**Crypto-Wallet-API 프로그램 명세서**

목 차

[I. 시스템 구성 2](#_Toc1596902)

[II. 유의 사항 2](#_Toc1596903)

[III. 지갑 동기화 프로세스 3](#_Toc1596904)

[IV. 개발환경 3](#_Toc1596905)

[V. 초기 데이터베이스 구성 4](#_Toc1596906)

[VI. 주요 설정파일 4](#_Toc1596907)

[1. application.yml 4](#_Toc1596908)

[VII. 신규 코인 추가 시 절차 5](#_Toc1596909)

[1. 새로 추가하는 암호화폐의 Service를 추가하는 방법 5](#_Toc1596910)

[VIII. Rest API (WalletApiController.java) 6](#_Toc1596911)

[1. 새 주소 발급: /address 6](#_Toc1596912)

[2. 주소 검증: /validate 6](#_Toc1596913)

[3. 송금: /send 6](#_Toc1596914)

[4. 지정 주소에서 출금: /sendfrom 7](#_Toc1596915)

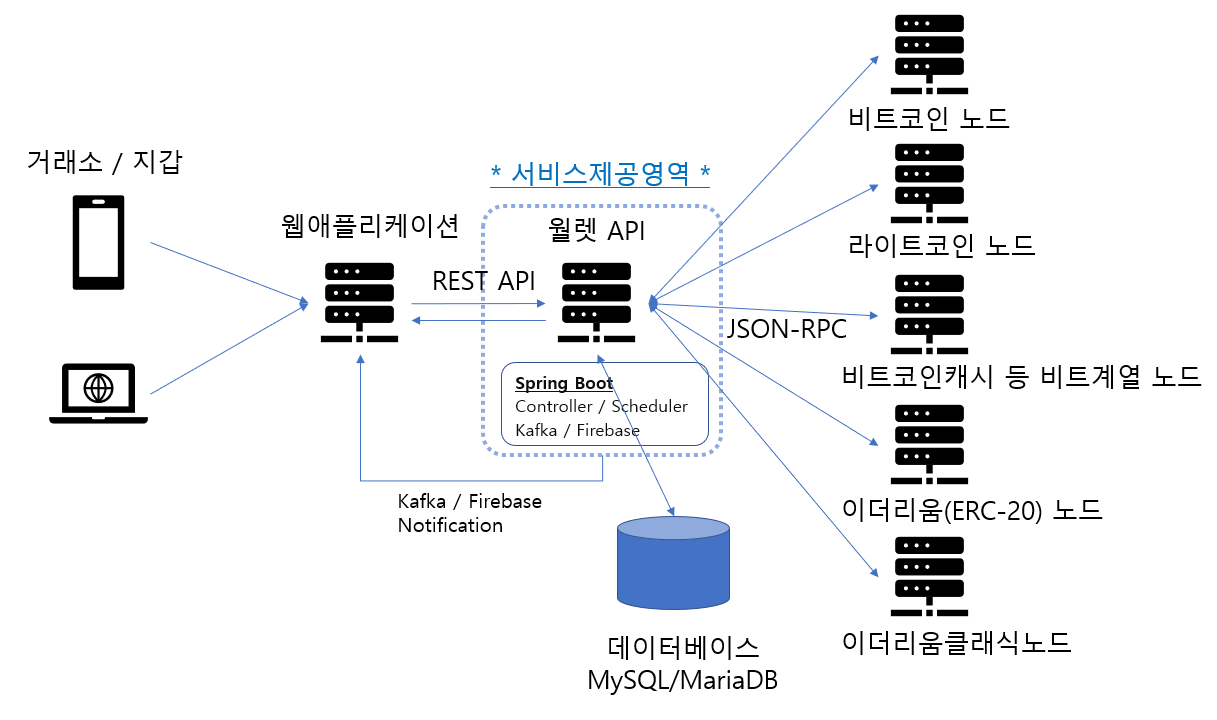
[5. 지정 주소 잔고 조회: /balance 7](#_Toc1596916)

[IX. 주요 테이블 7](#_Toc1596917)

[X. Todo 8](#_Toc1596918)

## 시스템 구성

* 아래와 같은 서비스 인프라 구성이 필요
* 점선 영역이 월렛API가 지원하는 범위
* (1) 지갑 Rest API와 (2) 블록 내 입출금 트랜잭션 데이터 검사 (TX Sync)



## 유의 사항

* + - 거래소 지갑 방식과 사용자 직접 관리 방식이 다름, 본 설계는 거래소 지갑 중심으로 설계되었으나, 약간의 수정으로 사용자 직접 관리하는 (모바일 월렛 방식)으로 사용 가능

|  |  |  |
| --- | --- | --- |
| **기능** | **거래소 방식** | **사용자 관리 방식** |
| 지갑 출금 | 한 개 내지 몇 개의 송금 전용 주소를 운영. 개인 주소의 잔고를 송금 주소로 일괄로 모아놓음 | 사용자 각자의 주소에서 직접 송금 |
| 입출금 수수료 | 구현 가능 | 구현 불가 (2번 송금해야 함 + 2번의 블록체인 수수료) |
| 잔고 확인 | 허수로 데이터베이스에서 관리 | 블록체인에서 조회 |
| 송금 처리 | Batch 스케줄링 일괄 송금 | 실시간 송금 |
|  |  |  |

## 지갑 동기화 프로세스

* + - WalletSyncScheduler.java에서 순차적으로 실행

|  |  |  |
| --- | --- | --- |
| **No** | **호출 메소드** | **설 명** |
| 1 | updateCryptoMaster() | 블록체인에서 가장 최근 블록 숫자를 가져옴 |
| 2 | syncWalletBalances() | 블록체인상의 개별 주소들의 잔고를 조회하여 DB에 업데이트 |
| 3 | openBlocksGetTxsThenSave() | 스케줄러에서 가장 마지막 체크한 블록부터 블록체인의 최근 블록 숫자까지 체크하며 블록의 트랜잭션을 열어, API에서 발행한 주소에 대한 TX만 추출 => DB에 상태 업데이트 |
| 4 | batchSendTransaction() | 송금 요청한 송금 건들을 일괄 송금 |
| 5 | updateSendConfirm() | 송금처리 건의 컨펌 수 업데이트 체크 |
| 6 | updateReceiveConfirm() | 입금 건의 컨펌 수 업데이트 체크 |
|  |  |  |

## 개발환경

|  |  |  |
| --- | --- | --- |
| **No** | **Platform** | **설 명** |
| 1 | JAVA | JDK 1.8 |
| 2 | IDE | Spring Tools Suite (권장) 또는 Intelli J 또는 Eclipse |
| 3 | Lombok | @Data 어노테이션 관련  <https://countryxide.tistory.com/16>   * + - Intelli J의 경우 lombok 플러그인 필요 |
| 4 | 선택) Kafka | Kafka를 사용하고자 할 경우(주로 거래소 애플리케이션과 연동하여 입출금 알림용) 브로커 구성필요 |
| 5 | 선택) FCM | 모바일 지갑에 실시간 입출금 PUSH가 필요한 경우.   * + - Google Firebase 콘솔 설정한 후     - /src/main/resources/serviceAccount.json 설정 |

## 초기 데이터베이스 구성

* MySQL / MariaDB에서 DB 생성권한이 있는 유저로 wallet.sql을 실행
* /src/main/resources/wallet.sql => 내용을 copy해서 SQL 에디터에 붙여넣고 실행

## 주요 설정파일

### application.yml

|  |  |  |
| --- | --- | --- |
| **1뎁스** | **2뎁스** | **설명** |
| server: | port: | Rest API 제공 포트 |
| app: | enabledSymbols: | 사용할 암호화폐 명  (토큰명 포함) |
| enabledERC20s: | 사용할 ERC20 토큰명 |
| crypto: | rpcurl: | 노드가 제공하는 JSON-RPC URL:포트 |
| rpcid: | RPC 유저ID |
| rpcpw: | RPC 패스워드 |
| sendaccount: | 송금을 전담하는 계정 (거래소의 경우 1개의 주소에서 일괄 송금처리 함, 비트계열의 경우 계정과 주소가 함께 있음) |
| sendaddr: | 송금을 전담하는 주소 |
| testaddr: | JUNIT 테스트 시 사용하는 테스트 용 주소 |
| pp: | 노드 지갑 잠금해제 패스워드 |
| coldaddr: | 암호화폐를 백업할 콜드월렛 주소 (현재 미사용) |
| decimals: | 암호화폐 소수점 자리수 |
| initialblock: | 동기화 처음 시작할 블록 주소 (본 서비스와 연관있는 트랜잭션이 포함된 최초의 블록 숫자 기재) |
| minconfirm: | 해당 암호화폐의 입출금이 확정이라고 판단할 최소 컨펌 수 |
| minamtgather: | (거래소) 송금 전용 주소로 잔고를 모을 때 개인 주소에 최소 보유량 (현재 미사용) |

## 신규 코인 추가 시 절차

|  |  |  |
| --- | --- | --- |
| **No** | **작업** | **설명** |
| (1) | 추가될 코인 영문심볼 확인 | 해당 코인의 3~4자리 영문 심볼. BTC / LTC / XMR / ADA 등 (거래소마다 다른경우 있으므로 자체 사용할 약자를 결정하면 됨) |
| (2) | WalletConst.java | 해당 심볼이 본 파일에 없으면 추가  ex) String SYMBOL\_DOGE = “DOGE”; |
| (3) | package crypto.wallet.service.impl | 추가된 코인에 대한 서비스가 없으면 여기에 추가 (상속받을 클래스 확인 필요) |
| (4) | CoinFactory.java | 해당 심볼에 대한 서비스가 본 파일에 없으면 추가  = getService(String symbol) 메소드의 반환값 |
| (5) | application.yml | 해당 심볼에 대한 속성이 crypto: 속성 아래에 없으면 추가 (다른 암호화폐 설정 참고) |

### 새로 추가하는 암호화폐의 Service를 추가하는 방법

* [BTC계열] BitcoinAbstractService.java를 상속받은 클래스를 생성해서 getter, setter를 추가. BTCRpcServiceImpl.java 참고
* [이더리움계열] EthereumAbstractService.java를 상속받은 클래스를 생성해서 getter, setter를 추가.
* [ERC20토큰계열] ERC20AbstractService.java를 상속받은 클래스를 생성해서 getter, setter를 추가.
* [멀티체인계열] =>
* [기타] 새로 만들어야 함

※ 참고) 비트코인 계열 암호화폐

* BTC / LTC / BTG / BCD / BCH / DOGE / BSV (BCH포크, 확인필요)

※ 참고) 이더리움 계열 암호화폐

- ETH, ETC, CLO

## Rest API (WalletApiController.java)

* + - API URL: http://{ip}:{port}/api/v1/{메소드명}

|  |  |  |
| --- | --- | --- |
| **No** | **메소드** | **설명** |
| 1 | address() | 새 주소 발급 |
| 2 | validate() | 주소 검증 (유효성만 검증, 실재하는지는 알 수 없음) |
| 3 | send() | 송금 전용 주소에서 암호화폐 송금(출금) |
| 4 | sendfrom() | 지정된 주소에서 암호화폐 송금(출금)  => 기능 동작여부 확인 필요 (최근 체크 못함) |
| 5 | balance() | 해당 주소의 지갑 잔고조회   1. 거래소의 경우 DB테이블로 관리함 2. 모바일 월렛 같은 경우 실 주소 잔고로 체크하기도 함   (1), (2) 방식 중에 선택 필요 |

### 새 주소 발급: /address

|  |  |  |
| --- | --- | --- |
| **타입** | **파라미터키** | **설명** |
| String | symbol | 암호화폐 심볼. Ex) BTC / ETH |
| int | uid | 유저ID (정수 고유 값) |
| String | brokerId | KAFKA를 사용하는 경우 유저가 할당된 브로커의 ID. 미사용 시 하드코딩 문자열 |
|  |  |  |
|  |  |  |

### 주소 검증: /validate

|  |  |  |
| --- | --- | --- |
| **타입** | **파라미터키** | **설명** |
| String | symbol | 암호화폐 심볼. Ex) BTC / ETH |
| String | address | 지갑 주소 |
|  |  |  |
|  |  |  |

### 송금: /send

|  |  |  |
| --- | --- | --- |
| **타입** | **파라미터키** | **설명** |
| String | symbol | 암호화폐 심볼. Ex) BTC / ETH |
| int | uid | 송금 요청 유저ID |
| String | orderId | 송금요청 일련번호 (중복요청 방지) |
| String | toAddress | 수신자 암호화폐 주소 |
| double | amount | 암호화폐 송금 수량 |
|  |  |  |

### 지정 주소에서 출금: /sendfrom

|  |  |  |
| --- | --- | --- |
| **타입** | **파라미터키** | **설명** |
| String | symbol | 암호화폐 심볼. Ex) BTC / ETH |
| int | uid | 송금 요청 유저ID |
| String | orderId | 송금요청 일련번호 (중복요청 방지) |
| String | toAddress | 수신자 암호화폐 주소 |
| double | amount | 암호화폐 송금 수량 |
| String | fromAddress | 송금자 주소 |
| String | fromAccount | 송금자 계정 (비트코인 계열) |
|  |  |  |

### 지정 주소 잔고 조회: /balance

|  |  |  |
| --- | --- | --- |
| **타입** | **파라미터키** | **설명** |
| String | symbol | 암호화폐 심볼. Ex) BTC / ETH |
| String | address | 지갑주소 (모바일 월렛의 경우, 블록체인 주소 잔고 조회) |
| int | uid | 유저ID (거래소의 경우, DB 장부 잔고 조회) |
|  |  |  |
|  |  |  |

## 주요 테이블

|  |  |  |
| --- | --- | --- |
| **No** | **테이블명** | **설명** |
| 1 | tb\_crypto\_master | 암호화폐별 개요) 마지막 Sync 블록번호, 잔고, 최근 송금Fee 등 |
| 2 | tb\_address | 사용자 UID에 매칭되는 블록체인 주소 |
| 3 | tb\_send\_request | 송금 요청 건 목록 |
| 4 | tb\_send | 본 노드에서 송금된 이력 (블록체인에서 조회된 이력) |
| 5 | tb\_recv | 본 노드로 입금된 이력 |
| 6 | tb\_orphan\_address | 잔고가 있으나 관리되고 있지 않은 주소 목록 |
|  |  |  |

## Todo

* + - WallSyncScheduler.java 가 싱글 쓰레드 인데, 암호화폐 다수를 관리하는 경우 API 영역과 스케줄러 인스턴스를 분리하는 방법도 필요 (스케줄러 부하가 크면 API가 지연될 수 있음)
    - 암호화폐 다수 관리하는 경우 스케줄러도 암호화폐 별로 Grouping하여 몇 개의 스케줄 애플리케이션으로 나눌 필요 있음 (ERC20)에서 부하가 많이 걸림.

