MOMO TRANSACTIONS API
Security and Endpoint Report

INTRODUCTION TO API SECURITY

For the MoMo Transactions API, security is not optional — it is a fundamental requirement.

The system handles highly sensitive financial data, including user balances, personally identifiable information (PII), and transaction history. Without proper protection, these endpoints would be vulnerable to unauthorized access, data scraping, and malicious manipulation.

To mitigate these risks, the application implements a middleware-based security model using HTTP Basic Authentication.

This security layer acts as a gatekeeper that ensures only authorized administrators can access protected resources.

SECURITY ARCHITECTURE (MIDDLEWARE GATEKEEPER)

Authentication is enforced using a require_auth decorator that intercepts requests before they reach the core business logic.

How it works:

Interception
Every request sent to protected endpoints is intercepted by the middleware.

Validation
The middleware extracts the Authorization header, decodes the Base64 credentials, and validates them against secure environment variables stored in a .env file. No credentials are hardcoded.

Enforcement
Requests that lack valid credentials are immediately rejected with a 401 Unauthorized response. Only authenticated users are allowed to proceed.

API ENDPOINT DOCUMENTATION

Tests for these endpoints are included in the repository as screenshots.

RETRIEVAL ENDPOINTS (READ)

GET /list-transactions

Description
Returns a complete list of all transactions in the system.

Database joins include:
• transactions
• transaction_categories
• users (joined twice for sender and receiver)

Success Response
200 OK

Key fields returned:
• sender – name of the sender
• receiver – name of the receiver
• transaction_category – descriptive category name (e.g., Airtime)
• raw_message – original SMS content for verification

GET /transaction-details/<transaction_id>

Description
Retrieves every detail of a specific transaction using its unique ID.

Responses
200 OK – transaction found
404 Not Found – transaction does not exist

CREATION ENDPOINT (CREATE)

POST /create-transaction

Description
Processes and records a new transaction. This endpoint does not create new users. Instead, it maps the transaction to existing sender, receiver, and category records.

Request Body (JSON example)

{
"external_ref_id": "16803066185",
"transaction_date": "2024-11-12 23:47:47",
"amount": 5000.00,
"category_id": 4,
"sender_id": 1,
"receiver_id": 12,
"status": "FAILED",
"raw_message": "...SMS Body..."
}

Database Logic

Insert into transactions table

Create two entries in user_transactions
• one for the sender
• one for the receiver

Success Response
201 Created

UPDATE ENDPOINT (UPDATE)

PUT /update-transaction/<tx_id>

Description
Updates the audit trail and system logs for a specific transaction. This is typically used to refine failure reasons or adjust log severity.

Request Body (JSON example)

{
"log_level": "ERROR",
"status": "FAILED",
"message": "Failed due to an airtime loan that isn't paid yet"

}

Database Logic
Updates the status, log_level, and message fields in the system_logs table.

Success Response
200 OK

DELETION ENDPOINT (DELETE)

DELETE /delete-transaction/<tx_id>

Description
Completely removes a transaction and all related data.

Referential Integrity Strategy
To avoid foreign key constraint errors, records are deleted in the following order:

system_logs

user_transactions

transactions

Responses
200 OK – successfully deleted
404 Not Found – transaction not found

BASIC AUTHENTICATION LIMITATIONS

While HTTP Basic Authentication prevents casual unauthorized access, it should be considered only a minimum viable security measure. For a financial system, it presents significant risks and is not suitable for production environments.

Key limitations include:

Credential Interception Risk

Basic Auth does not encrypt credentials. They are only Base64 encoded. Without HTTPS, they can be easily decoded by attackers monitoring network traffic.

High Attack Surface
Passwords must be transmitted with every request, increasing the likelihood of compromise.

Lack of Granular Control
Basic Auth only supports simple access control. It does not allow role-based permissions such as read-only or restricted access users.

Revocation Difficulties
There is no easy way to revoke a single session. If credentials are compromised, the password must be changed for the entire system.

STRONGER ALTERNATIVES

Modern enterprise systems typically use OAuth 2.0 or JSON Web Tokens (JWT).

OAuth 2.0
Ideal for delegated access and third-party integrations.

JSON Web Tokens (JWT)
A stateless, token-based authentication mechanism that provides improved security and scalability.

RECOMMENDATION: UPGRADE TO JWT

JWT is strongly recommended for the next phase of this API.

Benefits include:

Stateless Scalability
The server validates tokens cryptographically without database lookups, improving performance and reducing load.

Minimized Credential Exposure
Users send their password only once during login. They then receive a temporary token that expires automatically.

Embedded Permissions
Tokens can carry role and permission information, enabling fine-grained access control without extra queries.

Example claims:
role: admin
permissions: read, write

Data Structures & Algorithms (DSA Integration)
Overview:
The purpose of this analysis was to compare the efficiency of two data handling methods: Linear Search and Dictionary Search (Hash-Map Indexing). This was done to ensure that the system can handle large volumes of transaction data without performance degradation.

We used a dataset of over 1600 parsed transactions, and to get good results, we conducted a test by searching for the same record 100,000 times. To simulate the worst-case scenario, where algorithms are most heavily tested, we searched for the last record in the dataset.

Comparative results;

Method	Complexity	Total Time (100000 runs)	Performance
Linear Search	O(n)	~10.69s	Inefficient for Large data
Dictionary Search	O(1)	~0.02s	More optimized for large datasets

Key Findings

- Linear Search (O(n)): The execution time of approximately 10 seconds shows the inefficiency of iterating through lists for every request. As a dataset grows, the search time will increase linearly, which has the potential to cause API timeouts.
- Dictionary Search (O(n)): By pre-indexing IDs into a Hash Map, we were able to retrieve information almost immediately. The search time remains constant regardless of whether the dataset contains 20 records or 10000 records.

Conclusion
Based on these results, we implemented the dictionary-based lookup for all the transaction retrieval endpoints. This ensures that the application

provides a seamless user experience by maintaining high speed as the transactions grow over time.