# \*\*1000 Project Middleware README\*\*

## \*\*Overview\*\*

The 1000 Project middleware is the backbone of our off-chain operations, enabling efficient data processing, filtering, and interaction with blockchain components. This modular framework minimizes gas fees and computational overhead, supporting hybrid on/off-chain functionality to optimize the reward and burn mechanisms.

---

## \*\*Features\*\*

- \*\*Snapshot Optimization\*\*: Incremental updates minimize the need for full daily snapshots, improving data efficiency.

- \*\*Eligibility Filtering\*\*: Dynamic logic for wallet qualification based on balance, cooldown status, and blacklist criteria.

- \*\*Change-Detection Logic\*\*: Detects and processes only altered data for streamlined performance.

- \*\*Weighted Scoring Mechanism\*\*: Ensures fairness by dynamically scoring wallets based on holding behavior and transaction history.

- \*\*Integration with Chainlink VRF\*\*: Ensures randomness in wallet selection for rewards.

---

## \*\*Setup Instructions\*\*

### \*\*1. Clone Repository\*\*

```bash

git clone https://github.com/1000Project/middleware.git

cd middleware

```

### \*\*2. Install Dependencies\*\*

Run the setup script to install all required Python libraries:

```bash

python3 setupanddependencies.py

```

### \*\*3. Configuration\*\*

Modify the `config.json` file to include:

- Blockchain API endpoints

- Chainlink VRF keys

- Database credentials

### \*\*4. Execute Scripts\*\*

- \*\*Fetch Wallet Data\*\*:

```bash

python3 fetch\_wallet\_data.py

```

- \*\*Filter Eligible Wallets\*\*:

```bash

python3 filter\_eligible\_wallets.py

```

- \*\*Detect P2P Transfers\*\*:

```bash

python3 detect\_p2p\_transfers.py

```

- \*\*Export Selected Wallets\*\*:

```bash

python3 export\_selected\_wallets.py

```

- \*\*Call Chainlink VRF\*\*:

```bash

python3 call\_chainlink\_vrf.py

```

- \*\*Run Main Functionality\*\*:

```bash

python3 mainfunction.py

```

- \*\*Run Weighted Scoring\*\*:

```bash

python3 score\_wallets.py

```

---

## \*\*Workflow Overview\*\*

### \*\*Daily Process\*\*

1. \*\*Incremental Data Retrieval\*\*:

- Fetch new or updated wallet data only.

- Store metadata in the middleware database.

2. \*\*Eligibility Filtering\*\*:

- Check wallets against criteria:

- Minimum balance requirement.

- Cooldown flags.

- Blacklist exclusions.

- Apply weighted scoring to refine wallet eligibility.

3. \*\*Random Wallet Selection\*\*:

- Use Chainlink VRF for secure, unbiased randomness.

4. \*\*Execute Reward/Burn Logic\*\*:

- Distribute rewards or burn tokens as per the daily cycle.

5. \*\*Log Transactions\*\*:

- Record all operations in the middleware database and on-chain where applicable.

---

## \*\*Key Changes and Updates\*\*

- \*\*Incremental Updates\*\*: Reduced computational load by processing only changed data.

- \*\*Enhanced Filtering Logic\*\*: Added weighted scoring and P2P transfer anomaly detection.

- \*\*Optimized Workflow\*\*: Consolidated steps for improved efficiency and cost savings.

- \*\*New Script Additions\*\*: Introduced `score\_wallets.py` for weighted scoring.

---

## \*\*Directory Structure\*\*

```

middleware/

├── config.json # Configuration file

├── scripts\_and\_executables/

│ ├── setupanddependencies.py

│ ├── fetch\_wallet\_data.py

│ ├── fetch\_wallet\_data.ver2.py

│ ├── filter\_eligible\_wallets.py

│ ├── filter\_eligible\_wallets.ver2.py

│ ├── detect\_p2p\_transfers.py

│ ├── detect\_p2p\_transfers.ver2.py

│ ├── export\_selected\_wallets.py

│ ├── call\_chainlink\_vrf.py

│ ├── mainfunction.py

│ ├── mainfunction.ver2.py

│ ├── score\_wallets.py

├── logs/ # Logs for debugging

├── database/ # Local database for snapshots

├── README.md # Documentation

```

---

## \*\*Testing and Debugging\*\*

### \*\*1. Unit Tests\*\*

Run unit tests to validate individual scripts:

```bash

python3 -m unittest discover tests/

```

### \*\*2. Debug Logs\*\*

Check the `logs/` directory for detailed error messages and execution summaries.

---

## \*\*Next Steps\*\*

1. \*\*Middleware Optimization\*\*: Further streamline incremental updates.

2. \*\*Integration Testing\*\*: Ensure seamless interaction between scripts.

3. \*\*AI Logic Integration\*\*: Incorporate AI-driven enhancements for scoring and decision-making.

---

## \*\*Contact\*\*

For support or contributions, reach out to the project team at:

- Email: 1000cryptoai@gmail.com

- Telegram: https://t.me/The1000Project

- X (formerly Twitter): [@1000CryptoAI](https://x.com/1000CryptoAI)