Project Report: Momo App Backend

Introduction

The **Momo App Backend** is a secure and efficient system designed for handling Mobile Money transactions. This report outlines the architectural choices made during development to ensure security, performance, and reliability.

Technical Design Choices

1. Authentication: JWT vs. Basic Auth

We implemented **JSON Web Tokens (JWT)** for user authentication instead of traditional **Basic Auth**. * **Rationale:** Basic Auth requires sending credentials with every request, which increases the risk of interception. JWT allows for a single login event that generates a temporary, cryptographically signed token. * **Security:** By using tokens, the server remains stateless and does not need to store session data. Additionally, we use a **blacklist** to immediately invalidate tokens when a user logs out. * **Error Handling:** To prevent account guessing attacks, the system provides a generic error message ("Invalid email or password") regardless of whether the email or password was the incorrect field.

2. Data Security: Password Hashing

User passwords are never stored in plain text. We utilize the **bcrypt** algorithm for hashing. * **Implementation:** Bcrypt automatically incorporates a unique "salt" for every password, ensuring that the same password used by different users results in different hashes. This is one way process, the password cannot be retrieved from the hash, which makes it more secure even when hackers somehow get the hash.

3. Performance: Transaction Lookup

Optimizing data retrieval was a primary focus for this project. We evaluated two search methods for transaction records:

Linear Search Initially, the system retrieved transactions by iterating through a list until a match was found. * **Limitation:** This approach has a time complexity of $O(n)$, meaning search time increases as the number of transactions grows. For 10,000 records, the system might need to scan every record. Linear Search Context

Dictionary Lookup (Indexed) To improve performance, we also implemented an indexed lookup using a Python Dictionary. * **Optimization:** This provides a time complexity of $O(1)$, ensuring that retrieval remains nearly instantaneous regardless of whether there are ten or ten million records. Indexed Lookup Performance

API Reference

The following table summarizes the primary endpoints available in the system:

Endpoint	Method	Auth	Purpose
`/auth/register`	`POST`	No	Creates a new user account.
`/auth/login`	`POST`	No	Authenticates user and generates a JWT.
`/auth/logout`	`POST`	Yes	Token invalidation and session termination.
`/transactions/`	`POST`	Yes	Initiates a money transfer.
`/transactions/me`	`GET`	Yes	Retrieves the current user's history.
`/transactions/<id>`	`GET`	Yes	Retrieve by ID (Linear Search).
`/indexed_transactions/<id>`	`GET`	Yes	Retrieve by ID (Dictionary Lookup).
`/transactions/<id>`	`PUT`	Admin	Update transaction metadata (Admin only).
`/transactions/<id>`	`DELETE`	Admin	Remove a transaction record (Admin only).

Generated: February 2, 2026