**Development Process & Decisions Documentation**

### ****1. Overview of Changes Made****

This document outlines the changes made to the original useHandleSwap module during the refactoring process. The goal was to improve the code's readability, modularity, testability, and maintainability, ultimately ensuring the module is easier to work with and extend in the future.

#### **1.1 Refactoring the Code into Separate Functions**

* **Change:** The original monolithic function was split into separate functions: handleSwapFromCardanoWallet and handleSwapFromTronLinkWallet.
* **Reason:** The Cardano and Tron wallet interactions had different protocols for handling swaps. Combining both into a single function created unnecessary complexity and reduced clarity. Separating them allows for focused, wallet-specific logic, making the module more modular and easier to maintain.

#### **1.2 Implementing Centralized Error Handling**

* **Change:** Introduced a custom hook, useHandleApiError, to centralize error handling for the various API calls and wallet interactions.
* **Reason:** The original code had inconsistent error handling, with errors scattered across the logic. By centralizing the error handling, we ensure that any errors in the module are caught uniformly and handled in a predictable manner, improving the user experience by providing clear and consistent feedback.

#### **1.3 Modularizing External Interactions**

* **Change:** The wallet interactions and API calls were moved into dedicated service functions (swapCardano, swapTron, etc.).
* **Reason:** The original code directly handled these interactions within the same function. By modularizing the external services, we reduce duplication, increase testability, and allow for easier modifications in the future. For example, swapping out an API call or wallet service can now be done without impacting the core functionality of the module.

#### **1.4 Adding Unit Tests**

* **Change:** Unit tests were added using Jest to ensure the correctness of the refactored logic, particularly the core wallet interaction functions.
* **Reason:** The original code had no unit tests, which made it risky to make changes or enhancements. Adding tests ensures that the refactored module behaves as expected and that any future modifications do not introduce bugs.

### ****2. Reasoning Behind Changes****

The changes made were driven by the following goals:

* **Improving Maintainability:** By breaking the code into smaller, focused functions, each part of the module is easier to understand and maintain. For example, if a change is needed in how Cardano wallets interact, it can be done without affecting the Tron wallet logic.
* **Ensuring Scalability:** The refactored code is more modular, which makes it easier to extend in the future. New wallet types or blockchain networks can be added without disrupting existing functionality.
* **Enhancing Testability:** By modularizing the code and separating concerns (e.g., API calls, wallet interactions), unit tests can now be written for each isolated function. This makes it easier to verify that each part of the code works as expected.
* **Improving User Experience:** Centralized and consistent error handling ensures that users receive clear, actionable feedback if anything goes wrong during the wallet interactions or API calls.

### ****3. Challenges Encountered****

#### **3.1 Managing Different Wallet Interaction Protocols**

* **Challenge:** The Cardano and Tron wallets have different interaction protocols, which made it difficult to write a unified, one-size-fits-all solution.
* **Solution:** By splitting the functionality into two separate functions (handleSwapFromCardanoWallet and handleSwapFromTronLinkWallet), each function could be tailored to handle the specific logic for its respective wallet. This made the code more flexible and easier to maintain.

#### **3.2 Error Handling Across Multiple Services**

* **Challenge:** The error handling in the original code was inconsistent, with errors being handled in multiple places and in different formats.
* **Solution:** We implemented a centralized error handling mechanism using a custom hook (useHandleApiError). This approach ensures that errors are captured and handled in a uniform way across the module, improving both consistency and readability.

#### **3.3 Mocking External Services for Unit Testing**

* **Challenge:** The original module relied heavily on external services, such as wallet APIs and blockchain interactions, which made it challenging to write unit tests.
* **Solution:** We used Jest’s mocking functionality to mock external services, allowing us to write unit tests for the core logic of the module without actually calling the blockchain or wallet APIs. This isolated testing ensured that the refactored logic was working correctly in a controlled environment.

#### **3.4 Balancing Code Modularity and Complexity**

* **Challenge:** While it was necessary to modularize the code, it was also important not to over-complicate the structure with excessive abstractions. Too many separate modules or services could create unnecessary complexity.
* **Solution:** We aimed to strike a balance by modularizing the code in a way that improved readability and maintainability, without creating excessive layers of abstraction. The refactored structure is simple and focused, but flexible enough to accommodate future extensions.

### ****4. Conclusion****

The refactoring of the useHandleSwap module significantly improved the code’s modularity, readability, and maintainability. The changes made allowed for better handling of Cardano and Tron wallet interactions, centralized error handling, and the introduction of unit tests, all of which contribute to a more reliable and scalable module. While there were challenges in dealing with different wallet protocols and external services, the solutions implemented have set a solid foundation for future development and improvements.