4
3.3.1 successful GET request to /transactions 4
3.3.2. Unauthorized 5
3.3.3. successful POST 6
3.3.4 successful PUT 6
3.3.5 successful DELETE 7
4 Reflection on Basic Auth Limitations 7
4.1 Credentials Sent in Every Request 7
4.2 No Encryption Without HTTPS 7
4.3 No Session Management 8
4.4 No Role-Based Access Control 8
4.5 Vulnerability to Brute Force Attacks 8
4.6 Better Alternatives 8
Conclusion 8

1. Introduction to API Security

API security refers to the set of practices and protocols used to protect Application Programming Interfaces (APIs) from malicious attacks, unauthorized access, and data breaches. As APIs serve as the communication bridge between clients and servers, securing them is critical to maintaining data integrity and user privacy.

In this project, we built a plain-Python REST API that exposes MoMo SMS transaction data. To protect this API, we implemented HTTP Basic Authentication, which requires every request to include a valid Authorization header containing a Base64-encoded username and password.

1.1 Why API Security Matters

- Data Protection: Our API handles financial transaction data (amounts, phone numbers, timestamps). Unauthorized access could expose sensitive user information.
- Access Control: Only authenticated users should be able to read, create, update, or delete transaction records.
- Integrity: Without authentication, any client could modify or delete records, compromising the reliability of the data.

1.2 How Our Authentication Works

1. Checks for the Authorization header.
2. Decodes the Base64-encoded credentials.
3. Compares the provided username and password against the server-side values API USER and API PASS, defaulting to admin / admin123).
4. If authentication fails, the server responds with 401 Unauthorized.

2. Documentation of API Endpoints

Base URL: http://127.0.0.1:8000
Authentication: Basic Auth
(default credentials: admin / admin123)

GET /transactions

- Description: Retrieve all transactions.

- Success Response: 200 OK – Returns a JSON array of transaction objects.
- Error Responses: 401 Unauthorized, 400 Bad Request (invalid limit value).

GET /transactions/{id}

- Description: Retrieve a single transaction by its ID.
- Success Response: 200 OK– Returns the transaction object.
- Error Responses: 401 Unauthorized, 404 Not Found.

POST /transactions

- Description: Create a new transaction. The request body must be a JSON object.
- Success Response: 201 Created – Returns the newly created transaction with its assigned ID.
- Error Responses: 400 Bad Request, 401 Unauthorized, 409 Conflict (duplicate ID).

PUT /transactions/{id}

- Description: Update an existing transaction by ID. The request body must be a JSON object with the updated fields.
- Success Response: 200 OK – Returns the updated transaction.
- Error Responses: 400 Bad Request, 401 Unauthorized, 404 Not Found.

DELETE /transactions/{id}

- Description: Delete a transaction by ID.
- Success Response: 200 OK – Returns {"deleted": "<id>"}.
- Error Responses: 401 Unauthorized, 404 Not Found.

3. Results of DSA Comparison

3.1 Overview

We implemented and compared two data structure approaches for searching transaction records:

1. Linear Search (List): Iterates through a Python list of transaction dictionaries one by one until the target ID is found. Time complexity: O(n).
2. Dictionary Lookup (Hash Map): Builds a dictionary (hash map) indexed by transaction ID, then retrieves the record in constant time. Time complexity: O(1) average.

3.2 how the comparison works.

The script search_compare.py loads 20 transaction records, randomly selects a target ID, and performs 10,000 repeated lookups using each method. The total time is measured using Python's

Results

Method	Time (10,000 lookups)	Complexity
Linear Search	~0.01–0.03s	O(n)
Dictionary Lookup	~0.001–0.003s	O(1)
Speedup	~5–15x faster	

The dictionary lookup consistently outperformed linear search by a significant margin. As the dataset grows, this difference becomes even more pronounced since linear search time increases proportionally with the number of records, while dictionary lookup remains constant.

3.3 Screenshot Evidence

3.3.1 successful GET request to /transactions

Shows a successful GET request to /transactions with valid Basic Auth credentials, returning 200 OK with transaction data.

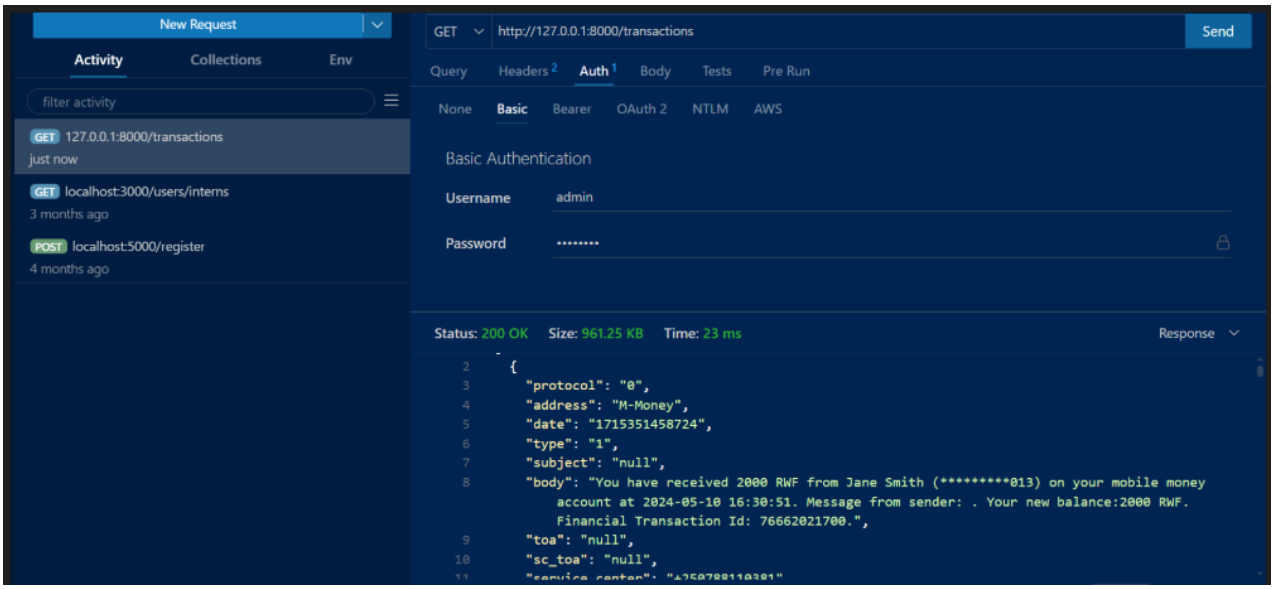

Figure 1: successful

3.3.2. Unauthorized

Shows a GET request without valid credentials, returning 401 Unauthorized, demonstrating that the API correctly rejects unauthenticated requests.

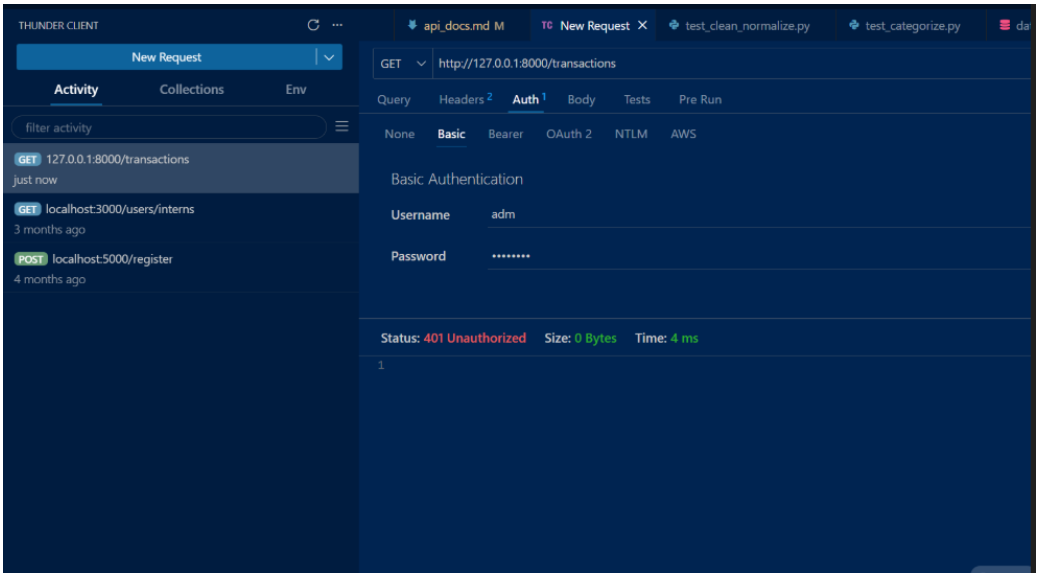

Figure 2 Unauthorized

3.3.3. successful POST

Shows a successful POST request creating a new transaction, returning 201 Created

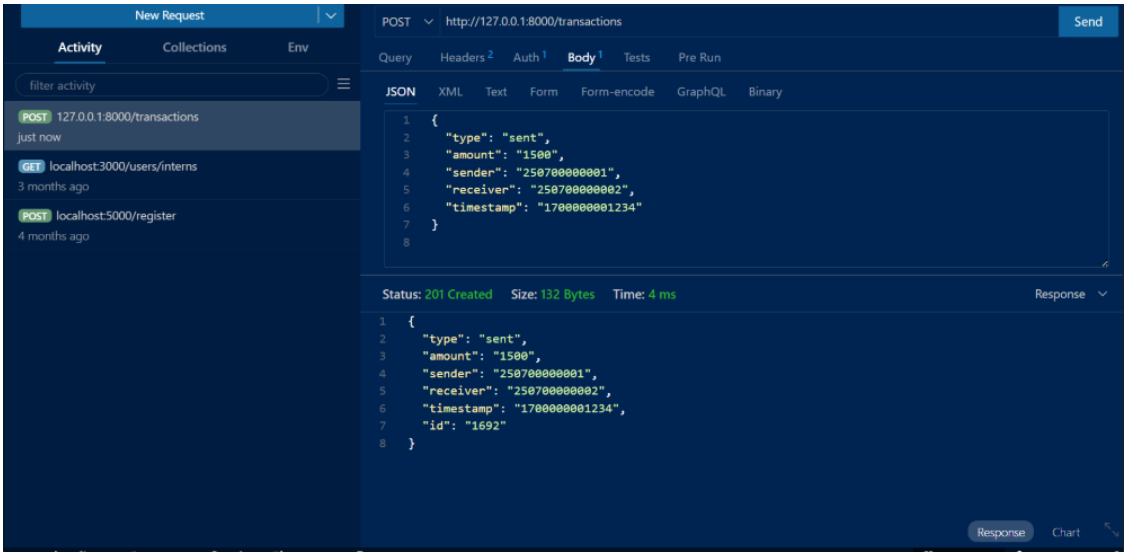

Figure 3 successful POST

3.3.4 successful PUT

Shows a successful PUT request updating an existing transaction, returning 200 OK

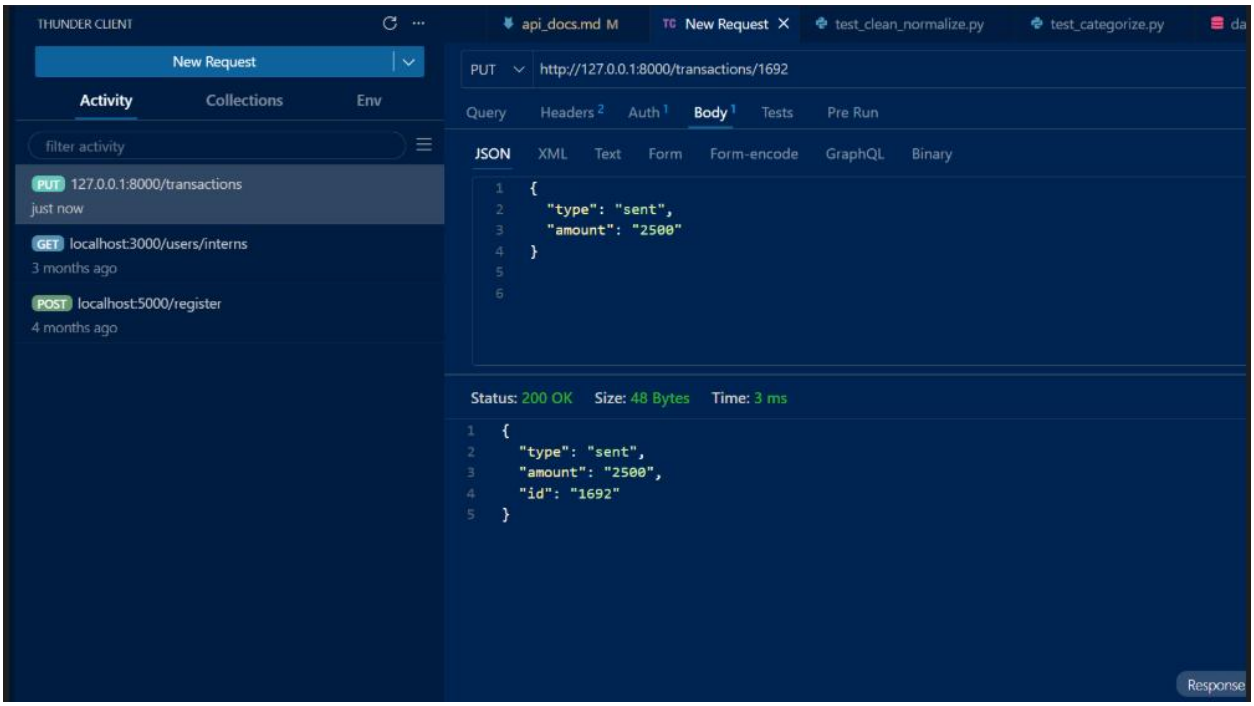

Figure 4successful PUT

3.3.5 successful DELETE

Shows a successful DELETE request removing a transaction, returning 200 OK with the deleted ID.

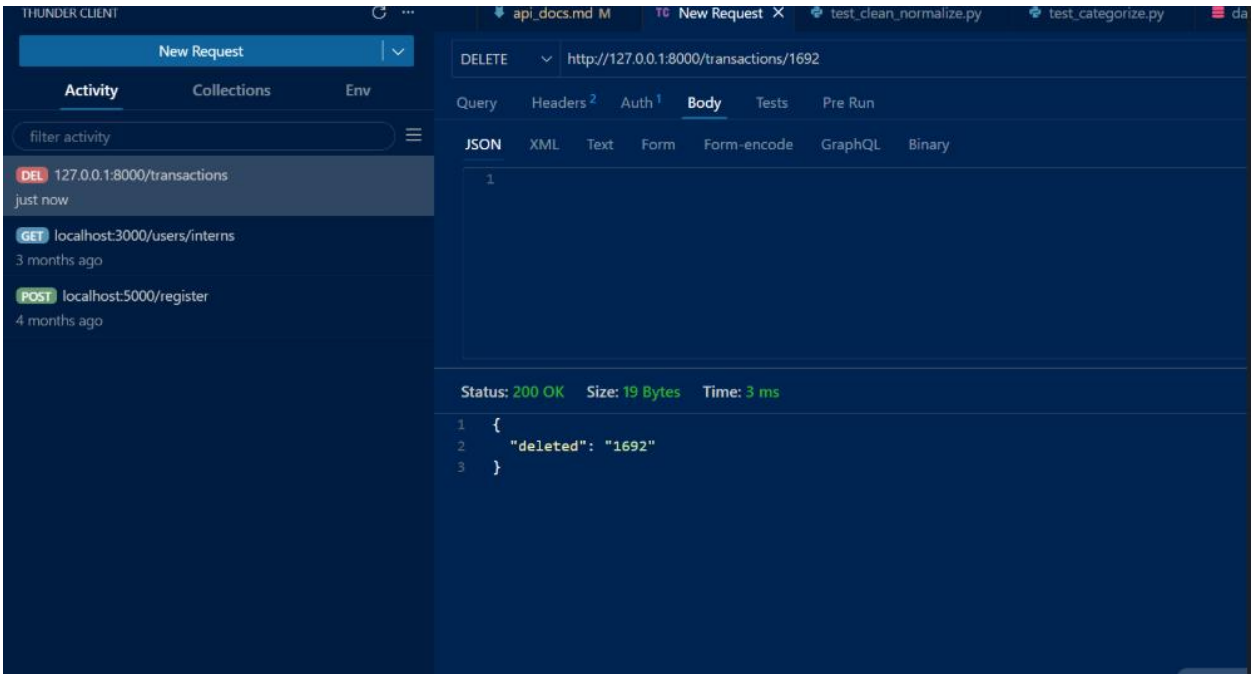

Figure 5 Successful DELETE

4 Reflection on Basic Auth Limitations

While HTTP Basic Authentication was suitable for this academic project, it has several significant limitations that make it inappropriate for production use:

4.1 Credentials Sent in Every Request

Basic Auth requires the username and password to be sent with **every single request**. The credentials are only Base64-encoded, **not encrypted**. Base64 is a reversible encoding, not a security measure — anyone who intercepts the request can decode the credentials instantly.

4.2 No Encryption Without HTTPS

If the API is served over plain HTTP (as ours is, on http://127.0.0.1:8000), the credentials travel in **plain text** over the network. Any attacker performing a man-in-the-middle (MITM) attack can read them. Basic Auth should **never** be used without HTTPS/TLS in a real deployment.

4.3 No Session Management

Basic Auth is stateless — there is no concept of login, logout, or session expiration. The server cannot revoke access for a specific user without changing the password for everyone. There are no session tokens, no refresh mechanisms, and no way to track active sessions.

4.4 No Role-Based Access Control

Our implementation uses a single set of credentials. There is no way to differentiate between users or assign different permission levels (e.g., read-only vs. admin). All authenticated users have identical access to all endpoints.

4.5 Vulnerability to Brute Force Attacks

Our API has no rate limiting or account lockout mechanism. An attacker could attempt unlimited username/password combinations without being blocked.

4.6 Better Alternatives

Alternative	Advantage
Token-Based Auth (JWT)	Credentials sent once; token used for subsequent requests; supports expiration and revocation.
OAuth 2.0	Industry standard for delegated authorization; supports scopes and third-party access.
API Keys	Simple to implement; can be rotated independently; supports per-key rate limiting.
Session Cookies	Server-managed sessions with expiration and logout capability.

Conclusion

Basic Auth served as a straightforward learning exercise for understanding authentication concepts. However, for any real-world application handling financial data like MoMo transactions, a more robust solution such as JWT or OAuth 2.0, combined with HTTPS, rate limiting, and role-based access control, would be essential.