MoMo Data Analysis Assignment Report

Student Names: Eddy Irasetsa, Hasbiyallah Umutoniwabo, Jean Muhire, Tresor Nkurunziza
Course: Enterprise Web Development
Date: 2025-06-16

---

1. Introduction

This project showcases full-stack development capabilities through the creation of a dashboard for analyzing and visualizing MTN MoMo SMS transaction data. The development process includes data extraction, cleaning, structuring, backend API development, and frontend visualization.

---

2. Approach & Architecture

- **Data Extraction**: Used Python to parse raw SMS data from an XML file.
- **Data Cleaning & Categorization**: Employed regular expressions to extract structured fields (amount, sender, date, transaction type) from unstructured SMS content.
- **Database Design**: Designed a normalized SQLite database schema to store and manage categorized transaction records.
- **Backend API:**Built using **Flask** to serve transaction data in **JSON** format. The API supports dynamic queries using URL parameters (such as category, date, etc.) to filter the transaction results.
- **Frontend Dashboard:** Created using **React JS** and **Tailwind CSS**, with **Chart.js** for rendering interactive charts and visual filters. The frontend communicates with the Flask backend using fetch calls to load and display data based on user interactions.

---

3. Key Design Decisions

- **Separation of Concerns**:
  Cleanly split the project into backend/ and frontend/ directories to improve maintainability, scalability, and collaboration.
- **Technology Choices**:

- **Flask**: Chosen for its simplicity, flexibility, and wide community support. Flask enabled us to quickly build a RESTful API to serve the transaction data.
- **SQLite**: A lightweight, serverless, file-based database ideal for local development and prototyping.
- **React JS**: Used to build a dynamic and responsive dashboard. React's component-based architecture made it easy to manage UI state and interactions.
- **Tailwind CSS**: Enabled rapid and responsive UI development with utility-first classes for styling.
- **Chart.js**: Selected for its simplicity and effectiveness in rendering interactive data visualizations.

- **Error Logging**: Implemented logging to flag uncategorized or malformed SMS entries for further analysis.

---

4. Challenges & Solutions

- **Free-Form SMS Parsing**:
  - Problem: Messages came in inconsistent formats.
  - Solution: Built regex-based parsing and fallback logic to ensure resilience.
- **Frontend-Backend Communication**:
  - Problem: Blocked cross-origin requests.
  - Solution: Enabled CORS in FastAPI to allow frontend JavaScript to fetch backend data.
- **Data Normalization**:
  - Standardized date formats, amounts, and transaction categories to support accurate querying and visualization.

---

5. Results & Insights

- Users can filter transactions by category and date.
- Visual reports (bar and pie charts) help identify high-frequency transaction types.
- Highlights spending patterns, income flows, and useful insights for personal finance tracking.

---

6. Future Improvements

- Add authentication to restrict access to transaction data.
- Add anomaly detection or spending trend alerts.
- Store data in a cloud database (e.g., PostgreSQL) for scalability.
- Host the dashboard on cloud platforms like Vercel, Heroku, or Netlify for public access.

---

7. How to Run

Refer to the provided README.md file for setup, dependencies, and usage instructions. The backend and frontend can be run locally or deployed.

End of Report