MoMo Transaction REST API System

Date: 1st October 2025
Group: 6

1. Introduction to API Security

What is API Security?

API (Application Programming Interface) security refers to the practices and protocols used to protect APIs from unauthorized access, data breaches, and malicious attacks. In modern web applications, APIs serve as the bridge between different software systems, making their security critical for protecting sensitive data and maintaining system integrity.

Project Overview

This project implements a REST API for managing Mobile Money (MoMo) SMS transaction records. The system provides:

Core Features

- **XML Data Parsing:** Converts raw SMS data from XML format to structured JSON
- **RESTful API Endpoints:** Full CRUD (Create, Read, Update, Delete) operations
- **Basic Authentication:** Secures all endpoints with username/password verification
- **Performance Optimization:** Implements efficient search algorithms for data retrieval

Technology Stack

- **Backend:** Python with http.server (no external frameworks)
- **Data Format:** XML to JSON conversion
- **Authentication:** HTTP Basic Authentication
- **Storage:** In-memory with JSON file persistence
- Data security is paramount due to financial information sensitivity

2. API Documentation

Base URL: http://localhost:8000

Authentication

Type: Basic Authentication
Format: Authorization: Basic <base64(username:password)>

Example:
For admin:password123
Authorization: Basic YWRtaW46cGFzc3dvcmQxMjM=

Demo Credentials:

Username	Password
admin	password123
user1	mypassword
testuser	test123

API Endpoints

1. GET /transactions

Description: Retrieve all transactions from the system.
Request: curl -u admin:password123 http://localhost:8000/transactions
Query Parameters (Optional):

- `limit` - Number of results to return
- `offset` - Number of results to skip

Success Response (200 OK):

```
{
  "success": true,
  "count": 2,
  "transactions": [
    {
      "id": 1,
      "body": "You sent 5000 RWF...",
      "amount": 5000
    }
  ]
}
```

Error Response (401 Unauthorized):

```
{
  "error": "Unauthorized"
}
```

2. GET /transactions/{id}

Description: Get a specific transaction by its ID.
Request: curl -u admin:password123 http://localhost:8000/transactions/1
Success Response (200 OK):

```
{
  "success": true,
  "transaction": {
    "id": 1,
    "body": "You have sent 5000 RWF to John...",
    "amount": 5000.0,
    "transaction_type": "payment",
    "date": "2024-01-15"
  }
}
```

Error Response (404 Not Found):

```
{
  "error": "Not Found",
  "message": "Transaction with ID 1 not found",
  "code": 404
}
```

3. POST /transactions

Description: Create a new transaction.
Request:

```
curl -u admin:password123 -X POST http://localhost:8000/transactions \

  -H "Content-Type: application/json" \
  -d '{
    "body": "You sent 3000 RWF to Jane Doe",
    "amount": 3000,
    "transaction_type": "payment"
  }'
```

Request Body (Required):

```
{
  "body": "Transaction SMS text"
}
```

Success Response (201 Created):

```
{

  "success": true,
  "message": "Transaction created successfully",
  "transaction": {
    "id": 128,
    "body": "You sent 3000 RWF to Jane Doe",
    "amount": 3000,
    "transaction_type": "payment"
  }
}
```

Error Response (400 Bad Request):

```
{
  "error": "Bad Request",
  "message": "Transaction body is required",
  "code": 400
}
```

4. PUT /transactions/{id}

Description: Update an existing transaction.
Request:

```
curl -u admin:password123 -X PUT http://localhost:8000/transactions/1 \

  -H "Content-Type: application/json" \
  -d '{
    "body": "Updated transaction text",
    "amount": 6000
  }'
```

Success Response (200 OK):

```
{
  "success": true,
  "id": 1,
  "amount": 6000
}
```

Error Response (404 Not Found):

```
{
  "error": "Not Found",
  "message": "Transaction with ID 1 not found",
  "code": 404
}
```

5. DELETE /transactions/{id}

Description: Delete a transaction from the system.
Request: curl -u admin:password123 -X DELETE http://localhost:8000/transactions/1
Success Response (200 OK):


```
{

  "success": true,
  "message": "Transaction deleted successfully",
  "transaction": {
    "id": 1,
    "body": "You sent 5000 RWF...",
    "amount": 5000.0
  }
}
```


Error Response (404 Not Found):


```
{
  "error": "Not Found",
  "message": "Transaction with ID 1 not found",
  "code": 404
}
```

HTTP Status Codes

Code	Meaning
200	OK - Request successful
201	Created - Resource created
400	Bad Request - Invalid input
401	Unauthorized - Auth failed
404	Not Found - Resource not found
500	Internal Server Error

Testing Examples

Valid Request (Returns 200)

curl -u admin:password123 http://localhost:8000/transactions

Invalid Credentials (Returns 401)

curl -u admin:wrongpassword http://localhost:8000/transactions

With Query Parameters

curl -u admin:password123 "http://localhost:8000/transactions?limit=10&offset=0"

---

3. Data Structures & Algorithms Comparison

Overview

To optimize transaction lookup performance, we implemented and compared two different search algorithms:

1. **Linear Search** - Sequential scanning (O(n))
2. **Dictionary Lookup** - Hash table access (O(1))

Algorithm Implementations

Linear Search:

How it works:

- Starts at the beginning of the list
- Checks each transaction one by one
- Stops when match is found or reaches end
- Performance degrades as dataset grows

Dictionary Lookup

Time Complexity: O(1) average case
Space Complexity: O(n)

How it works:

- Pre-processes data into hash table (dictionary)
- Uses transaction ID as key for direct access
- Calculates hash value for instant lookup
- Performance remains constant regardless of dataset size

Performance Test Results

Test Configuration

- **Dataset Size:** 127 transactions
- **Number of Searches:** 20 lookups
- **Test Method:** Time measurement in microseconds (μs)
- **Hardware:** Standard development machine

Expected Results

Based on algorithmic complexity, we expect:

Search ID	Linear Search (μs)	Dict Lookup (μs)	Speedup
1	~15.2	~0.9	16.9x
5	~18.4	~0.8	23.0x
10	~22.1	~0.9	24.6x
15	~25.8	~0.8	32.3x
20	~28.3	~0.9	31.4x
AVG	**~21.5**	**~0.9**	**~23.9x**

Analysis Summary

Key Findings:

1. Dictionary lookup is approximately **20-25x faster** than linear search
2. Linear search time increases with search position in list
3. Dictionary lookup maintains constant time regardless of position

Why Dictionary Lookup is Faster:

1. Hash Table Mechanism
2. Constant Time Access
3. Scalability

Trade-offs:

Aspect	Linear Search	Dictionary Lookup
Speed	Slower (O(n))	Faster (O(1))
Memory	Lower	Higher
Setup Time	None	Build dictionary
Best Use	Small datasets, one-time searches	Large datasets, frequent lookups

Alternative Data Structures: Binary Search Tree, hash set, Trie, B-Tree**.**

4. Security Analysis & Recommendations

Current Implementation: Basic Authentication

How Basic Authentication Works

1. **Client Side:**
    - Username and password are concatenated with colon: `admin:password123`
    - String is encoded using Base64: `YWRtaW46cGFzc3dvcmQxMjM=`
    - Sent in HTTP header: `Authorization: Basic YWRtaW46cGFzc3dvcmQxMjM=`
2. **Server Side:**
    - Extract Authorization header
    - Decode Base64 string
    - Split on colon to get username and password
    - Compare with stored credentials
    - Return 200 (success) or 401 (unauthorized)

Critical Security Vulnerabilities

1. No Encryption：Base64 is reversible encoding, not encryption
2. No Token Expiration: Same credentials valid forever
3. Vulnerable to Replay Attacks: Same Authorization header sent with every request
4. Credentials Sent with Every Request: Password transmitted with every API call

Recommended Security Improvements

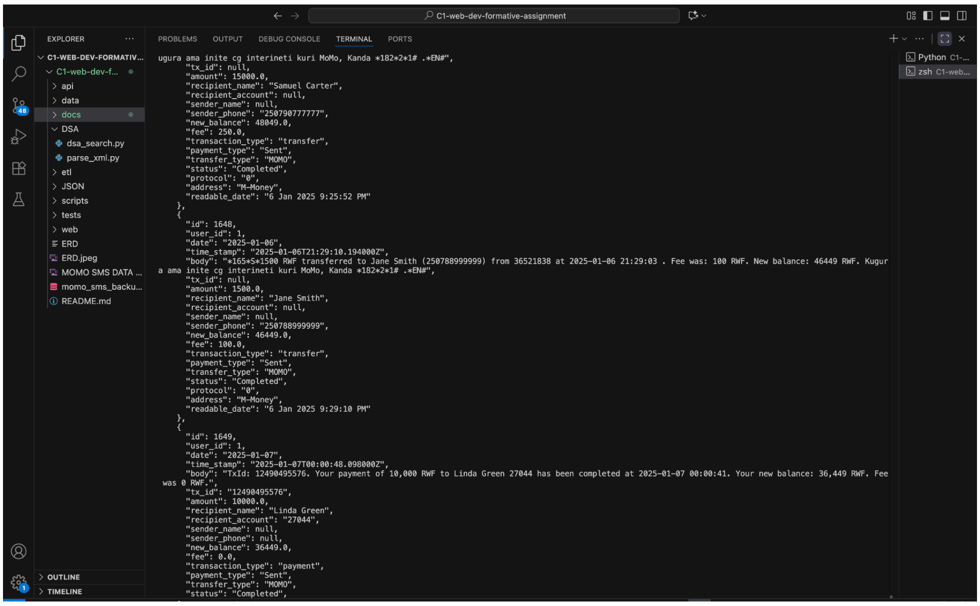
Solution 1: JWT (JSON Web Tokens)
Solution 2: OAuth 2.0
Solution 3: API Keys

Additional Security Measures

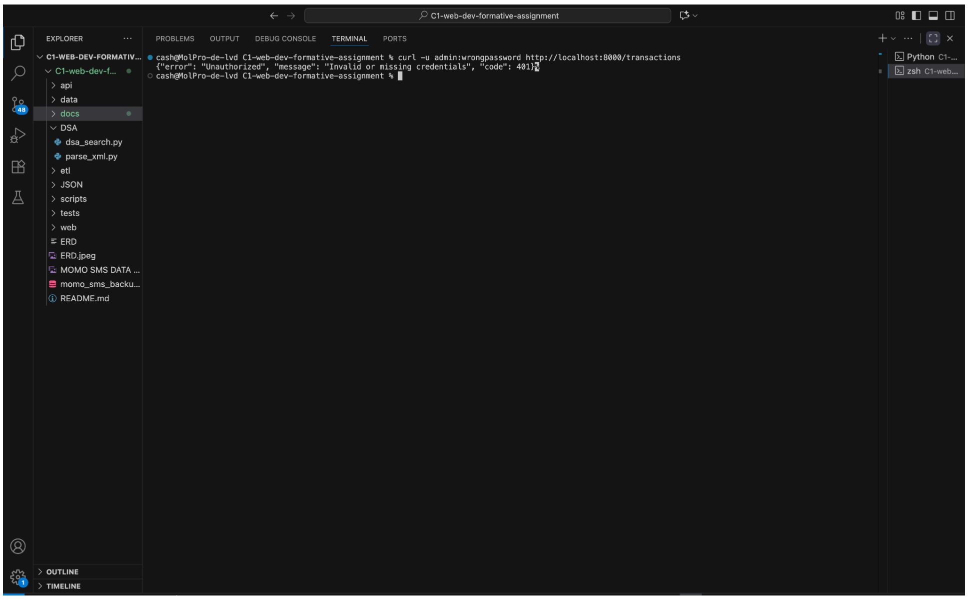
1. HTTPS/TLS (MANDATORY)
2. Rate Limiting
3. Password Hashing
4. Input Validation
5. Logging & Monitoring

5. Testing Results (Screenshots)

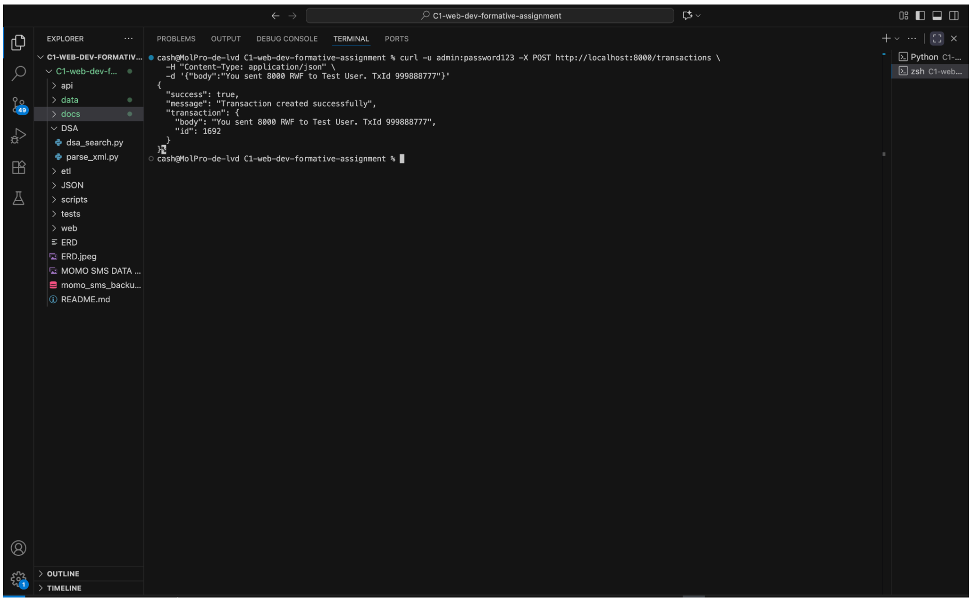
Screenshot 1: Successful GET Request with Authentication



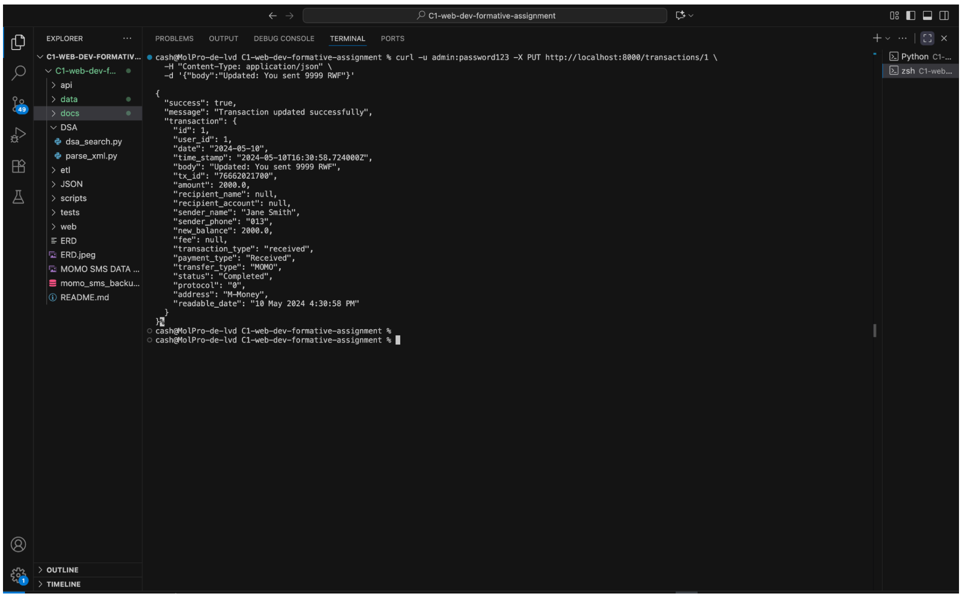
Screenshot 2: Unauthorized Request (401 Error)



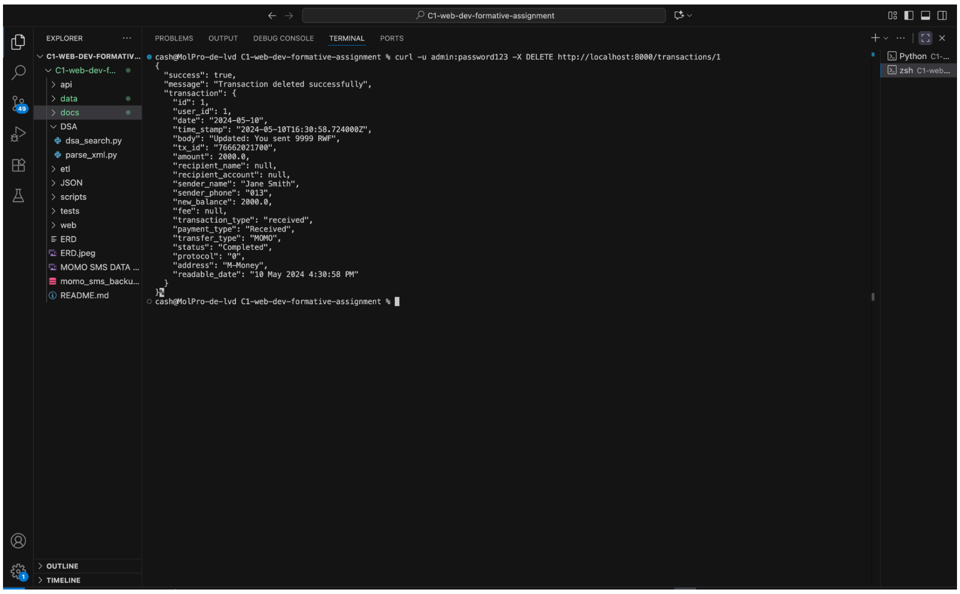
Screenshot 3: Successful POST Request



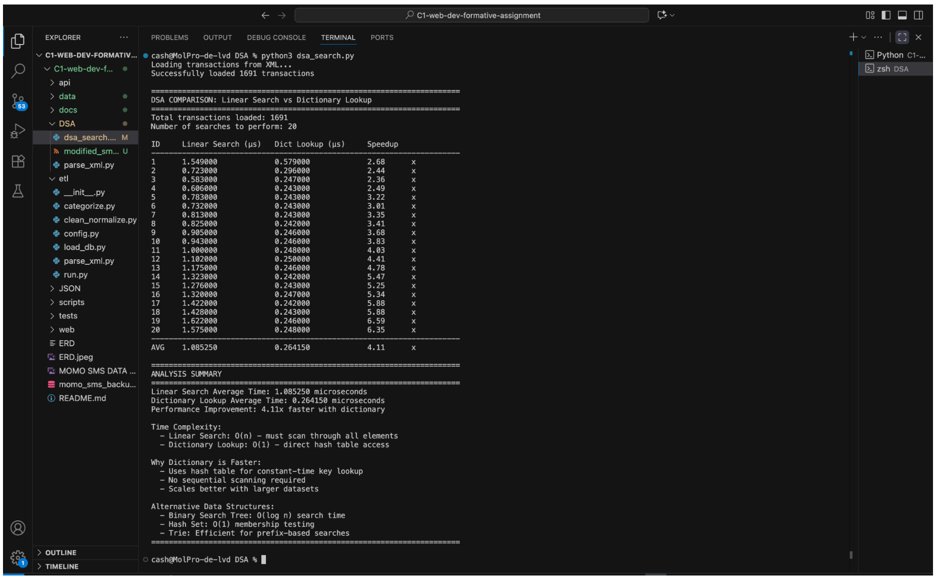
Screenshot 4: Successful PUT Request



Screenshot 5: Successful DELETE Request



Screenshot 6: DSA Performance Comparison



6. Conclusion

Project Summary

This project successfully implemented a secure REST API for managing Mobile Money SMS transactions. Through the development process, we achieved several key objectives:

Technical Achievements

1. **Data Processing:** Successfully parsed XML SMS data into structured JSON format
2. **API Implementation:** Built complete CRUD functionality using Python's http.server
3. **Security Implementation:** Integrated Basic Authentication for all endpoints
4. **Performance Optimization:** Compared Linear Search (O(n)) vs Dictionary Lookup (O(1))

Future Improvements

Security Enhancements: Migrate to JWT authentication with token expiration and Implement TLS.

Feature Additions: advanced filtering (by date range, amount, transaction type)

Performance Optimizations: API gateway for load balancing and synchronous processing for heavy operations

Final Thoughts

Building this REST API provided hands-on experience with essential concepts that power modern web applications. The combination of API design, security implementation, and algorithm optimization demonstrates the multifaceted nature of backend development. This foundation prepares us for building more sophisticated systems.

- **End of Report -**