Introduction to API Security

API security protects Application Programming Interfaces from unauthorized access and malicious attacks. For our MoMo SMS Analytics system, securing the API is critical because it exposes sensitive financial transaction data including user information, transaction amounts, and payment history. Key Security Principles

- **Authentication**: Verifying user identity before granting access. Our API uses Basic Authentication requiring valid credentials with each request.
- **Authorization**: Controlling what authenticated users can do. Currently all authenticated users have full CRUD access to transactions.
- **Confidentiality**: Protecting data from unauthorized disclosure. Best achieved through HTTPS encryption (not implemented in our development version).
- **Data Integrity**: Ensuring data isn't tampered with. Implemented through input validation and proper error handling.

Our Security Implementation

Security Measure	Status
Basic Authentication with credential verification	Implemented
401 Unauthorized responses for invalid credentials	Implemented
Input validation and JSON parsing	Implemented
Proper HTTP status codes	Implemented

API Endpoints

1. GET /transactions - List all transactions

- Description: Retrieves a list of all SMS transactions.
- Request Example:

```
GET /transaction HTTP/1.1
Host: localhost:8000
Authorization: Basic YWRtaW46bW9tb3Ntc2FkanFz
```

- Response Example (200 OK):

```
[
    {
        "id": 1,
        "sender": "Ama Mensah",
        "receiver": "Berima Kofi",
        "amount": 100,
        "type": "credit",
        "timestamp": "2026-02-01T10:00:00"
    }
]
```

- Error Codes:
    - 401 Unauthorized - Missing or wrong credentials

2. GET /transactions/{id} - Retrieve a single transaction

- Description: Get details of one transaction by ID
- Request Example:

```
GET /transactions/1 HTTP/1.1
Host: localhost:8000
Authorization: Basic YWRtaW46bW9tb3Ntc2FkanFz
```

- Response Example (200 OK):

```
{
    "id": 1,
    "sender": "Ama Mensah",
    "receiver": "Berima Kofi",
    "amount": 100,
    "type": "credit",
    "timestamp": "2026-02-01T10:00:00"
}
```

- Error Codes:

    - 401 Unauthorized - Invalid credentials
    - 404 Not Found - Transaction ID does not exist
    - 400 Bad Request - Invalid ID format

3. POST /transactions - Add a new transaction

- Description: Create a SMS transaction.
- Request Example:

```
POST /transactions HTTP/1.1

Host: Localhost:8000

Authorization: Basic YWRtaW46bW9tb3Ntc2FkanFz

Content-Type: application/json



{

    "sender": "David",

    "receiver": "Nicole",

    "amount": 50,

    "type": "debit",

    "timestamp": "2026-02-01T11:00:00"

}
```

- Response Example (201 Created):

```
{

    "id": 2,

    "sender": "David",

    "receiver": "Nicole",

    "amount": 50,
```

```
    "type": "debit",
    "timestamp": "2026-02-01T11:00:00"
}
```

- Error Codes:
    - 401 Unauthorized - Invalid Credentials
    - 400 Bad Request - Missing required fields or invald JSON

4. PUT /transactions/{id} - Update an existing transaction

- Description: Update fields of a transactions by ID.
- Request Example:

```
PUT /transactions/1 HTTP/1.1
Host: localhost:8000
Authorization: Basic YWRtaW46bW9tb3Ntc2FkanFz
Content-Type: application/json

{
    "amount": 120
}
```

- Response Example (200 OK):

```
{
    "id": 1,
    "sender": "Ama Mensah",
    "receiver": "Berima Kofi",
    "amount": 120,
    "type": "credit",
```

```
    "timestamp": "2026-02-01T10:00:00"
}
```

- Error Codes:
    - 401 Unauthorized - Invalid credentials
    - 404 Not Found - Transaction ID does not exist
    - 400 Bad Request - Invalid JSON

5. DELETE /transactions/{id} - Delete a transaction

- Description: Remove a transaction by ID.
- Request Example:

```
DELETE /transactions/1 HTTP/1.1
Host: localhost:8000
Authorization: Basic YWRtaW46bW9tb3Ntc2FkanFz
Response Example (200 OK):
{
    "message": "Transaction deleted"
}
```

- Error Codes:
    - 401 Unauthorized - Invalid credentials
    - 404 Not Found - Transaction ID does not exist

DSA Comparison + Implementation Details

We implemented and compared two search algorithms for retrieving transactions by ID:

- Linear Search (O(n)): Sequentially scans through the list comparing each transaction ID until found.

- Dictionary Lookup (O(1)): Stores transactions in a hash table (Python dictionary) where transaction ID is the key, enabling direct access

Benchmark Results

Test Parameters:

- Dataset: 1,691 transactions from modified_sms_v2.xml
- Number of searches: 100 random lookups
- Environment: Python 3.x

Metric	Linear Search	Dictionary Lookup	Improvement
Total time	6.35 ms	0.041 ms	154.9x faster
Average per search	0.0635 ms	0.000409 ms	155.25x faster
Average comparisons	~845	1	845x fewer
Time complexity	O(n)	O(1)	Constant time

Linear Search Process:

```
Search for ID 500:

→ Check transaction[0]

→ Check transaction[1]

→ ...

→ Check transaction[499] (Done)
```

Result: 500 comparisons

Dictionary Lookup Process:

```
Search for ID 500:

→ Compute hash(500)

→ Access dict[500] directly
```

Result: 1 operation

Basic Auth Limitations

Authorization header:

```
"admin:momosmsanalysis" → Base64 encode →

"YWRtaW46bW9tb3Ntc2FuYWx5c2lz"

Header: "Authorization: Basic YWRtaW46bW9tb3Ntc2FuYWx5c2lz"
```

Security Flaws

1. Not encrypted:

Base64 is encoding, not encryption. Anyone can decode it:

```
base64.b64decode("YWRtaW46bW9tb3Ntc2FuYWx5c2lz")

# Returns: "admin:momosmsanalysis"
```

Risk: Credentials transmitted in plain text. It's vulnerable to Man-in-the-Middle attacks, especially on public WiFi.

2. Credentials sent with every request:

Unlike token-based systems, credentials must be included in every single request, increasing exposure risk with each API call.

3. Poor Scalability:

    - Single shared password for all users in our implementation
    - Cannot distinguish between users
    - No role-based access control
    - Difficult to audit who performed which actions