Introduction to api security

The practice of safeguarding Application Programming Interfaces (APIs) against malevolent attacks, abuse, or unauthorized access is known as API security. APIs serve as a conduit for data exchange and communication between various software programs. Securing APIs is crucial to preventing data breaches, service interruptions, and other security incidents since they frequently reveal sensitive information or important functionality.

Typical API security measures include rate limiting to stop misuse, input validation to stop attacks like SQL injection or cross-site scripting, encryption of data in transit, authorization to make sure users can only access resources that are allowed, and authentication to confirm the identity of users or applications. By limiting access to the API to trusted clients, robust API security protects user data and the system.

Documentation of endpoints

MoMo Transactions API Documentation

This API provides CRUD operations for managing mobile money (MoMo) transactions.
All data is stored in transactions_dict.json.

Authentication
All endpoints require Basic Authentication.

- Username: admin
- Password: password123

Unauthorized requests will return 401 Unauthorized.

Example (with curl):
curl -u admin:password123 http://127.0.0.1:5000/transactions

Endpoints

1. Get All Transactions

- Method: GET
- URL: /transactions

Request Example:
curl -u admin:password123 http://127.0.0.1:5000/transactions

Response Example:

{
"123456789012": {
"transaction_id": "123456789012",
"sender": "Ishimwe",
"receiver": "Jean Claude",
"amount": 100
},
"987654321098": {
"transaction_id": "987654321098",

"sender": "John",
"receiver": "Mary",
"amount": 250
}
}

Errors:

- 401 Unauthorized

---

2. Get Transaction by ID

- Method: GET
- URL: /transactions/<transaction_id>

Request Example:
curl -u admin:password123 http://127.0.0.1:5000/transactions/123456789012

Response Example:
{
"transaction_id": "123456789012",
"sender": "Alice",
"receiver": "Bob",
"amount": 100
}

Errors:

- 401 Unauthorized
- 404 Not Found – if transaction ID doesn't exist

---

3. Create a Transaction

- Method: POST
- URL: /transactions

Request Example:
curl -u admin:password123 -X POST http://127.0.0.1:5000/transactions -H "Content-Type: application/json" -d

'{"sender":"Alice","receiver":"Bob","amount":150}'

Response Example:
{
"transaction_id": "654321987654",
"sender": "Alice",
"receiver": "Bob",
"amount": 150
}

Notes:

- If no transaction_id is provided, one will be auto-generated.

Errors:

- 400 Bad Request – if no data provided
- 401 Unauthorized

---

4. Update a Transaction

- Method: PUT
- URL: /transactions/<transaction_id>

Request Example:
curl -u admin:password123 -X PUT http://127.0.0.1:5000/transactions/123456789012 -H "Content-Type: application/json" -d '{"amount":200,"receiver":"Charlie"}'

Response Example:
{
"transaction_id": "123456789012",
"sender": "Alice",
"receiver": "Charlie",
"amount": 200
}

Notes:

- transaction_id cannot be changed.
- Only other fields are updated.

Errors:

- 401 Unauthorized
- 404 Not Found – if transaction doesn't exist

---

5. Delete a Transaction

- Method: DELETE
- URL: /transactions/<transaction_id>

Request Example:
curl -u admin:password123 -X DELETE
http://127.0.0.1:5000/transactions/123456789012

Response Example:
{
"message": "Transaction deleted"
}

Errors:

- 401 Unauthorized
- 404 Not Found – if transaction doesn't exist

Error Codes Summary

- 400 Bad Request – Missing or invalid request body
- 401 Unauthorized – Invalid or missing credentials
- 404 Not Found – Resource doesn't exist

Running the API
Run the API locally with:
python api/transactions_api.py

Server starts at:

http://127.0.0.1:5000

File Storage
Transactions are stored persistently in:
08_09_2025/momo-sms-dashboard/data/processed/transactions_dict.json

---

Example Workflow

1. Create a transaction (POST /transactions)
2. Retrieve it (GET /transactions/<id>)
3. Update it (PUT /transactions/<id>)
4. Delete it (DELETE /transactions/<id>)

Results of DSA comparison

Two search methods were implemented to find a transaction by its ID: Linear Search and Dictionary Lookup. The Linear Search method goes through each record one by one until it finds a match, while the Dictionary Lookup method accesses the transaction directly using its ID as a key.

After testing both methods on a dataset of 24 transactions, the Dictionary Lookup proved to be much faster. Linear Search took longer because it had to check each transaction sequentially, while the Dictionary Lookup retrieved the record instantly.

This is because dictionaries in Python are based on hash tables, which allow constant-time access to elements. Linear search has a time complexity of O(n), meaning it becomes slower as the dataset grows, while dictionary lookup has a time complexity of O(1).

Reflection of basic Auth limitations

By requiring a username and password, Basic Authentication is a straightforward technique for protecting APIs. Despite being simple to use, it has a lot of drawbacks. Every request includes credentials, which are frequently sent in base64 encoding. This encoding is not encrypted and is susceptible to interception if HTTPS is not used. It is also unsuitable for complex applications with multiple user roles because it does not have session management or granular access control.

If credentials are compromised, they must be manually changed because Basic Auth does not support token expiration or revocation. More secure techniques like OAuth, JWT, or API keys are advised for production or large-scale systems. Basic Auth is sufficient for testing and demonstration in the context of the MoMo Transactions API, but it would not be sufficient for practical deployment where sensitive financial data is involved.