Compare the guide you provided against the following one and let me know which one is best

```

## SUI Integration Points (End-to-End Flow & Payment Hooks)

Integrating SUI blockchain payments transforms Coin Clash from a simulation into a game with real economic stakes. This section details the end-to-end flow for handling entry fees and winner payouts using SUI, identifying the crucial connection points ("hooks") within the proposed architecture (including the API layer from Section 9).

\*\*The Goal:\*\* Players pay an entry fee in SUI to join a match via the Telegram bot. The collected fees (potentially minus a small rake/fee) are automatically paid out in SUI to the winner(s) upon match completion.

\*\*End-to-End Economic Flow:\*\*

Here’s a step-by-step breakdown of how a typical match cycle would work with SUI integration:

1. \*\*Player Initiates (Telegram):\*\* The player uses a command like `/play` or `/join\_match` in the Telegram bot.

2. \*\*Bot Contacts API (Bot -> API):\*\* The Telegram bot sends a request to the API layer (e.g., `POST /matches/initiate` or similar). This request includes the player's Telegram ID.

3. \*\*API Prepares (API Layer):\*\*

\* The API verifies the player exists in the database (using `SqlPlayerRepo`).

\* It determines the required SUI entry fee (from configuration).

\* It generates a unique identifier for this specific match entry attempt (e.g., a temporary match ID or transaction nonce).

\* \*\*Crucially, it retrieves the game's designated SUI wallet address\*\* (from configuration).

4. \*\*API Instructs Player (API -> Bot -> Player):\*\* The API sends a response back to the bot, which then messages the player: "To enter the match, please send exactly `[Entry Fee Amount]` SUI to the address `[Game Wallet Address]`. IMPORTANT: Include this unique ID in the transaction memo/note: `[Unique ID]`."

5. \*\*Player Pays (Player -> SUI Blockchain):\*\* The player uses their SUI wallet to send the specified amount to the game's address, including the unique ID in the memo/note field if the wallet supports it (this greatly helps tracking). If memos aren't reliable, alternative tracking might involve generating unique deposit addresses per match, though this is more complex.

6. \*\*API Monitors for Payment (API <-> SUI Blockchain) - HOOK 1:\*\*

\* The API layer now actively monitors the SUI blockchain for an incoming transaction to the game's wallet that matches the expected amount (`[Entry Fee Amount]`) and, ideally, the unique ID (`[Unique ID]`) provided to the player.

\* This requires the API to have access to a SUI RPC endpoint (configured) and use a SUI SDK (like `pysui`) to query recent transactions or subscribe to events for the game's wallet address.

7. \*\*API Confirms Payment & Starts Match (API Layer -> Engine):\*\*

\* Once the API confirms the correct SUI payment on-chain (\*\*Hook 1 Success\*\*), it marks the player as paid for this match attempt.

\* It can now potentially group players if it's a multi-player match or proceed directly.

\* It calls the `MatchEngine` (`core/engine.py`) via its own internal logic or another API call (e.g., `POST /matches/start`) to begin the actual game simulation, passing the participating player/character details.

8. \*\*Engine Simulates Match (Engine):\*\* The `MatchEngine` runs the simulation as described in Section 7, independent of the blockchain at this stage.

9. \*\*Engine Reports Winner (Engine -> API Layer):\*\* The engine finishes and returns the result (winner ID(s), potentially prize pool info) to the API layer.

10. \*\*API Prepares Payout (API Layer):\*\*

\* The API receives the winner information.

\* It calculates the total payout amount (e.g., sum of entry fees minus rake, or a fixed prize from config).

\* \*\*It retrieves the winner's SUI wallet address\*\* from the database (using `SqlPlayerRepo`). This requires the address to have been collected previously. \*\*HOOK 2: Player SUI Address Storage/Retrieval.\*\*

11. \*\*API Executes Payout (API Layer -> SUI Blockchain) - HOOK 3:\*\*

\* Using the SUI SDK and the game wallet's securely stored private key/seed phrase, the API constructs, signs, and submits a transaction on the SUI blockchain to send the calculated payout amount from the game's wallet to the winner's SUI address.

12. \*\*API Monitors Payout Tx (API <-> SUI Blockchain) - HOOK 4:\*\*

\* The API monitors the status of its \*outgoing\* payout transaction on the SUI blockchain to confirm it was successfully processed.

13. \*\*API Finalizes & Notifies (API -> Bot -> Player):\*\*

\* Once the payout transaction is confirmed (\*\*Hook 4 Success\*\*), the API can optionally update internal records (e.g., a transaction log table in the database).

\* It sends a final status update to the bot, which notifies the winner (and potentially losers) about the match outcome and the successful payout.

\*\*Detailed Payment Hooks & Implementation Points:\*\*

\* \*\*HOOK 1: Entry Fee Payment Confirmation:\*\*

\* \*\*Location:\*\* Primarily within the API layer.

\* \*\*Mechanism:\*\* Requires code using a SUI SDK (e.g., `pysui`) to connect to a SUI RPC node (URL from config). Needs functions like `get\_transactions\_for\_address` or similar, filtering by recipient (game wallet), amount, and ideally checking memo/metadata for the unique ID. This might involve periodic polling or using websocket subscriptions if the RPC/SDK supports them.

\* \*\*Database:\*\* May need a temporary table to track pending entries (Player ID, Unique ID, Expected Amount, Status: Pending/Paid/Expired).

\* \*\*HOOK 2: Player SUI Address Management:\*\*

\* \*\*Location:\*\* `core/models.py` (add `sui\_address: str` field to `Player` model), `core/repositories.py` (update `SqlPlayerRepo` to handle saving/retrieving the address), and the API/Bot layer (for the user interface to submit/update their address).

\* \*\*Mechanism:\*\* Simple database storage and retrieval. Need validation to ensure the address format is correct. The bot needs commands like `/register\_wallet <sui\_address>`.

\* \*\*HOOK 3: Winner Payout Execution:\*\*

\* \*\*Location:\*\* Primarily within the API layer.

\* \*\*Mechanism:\*\* Requires the SUI SDK. Needs access to the game wallet's private key/seed phrase (stored \*very\* securely, e.g., environment variable, secrets manager, \*never\* in code or config files). Code to build a SUI transfer transaction (specifying recipient from Hook 2, amount), sign it with the game wallet's key, and submit it via the RPC node.

\* \*\*HOOK 4: Payout Transaction Confirmation:\*\*

\* \*\*Location:\*\* Primarily within the API layer.

\* \*\*Mechanism:\*\* After submitting the payout transaction (Hook 3), the API gets a transaction digest (hash). It uses the SUI SDK and RPC node to query the status of this specific transaction digest until it's confirmed (or fails).

\*\*Handling Failures & Transactionality:\*\*

This is critical for real money:

\* \*\*Payment Timeout:\*\* If a player doesn't pay within a certain time after instruction, the API should expire the entry attempt.

\* \*\*Payment Confirmation Failure:\*\* If the API can't confirm payment (e.g., player sent wrong amount, network issue), the match doesn't start for that player.

\* \*\*Engine Crash:\*\* If payment is confirmed but the engine fails mid-match, you need a policy. Refund entry fee? Requires tracking payment status robustly.

\* \*\*Payout Failure:\*\* If the engine finishes but the SUI payout transaction fails (e.g., network congestion, insufficient gas in game wallet), the API \*must\* detect this (Hook 4). It should implement a retry mechanism. If retries fail, log the error clearly for manual intervention (manual payout). The internal database should \*not\* reflect a successful payout until Hook 4 confirms it on-chain.

\* \*\*Ledger:\*\* Consider an internal database table `transactions` logging every SUI-related action (Attempted Entry, Confirmed Entry, Attempted Payout, Confirmed Payout) with player IDs, amounts, SUI transaction digests, and timestamps. This helps reconcile internal state with blockchain state.

\*\*Configuration Needs:\*\*

\* `SUI\_NETWORK` (e.g., 'devnet', 'testnet', 'mainnet')

\* `SUI\_RPC\_URL`

\* `GAME\_WALLET\_ADDRESS`

\* `GAME\_WALLET\_SECRET` (Handled via secure environment variable/vault)

\* `ENTRY\_FEE\_SUI`

\* `PAYOUT\_RAKE\_PERCENT` (Optional, if taking a fee)

Integrating SUI requires careful orchestration between your API layer, the SUI blockchain (via an SDK), and secure wallet management. Each hook represents a critical interaction point that must be implemented robustly and handle potential failures gracefully.  
```