File Overview

src/index.ts

The `index.ts` file serves as the entry point for the application. It establishes a WebSocket connection to listen for events on the Solana blockchain, specifically for new liquidity pools. The main functionalities include:

- **WebSocket Connection**: It creates a WebSocket connection to the Helius API and subscribes to logs related to the liquidity pool program.
- **Message Handling**: It processes incoming messages from the WebSocket, checking for specific logs that indicate a new liquidity pool has been created.
- **Transaction Handling**: Upon detecting a new liquidity pool, it fetches transaction details, performs a rug check on the token, and initiates a swap transaction if the checks pass.
- **Error Handling**: It includes error handling for JSON parsing and WebSocket errors, ensuring the application can recover from issues gracefully.

src/transactions.ts

The `transactions.ts` file contains functions that handle various transaction-related operations, primarily interacting with the Helius and Jupiter APIs. Key functionalities include:

- **Fetching Transaction Details**: The `fetchTransactionDetails` function retrieves transaction details from the Helius API using a transaction signature. It processes the response to extract relevant token mint information.
- **Creating Swap Transactions**: The `createSwapTransaction` function generates a swap transaction using the Jupiter API, allowing users to swap SOL for a specified token. It handles retries for specific errors and serializes the transaction for submission.
- **Rug Check**: The `getRugCheckConfirmed` function checks if a token is a potential rug pull by querying the RugCheck API and evaluating various risk factors.
- **Saving Swap Details**: The `fetchAndSaveSwapDetails` function fetches additional details about a swap transaction and saves the relevant information to a database.
- **Creating Sell Transactions**: The `createSellTransaction` function allows users to sell a specified token for SOL, following a similar process to the swap transaction creation.

src/types.ts

The `types.ts` file defines TypeScript interfaces and types used throughout the application to ensure type safety and clarity. Key components include:

- **DisplayDataItem**: Represents the structure of data related to token mints.
- **QuoteResponse**: Defines the expected structure of the response from the Jupiter API when requesting a swap quote.
- **SerializedQuoteResponse**: Represents the structure of a serialized swap transaction response.
- **RugResponse**: Defines the structure of the response from the RugCheck API, detailing risks associated with a token.

- **TransactionDetailsResponse**: Describes the structure of transaction details returned from the Helius API, including information about fees, instructions, and events.
- **SwapEventDetailsResponse**: Represents the details of a swap event, including program information and token inputs/outputs.
- **HoldingRecord**: Defines the structure for holding records in the database, capturing details about token transactions.

These interfaces and types help maintain consistency and reduce errors when working with data throughout the application.

`src/tracker/index.ts`

The `index.ts` file is responsible for tracking the holdings of a cryptocurrency wallet, specifically focusing on the performance of tokens held in the wallet. It periodically checks the current prices of these tokens, calculates unrealized profit and loss (PnL), and automatically sells tokens based on predefined take profit and stop loss percentages.

Key Functionalities

1. **Environment Configuration**:
    - Loads environment variables from a `.env` file using the `dotenv` package to access configuration parameters.
2. **Database Connection**:
    - Connects to a SQLite database to manage and store token holdings.
    - Creates a table for holdings if it does not already exist.
3. **Price Fetching**:
    - Retrieves the current prices of tokens from the Jupiter API.
    - Handles multiple tokens by constructing a query string from the token IDs stored in the database.
4. **PnL Calculation**:
    - For each token held, calculates the unrealized PnL in both USD and percentage terms.
    - Displays a visual indicator (green or red) based on whether the PnL is positive or negative.
5. **Automatic Selling**:
    - Checks if the unrealized PnL exceeds the configured take profit percentage or falls below the stop loss percentage.
    - If conditions are met, it triggers a sell transaction using the `createSellTransaction` function from the `transactions.ts` file.
6. **Logging**:
    - Logs the current status of each holding, including the amount held, the price paid, and the current unrealized PnL.

- Outputs the logs to the console for easy monitoring.

7. **Periodic Execution**:
   - The `main` function is set to run every 5 seconds, allowing for continuous tracking of holdings and price updates.

Usage Instructions

1. **Setup Environment**:
   - Ensure that the `.env` file is properly configured with the necessary API keys and database names.
2. **Database Initialization**:
   - The script will automatically create the necessary database and table for holdings if they do not exist.
3. **Run the Tracker**:
   - Execute the script using Node.js:

```
node src/tracker/index.ts
```

4. **Monitor Output**:
   - The console will display the current status of your token holdings, including any actions taken (e.g., selling tokens).
5. **Adjust Configuration**:
   - Modify the `config` file to set your desired take profit and stop loss percentages, as well as any other parameters relevant to your trading strategy.
6. **Stop the Tracker**:
   - To stop the tracker, simply terminate the Node.js process (e.g., by pressing `Ctrl + C` in the terminal).

Important Notes

- Ensure that the `createSellTransaction` function in `transactions.ts` is correctly implemented to handle sell transactions.
- The script assumes that the tokens are already present in the database. Ensure that the database is populated with holdings before running the tracker.
- The script will continuously run and check for updates every 5 seconds. Adjust the timing as necessary based on your requirements.