MoMo Analyzer Development Report

Project Overview

I built the MoMo Analyzer for the Enterprise Web Development class final project. This web app helps users track and analyze their mobile money transactions in a simple way. This report covers how I built it, what problems I faced, and the main choices I made.

Development Approach

Technology Selection

I picked technologies that were simple and that I knew well:

- **Backend Framework**: I used Express.js because it's simple and works with JavaScript, which I already know. Even with little backend experience, I found Express easy to learn.
- **Database**: I chose SQLite because it's small and easy to set up. No complex configuration was needed.
- **Frontend**: I went with plain HTML, CSS (using Tailwind), and JavaScript instead of a big framework. This kept the code simple and made it easy to serve files directly from the `/public` folder with Express.
- **Visualization**: I used Chart.js to create graphs and charts for showing transaction data.

Development Process

I built the project in these steps:

1. **XML Parsing**: I found a good library to read XML files and extract data from them.
2. **Transaction Sorting System**: I made a system with a main function called `categorizeTransaction` that takes SMS messages and uses helper functions to create transaction records.
3. **Pattern Finding**: I spent a lot of time finding patterns in the messages and writing regex code to pull out important information.
4. **Database Setup**: I set up SQLite with a simple structure to store and get transaction data.
5. **User Interface**: I created a clean and simple interface using HTML, CSS (Tailwind), and JavaScript, and used Chart.js to show data visually.
6. **Deployment**: I set up the application on university servers using Nginx to manage web traffic.

Technical Challenges

Challenge 1: SMS Message Pattern Recognition

Problem: Transaction SMS messages followed various formats with inconsistent structures, making data extraction difficult.

Solution: I developed a modular approach with specialized helper functions for different transaction types. Each function used specific regex patterns to extract relevant information from messages. The central `categorizeTransaction` function orchestrated these helpers, building complete transaction objects regardless of message format.

Outcome: This approach successfully handled various message formats while maintaining code readability and extensibility.

Challenge 2: Edge Case Management

Problem: Continuous discovery of edge cases in SMS formats required frequent code adjustments.

Solution: I implemented an iterative development process, regularly testing and refining regex patterns to accommodate newly discovered formats. This approach allowed for incremental improvement of the parsing logic.

Outcome: The final system robustly handles a wide range of message formats with high accuracy.

Challenge 3: Deployment Configuration

Problem: The application needed to be accessible through university servers.

Solution: I researched and implemented Nginx as a reverse proxy, redirecting connections to the appropriate localhost port where the Express application runs.

Outcome: Successful deployment with minimal configuration complexity, making the application accessible to users through the university domain.

Key Decisions

Modular Code Structure

Decision: Breaking down transaction processing into discrete, focused functions rather than a monolithic solution.

Rationale: This approach improved code maintainability, made debugging easier, and allowed for systematic handling of different message formats. Each function had a clear responsibility, making the codebase easier to understand and extend.

Frontend Simplicity

Decision: Using pure HTML/CSS/JavaScript instead of a frontend framework.

Rationale: Given the application's straightforward requirements, introducing a complex frontend framework would have been unnecessary overhead. The chosen approach resulted in fast loading times and simplified development while still achieving all functional goals.

Data Visualization

Decision: Implementing Chart.js for transaction visualization.

Rationale: Chart.js provided an excellent balance between ease of implementation and powerful visualization capabilities, allowing users to better understand their transaction patterns without requiring complex development.

Server Deployment with Nginx

Decision: Using Nginx as a reverse proxy for deployment.

Rationale: This approach allowed for clean URL paths while leveraging existing infrastructure, with minimal additional configuration required.

Conclusion

Developing the MoMo Analyzer was both challenging and rewarding. By focusing on simplicity and modularity, I was able to create a functional application that effectively addresses the need for mobile money transaction analysis.

The project provided valuable learning experiences in several areas, including:

- Regex pattern development and refinement
- Modular backend architecture
- Data extraction from semi-structured text
- Data visualization implementation
- Server deployment with reverse proxy configuration

The technical approach taken—prioritizing simplicity, modularity, and focused problem-solving—resulted in a robust application that achieves its core objectives while remaining maintainable and extensible for future development.

Overall, this project demonstrates that effective solutions don't necessarily require complex technologies—thoughtful architecture and well-structured code can create powerful tools even with relatively simple technology stacks.