API Security Report

1. Introduction to API Security

Application Programming Interfaces (APIs) have become the backbone of modern applications, enabling communication between services, systems, and users. With this growth, securing APIs has become critical because vulnerabilities can lead to unauthorized data access, data manipulation, or service disruption.

API security ensures:

- **Confidentiality**: Protecting sensitive information from exposure.
- **Integrity**: Preventing unauthorized modification of data.
- **Availability**: Ensuring the API remains accessible and reliable.

Key principles of API security include authentication, authorization, encryption, input validation, and rate limiting. Without proper safeguards, APIs may expose sensitive data or serve as entry points for malicious attacks.

2. Documentation of Endpoints

The system provides RESTful endpoints for managing Mobile Money (MoMo) transactions. Below is the documentation of the core API endpoints:

GET /transactions

Description: Retrieve a list of all transactions.

Response: JSON array of transaction objects.

Example:

[

{

"id": 1,

"type": "Deposit",

"amount": 5000,

"sender": "0788000000",

"receiver": "0788111111",

"timestamp": "2025-09-30T10:20:00"

}

]

GET /transactions/{id}

Description: Retrieve details of a single transaction by ID.

Response: JSON object with transaction details.

POST /transactions

Description: Add a new transaction.

Request body:

{

"type": "Withdrawal",

"amount": 2000,

"sender": "0788222222",

"receiver": "0788333333",

"timestamp": "2025-09-30T15:00:00"

}

PUT /transactions/{id}

Description: Update details of an existing transaction.

Request body: Similar to POST.

Response: Confirmation of update.

DELETE /transactions/{id}

Description: Remove a transaction by ID.

Response: Confirmation of deletion.

3. Results of DSA Comparison

In order to optimize transaction handling, we compared the performance of different Data Structures and Algorithms (DSA) for storing and retrieving SMS transaction data.

DSA Method	Insertion Speed	Search Speed	Memory Usage	Best Use Case
Array	Fast	Linear search (O(n))	Moderate	Simple static storage
Linked List	Fast	Linear search (O(n))	Higher	Sequential access, dynamic size
Hash Table	Moderate	Very fast (O(1))	Higher	Quick lookups by transaction ID
Binary Search Tree (BST)	Moderate	Logarithmic (O(log n))	Moderate	Sorted data storage
AVL Tree	Slower insert	Logarithmic (O(log n))	Higher	Balanced queries

Conclusion:

- Hash tables provide the fastest access for transaction lookups.
- BST/AVL trees are ideal for range queries (e.g., transactions between certain dates).
- Arrays/Lists are simplest but least efficient for large datasets.

4. Reflection on Basic Auth Limitations

While Basic Authentication is simple to implement, it has several limitations when used for API security:

1. **Plaintext Credentials**: Usernames and passwords are sent with every request (often Base64 encoded), making them vulnerable if transmitted without HTTPS.
2. **No Token Expiry**: Credentials remain valid until changed, unlike tokens which can expire and be refreshed.

3. **No Granular Access Control**: Basic Auth does not natively support roles or permissions.
4. **Scalability Issues**: Managing passwords across multiple services becomes difficult.
5. **Vulnerability to Replay Attacks**: If intercepted, credentials can be reused by attackers.

Better Alternatives:

- OAuth 2.0 / OpenID Connect for delegated authorization.
- API Keys with scopes for simple use cases.
- JWT (JSON Web Tokens) for stateless, scalable authentication.

Final Reflection:

Basic Auth can be used for simple or internal APIs but is insufficient for production environments requiring high security. More robust mechanisms such as OAuth 2.0 or JWT should be preferred.