MoMo Data Analysis (MDA) Project Report

Group 12: Kevin Kaneza Mbonimpaye, Yvan Muhirwa, Karabo Innoncent Pamela Ojiambo, Emmanuel Mukasa Simiyu

Assembled by: Karabo Innocent Pamela Ojiambo

June 2025

Introduction

Our team built the MoMo Data Analysis (MDA) project to help MTN MoMo users in Rwanda track their transactions. The application takes an XML file with about 1,600 SMS messages, sorts them into categories like payments or withdrawals, saves them in a SQLite database, and shows the data on a web dashboard. This report explains how we built it, what problems we faced, why we made confident choices, and what we learned. Mobile money services like MTN MoMo are critical in Rwanda, handling millions of daily transactions (Ndikubwimana, 2018).

Project Overview:

- The MDA project gives MoMo users a clear view of their spending habits. It processes SMS messages, organizes them, and stores them in a database. The dashboard lets users search, filter, and see charts of their transactions, like how much they spent each month or what types of payments they made. It is designed for anyone using MoMo who wants to understand their money flow better.

Approach

Team Organization:

- We used Google Spaces to plan and discuss the project. To keep our work organized, we set up two GitHub branches for the main project and another for the backend code, so our commits would not clash. We had different schedules but found meeting times that worked for everyone.
- Technologies Used For the backend, we used Python 3.8 to parse XML files with the xml.etree.ElementTree library categorizes messages with regex patterns (Python Software Foundation, 2025). We stored data in SQLite because it is simple and does not need a server. The frontend used HTML for the layout, CSS with Bootstrap for styling, and JavaScript with Chart.js to create charts and add features like a dark mode switch.
- Data Processing We started with the XML file (modified_sms_v2.xml) and used sms_data.py to pull out details like amounts and dates. Regex helped us sort messages into categories like "Incoming Money" or "Payments." The cleaner_script.py script fixed messy data, like inconsistent dates, and saved

it as cleaned_sms.json. Messages we could not process went into unprocessed_sms.log to help us debug.

Database Design:

- We created a SQLite database (transactions.db) with tables for transactions and entities, like senders or recipients. We added a UNIQUE constraint on message text to avoid saving the same message twice, making the database reliable and ready for future updates.

Frontend Development:

- Our dashboard, built with index.html and transactions.html, has a sidebar, navbar, and main area that adjusts to different screen sizes. Chart.js creates visuals, like pie charts showing transaction types and bar charts for monthly spending. Users can switch between light and dark modes, search for specific transactions, or filter by category.

Challenges Technical Challenges

- Frontend: Our Chart.js charts did not show up when we opened index.html directly in a browser. We fixed this using VS Code Live Server, which we learned from online tutorials.
- Backend: The SMS messages in modified_sms_v2.xml had different formats, making it hard to write a regex that worked for all of them. Some messages missed key details, like amounts, so we added checks to catch these issues and logged them in unprocessed_sms.log.

Teamwork Challenges

- Our team had conflicting schedules, but we sorted it by picking meeting times everyone could make. We worked well together, sharing ideas and solving problems as a group.

Key Decisions

- Database: We picked SQLite because it is easy to set up and works well for a small project like ours.
- Data Processing: To keep the code organized and reusable, we split the work into two scripts (sms_data.py and cleaner_script.py). JSON was used for cleaned_sms.json because it is easy to check for errors.
- Frontend Design: We went for a clean dashboard with a sidebar and navbar that's easy to use on phones or computers, plus a dark mode for better viewing.
- Code Structure: We kept the backend and frontend code separate to make the project easier to update later.

Features Implemented

- Backend: Reads XML, sorts messages with regex, cleans data into JSON, saves it in SQLite with no duplicates, and logs errors.
- Frontend: A responsive dashboard with light/dark mode, search, category filters, Chart.js charts (pie and bar), and a detailed transaction page in transactions.html.

Testing:

- We tested the backend to ensure that sms_data.py and cleaner_script.py correctly pulled and saved data. We checked that charts loaded for the frontend, the dashboard worked on different devices, and features like search and filters ran smoothly using VS Code Live Server.

Lessons Learned

- Using CSS variables made styling faster and easier to change.
- Checking data before saving it kept our database clean and error-free.
- Designing a flexible database saved us from future headaches.
- Logging errors helped us find and fix problems without losing data.

Conclusion:

- The MDA project gives MoMo users a practical way to analyze their transactions. Despite tricky SMS formats and Chart.js issues, our teamwork and clear code structure led to a working dashboard. We could improve it by adding a REST API to fetch data live or adding charts for extra insights.

References

Hipp, D. R. (2025). SQLite documentation. Retrieved from https://www.sqlite.org/docs.html

Python Software Foundation. (2025). Python 3.8 documentation. Retrieved from https://docs.python.org/3.8