Building and Securing a REST API for MoMo SMS Transactions

1. Introduction to API Security

An Application Programming Interface (API) acts as a bridge that allows different software systems to communicate with each other. Because APIs often expose sensitive data and critical operations, security is a core requirement in API design.

In this project, a REST API was developed to expose Mobile Money (MoMo) SMS transaction data parsed from an XML dataset. The API supports CRUD operations (Create, Read, Update, Delete) on transaction records. Since this data represents financial transactions, unrestricted access would pose serious security risks, including data leakage and unauthorized manipulation.

To address this, Basic Authentication was implemented to ensure that only authorized users can access the API endpoints. Authentication checks are performed before any request is processed, and unauthorized requests are rejected with an appropriate HTTP status code.

2. API Endpoint Documentation

The API was implemented using Python's built-in http.server module and follows REST principles.

Base URL

http://localhost:9000

All endpoints require **Basic Authentication**.

2.1 GET /transactions

Description:
Returns a list of all SMS transaction records.

Authentication: Required (Basic Auth)

Request Example:

curl -u group7:EWD http://localhost:9000/transactions

Successful Response (200 OK):

[

```
  {
    "id": 1,
    "transaction_type": "received",
    "sender": "Jane Smith",
    "receiver": null,
    "amount": 2000,
    "readable_date": "10 May 2024 4:30:58 PM"
  }
]
```


Error Responses:

- 401 Unauthorized – Invalid or missing credentials
- 404 Not Found – Invalid endpoint

2.2 GET /transactions/{id}

Description:
Returns a single transaction based on its ID.

Authentication: Required

Request Example:

curl -u group7:EWD http://localhost:9000/transactions/2

Successful Response (200 OK):


```
{
  "id": 2,
  "transaction_type": "payment",
  "sender": "Alice",
  "receiver": "Bob",
  "amount": 1000,
  "readable_date": "10 May 2024 4:31:46 PM"
}
```


Error Responses:

- 400 Bad Request – Invalid transaction ID

- 401 Unauthorized – Invalid credentials
- 404 Not Found – Transaction does not exist

2.3 POST /transactions

Description:
Adds a new transaction to the dataset.
The API validates input and **rejects unwanted fields**.

Authentication: Required

Request Example:

```
curl -u group7:EWD -X POST http://localhost:9000/transactions \
-H "Content-Type: application/json" \
-d '{
  "transaction_type": "payment",
  "sender": "Alice",
  "receiver": "Bob",
  "amount": 1500,
  "readable_date": "02 Feb 2026 3:00 PM"
}'
```

Successful Response (201 Created):

```
{
  "id": 1694,
  "transaction_type": "payment",
  "sender": "Alice",
  "receiver": "Bob",
  "amount": 1500,
  "readable_date": "02 Feb 2026 3:00 PM"
}
```

Error Responses:

- 400 Bad Request – Missing or invalid fields
- 401 Unauthorized – Invalid credentials

2.4 PUT /transactions/{id}

Description:
Updates an existing transaction.

Authentication: Required

Request Example:

```
curl -u group7:EWD -X PUT http://localhost:9000/transactions/2 \
-H "Content-Type: application/json" \
-d '{"amount": 2000}'
```

Successful Response (200 OK):


```
{
  "id": 2,
  "transaction_type": "payment",
  "sender": "Alice",
  "receiver": "Bob",
  "amount": 2000,
  "readable_date": "10 May 2024 4:31:46 PM"
}
```


2.5 DELETE /transactions/{id}

Description:
Deletes a transaction by ID.

Authentication: Required

Request Example:

```
curl -u group7:EWD -X DELETE http://localhost:9000/transactions/2
```

Successful Response:

- 204 No Content

Error Responses:

- 401 Unauthorized
- 404 Not Found

3. Data Structures & Algorithms (DSA) Comparison

Two approaches were implemented to retrieve transactions by ID:

3.1 Linear Search

- Transactions stored in a **list**
- Each search scans records one by one
- Time complexity: **O(n)**

Example:

```
next((t for t in transactions if t["id"] == target_id), None)
```

3.2 Dictionary Lookup

- Transactions stored in a **dictionary** using id as the key
- Direct access to records
- Time complexity: **O(1)**

Example:

```
transactions_dict[target_id]
```

Results (20+ Records)

Method	Search Time	Efficiency
Linear Search	Increases with dataset size	Slower
Dictionary Lookup	Constant time	Faster

Conclusion:
Dictionary lookup is significantly faster because it avoids scanning the entire dataset. The hash-based structure allows direct access to records using unique keys.

Suggested Improvement:
Using a database index (e.g., B-tree index in SQL) would further improve performance and scalability for large datasets.

4. Reflection on Basic Authentication Limitations

While Basic Authentication was suitable for this academic project due to its simplicity, it has several weaknesses:

1. Credentials are sent with **every request**
2. Base64 encoding is **not encryption**
3. No session expiration or token revocation
4. Vulnerable without HTTPS

Stronger Alternatives

- **JWT (JSON Web Tokens):**
  Stateless, time-limited tokens with better scalability
- **OAuth2:**
  Secure delegated access without exposing passwords
- **API Keys with HTTPS:**
  Easier to rotate and revoke than passwords

In a production system handling financial data, Basic Authentication should only be used in combination with HTTPS or replaced entirely by token-based authentication.