1. Introduction to API Security

Overview
API Security refers to the practices and protocols used to protect Application Programming Interfaces (APIs) from malicious attacks, misuse, and data breaches. As APIs expose application logic and sensitive data, they are primary targets for attackers.

Core Principles
Authentication: Verifying the identity of the client attempting to access the API. In this project, we implemented Basic Authentication which requires a username and password with every request.
Authorization: ensuring that the authenticated user has permissions to perform specific actions (e.g., only admins can delete transactions).
Input Validation: Ensuring that data sent to the API is valid and safe to prevent injection attacks or data corruption.

2. API Endpoint Documentation

Base URL: http://127.0.0.1:5000
Authentication: Basic Auth (Username: admin, Password: secret)

2.1 GET /transactions

Description:Retrieves a complete list of all parsed MoMo SMS transactions.
Method: GET
Protected: Yes
Response (200 OK):

```
json
[
  { "id": "Pay1", "sms": { "body": "Payment of...", "amount": "500" } },
  { "id": "Rec2", "sms": { "body": "Received...", "amount": "1000" } }
]
```

2.2 GET /transactions/id

Description: Retrieves specific details for a single transaction by its unique ID.
Method: GET
Protected: Yes
Response (200 OK):JSON object of the specific transaction.
Error (404 Not Found): {"error": "Id not found"}

2.3 POST /transactions/

Description: Creates a new transaction record. The API automatically generates an ID based on the content of the SMS body.
Method:POST
Header:Content-Type: application/json
Body:

```
json
{
  "body": "Payment of 2000 RWF",
  "address": "MOMO_PAY",
  "readable_date": "Jan 01 2024"
}
```

Response (200 OK): The created object with the assigned ID.

2.4 PUT /transactions/id
Description: Updates an existing transaction's details.
Method: PUT
Body: JSON object containing fields to update.
Response (200 OK): The updated transaction object.

2.5 DELETE /transactionsid
Description: Permanently removes a transaction from the dataset.
Method: DELETE
Response (200 OK): {"success": "Pay1 removed successfully"}

3. Results of DSA Comparison

Objective
To analyze the efficiency of data retrieval using different Data Structures and Algorithms.
The Experiment We compared the time complexity of finding a specific transaction by ID.

1. Linear Search (List/Array):
Mechanism: Iterating through the list records one by one until the matching id is found.
Big O Complexity: O(n). As the data grows (e.g., 10,000 SMS messages), the search time increases linearly.
Observation: acceptable for small datasets (like our XML file) but becomes slow as the dataset scales.

2. Hash Map Lookup (Python Dictionary): Mechanism: Storing records where the Key is the id and the Value is the record.
Big O Complexity:O(1). Immediate retrieval regardless of dataset size.
Observation: significantly faster for lookups, specific updates (PUT), and deletions (DELETE).

Conclusion
While our current implementation uses a List (resulting in O(n) for DELETE and GET-Single operations), refactoring to a Dictionary would drastically improve performance for the delete and get_by_id endpoints in a production environment with millions of transactions.

4. Reflection on Basic Auth Limitations

Current Implementation

We utilized HTTP Basic Authentication, where the client sends Authorization: Basic base64(username:password) with every request.

Limitations Identify

1. Security Risk (Base64 is not Encryption):Base64 is merely encoding. If this API is accessed over HTTP (non-secure), anyone intercepting the traffic (Man-in-the-Middle attack) can easily decode the header and steal the admin credentials.
2. No Session Management: The browser or client must send the password with every single request. This increases the attack surface compared to sending it once to get a session token.
3. Cannot Log Out: Basic Auth does not support a true "logout" mechanism on the server side without closing the browser, as the browser caches the credentials.
4. Granularity: Basic Auth usually applies a "blanket" access level. It is difficult to implement complex roles without adding significant backend logic.