**README for the 1000 Project Middleware**

**Overview**

The middleware for the 1000 Project is a modular system designed to streamline off-chain computations, optimize cost efficiency, and support the reward and burn logic. This document outlines the updated workflow, individual script functionalities, and setup instructions.

**Updated Workflow**

1. **Data Collection (Off-Chain)**
   * **Script**: fetch\_wallet\_data.py
   * Fetch incremental wallet data updates from the blockchain.
   * Store changes in the middleware database for further processing.
2. **Filter Eligible Wallets (Off-Chain)**
   * **Script**: filter\_eligible\_wallets.py
   * Exclude wallets that:
     + Hold less than the minimum required balance.
     + Are flagged for cooldown periods or suspected behavior.
     + Are contract or blacklisted wallets.
3. **Score Eligible Wallets (Off-Chain)**
   * **Script**: score\_wallets.py
   * Assign weighted scores based on criteria:
     + Holding duration.
     + Balance stability.
     + Other dynamic metrics.
   * Store wallet scores in the middleware database.
4. **Select Wallets Using Chainlink VRF (On-Chain)**
   * **Script**: call\_chainlink\_vrf.py
   * Randomly select 10% of eligible wallets (max 1,000).
   * Use Chainlink VRF for randomness verification.
5. **Execute Daily Operations (On-Chain)**
   * **Script**: export\_selected\_wallets.py
   * On Reward Days:
     + Distribute 1% of the reward wallet to selected wallets.
     + Log transactions for transparency.
   * On Burn Days:
     + Execute token burns (1% of the reward wallet).
6. **Post-Execution Updates (Off-Chain)**
   * Update middleware database to reflect new balances and cooldown flags.
   * Notify holders via Telegram/Discord bots of rewards, cooldowns, or updates.

**Script Descriptions**

1. **fetch\_wallet\_data.py**
   * Purpose: Fetch wallet data from the blockchain with incremental updates.
   * Key Features: Reduces overhead by only updating modified or new wallet data.
2. **filter\_eligible\_wallets.py**
   * Purpose: Apply eligibility filters to wallet data.
   * Filters: Minimum balance, cooldown flags, blacklist exclusion.
3. **score\_wallets.py**
   * Purpose: Assign scores to eligible wallets based on weighted criteria.
   * Scoring Logic: Holding duration, balance stability, dynamic metrics.
4. **call\_chainlink\_vrf.py**
   * Purpose: Use Chainlink VRF to randomly select wallets.
   * Output: List of randomly selected wallets.
5. **export\_selected\_wallets.py**
   * Purpose: Distribute rewards or execute burns based on daily operations.
   * Includes: Logging transactions on-chain.
6. **setupanddependencies.py**
   * Purpose: Install required Python libraries and set up dependencies.
7. **mainfunction.py**
   * Purpose: Orchestrates all middleware scripts for seamless workflow execution.

**Setup Instructions**

1. **Clone the Repository**:
2. git clone <repository-url>

cd <repository-folder>

1. **Install Dependencies**: Run the setupanddependencies.py script to install required Python libraries.

python3 setupanddependencies.py

1. **Run the Middleware**: Execute the main function to trigger the workflow.

python3 mainfunction.py

1. **Configure Settings**:
   * Update blockchain URLs, APIs, and wallet data settings in the respective scripts.
   * Modify scoring weights and filters in score\_wallets.py and filter\_eligible\_wallets.py as needed.

**Changelog**

**Latest Updates**

1. **Incremental Data Updates**:
   * Optimized fetch\_wallet\_data.py for change-detection logic.
2. **Streamlined Filtering**:
   * Combined multiple checks into filter\_eligible\_wallets.py.
3. **Weighted Scoring Mechanism**:
   * Added score\_wallets.py for scoring eligible wallets.
4. **Cost Optimization**:
   * Prioritized off-chain computations to reduce on-chain gas costs.

**Next Steps**

* **Testing**: Validate individual script functionality and full workflow integration.
* **Documentation**: Keep this README updated with further changes.
* **Feedback**: Collaborate with the dev team to refine scripts and processes.