

Monitoring Solana Wallet (Part 2)

Dalam ekosistem kripto, memantau aktivitas wallet tertentu, seperti wallet whale atau untuk copy trading, adalah hal yang umum dilakukan. Dengan menggunakan RPC dari QuickNode, kita bisa mengakses data transaksi detail seperti perubahan saldo sebuah wallet dan **bikin sistem wallet monitoring dan copy trading bot sendiri**. Di sini kita akan fokus belajar untuk chain Solana, jadi sebelum masuk ke code, akan dijelaskan beberapa konsep penting.

Contents

Prerequisites	3
Concepts	3
System Program	3
Solana's Account Model and Token Accounts.....	3
Mint Account.....	3
Associated Token Account (ATA)	3
Raydium Liquidity Pools (LP)	3
Coding Guide.....	4

Prerequisites

- Pemahaman tentang smart contract dan blockchain dasar
- Pemahaman tentang Node JS
- Pemahaman cara kerja Liquidity Pool di Solana
- Pemahaman "Monitoring Solana Wallett (Part 1)"

Concepts

Sifat blockchain Solana berbeda dengan EVM. Salah satu yang menonjol adalah Solana perlu ada rent deposits untuk setiap account di chainnya agar bisa digunakan. Maka biasanya, kita ngga bisa narik seluruh Solana dari account kecuali di burn. Jumlahnya kecil, paling 0.002 SOL per akun, tapi kalau punya banyak cukup buat makan. 😊

System Program

System Program di Solana adalah program inti yang digunakan untuk basic operation seperti transfer SOL, account creation, dan mengurus rent fee. Alamat dari System Program di Solana adalah:

11111111111111111111111111111111

Banyak transaksi di Solana berinteraksi dengan alamat ini, tetapi biasanya tidak relevan saat kita memantau token spesifik seperti SPL tokens dalam Liquidity Pool (LP).

Solana's Account Model and Token Accounts

Di Solana, **wallet adalah account milik System Program** dan sistem kepemilikan tokennya berbeda dari blockchain berbasis EVM seperti Ethereum. Di EVM-based chain, semua token terhubung ke 1 account address, dan saldo token dicek menggunakan fungsi **balanceOf** dari token contract. Sementara itu, Solana menggunakan **subaccount** terpisah yang disebut **token accounts** untuk setiap jenis token yang dimiliki oleh sebuah main account atau **pubkey**. Jadi, jika sebuah wallet di Solana memiliki beberapa jenis token, akan ada **subaccount terpisah** untuk masing-masing token tersebut. Konsep ini memungkinkan pengelolaan saldo yang lebih fleksibel dan terpisah.

Mint Account

Mint adalah istilah yang digunakan di Solana untuk merujuk pada token jenis tertentu atau contract yang bertanggung jawab untuk mengelola creation dan distribution token tersebut. Setiap token di Solana memiliki Mint yang unik, yang mengelola atribut seperti **token supply, decimals, and account roles (mint-authority dan freeze-authority)**.

Associated Token Account (ATA)

Associated Token Account (ATA) adalah sebuah account khusus di Solana yang digunakan untuk menyimpan token tertentu seperti **token accounts**. Cara kerjanya sedikit berbeda dimana dengan token accounts, penerima perlu punya token account untuk bisa menerima sebuah token, sedangkan di ATA, hal ini tidak diperlukan karena address ATA ditentukan otomatis dari mint dan owner account sehingga 1 owner hanya ada 1 ATA khusus 1 token. Beda dari token account yang bisa saja ada lebih dari 1 untuk 1 token.

Raydium Liquidity Pools (LP)

Solana tidak ada single standard LP seperti EVM yang mostly fork dari Uniswap. Jenis LP di Solana ada banyak tapi kita akan meng-cover 2 yang paling umum dari Raydium.

- a. Concentrated Liquidity Market Maker(CLMM)

LP contract punya 2 token account untuk pair-nya. Umumnya WSOL (wrapped SOL) dan untuk mint account. Owner contract ini adalah:

CAMMCzo5YL8w4VFF8KVHrK22GGUsp5VTaW7grrKgrWqK

Jadi, jika ada transaksi yang melibatkan contract yang owner-nya address di atas, maka itu adalah sebuah LP. Biasa nama contract LP-nya menggunakan format “Raydium (PAIR) Market”

Overview		More info	
SOL Balance	0.011637 SOL (\$1.7641)	Public name	Raydium (SOL-BILLY) Market
Token Balance		Owner	Raydium Concentrated Liquidity 🔗
 WSOL	540.03 (\$82,171.68)	isOnCurve ⓘ	False
 BILLY	9,863,133.49 (\$524,235.4)	Tags	raydium 2+

- b. ATA owned by Raydium Authority (currently V4 or version 4)

Jadi address Raydium Authority ini (5Q544fKrFoe6tsEbD7S8EmxGTJYAKtTVhAW5Q5pge4j1) memiliki banyak ATA untuk setiap token LP. Kalau di lihat portfolio-nya, namanya mengandung kata “Pool”.

Account	Token	Token Balance
Raydium (MAGAPEPE-SOL) Pool 1 🔗	 MAGAPEPE	3,550,900,624,925,346
Raydium (SOLPAC-SOL) Pool 1 🔗	 SOLPAC	6,253,784,891,048,510
Raydium (CHWY-SOL) Pool 1 🔗	 CHWY	4,761,334,899,754,892

Karena LP menyediakan likuiditas, cara kerjanya, jika token di beli dari LP dengan SOL, maka SOL di LP bertambah dan token di LP berkurang, berlaku sebaliknya.

Feel free to try it yourself before reading the guide.

Coding Guide

Melanjutkan dari part 1, di sini kita akan selesaikan logging transactions dan execute trade.

1. Helper functions

Pertama-tama, Pertama, buat beberapa **helper functions** untuk memudahkan interaksi dengan blockchain Solana. Berikut module yang diperlukan dan functions-nya.

- Import modules

```
import { programs } from '@metaplex/js';
const { Metadata } = programs.metadata;
import * as solanaWeb3 from '@solana/web3.js';
```

- Environment variables

Declare beberapa variable yang diambil dari file **.env**. File ini wajib masuk **.gitignore** kalau mau di-publish ke Git. Berikut panduan nilai di **.env**:

- SLIPPAGE adalah batas toleransi perubahan harga saat melakukan swap. Nilai ini diatur dalam basis points (bps), di mana 100 bps = 1%. Misalnya, nilai 50 berarti kamu mengizinkan slippage maksimal 0.5%.

- `PRIVATE_KEY` adalah private key dari wallet-mu untuk sign transaction.

```
import dotenv from 'dotenv';
dotenv.config();

const SLIPPAGE = parseInt(process.env.SLIPPAGE) || 50;
const myWallet = new Wallet(solanaWeb3.Keypair.fromSecretKey(bs58.decode(process.env.PRIVATE_KEY
|| '')));
```

- **getTokenMetadata:** Mengambil metadata token seperti token name berdasarkan mint address. Di Solana, metadata token disimpan di **metadata account** yang merupakan PDA (Program Derived Address), simpelnya, ini auto-generated. Maka pertama kita perlu `getPDA` untuk bisa `load` metadata-nya. Detail struktur bisa cek dokumentasi Metaplex.

```
async function getTokenMetadata(connection, mint) {
  try {
    const metadataPDA = await Metadata.getPDA(mint);
    const metadata = await Metadata.load(connection, metadataPDA);
    return metadata.data.data;
  } catch (error) {
    console.error(`Error fetching metadata for mint ${mint}: `, error);
    return 'Unknown Token';
  }
}
```

- **getTokenOwner:** Mengambil account owner dengan lewat `connection.getAccountInfo()`. Terus, hasilnya dikonversi ke format Base58 pakai `toBase58()`, biar account-nya jadi readable address. Kalau akun nggak ditemukan, fungsi ini bakal balikin empty string.

```
async function getTokenOwner(connection, account) {
  const accountInfo = await connection.getAccountInfo(new solanaWeb3.PublicKey(account));
  return (accountInfo) ? accountInfo.owner.toBase58() : "";
}
```

- **swapTokens:** Menukar satu token dengan token lain secara otomatis via Jupiter Aggregator API. Pertama, akses endpoint quote (<https://quote-api.jup.ag/v6/quote>) untuk mendapat harga terbaik berdasarkan input/output token, nominal beli dalam input token, dan slippage. Setelah itu, kirim request ke endpoint swap (<https://quote-api.jup.ag/v6/swap>) untuk dapatkan object transaction yang siap dieksekusi. Transaksi ini kemudian di-deserialized, di-sign, dan dikirim ke blockchain. Terakhir, fungsi ini nunggu transaction confirmation dan kasih tahu status-nya.

```
async function swapTokens(connection, inputMint, inputAmount, outputMint, priorityFee) {

  console.log("Swapping from " + inputMint + " to " + outputMint);
  try {
    const response = await axios.get('https://quote-api.jup.ag/v6/quote', {
      params: {
        inputMint: inputMint,
        outputMint: outputMint,
        amount: inputAmount,
        slippageBps: SLIPPAGE
      }
    });
    const quoteResponse = response.data;
    // console.log(quoteResponse);

    // get serialized transactions for the swap
  }
}
```

```

// Get swap transaction
const swapResponse = await axios.post('https://quote-api.jup.ag/v6/swap', {
    quoteResponse,
    userPublicKey: myWallet.publicKey.toString(),
    wrapAndUnwrapSol: true,
    dynamicComputeUnitLimit: true,
    prioritizationFee: priorityFee, // prioritization fee in lamports
}, {
    headers: {
        'Content-Type': 'application/json'
    }
});

// Destructure to get the swapTransaction
const { swapTransaction } = swapResponse.data;
// console.log(swapTransaction);

// deserialize the transaction
const swapTransactionBuf = Buffer.from(swapTransaction, 'base64');
var transaction = solanaWeb3.VersionedTransaction.deserialize(swapTransactionBuf);
// sign the transaction
transaction.sign([myWallet.payer]);

// get the latest block hash
const latestBlockHash = await connection.getLatestBlockhash();

// Execute the transaction
const rawTransaction = transaction.serialize()
const txid = await connection.sendRawTransaction(rawTransaction, {
    skipPreflight: true,
    maxRetries: 2
});

console.log("Transaction signed, processing tx " + txid);

await connection.confirmTransaction({
    blockhash: latestBlockHash.blockhash,
    lastValidBlockHeight: latestBlockHash.lastValidBlockHeight,
    signature: txid
});
console.log(`Transaction Successful. View on Solscan: https://solscan.io/tx/${txid}`);
console.log();
return true;
}
catch (error) {
    console.log('Error swapping tokens:', error.message);
    return false;
}
}
}

```

Helper function ini akan kita masukkan ke file baru, sebut saja **helper.js**. File ini jadi semacam library, jadi jangan lupa export function-nya.

```
export { getTokenMetadata, getTokenOwner, swapTokens };
```

2. Menentukan BUY/SELL

Sebelumnya kita sudah sampai melihat pre-balance, post-balance, dan metadata token yang terlibat dalam transaksi. Dari situ, kita bisa menentukan jenis transaksi dan jumlah token yang ditransaksikan. Sebelum lanjut, import dulu helper functions.

```
import { getTokenMetadata, getTokenOwner, swapTokens } from './helper.js';
```

Code di bawah ngecek jenis transaksi **buy** atau **sell**, memastikan ada interaksi ke LP, dan menghitung jumlah token yang terlibat. Pertama, `preBalance` dan `postBalance` dicek buat melihat perubahan saldo token. Terus, dicek apakah tokennya dari LP dengan `isLP`, di sini kita fokus ke LP Raydium (khusus Raydium Authority V4), dicek owner dari `actualOwner` cek penjelasan di [Concepts](#)). Kita menggunakan `tokenChanged` dan `solChanged`, untuk menentukan transaksi **buy** atau **sell**. `tokenName` dan `tokenChanged` merupakan info token dari metadata-nya.

```
const accountKeys = txnDetails[i].transaction.message.accountKeys;
let lpCount = 0;
let tokenName, tokenPubkey;
let tokenChanged = 0, solChanged = 0;
let isBuy = false;

for (let j = 0; j < accountKeys.length; j++) {
    const accountKey = accountKeys[j].pubkey;
    let preAmount = 0, postAmount = 0;

    if (accountKey !== solanaWeb3.SystemProgram.programId) {
        const preBalance = meta.preTokenBalances.find(a => a.accountIndex === j);
        const postBalance = meta.postTokenBalances.find(a => a.accountIndex === j);

        if (preBalance || postBalance) {
            const balance = postBalance ? postBalance : preBalance;
            const owner = balance.owner;
            const actualOwner = await getTokenOwner(connection, owner);
            const tokenMetadata = await getTokenMetadata(connection, new
solanaWeb3.PublicKey(balance.mint));

            const isLP = RAYDIUM_LP OWNERS.includes(actualOwner) ||
RAYDIUM_LP OWNERS.includes(owner);
            if (isLP) {
                lpCount++;
                if (preBalance) {
                    preAmount = parseFloat(preBalance.uiTokenAmount.uiAmountString).toFixed(5);
                    // console.log(`PRE: ${preAmount} ${tokenMetadata.name}`);
                }
                if (postBalance) {
                    postAmount =
parseFloat(postBalance.uiTokenAmount.uiAmountString).toFixed(5);
                    // console.log(`POST: ${postAmount} ${tokenMetadata.name}`);
                }
                if (balance.mint == WRAPPED_SOL_MINT) {
                    solChanged = parseFloat(postAmount) - parseFloat(preAmount);
                    isBuy = solChanged > 0 ? true : false;
                } else {
                    tokenChanged = parseFloat(postAmount) - parseFloat(preAmount);
                    tokenName = tokenMetadata.name;
                    tokenPubkey = new solanaWeb3.PublicKey(balance.mint);
                    isBuy = tokenChanged > 0 ? false : true;
                }
            }
        }
    }
}
```

```

        }
    }
}

```

3. Log the Transaction

We got all the information we need, so let's log it. Jika `lpCount > 0`, artinya ini transaksi swap. Kemudian, variabel `action` digunakan untuk menentukan apakah transaksi tersebut merupakan **BUY** atau **SELL**, berdasarkan nilai `isBuy`. Jika `isBuy` bernilai `true`, transaksi dianggap sebagai **BUY**, dan sebaliknya. Lalu jumlah perubahan token dikasih `Math.abs` biar muncul nilai absolutnya semisal nilainya negatif.

```

if (lpCount > 0) {
    let action = isBuy ? "BUY" : "SELL";
    console.log(${action} ${Math.abs(tokenChanged)} ${tokenName} FOR ${Math.abs(solChanged)}
    SOL);
}

```

4. Execute the swap

Pertama kita cek balance SOL dan token. Jika transaksi adalah **BUY** (`isBuy === true`), maka jika saldo SOL lebih besar dari nilai yang ditentukan di `SOL_BUY_AMT`, `swapTokens` akan dieksekusi untuk membeli token. Sebaliknya jika transaksi adalah **SELL** (`isBuy === false`), maka cek apakah saldo token lebih besar dari 0. Jika saldo token tersedia, maka `swapTokens` dieksekusi untuk jual semua token.

```

if (lpCount > 0) {
    let action = isBuy ? "BUY" : "SELL";
    console.log(${action} ${Math.abs(tokenChanged)} ${tokenName} FOR ${Math.abs(solChanged)}
    SOL);
    const myBalance = await connection.getBalance(myWallet.publicKey);
    const tokenBalance = await
connection.getTokenAccountsByOwner(tokenPubkey)(myWallet.publicKey, { mint: (tokenPubkey) }); // 
get token account of the token to be swapped
    if (isBuy) {
        if (myBalance > SOL_BUY_AMT)
            swapTokens(connection, WRAPPED_SOL_MINT, SOL_BUY_AMT, tokenPubkey, "priority_fee");
    }
    else if (!isBuy) {
        if (tokenBalance > 0)
            swapTokens(connection, tokenPubkey, tokenBalance, WRAPPED_SOL_MINT, "priority_fee");
    }
}

```

Selesai sampai eksekusi transaksi, code bisa di enhance untuk LP lain dan jika mau memfilter kriteria token yang mau kita ikut copytrade, misal dari marketcap atau tanggal token launch. Bisa juga mengatur jumlah sell. Full code di [repository ini](#).