## SMS transaction data was parsed and structured for API consumption

The raw modified_sms_v2.xml dataset was parsed using Python to extract structured transaction information such as transaction type, amount, sender, receiver, and timestamp. Each SMS record was processed and normalized into a consistent internal representation.

Parsed records were converted into JSON-compatible objects, enabling seamless integration with the REST API layer. This transformation step ensured that downstream API operations could work with clean, predictable data structures.

The parsing logic was designed to be reusable and extensible, allowing the dataset to be reprocessed or expanded without changes to the API implementation.

## Core CRUD API endpoints were implemented using plain Python

The POST /transactions endpoint is meant to be used as a means by which authenticated users can insert new transaction records into the MoMo SMS transaction database. When a request is received by the API, Basic Authentication is used to authenticate the user. After authentication, the endpoint then reads and authenticates the content of the request body and checks to make sure it has all the necessary fields (external_ref, amount, raw_data, and transaction_date) in it. In case of lack of any of the necessary fields or a malformed format of the JSON, the API issues 400 Bad Request with a descriptive error message. This endpoint then interprets the transaction data by breaking out the transaction date into the ISO 8601 format, giving the transaction a default category (TRANSFER) and default currency (RWF) if not specified, and linking the transaction to the default system user. In case a fee amount is provided, the endpoint will generate a distinct record of fee that is attached to the transaction. When a successful creation has taken place, the transaction is recorded in the database, where it receives a transaction-id, and the API responds with a 201 Created response with the entire transaction object. Any attempt to make a transaction with improper authentication will result in a 401

Unauthorized response, whereas invalid request syntax or a lack of the necessary fields will result in a 400 Bad Request response.

## Authentication was added to protect all API endpoints

Basic Authentication was implemented to restrict API access to authorized users only. All endpoints require valid credentials to be supplied through the Authorization header before any operation is performed.

Authentication checks are enforced centrally, ensuring that unauthorized requests are consistently rejected with a 401 Unauthorized response and a proper authentication challenge header.

To improve maintainability and demonstrate modular design, the authentication logic was abstracted into a dedicated helper module, separating security concerns from request-handling logic.

## Transaction deletion was implemented with proper authorization and safeguards

We implemented a DELETE /transactions/{id} endpoint to allow authorized clients to remove transactions by ID. The endpoint validates both authentication and request format before attempting any database operation.

If the specified transaction does not exist, the API responds with a 404 Not Found error to prevent unintended behavior. Valid transactions are deleted using controlled database sessions to maintain data integrity.

Successful deletions return a 204 No Content response, following REST best practices by confirming the action without returning unnecessary data.

## Efficient data lookup was demonstrated using alternative data structures

To demonstrate data structure efficiency, transaction lookup was implemented using both linear search and dictionary-based access. Linear search scans the transaction list sequentially to locate a record by ID, while dictionary lookup retrieves records directly using key-based access.

Performance comparisons were conducted on datasets of at least 20 records to highlight differences in lookup efficiency. Dictionary-based lookup consistently outperformed linear search due to constant-time access.

This comparison illustrates why hash-based data structures are preferable for frequent lookup operations and highlights how data structure choice impacts system performance.

## API functionality was tested and validated using command-line tools

The API was tested using tools such as curl to verify correct behavior across all endpoints. Tests included successful authenticated requests, unauthorized access attempts, and CRUD operations.

Screenshots were captured to demonstrate successful GET requests with authentication, rejection of unauthorized requests, and successful POST, PUT, and DELETE operations.

These tests confirm that the API behaves as expected under both valid and invalid conditions, ensuring reliability and correctness.