Building and Securing a REST API - Project Report

Team Name: ALU MoMo API Team

Team Members:
Darlene Ayinkamiye
Chely Kelvin Sheja
Solomon Leek

Course: Software Engineering
Institution: African Leadership University
Date: January 24, 2026

Executive Summary

This project demonstrates the implementation of a secure REST API for managing Mobile Money (MoMo) transaction data. Built using Python's http.server module, the API provides full CRUD functionality with Basic Authentication security. The project also includes a comprehensive analysis of data structure algorithms comparing linear search versus dictionary lookup performance.

1. Introduction to API Security

Overview

API security is critical for protecting sensitive financial transaction data from unauthorized access, data breaches, and malicious attacks. Our implementation focuses on authentication, authorization, and secure data handling.

Security Implementation: Basic Authentication

What is Basic Authentication?
Basic Authentication is an HTTP authentication scheme where credentials (username and password) are encoded in Base64 and sent in the Authorization header with each request.

How We Implemented It:

```
  Authorization: Basic YWRtaW46cGFzc3dvcmQ=
# Decoded: admin:password
```

Security Flow:

1. Client sends request with Authorization header
2. Server decodes Base64 credentials
3. Server validates credentials against stored users
4. Returns 200 OK for valid credentials, 401 Unauthorized for invalid

Code Example:

```
def authenticate_request(handler):
    auth_header = handler.headers.get('Authorization')
    if not auth_header:
        return False
    credentials = parse_basic_auth_header(auth_header)
    if not credentials:
        return False
    username, password = credentials
    return VALID_CREDENTIALS.get(username) == password
```

2. Why Basic Auth is Weak

- **Credentials in Every Request:** Username and password sent with every API call, increasing exposure to interception.
- **Base64 Encoding is NOT Encryption:** Anyone intercepting traffic can decode credentials instantly.
- **No HTTPS Requirement:** Credentials can be transmitted in cleartext over network.
- **No Expiration:** Credentials valid indefinitely until changed.
- **No Fine-Grained Access Control:** Simple username/password check, no role-based permissions.

Real-World Risks: Man-in-the-middle attacks, replay attacks, brute force, credential theft.

3. Stronger Security Alternatives

Recommended: JSON Web Tokens (JWT)

1. User authenticates once with username/password
2. Server generates signed JWT token with expiration
3. Client stores token and sends with requests
4. Server validates token signature (no database lookup)

Advantages: Token expires automatically, stateless authentication, contains user claims, cryptographic signature.

Alternative: OAuth 2.0

- Delegated authorization framework
- User authenticates with trusted provider (Google, GitHub)
- App receives access token, never sees password
- Tokens have limited scope and expiration

4. API Documentation

Base URL

```
http://localhost:8000
```

Authentication

```
Authorization: Basic base64(username:password)
```

Valid Credentials

- admin : password
- user1 : test123
- developer : devpass

Endpoints

- **GET /transactions** - Retrieve all transaction records
- **GET /transactions/{id}** - Retrieve a single transaction by ID
- **POST /transactions** - Create a new transaction record
- **PUT /transactions/{id}** - Update an existing transaction
- **DELETE /transactions/{id}** - Delete a transaction record

Error Codes

Status Code	Description	Example Scenario
200 OK	Request successful	GET, PUT, DELETE successful
201 Created	Resource created	POST successful
400 Bad Request	Invalid input data	Missing required fields
401 Unauthorized	Invalid credentials	Wrong username/password
404 Not Found	Resource not found	Invalid transaction ID
405 Method Not Allowed	Invalid HTTP method	POST to /transactions/{id}
500 Internal Server Error	Server error	Unexpected exception

5. Data Structures & Algorithms Analysis

Compared linear search (O(n)) and dictionary lookup (O(1)) for finding transaction records by ID. Dictionary lookup is 5x faster for our dataset and scales better for large data.

6. Testing & Validation

- Authentication: Valid/invalid credentials, missing header
- CRUD: GET all, GET by ID, POST, PUT, DELETE, error cases
- Test evidence: Screenshots in `/screenshots/` directory

7. Conclusion

- REST API with all CRUD operations
- Basic Authentication with error handling
- Security vulnerabilities analyzed
- Stronger alternatives (JWT, OAuth 2.0) recommended
- DSA performance analysis
- Comprehensive testing

Individual Contributions

- **Darlene Ayinkamiye:** API server, CRUD, project coordination
- **Chely Kelvin Sheja:** Data parsing, GET endpoints, testing, docs
- **Solomon Leek:** Authentication, security docs, endpoint validation

References

1. Mozilla Developer Network (MDN) - HTTP Authentication
2. OWASP API Security Project
3. Python Documentation - http.server module
4. RFC 7617 - The 'Basic' HTTP Authentication Scheme
5. JWT Introduction - jwt.io
6. Time Complexity Analysis - Big-O Cheat Sheet

Submitted by: Darlene Ayinkamiye, Chely Kelvin Sheja, Solomon Leek
African Leadership University
January 24, 2026