Vision:

Our vision for the system was to build a comprehensive and user-friendly blockchain based format that enhances transparency, efficiency, and security whilst utilizing OCaml's unique functionality and abstraction. After continuous research and testing, our original plans of utilizing a myriad of OCaml libraries were altered with a manual hands on approach to block chain implementation. With the use of databases as well as a comprehensive GUI (Raylib, Raygui) to enhance our user experience, these changes ultimately reflect an adaptive nature to our team's approach. Ultimately, since our last report, the majority of our ideas have come into fruition. We heavily utilized OCaml's type abstractions to visualize a proper modularized implementation of a blockchain transactionary system. Furthermore, we've also left opportunities for further scalability. Although our approach utilizes certain libraries such as Cryptolib, given the continuous development of OCaml encryption libraries, future generative capabilities in key-encryption philosophies can be utilized. After research, we settled on a Caesar encryption shifting public and private keys within the backend.

Summary of Progress:

Since MS2, our implementation has shared the same foundational approach whilst making changes to better suit our preferences. Since our team lost a member, certain core functionalities had to have been picked up by each member of the team. We successfully implemented a modularized blockchain, separated by a transaction and accounts module which are connected via record types. Furthermore, we provided functionality, as previously mentioned, to an encryption - key system which we implemented manually after a caesar encryption. We utilized an SHA256 hash within the Cryptokit library to hash transactions as well as blocks, and left plenty of room for future scalability. We found that certain libraries such as Mirage_crypto required too much complexity and weren't type compatible with our database implementations. The bulk of this was implemented within transaction.ml, accounts.ml, and blockchain.ml within their respective mli files as well.

Furthermore, a GUI was implemented to enhance the UI experience with libraries Raylib and Raygui. The bulk of this was implemented within menu.ml and associated accounts.ml. This was done after an arduous process of compartmentalizing work and proper communication with backend implementation to properly integrate a type safe implementation of our final deliverable. As a whole, the GUI took in the most amount of code in comparison to other individual modules. It likely embodied the most gruelling aspect of the project which we didn't expect, integrating the backend.

Activity Breakdown:

Sebastian: Implemented the bulk of the GUI within menu.ml and integrated a great deal of the backend alongside said implementation. Although total hours aren't counted this took days to complete, alongside communication with teammates.

Chris: Implemented the backend in transaction.ml and the frontend with GUI implementation in menu.ml as well as further fixes in accounts.ml. Total hours weren't counted, spanned days as well.

Ethan: Implemented the backend, focusing on transaction.ml and blockchain.mli, integrating different libraries and encryptions. Total time wasn't counted, but the efforts spanned days as well.

Productivity Analysis:

As a team we were productive in our sprints, especially towards the latter stages of the project's implementation. Integrating the GUI with the backend was an arduous task that took much longer than expected to implement. Although there were times where disagreements occurred in implementation ideas, our team powered through with further communication and understanding. Each individual member was responsible and communicative throughout the process despite setbacks and issues with proper integration. The most difficult part of this project was definitely the integration issues like previously mentioned. The GUI took much longer than expected to implement, and certain libraries were tried and tested to no avail, most notably mirage_crypto. Type compatibility issues drove us away from utilize an generate_keys function, and although it would have provided greater encryption, our more hands on approach provided a level of clarity to the concept that we greatly appreciate. Peace, love, and 3110!