1. Title & Introduction

The goal of this project was to create a secure REST API that provides mobile money (MoMo) SMS transaction data. The dataset, provided in XML format (modified_sms_v2.xml), includes transaction records like transfers, payments, deposits, and withdrawals.

The assignment required:

- Parsing the XML into JSON,

- Implementing CRUD endpoints in a plain Python HTTP server,

- Securing the endpoints with Basic Authentication,

- Documenting the API for other developers,

- Demonstrating the efficiency of different data structures and algorithms for searching transactions,

- Validating the system with testing using curl/Postman.

This report details the implementation, security considerations, and performance comparison results.

2. Data Parsing

The first step involved converting the XML SMS records into structured JSON.

Tool: Python’s xml.etree.ElementTree module.

Output: A JSON file, transactions.json, that contains a list of dictionaries (each representing a transaction).

Each record includes:

- id (unique integer),

- address (SMS sender),

- body (SMS text),

- transaction object (extracted fields: type, amount, currency, name, phone).

Sample Parsed JSON (first 2 objects):


```
[
 {
   "id": 1,
   "address": "M-Money",
   "body": "You have received 2000 RWF from John Doe (2507xxxxxxx)...",
   "transaction": {
     "type": "receive",
     "amount": 2000,
     "currency": "RWF",
     "name": "John Doe",
     "phone": "(2507xxxxxxx)"
   }
 },
 {
   "id": 2,
   "address": "M-Money",
   "body": "Your payment of 500 RWF to Jane Smith (2507yyyyyyy)...",
   "transaction": {
     "type": "payment",
     "amount": 500,
     "currency": "RWF",
```

```
    "name": "Jane Smith",
    "phone": "(2507yyyyyyy)"
   }
  }
]
```


3. API Implementation

A REST API was built using Python's built-in http.server.

Endpoints:

- GET /transactions → List all transactions
- GET /transactions/{id} → View one transaction
- POST /transactions → Add a new transaction
- PUT /transactions/{id} → Update a transaction
- DELETE /transactions/{id} → Remove a transaction

Features:

- In-memory storage using both a list and dictionary (for data structure and algorithm comparison).
- Changes are saved back into transactions.json.
- Basic Authentication is required for all endpoints.

Example Code (Authorization check):

```
def auth_required(self):
    auth = self.headers.get('Authorization')
    if not check_auth_header(auth):
```

```
        self.send_response(401)
        self.send_header('WWW-Authenticate', 'Basic realm="MoMoAPI"')
        self.end_headers()
        return True
    return False
```

4. Authentication & Security

The authentication method used was HTTP Basic Authentication. The client sends username:password encoded in Base64 with each request.

Example:

Authorization: Basic c3R1ZGVudDpwYXNzd29yZDEyMw==

Limitations of Basic Auth:

- Base64 is not encryption (credentials can be easily decoded).
- Credentials are sent with each request → higher risk if intercepted.
- There is no built-in session expiry or token revocation.
- It is only secure if combined with HTTPS (TLS).

Stronger Alternatives:

- JWT (JSON Web Tokens): Short-lived signed tokens for stateless authentication.
- OAuth2: Industry-standard for delegated access; supports scopes and refresh tokens.
- Mutual TLS: Both server and client authenticate using certificates.
- HMAC-signed API keys: Secure server-to-server communication.

5. Data Structures & Algorithms (DSA Comparison)

To find transactions by ID, two approaches were implemented:

Method	Complexity	Description
Linear Search	O(n)	Iterate through the list until the matching ID is found.
Dictionary Lookup	O(1) avg	Direct access using the ID as a dictionary key.

Experimental Results (20 lookups):

- Linear search total time: 0.004123 s
- Dict lookup total time: 0.000056 s
- Per-op approx: linear 0.206 ms, dict 0.002 ms

Reflection:

Dictionary lookups are faster because Python dictionaries are hash tables.

For larger datasets, linear search becomes inefficient.

Alternative structures:

- Binary search trees (BST) / B-Trees → good for sorted ranges (e.g., by date).
- Inverted index → efficient text search in transaction bodies.
- Database with indexes (SQLite/Postgres) for large-scale data.

6. Testing & Validation

Testing was done using both curl and Postman.

Screenshots included:

- Successful GET with authentication

- Unauthorized request (wrong/missing credentials)

- Successful POST (new record created, new ID returned)

- Successful PUT (record updated)

- Successful DELETE (record deleted)

- DSA comparison script output

7. Conclusion & Future Work

The project successfully delivered a secure REST API for MoMo transactions:

- XML was converted to structured JSON.

- CRUD endpoints were created with Basic Authentication.

- The DSA comparison showed that dictionary lookup is significantly faster than linear search.

- Documentation was provided for all endpoints and test cases.

Future Improvements:

- Replace Basic Auth with JWT or OAuth2.

- Store transactions in a proper database with indexes.

- Add input validation, error handling, and logging.

- Deploy with HTTPS and TLS certificates.

- Implement rate limiting and request quotas for better security.