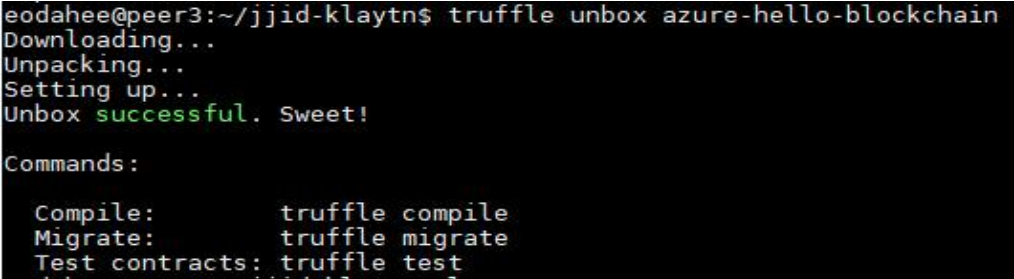

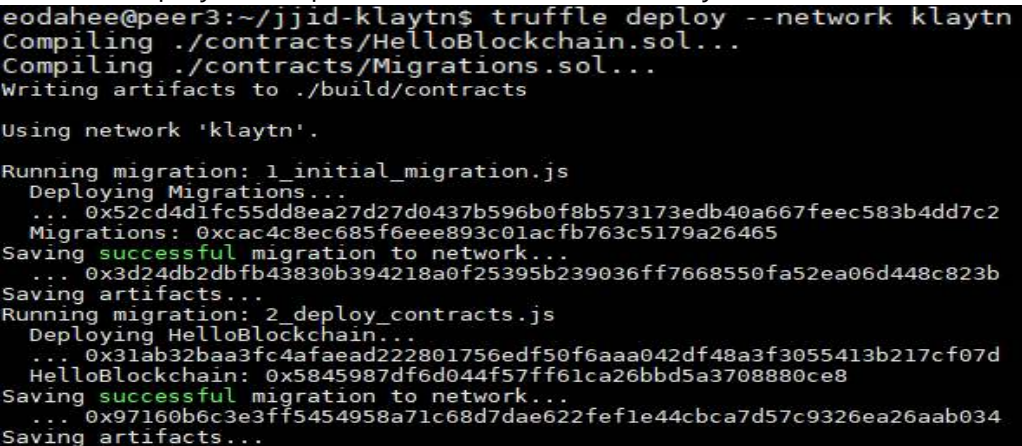


< Klaytn을 통한 B-App 개발하기 >

1. 환경설정

번 호	코 드
① node 설치	<pre>\$ curl -sL https://deb.nodesource.com/setup_10.x sudo -E bash - \$ sudo apt-get install -y nodejs \$ node --version</pre>
② npm 설치	<pre>\$ sudo npm install -g npm \$ sudo npm rebuild</pre>
③ truffle 설치	<pre>\$ sudo npm install -g truffle@4.1.15</pre> <p>→ truffle version은 5까지 나왔지만, klaytn이 4.1.15로 개발되었기 때문에 환경 맞춰주기</p>
④ 템플릿 다운로드 (https://www.trufflesuite.com/boxes/azure-hello-blockchain)	<pre>\$ truffle unbox azure-hello-blockchain</pre>  <p>→ But, 이더리움 버전이기 때문에 클레이튼에 맞게 코드 수정이 필요!!</p>
	 <p>smart contract를 배포하기 위해 contracts/Migrations.sol은 반드시 필요!! 그와 관련된 로직이 migrations 디렉토리에 들어있음. 여기에 웹개발에 필요한 html이나 js 파일은 알아서 추가!!</p> <p>package.json 파일을 만들어 klaytn 개발에 필요한 caver(이더리움의 web3)같은 라이브러리 작성해주기!!</p> <p>truffle.js는 환경변수 설정을 담당 (어느 네트워크에 SC를 배포할지 설정) artifact라고 불리는 HelloBlockchain.json 파일은 해당 contract의 abi 정보를 비롯한 관련 모든 정보를 가지고 있음</p>
⑤ 개발시작!! (스마트 계약 배포 및 재배포)	<pre>\$ truffle deploy --network klaytn \$ truffle deploy --compile-all --reset --network klaytn</pre> 

2. solidity 문법 참고

구 분	내 용
① 참고 블로그	https://d2fault.github.io/2018/03/19/20180319-about-solidity-1/

3. EN(Endpoint Node) 실행하기

번 호	코 드
① EN 다운로드	https://docs.klaytn.com/node/endpoint-node/installation-guide/download \$ tar zxf ken-baobab-vX.X.X-X-linux-amd64.tar.gz \$ export PATH=\$PATH:\$PWD/ken-linux-amd64/bin
② config EN network	https://docs.klaytn.com/getting-started/quick-start/launch-an-en \$ vi ken-linux-amd64/conf/kend.conf → 파일 열어서 내용 수정(RPC_API에 net 추가 및 DATA_DIR 변경)
③ EN 시작하기	\$ kend start
④ EN 상태 확인	\$ kend status
⑤ EN log 확인	\$ tail -f ./logs/kend.out
⑥ 콘솔에 연결하기	<pre>\$ ken attach ./klay.ipc eodahee@peer3:~/jjid-klaytn/endpoint-node\$ ken attach ./klay.ipc Welcome to the Klaytn JavaScript console! instance: klaytn/v1.1.1/linux-amd64/go1.12.5 datadir: /home/eodahee/jjid-klaytn/endpoint-node modules: admin:1.0 debug:1.0 governance:1.0 istanbul:1.0 klay:1.0 miner:1.0 net:1.0 personal:1.0 rpc:1.0 txpool:1.0 web3:1.0 > klay.blockNumber 1304680 > █</pre> → blockNumber 등 console에서 확인 가능(싱크가 맞아야 사용 가능)
⑦ Klaytn 계정 생성하기	<pre>> personal.newAccount() > personal.newAccount() Passphrase: Repeat passphrase: "0xd2ee0cc0bfcd18372a2d0120e012f7d54e0b7735"</pre> → ./keystore/UTC-2019-09-27T08-32-40.467111570Z-d2ee0cc0bfcd18372a2d0120e012f7d54e0b7735 키스토어 아래에 계정이 생성됨
⑧ 계정 잠금해제	<pre>> personal.unlockAccount('d2ee0cc0bfcd18372a2d0120e012f7d54e0b7735') > personal.unlockAccount('d2ee0cc0bfcd18372a2d0120e012f7d54e0b7735') Unlock account d2ee0cc0bfcd18372a2d0120e012f7d54e0b7735 Passphrase: true</pre>
⑨ 잔액 확인	> klay.getBalance('d2ee0cc0bfcd18372a2d0120e012f7d54e0b7735')
참고사항	https://docs.klaytn.com/getting-started/quick-start/install-development-tools

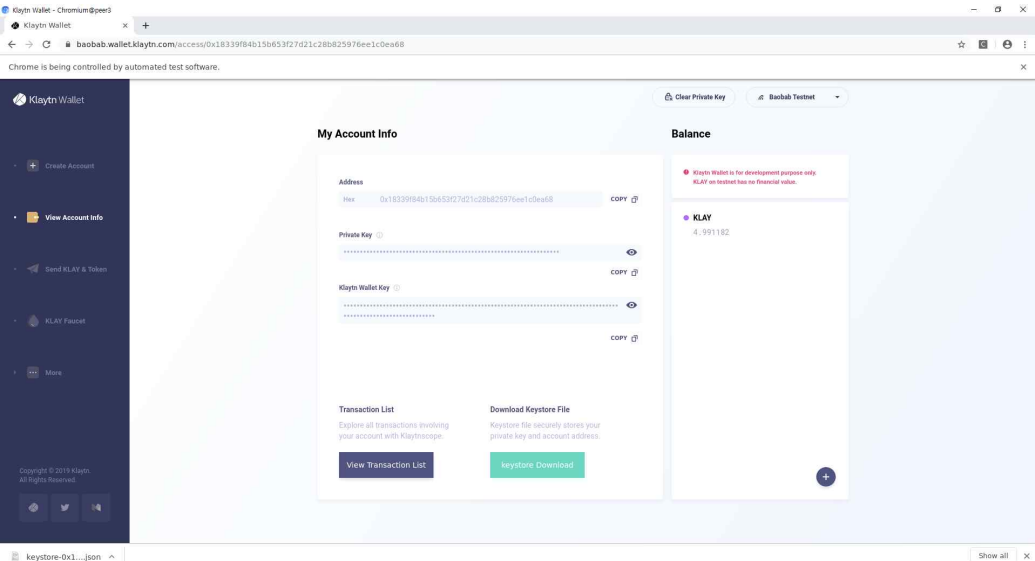
4. BApp 개발하기

번 호	코 드
① BApp 개발을 위한 directory 생성	<pre>eodahee@peer3:~/jjid-klaytn/klaytn-testboard\$ tree</pre> <pre> ├── build │ ├── contracts │ │ ├── Enroll_user_authen.json │ │ ├── Migrations.json │ │ └── Mortal.json │ ├── contracts │ │ ├── Enroll_user_authen.sol │ │ ├── Migrations.sol │ │ ├── deployedABI │ │ ├── deployedAddress │ │ └── migrations │ │ ├── 1_initial_migration.js │ │ └── 2_deploy_contract.js │ └── test │ ├── test-klaytn.js │ └── truffle-config.js </pre> <p>https://www.trufflesuite.com/boxes 를 통해 샘플 하나를 다운받으면 기본적인 파일 구조는 위 그림과 같음.(deployedABI와 deployedAddress는 생성한 것)</p>
② smart contract 개발	<p>contracts 디렉토리 아래 원하는 SC 개발(http://ide.klaytn.com/ 이용)</p> <pre> Enroll_user_authen.sol Migrations.sol JS 1_initial_migration.js JS 2_deploy_contract.js JS truffle-con C: > Users > KISMI06 > AppData > Local > Temp > Mxt111 > RemoteFiles > 133600_2_11 > Enroll_user_authen.sol 1 pragma solidity >=0.4.21 <0.6.0; 2 3 contract Enroll_user_authen { 4 address public owner; 5 6 constructor() public { 7 owner = msg.sender; 8 } 9 10 function getBalance() public view returns (uint) { 11 return address(this).balance; 12 } 13 14 function deposit() public payable { 15 require(msg.sender == owner); // msg를 보낸 사람이 owner 일 때만 deposit이 가능하도록 16 } 17 18 function transfer(uint _value) public returns (bool) { 19 require(getBalance() >= _value); // 현재 남아있는 잔액이 _value보다 많이 있는지 확인 20 msg.sender.transfer(_value); 21 return true; 22 } 23 } </pre> <p>이 때, 주의 할 사항은 반드시 파일명과 함수의 이름이 같아야 deploy 간 오류 안남! 다를 경우 발생하는 오류(찾을 수 없다고 나옴) 즉, 두 이름이 같아야 build/contracts/ 생기는 json 파일명이 동일하게 생성 https://github.com/trufflesuite/truffle/issues/724 trouble shooting 참고</p> <pre> Error: Could not find artifacts for ../contracts/jjid from any sources at Resolver.require (/home/eodahee/jjid-klaytn/node_modules/truffle/build/cli.bundled.js:60324:9) at Object.require (/home/eodahee/jjid-klaytn/node_modules/truffle/build/cli.bundled.js:70083:36) at ResolverIntercept.require (/home/eodahee/jjid-klaytn/node_modules/truffle/build/cli.bundled.js:198803:32) at /home/eodahee/jjid-klaytn/klaytn-testboard/migrations/2_deploy_contract.js:1:24 at Script.runInContext (vm.js:133:20) at Script.runInNewContext (vm.js:139:17) at /home/eodahee/jjid-klaytn/node_modules/truffle/build/cli.bundled.js:102210:14 at FSReqWrap.readFileAfterClose [as oncomplete] (internal/fs/read_file_context.js:53:3) </pre>
③ migrations 개발	<p>②에서 개발한 smart contract를 deploy 하기 위해 반드시 필요한 함수로 기본적인 구조는 아래와 같음(deployedABI와 deployedAddress를 위해 추가 가능)</p> <pre> Enroll_user_authen.sol Migrations.sol JS 1_initial_migration.js JS 2_deploy_contract.js C: > Users > KISMI06 > AppData > Local > Temp > Mxt111 > RemoteFiles > 133600_2_1 > JS 1_initial_migration.js 1 const Migrations = artifacts.require("../contracts/Migrations.sol"); 2 3 module.exports = function(deployer) { 4 deployer.deploy(Migrations); 5 }; </pre>

4. BApp 개발하기(이어서)

<p>④ 현재까지 작성된 부분 배포하기</p>	<pre>\$ truffle deploy --network klaytn</pre>  <p>위 그림과 같이 배포가 완료되고, build/contracts/에 관련 json 파일이 생성 network 이름은 truffle-config.js 파일에서 지정한 대로 사용 가능!!</p> 
<p>⑤ Front-end 개발하기</p>	<pre>\$ mkdir src \$ vi src/index.html → UI \$ vi src/index.js → UX</pre>
<p>참고사항</p>	<p>body-parser 모듈 참고 블로그 : https://backback.tistory.com/336</p> <p>webpack을 활용한 개발 참고 블로그 : https://jusungpark.tistory.com/52 자바스크립트 모듈을 관리해주는 모듈 번들러!!</p> <p>html에서 ajax로 값을 보내고, node에서 값을 받으면 반드시 JSON.parse를 통해 type 변환을 다시 한 번 시켜줘야 한다!!(아님 string으로 인식해버림..)</p> <p>npm install async 꼭 해주기^^</p> <p>js error 모음 : https://blog.meeta.io/10 webpack으로 여러 모듈 호출하기 : https://choiyb2.tistory.com/96</p>

5. selenium을 활용하여 klaytn keystore file 다운로드 받기

번 호	코 드
① selenium 사용을 위한 환경설정	<p>\$ pip install selenium → 필요에 따라 pip upgrade 해야 할 수 있음</p> <p>\$ sudo apt-get install chromium-chromedriver → 클라우드 환경에서 실행하기 때문에 그냥 chromedriver를 깔면 실행이 안됨. https://github.com/SeleniumHQ/selenium/wiki/ChromeDriver#requirements 위 웹페이지 참고하여 설치(좀 오래걸림) → 다운로드 된 chromedriver 위치는 /usr/bin/ <pre>eodahee@peer3:~/jjid-klaytn/klaytn-testboard\$ ll /usr/bin/chromedriver -rwxr-xr-x 1 root root 12068656 Sep 21 02:28 /usr/bin/chromedriver*</pre> </p> <p>python file에서 import해와 scraping 실시 <pre># -*- coding: utf-8 -*- from selenium import webdriver driver = webdriver.Chrome('/usr/bin/chromedriver')</pre> </p>
② scraping 코드 개발	<p>개발자 도구 활용하여 xpath를 통한 data 입력과 click</p> <pre># -*- coding: utf-8 -*- from selenium import webdriver driver = webdriver.Chrome('/usr/bin/chromedriver') # driver.implicitly_wait(3) driver.get('https://baobab.wallet.klaytn.com/access') # 계좌조회 klaytn wallet driver.implicitly_wait(2) # private 입력창에 data 입력 privatekey_input = driver.find_element_by_xpath('//*[@id="input-privatekey"]') privatekey_input.send_keys('0xbf8268bafc30faa0d43caeffe1bbe669a162b70b8cc6c5228aea83c08a049b5e') # test모드에 따른 동의 체크박스 클릭 agree_btn = driver.find_element_by_xpath('//*[@id="root"]/div/section/div[2]/div[2]/div/div[2]/div/div[2]/div[11]') driver.execute_script("arguments[0].click();", agree_btn) # access 버튼 클릭 access_btn = driver.find_element_by_xpath('//*[@id="root"]/div/section/div[2]/div[2]/div/div[2]/div/button') access_btn.click() # keystore_file 다운로드 버튼 클릭 keystore_file_download_btn = driver.find_element_by_xpath('//*[@id="root"]/div/section/div[2]/div[2]/div[2]/div/div[4]/div/button') keystore_file_download_btn.click() # 모달창에 pw 입력(HLF txid) pw = driver.find_element_by_xpath('//*[@id="password"]') pw.send_keys('wioewjog456456!') # 확인버튼 클릭(진짜 file 다운로드) download_btn = driver.find_element_by_xpath('//*[@id="root"]/div/section/div[2]/div[2]/div[1]/div/div[3]/button[2]') download_btn.click()</pre>
③ 코드 실행	<p>cloud에서도 chrome이 실행되어 keystore file 다운로드 완료!!</p>  <p>The screenshot shows the Klaytn Wallet web application running in a browser. The interface includes a sidebar with navigation options like 'Create Account', 'View Account Info', 'Send KLAY & Token', 'KLAY Faucet', and 'More'. The main content area displays 'My Account Info' with fields for Address, Private Key, and Klaytn Wallet Key, each with a 'copy' button. Below this is a 'Transaction List' section. To the right, there's a 'Balance' section showing 'KLAY' balance. At the bottom, there's a 'Download Keystore File' button, which is highlighted in green.</p>
④ keystore file 확인	<p>최종 저장된 위치는 ~/Downloads/ <pre>eodahee@peer3:~/jjid-klaytn/klaytn-testboard\$ ls ~/Downloads/ keystore-0x18339f84b15b653f27d21c28b825976ee1c0ea68-2019-10-23.json</pre> </p>

5. selenium을 활용하여 klaytn keystore file 다운로드 받기(이어서)

번 호	코 드
⑤ file 읽어서 hash화	<p>nodejs의 fs라는 라이브러리를 이용하여 readFile로 읽어오기! 단, 이 함수는 비동기이기 때문에 각별한 주의가 필요(삼질 많이 함) → 분명 console.log에는 찍히는데 return하면 undefined 뜨는 매직... 아래 참고! https://stackoverflow.com/questions/34980249/returning-undefined-from-readfile</p> <p>노드 10에서는 사용 fs/promises 하고 피할 수 있습니다 util.promisify</p> <pre>const fs = require('fs').promises; (async() => { try { const result = await fs.readFile('readme.txt', 'utf8'); console.log(result); } catch(e) { console.error(e); } })();</pre>
⑥ Fabric과 Klaytn 결합 - klaytn의 controller.js 파일 import 해오기	<p>처음엔 두 개 다 api화를 해서 호출하는 식으로 사용할까 했지만, 우리 서비스에서는 DID를 저장하는 Fabric이 상위모듈이기 때문에 Klaytn 관련 파일을 모듈화하여 import 해서 사용하는 것으로 결정! 아래와 같이 총 4개의 함수를 module.export = {}를 통해 json화</p> <pre>const Caver = require("caver-js"); // klaytn 블록체인과 소통 할 수 있는 라이브러리 const config = { rpcURL: 'https://api.baobab.klaytn.net:8651' // 환경설정 변수(어떤 klaytn node에 연결해서 쓸지) } const cav = new Caver(config.rpcURL); // caver 환경변수로 넘겨서 인스턴스화 작업 // const jjContract = new cav.klay.Contract(DEPLOYED_ABI, DEPLOYED_ADDRESS); // contract 배포할 때 생성되는 2개 파일의 값을 const fs = require('fs').promises; // keystore file을 읽어오기 위한 라이브러리 const crypto = require('crypto'); // hash화를 위한 라이브러리 module.exports = { // 총 4개 함수(klaytn 계정생성, keystore file 다운로드, 계정조회, klay 송금) auth : { accessType: 'keystore', // 인증방식 keystore와 privatekey 방식이 있음 keystore: '', password: '' }, wallet : { account: '', date: '' }, createAccount : async function() { // 계좌 생성하기... }, downloadKeystore : async function() { // keystore file 다운로드... }, getAccount : async function() { // account의 잔액 불러오기... }, sendKlay : async function() { // klay 전송하기(send_tx)... } }</pre> <p>Fabric의 routes.js에서 <code>var klaytn = require('/home/eodahee/jjkd-klaytn/jjgo-klaytn/controller.js');</code> 불러와 사용</p>
⑦ Fabric과 Klaytn 결합 - 함수 사용하기 (enroll_wallet_authen)	<p>시나리오 : klaytn 함수를 먼저 호출해서 account를 생성하고, keystore file을 다운로드 받음 → file을 읽어서 crypto 라이브러리를 통해 hash화 → 두 개의 값(account, file_hash)을 Fabric에 전달해 블록에 저장</p> <p>문제점 : nodejs는 기본값이 비동기 통신이기 때문에 함수를 순차적으로 실행 하지 않음 / return 값이 string이나 json이 아닌 Promise로 떨어짐..</p> <p>해결방안 : 동기처리가 필요(⑤의 fs라이브러리 공부 및 setTimeout() 이용) Promise 객체를 string화 해주기(객체 open)</p> <pre>app.get('/Fabric_enroll_wallet_authen', function(req, res){ // klaytn의 정보를 wallet에 등록 var account_info; var keystorefile_hash; // promise 객체 까주기 klaytn.createAccount().then(re => { account_info = re; }); // promise 객체 까주기 klaytn.downloadKeystore().then(kv => { keystorefile_hash = kv; }); // 1초 후에 아래 함수 실행 setTimeout(function() { const data = {'account_info':account_info, 'keystorefile_hash':keystorefile_hash}; bdms.Fabric_enroll_wallet_authen(data, res); }, 1000); }); app.post('/fabric/enroll_wallet_authen', async function(req, res){ // klaytn의 정보를 wallet에 등록 var authen_key = req.body.authen_key; // account를 보유 할 did 결정 const account = await klaytn.createAccount(); // klaytn_controller.js를 통한 account 생성 const kv = await klaytn.downloadKeystore(authen_key); // klaytn_controller.js를 통한 keystore file 다운로드 const filename = kv.filename; const keystorefile_hash = kv.file_hash; const encrypt_pk = kv.encrypt_pk; const data = {'authen_key':authen_key, 'account':account, 'filename':filename, 'keystorefile_hash':keystorefile_hash, 'encrypt_pk':encrypt_pk}; bdms.Fabric_enroll_wallet_authen(data, res); // fabric server로 생성된 데이터 전송 });</pre>

5. selenium을 활용하여 klaytn keystore file 다운로드 받기(이어서)

- caver-js의 encrypt 함수를 활용해 keystore file 생성해서 간단히 문제 해결(selenium 안 쓰고)
<https://ko.docs.klaytn.com/bapp/sdk/caver-js/api-references/caver.klay.accounts#encrypt>
- filepath와 filename을 지정하고, fs.writeFileSync() 함수를 통해 file 생성하기

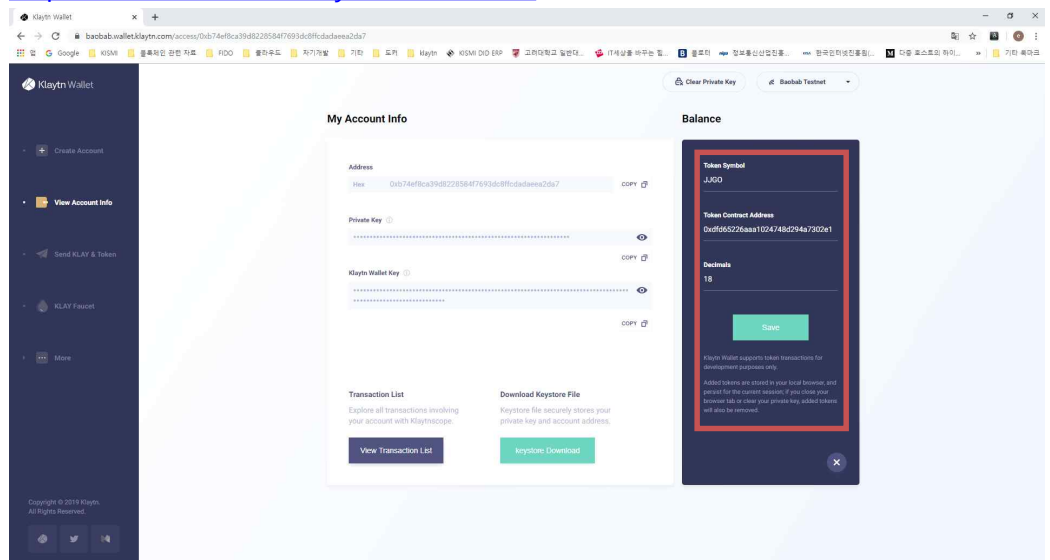
6. ERC20을 활용하여 JJGO 토큰 발행하기

번 호	코 드
① ERC20 표준을 따르는 오픈소스 다운로드	<p>두 개 블로그 참고하여 개발 https://medium.com/@ggomma/%EC%9D%B4%EB%8D%94%EB%A6%AC%EC%9B%80-%ED%86%A0%ED%81%B0-%EB%B0%9C%ED%96%89-5-openzeppelin-%EC%82%AC%EC%9A%A9%ED%95%98%EA%B8%B0-79b5eee68939 https://javaexpert.tistory.com/926</p> <p>\$ npm install openzeppelin-solidity --save → node_modules/ 디렉토리 안에 설치</p> <pre>eodahee@peer3:~/jjgo/klaytn_service/jjgo-klaytn\$ ls node_modules/openzeppelin-solidity/build CHANGELOG.md contracts LICENSE package.json README.md test</pre>
② openzeppelin을 활용하여 JJGO 토큰 생성 solidity 개발	<pre>pragma solidity ^0.4.24; import 'openzeppelin-solidity/contracts/token/ERC20/ERC20.sol'; import "openzeppelin-solidity/contracts/ownership/Ownable.sol"; contract JjgoToken is ERC20, Ownable { string public constant name = "JeJuGO token"; // 토큰이름 string public constant symbol = "JJGO"; // 통화단위 uint public constant decimals = 1; // 소숫점 아래 자리 uint public constant INITIAL_SUPPLY = 1000000000000000 * (10 ** decimals); constructor() public { // smartcontract가 생성 될 때 딱 1회 호출되는 함수 _mint(msg.sender, INITIAL_SUPPLY); } }</pre> <p>맨 위에 ERC20.sol file을 import해서 JjgoToken으로 덮어쓰는 방식</p>
③ truffle-config.js 파일에서 환경설정	<pre>const PrivateKeyConnector = require('connect-privkey-to-provider'); const URL = 'https://api.baobab.klaytn.net:8651'; module.exports = { networks: { klaytn: { provider: new PrivateKeyConnector(PRIV_KEY, URL), network_id: '1001', gas: '2000000', gasPrice: null, }, }, }</pre> <p>PrivateKeyConnector 모듈을 활용하여 klaytn이라는 networks 모듈 설정 network는 klaytn의 테스트넷인 baobab활용(gas비 부족하면 오류발생!!)</p>
④ smart contract 배포	<pre>\$ truffle deploy --network klaytn eodahee@peer3:~/jjgo/klaytn_service/jjgo-klaytn\$ truffle deploy --network klaytn --reset Compiling ./contracts/JjgoToken.sol... Compiling openzeppelin-solidity/contracts/math/SafeMath.sol... Compiling openzeppelin-solidity/contracts/token/ERC20/ERC20.sol... Compiling openzeppelin-solidity/contracts/token/ERC20/IERC20.sol... Writing artifacts to ./build/contracts Using network 'klaytn'. Running migration: 1_initial_migration.js Deploying Migrations... ... 0x883e76fdaabaaa9b7db0e83fa708c34e016c8f613dd0ce29825152bd6d9ac064 Migrations: 0xe5b128e5985ae0b4ced74cdd9a1ee964a557647c Saving artifacts... Running migration: 2_deploy_contracts.js Replacing JjgoToken... ... 0x52adbec7447f5d1d7851f0c41b0d0133ec9631c6ad3d29b219d43b122caa149c JjgoToken: 0xdfd65226aaa1024748d294a7302e100afaf75216 Saving artifacts... 파일에 ABI 입력 성공 파일에 주소 입력 성공</pre> <p>차례대로 smartcontract의 배포가 완료되며, 파란색 네모박스 안의 42자리 값이 해당 계약의 Address(token 발급 시 사용)</p>

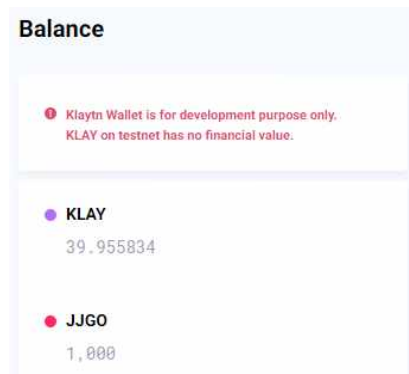
6. ERC20을 활용하여 JJGO 토큰 발행하기(이어서)

⑤ Klaytn Wallet을 활용하여 token 추가

<https://baobab.wallet.klaytn.com/access> 웹페이지에서 우측 하단 + 버튼 클릭



smart contract에서 개발한 바와 같이 token symbol과 address, decimals을 입력 후 Save 버튼을 누르면 토큰 발급 완료!



decimal을 18로 입력했기 때문에 1,000JJGO

⑥ JJGO smart contract가 사용할 수 있는 함수

```
eodahee@peer3:~/jjgo/klaytn_service/jjgo-klaytn$ node server.js
Start klaytn API port : 8000
object string
{ name: [Function: bound_createTx0bject],
  '0x06fdde03': [Function: bound_createTx0bject],
  'name()': [Function: bound_createTx0bject],
  approve: [Function: bound_createTx0bject],
  '0x095ea7b3': [Function: bound_createTx0bject],
  'approve(address,uint256)': [Function: bound_createTx0bject],
  totalSupply: [Function: bound_createTx0bject],
  '0x18160ddd': [Function: bound_createTx0bject],
  'totalSupply()': [Function: bound_createTx0bject],
  transferFrom: [Function: bound_createTx0bject],
  '0x23b872dd': [Function: bound_createTx0bject],
  'transferFrom(address,address,uint256)': [Function: bound_createTx0bject],
  INITIAL_SUPPLY: [Function: bound_createTx0bject],
  '0x2ff2e9dc': [Function: bound_createTx0bject],
  'INITIAL_SUPPLY()': [Function: bound_createTx0bject],
  decimals: [Function: bound_createTx0bject],
  '0x313ce567': [Function: bound_createTx0bject],
  'decimals()': [Function: bound_createTx0bject],
  increaseAllowance: [Function: bound_createTx0bject],
  '0x39509351': [Function: bound_createTx0bject],
  'increaseAllowance(address,uint256)': [Function: bound_createTx0bject],
  balanceOf: [Function: bound_createTx0bject],
  '0x70a08231': [Function: bound_createTx0bject],
  'balanceOf(address)': [Function: bound_createTx0bject],
  symbol: [Function: bound_createTx0bject],
  '0x95d89b41': [Function: bound_createTx0bject],
  'symbol()': [Function: bound_createTx0bject],
  decreaseAllowance: [Function: bound_createTx0bject],
  '0xa457c2d7': [Function: bound_createTx0bject],
  'decreaseAllowance(address,uint256)': [Function: bound_createTx0bject],
  transfer: [Function: bound_createTx0bject],
  '0xa9059cbb': [Function: bound_createTx0bject],
  'transfer(address,uint256)': [Function: bound_createTx0bject],
  allowance: [Function: bound_createTx0bject],
  '0xdd62ed3e': [Function: bound_createTx0bject],
  'allowance(address,address)': [Function: bound_createTx0bject] }
```


6. ERC20을 활용하여 JJGO 토큰 발행하기(이어서)

⑦ transfer 함수를 통한 토큰 전송 (⑤의 코드화)

baobab network에 배포 후 caver-js를 활용하여 transfer 함수에 접근

```
giveJjgoToken : async function(req) { // token 전송 contract
  var to_account = req.to_account;

  console.log("----- First 1000JJGO token to " + to_account + " -----");
  try {
    await cav.klay.accounts.wallet.add('0xc169851e8ece6ba15bed26d08e1d02c46f15558e0e4df249e3f79eb1544b33be');
    // console.log(JSON.stringify(cav.klay.accounts.wallet[0]));

    console.log(await jjgoContract.methods.balanceOf('0xb74ef8ca39d8228584f7693dc8ffcdadaeea2da7').call()); // 1,000,000
    console.log(await jjgoContract.methods.balanceOf(to_account).call()); // 0

    var send_res;
    jjgoContract.methods.transfer(to_account, 1000).send({
      from: cav.klay.accounts.wallet[0].address,
      gas: '2000000',
    })
    .once('transactionHash', (txHash) => {
      console.log(txHash);
    })
    .once('receipt', (receipt) => {
      console.log(receipt);
      send_res = receipt;
    })
    .once('error', (error) => {
      console.log(error);
    });

    console.log(await jjgoContract.methods.balanceOf('0xb74ef8ca39d8228584f7693dc8ffcdadaeea2da7').call()); // 999,000
    console.log(await jjgoContract.methods.balanceOf(to_account).call()); // 1000

    return send_res;
  } catch(e) {
    console.log('Fail to send jjgoToken : ${e}')
  }
},
```

no account 라는 error → 현재 wallet에 아무 데이터가 없기 때문에 발생!
cav.klay.accounts.wallet.add({privateKey})를 통해 wallet에 데이터 추가 후
contract의 send함수에 해당 address를 써주면 문제 해결!

⑧ transfer를 위해 해당 계정의 privateKey를 모바일 storage에 저장하기

Fabric_enroll_wallet_authen 함수 실행 간 keystorefile을 download 할 때
생성된 account의 privateKey를 양방향 암호화를 통해 암호화해 return

```
// mobile storage에 pk 저장하기(암호화)
const cipher = crypto.createCipher('aes-256-cbc', 'kismi');
let encrypt_pk = cipher.update(this.wallet_info.privatekey, 'utf8', 'base64');
encrypt_pk += cipher.final('base64'); // 암호화 할 내용이 16자를 넘어가면 update와 final로 내용이 분리됨
console.log('암호화된 pk = ' + encrypt_pk);
```

payJjgoToken 함수를 실행 할 때 암호화된 privateKey를 가져와 같은 방식으로
복호화 → 해당 값으로 transfer transaction을 생성해 토큰 전송

```
// mobile storage에 저장된 pk 가져와서 wallet에 추가하기(복호화)
const decipher = crypto.createDecipher('aes-256-cbc', 'kismi');
let decrypt_pk = decipher.update(encrypt_pk, 'base64', 'utf8');
decrypt_pk += decipher.final('utf8');
```

주의 및 참고사항

★ ④에서 주의사항 많음 ★

- truffle과 solidity의 version 문제
- klaytn은 0.4.24와 0.5.6을 제공하며, openzeppelin과 버전이 같아야 함!
(openzeppelin은 0.5.6쓰고, JjgoToken.sol은 0.4.24 써서 오류 발생했음)
- truffle은 4.1.15버전을 사용할 것(<https://www.infllearn.com/questions/9355>)
- ⑤에서 decimal 값에 따라 초기 발급되는 토큰값이 달라짐

<https://eips.ethereum.org/EIPS/eip-20> → 토큰표준 doc 참고

<https://medium.com/blockchannel/the-anatomy-of-erc20-c9e5c5ff1d02>

→ ERC20 토큰 관련 함수

<http://jaeyunkim.com/lockable-smart-contract-1/> → ERC20 토큰 참고 블로그

<https://github.com/0xSage/erc20-testnet> → ERC20 토큰 testnet 배포 시 참고

<https://github.com/0xSage/erc20-testnet> → ERC20 토큰 배포 시 참고

<https://github.com/0xSage/erc20-testnet> → ERC20 토큰 배포 시 참고

<https://github.com/0xSage/erc20-testnet> → ERC20 토큰 배포 시 참고

<https://steemit.com/kr-dev/@nida-io/2oduk2-erc20-1> → 이더리움 친절 블로그

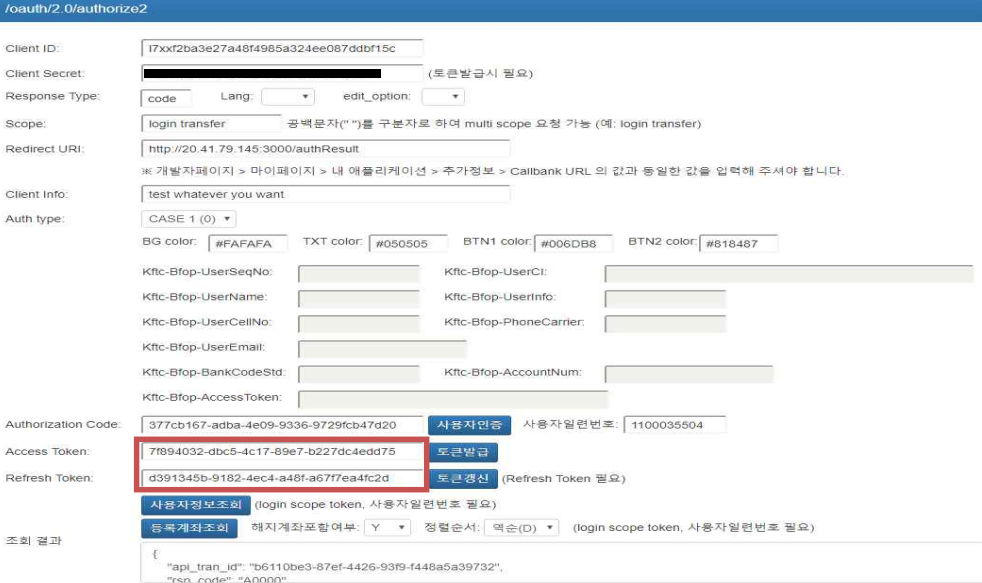
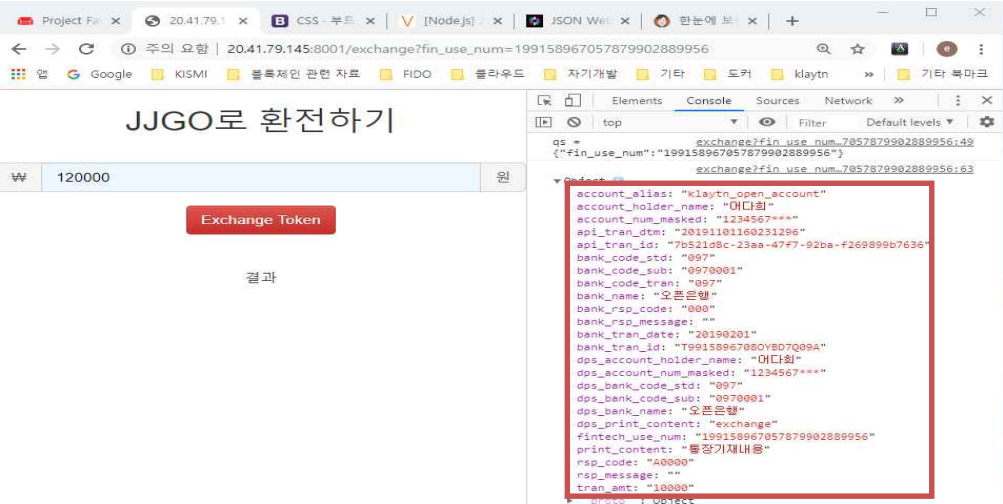
<https://www.zerocho.com/category/NodeJS/post/593a487c2ed1da0018df95d> → crypto 관련 블로그1

<https://nalwoo.tistory.com/36> → crypto 관련 블로그2

7. ERC20을 활용하여 JJGO 토큰 환전하기

번 호	코 드
① 토큰을 발행한 account에서 환전액 transfer해 주기	<p>6번 토큰 발행 후 첫 전송과 같이 contract를 deploy한 계정으로 환전 금액에 대한 JJGO 토큰을 보내줘야 함.</p> <pre>// wallet이 비었을 때만 add하도록 validation if(await cav.klay.accounts.wallet.length == 0) await cav.klay.accounts.wallet.add(admin_key); // console.log(JSON.stringify(cav.klay.accounts.wallet[0])); else { await cav.klay.accounts.wallet.remove(cav.klay.accounts.wallet[0].address); await cav.klay.accounts.wallet.add(admin_key); }</pre> <p>admin_key는 계정의 private_key로 case에 맞게 wallet에 계정정보 추가</p>
<p>② 원화의 살 때 환율 API를 통해 받아오기</p> <p>* JSON 정렬 사이트 : http://json.parser.online.fr/</p>	<pre>import requests headers = { 'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/50.0.2661.102 Safari/537.36' } url = 'https://quotation-api-cdn.dunamu.com/v1/forex/recent?codes=FRX.KRWUSD,FRX.KRWJPY,FRX.KRWKRW,FRX.KRWEUR' exchange = requests.get(url, headers=headers).json() for i in range(4): print(exchange[i]['country'], ' ', exchange[i]['currencyCode'], ' = ', exchange[i]['cashBuyingPrice'])</pre> <p>미국 USD = 1177.75 일본 JPY = 1077.0 중국 CNY = 173.94 유로 EUR = 1302.65</p> <p>업비트(두나무)의 환율 API를 활용하여 USD, JPY, CNY, EUR의 살 때 환율 정보를 받아오기 → 파싱을 통해 값만 klaytn API(exchange_token)로 넘기기</p> <pre>{ "code": "FRX.KRWUSD", "currencyCode": "USD", "currencyName": "달러", "country": "미국", "name": "미국 (KRW/USD)", "date": "2019-11-08", "time": "20:01:00", "recurrenceCount": 383, "basePrice": 1157.5, "openingPrice": 1160.6, "highPrice": 1160.6, "lowPrice": 1154.2, "change": "RISE", "changePrice": 1.5, "cashBuyingPrice": 1177.75, "cashSellingPrice": 1137.25, "t8BuyingPrice": 1146.2, "t8SellingPrice": 1168.8, "t6BuyingPrice": 1171.39, "t6SellingPrice": 1145.05, "exchangeCommission": 3.608, "usDollarRate": 1.0, "high52wPrice": 1223.5, "high52wDate": "2019-08-13", "low52wPrice": 1105.1, "low52wDate": "2018-12-04", "currencyUnit": 1, "provider": "KEB하나은행", "timestamp": 1573211244777, "id": 79, "createdAt": "2016-10-21T06:13:34.000+0000", "modifiedAt": "2019-11-08T11:07:25.000+0000", "signedChangePrice": 1.5, "signedChangeRate": 0.0012975779, "changeRate": 0.0012975779 }</pre> <p>위 코드의 전체 결과는 좌측과 같이 배열에 JSON 형식으로 값들이 들어감.</p> <p>country, basePrice, BuyPrice, SellPrice, timestamp 등 다양한 정보가 들어있으며 원하는 값만 추출해 사용 가능</p> <p>https://steemit.com/kr/@crazypp/python-usd → 참고 블로그</p>
③ caver-js를 활용한 JJGO token 환전	<pre>var send_res; const exchange_val = value * rate; console.log('환전할 금액 = ' + exchange_val); await jjgoContract.methods.transfer(to_account, exchange_val).send({ from: cav.klay.accounts.wallet[0].address, gas: '2000000', }) .once('transactionHash', (txHash) => { console.log(txHash); }) .once('receipt', (receipt) => { // console.log(receipt); send_res = receipt; }) .once('error', (error) => { console.log(error); }); return {'value' : exchange_val, 'receipt' : send_res};</pre> <p>환전하고자하는 값(value)과 ②에서 얻은 환율(rate)을 곱해 token의 양을 계산 → ④에서 등록한 account로부터 to_account에게 토큰 환전 → return값은 receipt</p>

8. 금결원 API 활용하여 계좌이체 받은 금액을 JJGO 토큰으로 환전하기

번 호	코 드
① 금결원 테스트베드 앱 생성 후 토큰발급(GUI에서)	<div>1) 앱 생성</div> <div>- API KEY = l7xxf2ba3e27a48f4985a324ee087ddbf15c</div> <div>- API Secret = ****</div> <div>2) 테스트페이지 다운로드(https://developers.open-platform.or.kr/guide/sdkdownload/2016041)</div> <div>3) kftcApiLocalTest_201808021047/WebContent/html/authorize2.html 페이지를 통해 본인인증 후 계좌생성</div> <div>- 계좌생성 정보 : 오픈(1234567890) / klaytn_open_account</div> <div>4) 본인인증 완료 후 code 값 기억!!</div> <div>http://20.41.79.145:3000/authResult?code=377cb167-adba-4e09-9336-9729fcb47d20&scope=login+transfer&client_info=test+whatever+you+want</div> <div>5) Authorization Code 입력을 통한 token 발급</div> <div></div> <div>- Access Token = 7f894032-dbc5-4c17-89e7-b227dc4edd75</div> <div>- Refresh Token = d391345b-9182-4ec4-a48f-a67f7ea4fc2d</div>
② 코딩으로 토큰발급 (①-2부터 5까지의 과정)	<div><pre>{ access_token: '0c437e4f-c4b1-4669-aabe-cc7b084c5d99', token_type: 'Bearer', expires_in: 7776000, refresh_token: '0fd27a0c-df3b-4c76-92ca-da29fffd3194e', scope: 'login inquiry transfer', user_seq no: '1100035504' }</pre></div> <div>0c437e4f-c4b1-4669-aabe-cc7b084c5d99</div>
③ test데이터 입력을 통한 출금이체(환전)	<div></div>
④ JJGO 토큰과 결합	6.에서 생성한 ERC20 기반 토큰인 JJGO와 결합해 토큰발급

8. 기타 참고사항

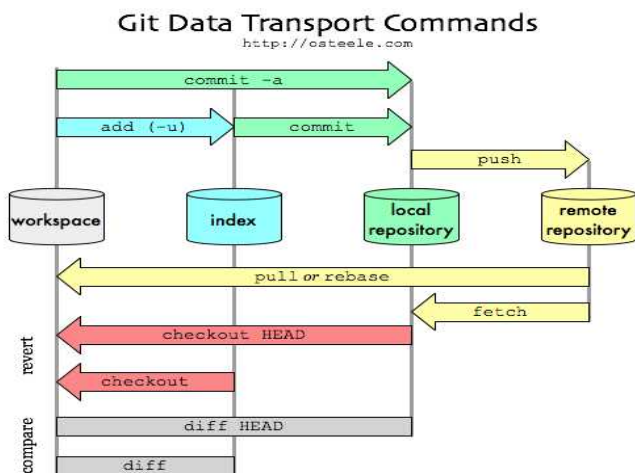
- git alias 설정하는 방법

```
$ vi ~/.gitconfig 파일 안에 아래 코드 넣으면 끝!  
st = status  
br = branch  
co = checkout  
ll = log --pretty=format:"%C(yellow)%h%Cred%dww %Creset%s%Cblueww [%cn]" --decorate --numstat  
ld = log --pretty=format:"%C(yellow)%hww %Cgreen%ad%Cred%dww %Creset%s%Cblueww [%cn]" --decorate --date=short --graph  
ls = log --pretty=format:"%C(green)%hww %C(yellow)%ad%Cred%dww %Creset%s%Cblueww [%cn]" --decorate --date=relative
```

- git ignore 설정하는 방법

```
$ vi ~/.gitignore 파일 안에 아래 코드 넣으면 끝!  
node_modules/  
build/  
*.swp  
*.swo
```

- 마크다운 문법 : <https://gist.github.com/ihoneymon/652be052a0727ad59601>



- python version change 하는 법 : <https://codechacha.com/ko/change-python-version/>

- python 삭제하는 법 : <https://ng1004.tistory.com/61>

- json parser = <http://json.parser.online.fr/>

- json.stringify와 parse의 차이점 = <https://blog.outsider.ne.kr/257>

pdf webpack 참고 = <https://github.com/blikblum/pdfkit-webpack-example>

nodejs 활용 open API 만들기 참고 블로그

→ return 값을 web page로 주냐, json형식으로 주냐의 차이

→ 입력방식은(GET, POST, PUT, DELETE) 선택이 가능하며, type도 지정 가능(express CORS 참고)

= <https://www.open-platform.or.kr/apt/content/openapi>

= <https://developers.open-platform.or.kr/openapi/restapi>

= <http://webframeworks.kr/tutorials/nodejs/api-server-by-nodejs-03/>

= <https://www.a-mean-blog.com/ko/blog/NodeJS-API-%EA%B8%B0%EB%B3%B8-REST-API-%EB%A7%8C%EB%B3%A4%EA%B8%B0>