

Introduction to API Security

We developed a RESTful API for the MoMo SMS Data Processing System to securely manage transaction records. The API includes endpoints for retrieving, adding, updating, and deleting transactions. To ensure security, we implemented **Basic Authentication**, which requires users to provide a username and password to access the endpoints.

Documentation of Endpoints

API Documentation

Overview

In this project, we created a RESTful API server for managing mobile money transactions. Our implementation uses Python's built-in `http.server` library to handle HTTP requests and responses, and stores transaction data in a JSON file. We designed the API to support basic authentication and CRUD operations.

How We Created the Server

- We used Python's **BaseHTTPRequestHandler** and **HTTPServer** classes to build the server.
- Before starting the server, we prompt the user for a username and password using the terminal. The password input is hidden for security using the `getpass` module.
- If the credentials are incorrect, the server prints "401 Unauthorized due to Invalid credentials" and exits.
- Once authenticated, the server listens on ``localhost:8080`` for incoming API requests.

API Authentication

- Every API request must include HTTP Basic Authentication headers.
- The server checks for the correct username (**team2**) and password (**alu@123**) in the request headers.
- If authentication fails, the server responds with a 401 Unauthorized status.

Why We Used "curl"

We used the `curl` command-line tool to interact with our API endpoints during development and testing.

`curl` is widely used for making HTTP requests from the terminal. It allows us to easily specify request methods (GET, POST, PUT, DELETE), add headers, send authentication credentials, and include request bodies.

Using curl helped us quickly verify that our API endpoints were working as expected and made it simple to test different scenarios and error cases.

API Endpoints and Examples

1. Get All Transactions

Endpoint:

GET /transactions

curl -u team2:alu@123 http://localhost:8080/transactions

Response:

```
[
  {
    "transaction_id": "76662021700",
    "type": "transfer",
    "amount": "2000",
    "sender": "Jane Smith (*****013)",
    "receiver": "You",
    "timestamp": "2024-05-10T16:30:51",
    "new_balance": "2000",
    "message": null,
    "financial_transaction_id": "76662021700",
    "service_center": "+250788110381",
    "fee": "0"
  }
]
```

etc..(Because we have more than 100 transactions)

2. Get a Single Transaction

Endpoint:

GET /transactions/{id}

We implemented this endpoint to retrieve a specific transaction by its ID.

curl -u team2:alu@123 http://localhost:8080/transactions/76662021700

Response:

```
{
  "transaction_id": "76662021700",
  "type": "transfer",
  "amount": "2000",
  "sender": "Jane Smith (*****013)",
```

```
"receiver": "You",
"timestamp": "2024-05-10T16:30:51",
"new_balance": "2000",
"message": null,
"financial_transaction_id": "76662021700",
"service_center": "+250788110381",
"fee": "0"
}
```

3. Add a New Transaction

Endpoint:
POST /transactions

We designed this endpoint to add a new transaction.

```
curl -X POST -u team2:alu@123 -H "Content-Type: application/json" -d '{
  "transaction_id": "1234567890",
  "type": "transfer",
  "amount": "5000",
  "sender": "You",
  "receiver": "John Doe",
  "timestamp": "2024-05-13T10:00:00",
  "new_balance": "15880",
  "message": null,
  "financial_transaction_id": "1234567890",
  "service_center": "+250788110381",
  "fee": "0"
}' http://localhost:8080/transactions
```

4. Update a Transaction

Endpoint:
PUT /transactions/{id}

We created this endpoint to update an existing transaction by its ID.

```
curl -X PUT -u team2:alu@123 -H "Content-Type: application/json" -d '{
  "transaction_id": "76662021700",
  "type": "transfer",
  "amount": "2500",
  "sender": "Jane Smith (*****013)",
  "receiver": "You",
  "timestamp": "2024-05-10T16:30:51",
  "new_balance": "2500",
  "message": null,
}
```

```
"financial_transaction_id": "76662021700",  
"service_center": "+250788110381",  
"fee": "0"  
}' http://localhost:8080/transactions/76662021700
```

5. Delete a Transaction

Endpoint:

DELETE /transactions/{id}

We implemented this endpoint to delete a transaction by its ID.

```
curl -X DELETE -u admin:alu@123 http://localhost:8080/transactions/76662021700
```

Error Codes

Error Code	Description
400	Bad Request (e.g., invalid JSON)
401	Unauthorized (invalid credentials)
404	Not Found (endpoint/ID does not exist)

Data Storage

- We stored all transactions in data/transactions.json.
- The server reads from and writes to this file for every CRUD operation.

Results of DSA Comparison

Data Structures & Algorithms (DSA) Comparison Results

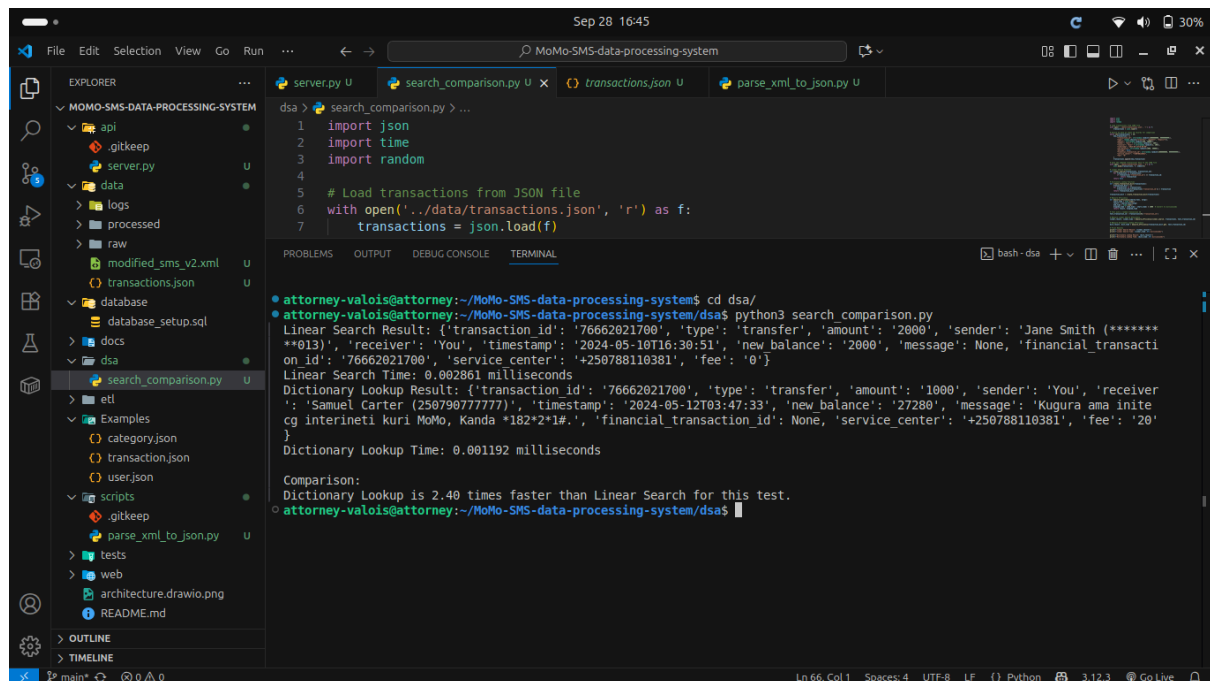
Results

We conducted a comparison between **Linear Search** and **Dictionary Lookup** to determine the efficiency of searching for transactions by ID. The results are as follows with a screenshot:

Linear Search Time: 0.002861 milliseconds

Dictionary Lookup Time: 0.001192 milliseconds

Comparison: Dictionary Lookup is 2.40 times faster than Linear Search for this test.



```
server.py U search_comparison.py U transactions.json U parse_xml_to_json.py U
dsa > search_comparison.py > ...
1 import json
2 import time
3 import random
4
5 # Load transactions from JSON file
6 with open('../data/transactions.json', 'r') as f:
7     transactions = json.load(f)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
attorney-valois@attorney:~/MoMo-SMS-data-processing-system$ cd dsa/
attorney-valois@attorney:~/MoMo-SMS-data-processing-system/dsa$ python3 search_comparison.py
Linear Search Result: {'transaction_id': '76662021700', 'type': 'transfer', 'amount': '2000', 'sender': 'Jane Smith (*****
**013)', 'receiver': 'You', 'timestamp': '2024-05-10T16:30:51', 'new_balance': '2000', 'message': None, 'financial_transacti
on_id': '76662021700', 'service_center': '+250788110381', 'fee': '0'}
Linear Search Time: 0.002861 milliseconds
Dictionary Lookup Result: {'transaction_id': '76662021700', 'type': 'transfer', 'amount': '1000', 'sender': 'You', 'receiver
': 'Samuel Carter (25079077777)', 'timestamp': '2024-05-12T03:47:33', 'new_balance': '27280', 'message': 'Kugura ama inite
cg interineti kuri MoMo, Kanda *182*2*1#.', 'financial_transaction_id': None, 'service_center': '+250788110381', 'fee': '20'
}
Dictionary Lookup Time: 0.001192 milliseconds

Comparison:
Dictionary Lookup is 2.40 times faster than Linear Search for this test.
attorney-valois@attorney:~/MoMo-SMS-data-processing-system/dsa$
```

After this we did a reflection about why is Dictionary Lookup Faster?

And we observed that dictionary lookup is faster because it uses a hash table to directly access the transaction by its key ('transaction_id'), resulting in an average time complexity of $O(1)$. In contrast, linear search scans each transaction sequentially, resulting in a time complexity of $O(n)$.

Alternative Data Structures

We considered other data structures that could be useful in different scenarios:

- **Binary Search Tree (BST):** If transaction IDs were sorted, a BST could provide a search time of $O(\log n)$.
- **Trie:** If transaction IDs had common prefixes, a trie could be used for efficient prefix-based searches.

Reflection on Basic Auth Limitations

This is our short report, explaining why Basic Auth is weak and suggesting stronger alternatives (JWT, OAuth2).

Why Basic Authentication is Weak

We implemented Basic Authentication to secure our API endpoints. However, we identified several weaknesses with this method:

- **No Encryption:** Credentials are only Base64-encoded, not encrypted. If intercepted, they can be easily decoded.
- **No Session Management:** Each request must include the credentials, which can be inefficient and insecure.
- **Vulnerable to Replay Attacks:** If an attacker intercepts the credentials, they can reuse them indefinitely.

Stronger Alternatives

To address these weaknesses, we suggest the following stronger alternatives:

- **JWT (JSON Web Tokens):** Tokens are signed and can include expiration times, making them more secure and scalable for modern applications.
- **OAuth2:** Provides authorization frameworks for third-party applications, supporting token expiration, refresh tokens, and scopes for fine-grained access control.