Project Report

1. Introduction to API Security

This project represents an efficient implementation of understanding REST API design for MoMo data transaction endpoints. In this scenario we are dealing with financial data and so API security is an important implementation in this process.

The main goal of API security is to ensure data integrity and confidentiality. For the MOMO analyzer, we implemented key security measures to prevent unauthorized access to sensitive transaction histories (sender, receiver, and balance). By requiring credentials for every request, the API ensures that only trusted clients can interact with the server.

Key Featuers:

- Authentication: Verifying who the user is.
- Authorization: Determining what the user is allowed to do.
- Encryption: Protecting data while it is “in transit” (typically via HTTPS).

2.Documentation of endpoints

Method	Endpoint	Description	Expected Response
GET	/transactions	List all parsed SMS transactions	200 OK (JSON Array)
GET	/transactions/{id}	View a specific transaction by ID	200 OK or 404 Not Found
POST	/transactions	Create a new transaction record	201 Created
PUT	/transactions/{id}	Update an existing transaction	200 OK
DELETE	/transactions/{id}	Delete a record from the list	204 No Content

Request Example (POST)

```
{
  "user_id": 1,
  "recepient_sender": "250788123456",
  "amount": 500.0,
  "category_id": 2
}
```


3.Results of DSA Comparison

DSA Integration: Search Efficiency Reflection

Comparative Analysis

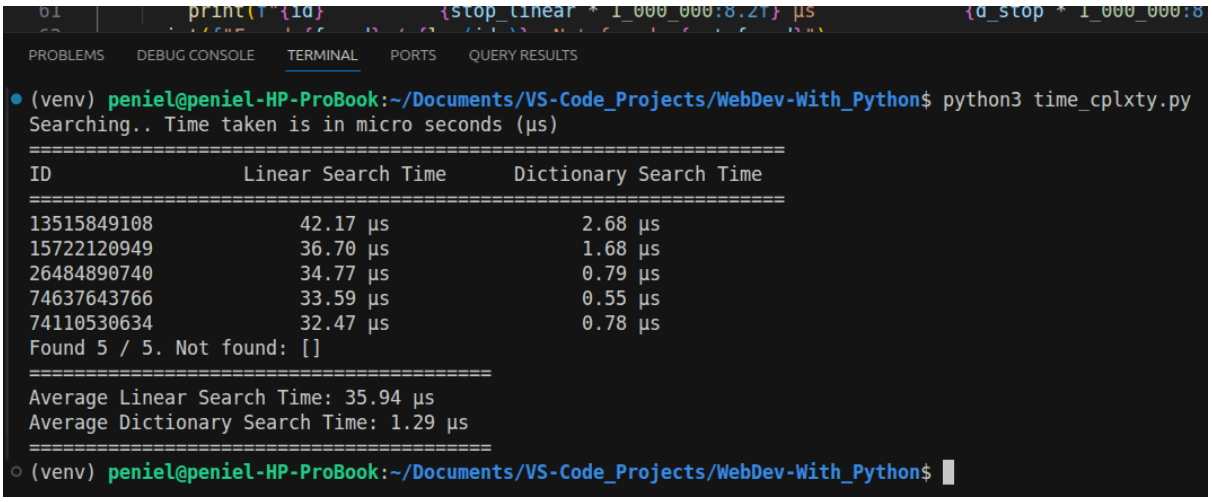
To evaluate search efficiency, we performed empirical testing on a dataset of transactions, increasing the volume of searches from 5 to 20 IDs. The results consistently demonstrated that **Dictionary Lookup** outperforms **Linear Search** by a significant margin.

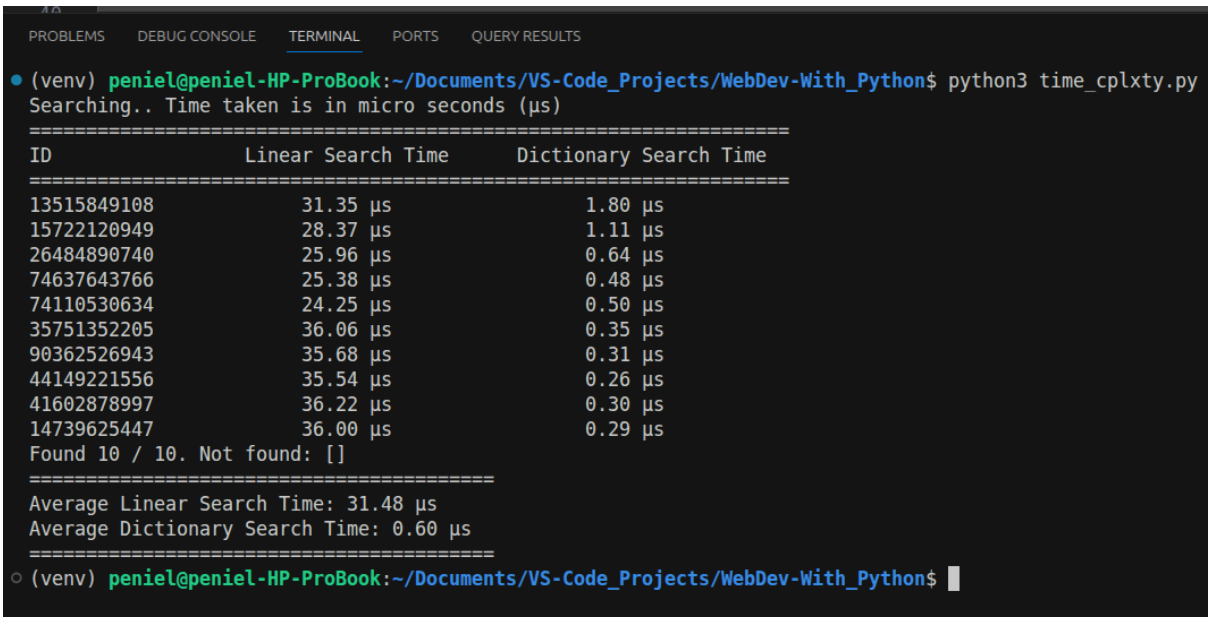
Test Results Summary (in microseconds µs)

Number of IDs	Avg. Linear Search Time	Avg. Dictionary Search Time
5 IDs	35.94 µs	1.29 µs
10 IDs	31.48 µs	0.60 µs
15 IDs	30.29 µs	0.52 µs
20 IDs	34.87 µs	0.35 µs

Performance Chart

SCREENSHOTS OF TERMINAL OUTPUT FOR TIME COMPLEXITY COMPARISON





As the test size increased, the Dictionary search time actually became more efficient on average, while the Linear search time fluctuated based on the index position of the IDs within the list.

Why Dictionary Lookup is Faster

The disparity in speed is a result of the underlying data structures:

- **Linear Search ($O(n)$):** This method uses a "brute force" approach. It starts at the first index of trans_list and compares the transaction_id of every single object until it finds a match. If you have 1,000 records and the target is at the end, it performs 1,000 checks.
- **Dictionary Lookup ($O(1)$):** Python dictionaries use **Hash Tables**. Instead of searching through a list, Python runs the id through a hash function to calculate the exact memory address where the transaction is stored. This "direct access"

means the time taken to find a record is constant, regardless of whether the list has 20 items or 20,000 items.

Alternative Method: Binary Search

A strong alternative to dictionary-based search is the **Binary Search**. While it is slightly slower than a dictionary ($O(\log n)$ vs $O(1)$), it is significantly faster than a linear search and more memory-efficient than a dictionary because it doesn't require storing a separate hash table.

How it Works

Binary search works on a "divide and conquer" principle. By starting in the middle of a **sorted** list, it eliminates half of the remaining items with every single comparison.

4. Authentication & Security Implementation

Overview of Implementation

The MoMo SMS API uses **HTTP Basic Authentication** to secure its transaction endpoints. This was implemented in auth.py using the flask_httpauth library, integrated with python-dotenv for secure credential management.

1. Protected Endpoints

Every endpoint in the api/ directory (e.g., GET /transactions, POST /transactions) is decorated with @auth.login_required. This creates a security layer that intercepts incoming requests before they reach the data logic.

2. Logic and Validation Flow

- **Credential Retrieval:** The system pulls the master username and password from a local .env file using os.getenv.
- **Secure Hashing:** Instead of comparing plain-text strings, the API uses the **PBKDF2-SHA256** algorithm via the werkzeug security library. This ensures that even if the environment variables were somehow exposed, the actual password remains mathematically obscured.
- **Verification:** The verify_password function serves as the gatekeeper.
    - If username and password match the stored hash, the request proceeds.
    - If they do not match, the system returns an **HTTP 401 Unauthorized** error.

3. Reflection: Why Basic Authentication is Weak

While Basic Auth is sufficient for this academic project, it has several critical vulnerabilities that make it unsuitable for production-grade financial or mobile money systems:

- **Credential Exposure:** Basic Auth sends the username and password in every single request header. If the connection is not encrypted (HTTPS), any "man-in-the-middle" can intercept the packets and read the credentials.
- **Encoding vs. Encryption:** Credentials are sent as Base64-encoded strings. **Base64 is not encryption**; it can be reversed instantly by any standard tool.
- **Lack of Revocation:** Once a user provides credentials, the browser often caches them indefinitely. There is no built-in "session logout" or token expiration.
- **No Granular Control:** Basic Auth is typically "all or nothing." It doesn't easily support different permission levels (e.g., "Viewer" vs. "Admin") without complex custom logic.

4. Stronger Alternatives

To scale this MoMo Project for a real-world scenario, we recommend the following industry-standard alternatives:

JWT (JSON Web Tokens)

- **How it works:** After a single login, the server provides a signed token. The client sends this token in the header of future requests.
- **Advantage:** Tokens are **stateless** and **time-limited**. If a token is stolen, it will expire automatically after a short period (e.g., 15 minutes), limiting the attacker's window.

OAuth 2.0

- **How it works:** A framework that allows for "delegated access." The client application never sees the user's password; instead, it receives an **Access Token** from a dedicated Identity Provider (like Google or a custom Auth server).
- **Advantage:** It supports "Scopes," allowing you to give a mobile app permission to *view* transactions but not *delete* them.