EOSJS 활용하기

hexlant.
조성은

RPC 란?

Remote Procedure Call의 약자로 이름 그대로 다른 대상의 procedure를 호출할때 사용된다. 대부분의 블록체인 노드(비트코인, 이더리움)들은 노드들과 통신을 위해 JSON RPC API를 제공한다.

geth(ethereum) : https://github.com/ethereum/wiki/wiki/JSON-RPC

bitcoin : https://en.bitcoin.it/wiki/Original_Bitcoin_client/API_calls_list

eos : https://developers.eos.io/eosio-nodeos/reference

RPC API 직접 호출해보기

curl --request POST --url http://jungle2.cryptolions.io/v1/chain/get_info


```
{
    "server_version": "75635168",
    "chain_id": "038f4b0fc8ff18a4f0842a8f0564611f6e96e8535901dd45e43ac8691a1c4dca",
    "head_block_num": 10421907,
    "last_irreversible_block_num": 10421579,
    "last_irreversible_block_id": "009f054b3cf62a67808e4607ab60c034145898ff31919cb01e8b75946f323b2c",
    "head_block_id": "009f069352d629e193ab4b04bef2f92d73b5a8d41aea80b22699decd27142a71",
    "head_block_time": "2018-08-19T02:53:27.000",
    "head_block_producer": "blockgenesys",
    "virtual_block_cpu_limit": 200000000,
    "virtual_block_net_limit": 1048576000,
    "block_cpu_limit": 200000,
    "block_net_limit": 1048576
}
```

EOSJS?

Nodeos RPC 스펙을 구현한 JavaScript API입니다. NPM 노드 모듈로 설치 할 수 있으며
CDN 방식의 클라이언트 방식도 지원을 합니다. 이더리움의 web3js 와 유사하며 현재는 Javascript API 만 제공 합니다.

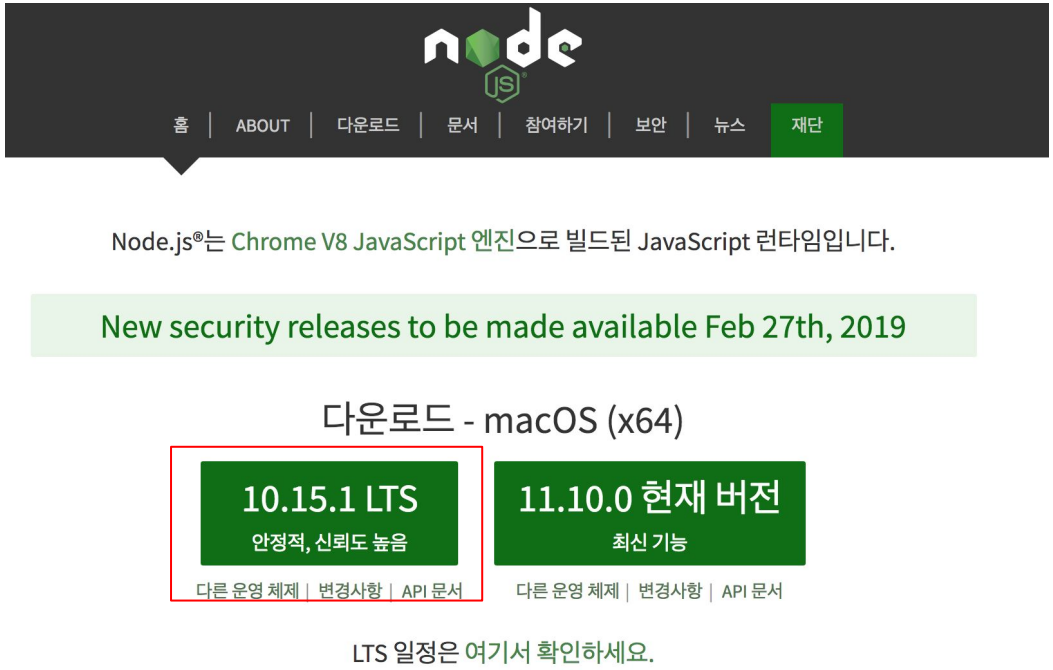
그럼 RPC 만 이용해도 가능한데 왜 EOSJS를 사용할까요?
만약 RPC 를 사용하여 EOS를 전송 할 경우 /v1/chain/get_info, /v1/chain/get_block, /v1/chain/push_transaction 3개의 API을 호출하고 해당 값을 조합하여 전송해야 하기 때문에 매우 복잡합니다. 이러한 이유로 transfer 와 같이 고수준의 API 를 제공하여 Dapp 개발을 조금 더 쉽게 도와 줍니다.

github : https://github.com/EOSIO/eosjs & gitbook : https://eosio.github.io/eosjs/

EOSJS 설치

nodejs 설치 (현재 10.15.1 추천)

- URL : https://nodejs.org/ko/



EOSJS 설치

서버사이드 방식 (NPM 사용) - 본 강의에서는 서버사이드 방식 사용

```
npm install eosjs
```

클라이언트 방식 (CDN 사용)

```
<script src="https://cdn.jsdelivr.net/npm/eosjs@16.0.7/lib/eos.min.js"
integrity="sha512-Lb1HeEaUSTxg0Y1KqlKOKeZCRi8XrEat+my3sWdg1INE81sUZNWogpfC9G
SIwPvON1Bqj5nQk2PyTE56lBR8ag==" crossorigin="anonymous"></script>
```

JsonRpc - 01_connectRpc.js

```
Just reading blockchain state only require an istance of 'JsonRpc'
connected to a node;
// < *npm install node-fetch >

const { JsonRpc } = require('eosjs');
const fetch = require('node-fetch');
const rpc = new JsonRpc('http://jungle2.cryptolions.io:80', { fetch
}};

async function getInfo() {
    const result = await rpc.get_info();
    console.log(result);
}

getInfo()
```

Api

To send transactions and trigger actions on the blockchain, you must have an istance of Api

```
const { Api, JsonRpc } = require('eosjs');
const { JsSignatureProvider } = require('eosjs/dist/eosjs-jssig');
const fetch = require('node-fetch');
const { TextDecoder, TextEncoder } = require('util');

const privateKeys = [ <privatekey> ];

const signatureProvider = new JsSignatureProvider(privateKeys);
const rpc = new JsonRpc('http://jungle2.cryptolions.io:80', { fetch });
const api = new Api({ rpc, signatureProvider, textDecoder: new TextDecoder(), textEncoder: new TextEncoder() });
```

Api (05_staking.js)

```
async function stake () {
   const result = await eos.transact({
       actions: [{
           account: 'eosio',
           name: 'delegatebw',
           authorization: [{
               actor: < Input account >,
               permission: 'active',
           }],
           data: {
                 from: '< Input Account >',
                 receiver: '< Input Account >',
                 stakenet: '< Input Balance >',
                 stakecpu: '< Input Balance >',
           }
       }]
   }, {
       blocksBehind: 3,
       expireSeconds: 30,
   })
   console.log('txID: ', result.transaction_id)
}
```

Api

```
async function stake () {
   const result = await eos.transact({
       actions: [{
           account: 'eosio',
           name: 'delegatebw',
           authorization: [{
               actor: < Input account >,
               permission: 'active',
           }],
           data: {
                 from: `< Input Account >',
                 receiver: '< Input Account >',
                 stakenet: '< Input Balance >',
                 stakecpu: '< Input Balance >',
           }
       }]
   }, {
       blocksBehind: 3,
       expireSeconds: 30,
   })
   console.log('txID: ', result.transaction_id)
}
```

setting.js

```
const { Api, JsonRpc } = require('eosjs');
const { JsSignatureProvider } = require('eosjs/dist/eosjs-jssig');
const fetch = require('node-fetch');
const { TextDecoder, TextEncoder } = require('util');
const privateKeys = [ < Input privateKeys > ];

const signatureProvider = new JsSignatureProvider(privateKeys);
const rpc = new JsonRpc('http://jungle2.cryptolions.io:80', { fetch
});
const eos = new Api({ rpc, signatureProvider, textDecoder: new
TextDecoder, textEncoder: new TextEncoder() });

module.exports = {
    eos
}
```

블록 정보 확인 - 02_get_block.js

```
const { eos } = require('./setting');

async function get_block (blockNumOrId) {
    const result = await eos.rpc.get_block(blockNumOrId);
    console.log(result);
}

get_block(1);
```

```
example $ node 02_get_block.js
{ timestamp: '2018-11-23T16:20:00.000',
  producer: '',
  confirmed: 1,
  previous: '0000000000000000000000000000000000000000000000000000000000000000',
  transaction_mroot: '0000000000000000000000000000000000000000000000000000000000000000',
  action_mroot: 'e70aaab8997e1dfce58fbfac80cbbb8fecec7b99cf982a9444273cbc64c41473',
  schedule_version: 0,
  new_producers: null,
  header_extensions: [],
  producer_signature: 'SIG_K1_111111111111111111111111111111111111111111111111111111111111111116uk5ne',
  transactions: [],
  block_extensions: [],
  id: '00000001a97e344b18ae9b1a33441ddeb98fc1d60bccdcc819dbe4949c01f7d2',
  block_num: 1,
  ref_block_prefix: 446410264 }
```

계정 정보 확인 - 03_get_account.js

```
const { eos } = require('./setting');

async function get_account(accountName) {
    const result = await eos.rpc.get_account(accountName);
    console.log(result);
}

get_account('hexeosjungle');
```

계정 정보 확인

```
example $ node 03_get_account.js
{ account_name: 'hexeosjungle',
  head_block_num: 41714201,
  head_block_time: '2019-07-30T05:14:10.500',
  privileged: false,
  last_code_update: '2019-07-19T06:29:21.000',
  created: '2019-07-11T10:12:35.000',
  core_liquid_balance: '145.8431 EOS',
  ram_quota: 242043,
  net_weight: 310000,
  cpu_weight: 310000,
  net_limit: { used: 729, available: 4187166, max: 4187895 },
  cpu_limit: { used: 2250, available: 1470022, max: 1472272 },
  ram_usage: 201869,
  permissions:
   [ { perm_name: 'active', parent: 'owner', required_auth: [Object] },
     { perm_name: 'owner', parent: '', required_auth: [Object] } ],
  total_resources:
   { owner: 'hexeosjungle',
     net_weight: '31.0000 EOS',
     cpu_weight: '31.0000 EOS',
     ram_bytes: 240643 },
  self_delegated_bandwidth:
   { from: 'hexeosjungle',
     to: 'hexeosjungle',
     net_weight: '31.0000 EOS',
     cpu_weight: '31.0000 EOS' },
  refund_request: null,
  voter_info:
   { owner: 'hexeosjungle',
     proxy: '',
     producers: [],
     staked: 2280000,
     last_vote_weight: '0.00000000000000000',
     proxied_vote_weight: '0.00000000000000000',
     is_proxy: 0,
     flags1: 0,
     reserved2: 0,
     reserved3: '0 ' } }
```

계정 정보 확인

```
account_name : lazylion5555    // 계정명
ram_quota: 45948               // RAM (byte 단위)
ram_usage: 3446                // 사용한 RAM
net_limit:                     // 계정이 가지고 있는 network bandwidth
cpu_limit:                     // 계정이 가지고 있는 cpu
permissions:                   // 계정의 퍼미션 정보
total_resources:               // 계정이 가지고 있는 전체 리소스(CPU,
Network)
// 다른 계정이 해당 계정에게 delegate 한 것도 포함.
voter_info:
    producers                  // 투표한 BP들 리스트
    proxy                      // 프록시 투표한 계정
```

EOS 잔고 확인 - 04_get_currency_balance.js

```
const { eos } = require('./setting');

// code: 컨트랙트 명    account: 계정 명    symbol: 심볼 ex) HEX

async function get_currecny_balance(code, account, symbol) {
    const result = await
eos.rpc.get_currency_balance(code,account,symbol)
    console.log(result);
}

get_currency_balance('eosio.token', 'hexeosjungle', 'EOS');
```

EOS Staking - 05_staking.js

```
const { eos } = require('./setting');

async function stake () {
    const result = await eos.transact({
        actions: [{
            account: 'eosio',
            name: 'delegatebw',
            authorization: [{
                actor: < Input Account >
                permission: 'active'
            }],
            data: {
                from: < Input Account >
                receiver: < Input Account >
                stakenet: < Input Amount >
                stakecpu: < Input Amount >
            }]
        }, {
                blocksBehind: 3,
                expireSeconds: 30,
        });
        console.log(result);
}
```

EOS Unstaking - 06_unstaking.js

```
const { eos } = require('./setting');

async function unstake () {
    const result = await eos.transact({
        actions: [{
            account: 'eosio',
            name: 'undelegatebw',
            authorization: [{
                actor: < Input Account >,
                permission: 'active'
            }],
            data: {
                from: < Input Account >,
                receiver: < Input Account >,
                unstakenet: < Input Amount >,
                unstakecpu: < Input Amount >
            }]
        }, {
            blocksBehind: 3,
            expireSeconds: 30,
        })
        console.log(result);
}
```

RAM 구매 - 07_buy_ram.js

```
const { eos } = require('./setting');

async function buyram () {
    const result = await eos.transact({
        actions: [{
            account: 'eosio',
            name: 'buyrambytes',
            authorization: [{
                actor: < Input Account>,
                permission: 'active',
            }],
            data: {
                payer: < Input Account >,
                receiver: < Input Account >,
                bytes: < Input Size >,
            }],
        }, {
            blocksBehind: 3,
            expireSeconds: 30,
        })
        console.log(result);
}
```

이오스 전송 - 08_transfer.js

```
const { eos } = require('./setting');

async function transfer () {
    const result = await eos.transact({
        actions: [{
            account: 'eosio.token',
            name: 'transfer',
            authorization: [{
                actor: < Input Account >
                permission: 'active',
            }],
            data: {
                from: < Input Account >
                to: < Input Account >
                quantity: < Input Amount >
                memo: < Input String >
            }]
        }, {
           blocksBehind: 3,
           expireSeconds: 30,
        })
        console.log(result);
}
```

EOS 전송 확인

```
example $ node 08_transfer.js
{ transaction_id: '94a412f6dee6b23abf9ef89f047e8ccb94ecc84f5c5573e4ac59e0e773bbc664',
  processed:
   { id: '94a412f6dee6b23abf9ef89f047e8ccb94ecc84f5c5573e4ac59e0e773bbc664',
     block_num: 41716448,
     block_time: '2019-07-30T05:33:09.000',
     producer_block_id: null,
     receipt: { status: 'executed', cpu_usage_us: 231, net_usage_words: 17 },
     elapsed: 231,
     net_usage: 136,
     scheduled: false,
     action_traces: [ [Object] ],
     account_ram_delta: null,
     except: null,
     error_code: null } }
```

https://jungle.bloks.io/transaction/94a412f6dee6b23abf9ef89f047e8ccb94ecc84f5c5573e4ac59e0e773bbc664

컨트랙트 호출

cleos 에서는 어떻게 컨트랙트의 Action을 호출하였는가?

```
./cleos.sh push action  contract transfer
'["fromAccount","toAccount","1.0000 CLUB","memo"]' -p
fromAccount@active
```

push action 을 사용하여 컨트랙트의 Action을 호출하였습니다.

https://developers.eos.io/eosio-nodeos/reference#push_transaction

contract 작성 - contracts/hello.cpp

```
// contracts/hello.cpp

#include <eosiolib/eosio.hpp>
using namespace eosio;
class hello : public eosio::contract {
 public:
     using contract::contract;

     [[eosio::action]]
     void hi( account_name user ) {
        print( "Hello, ", name{user} );
     }
};

EOSIO_ABI( hello, (hi) )
```

contract 컴파일

```
$ eosio-cpp -o hello.wasm hello.cpp -abigen
```

contract 배포

```
$ cleos set contract tutorialcall ./
```

컨트랙트 ABI 확인 - 09_get_abi.js

```
const { eos } = require('./setting');

async function get_abi (contract) {

    const result = await eos.rpc.get_abi(contract);

    console.log(JSON.stringify(result, null, 2);

}

get_abi(tutorialcall);
```

컨트랙트 ABI 확인 - 09_get_abi.js

```
example $ node get_abi.js
{
  "account_name": "tutorialcall",
  "abi": {
    "version": "eosio::abi/1.0",
    "types": [
      {
        "new_type_name": "account_name",
        "type": "name"
      }
    ],
    "structs": [
      {
        "name": "hi",
        "base": "",
        "fields": [
          {
            "name": "user",
            "type": "name"
          }
        ]
      }
    ],
    "actions": [
      {
        "name": "hi",
        "type": "hi",
        "ricardian_contract": ""
      }
    ],
    "tables": [],
    "ricardian_clauses": [],
    "error_messages": [],
    "abi_extensions": [],
    "variants": []
  }
}
```

cleos에서 컨트랙트 호출

```
$ cleos push action tutorialcall hi ‘[“hexlant”] -p tutorialcall
```

```
example $ cleos push action tutorialcall hi '["hexlant"]' -p tutorialcall
executed transaction: b991e26119aa384308956e5b1976db09d006751b99cdcc55113500def91649f7  104 bytes  183 us
#  tutorialcall <= tutorialcall::hi            {"user":"hexlant"}
>> Hello, hexlant
warning: transaction executed locally, but may not be confirmed by the network yet         ]
```

https://jungle.bloks.io/transaction/b991e26119aa384308956e5b1976db09d006751b99cdcc55113500def91649f7

컨트랙트 콜 - 10_contract_call.js

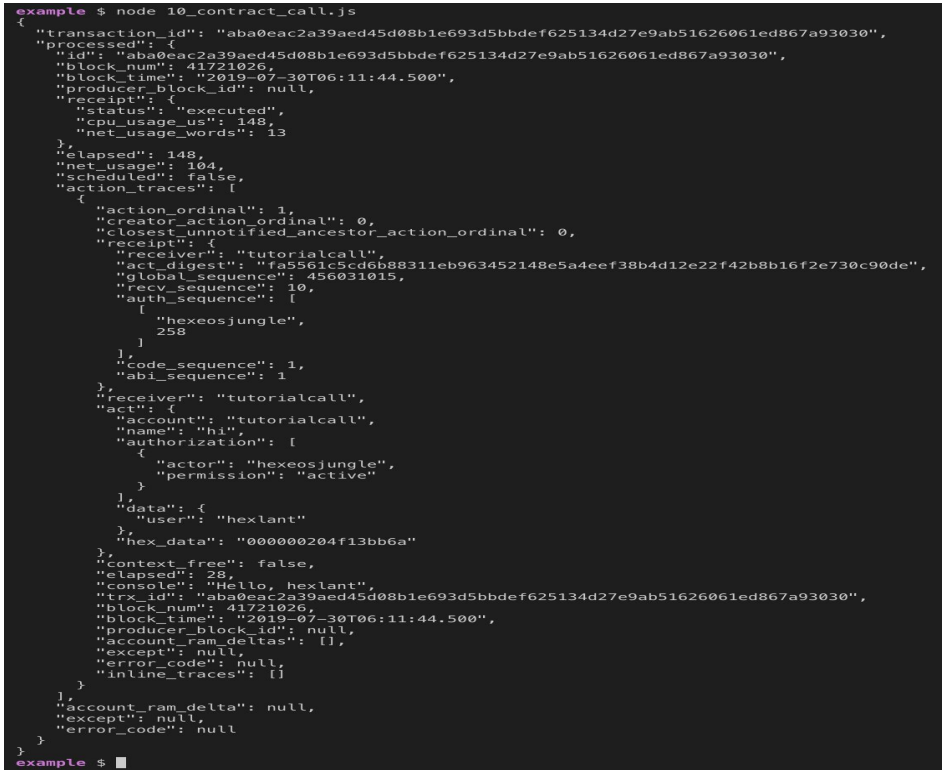
```
const { eos } = require('./setting');

async function invoke () {
    const result = await eos.transact({
        actions: [{
            account: < Input Contract >,
            name: < Input Action >,
            authorization: [{
                actor: < Input Account >
                permission: 'active',
            }],
            data: {
                < Input Actions Parameter >
            }],
        }, {
           blocksBehind: 3,
           expireSeconds: 30,
        })
        console.log(result);
}
```

그럼 Eosjs를 이용하여 Contract Call을 해봅시다

```
const { eos } = require('./setting');

async function invoke () {
    const result = await eos.transact({
        actions: [{
            account: 'tutorialcall',
            name: 'hi',
            authorization: [{
                actor: 'hexeosjungle'
                permission: 'active',
            }],
            data: {
                user: 'hexlant'
            }],
        }, {
           blocksBehind: 3,
           expireSeconds: 30,
        })
        console.log(result);
}
```



https://jungle.bloks.io/transaction/aba0eac2a39aed45d08b1e693d5bbdef625134d27e9ab51626061ed867a93030

cleos에서 계정 생성

```
$ cleos system newaccount < creator > < newAccount> <Ownerkey>
[<ActiveKey> --stake-net < amount > --stake-cpu < amount >
--buy-ram-kbytes < amount >
```

```
$ cleos system newaccount hexeosjungle mynewaccount
EOS6iugFh9id7UPuEovcbRjPfFerqbg3y4g7Wr95sd4ABdNRNMrKh
EOS77BZra7AJbRNd9CfUppcbSzBKWqpdCZJV3mEck5P55Kz8Zcrk3
--stake-net “1.0000 EOS” --stake-cpu “1.0000 EOS” --buy-ram-kbytes 3
```

계정 생성 - 11_create_account.js

```
const { eos } = require('./config');

async function createAccount () {
   const result = await eos.transact({
       actions: [{
         account: 'eosio',
         name: 'newaccount',
         authorization: [{
           actor: config.actor,
           permission: 'active',
         }],
         data: {
           creator: config.creator,
           name: config.name,
           owner: {
             threshold: 1,
             keys: [{
               key: config.ownerkeys,
               weight: 1
             }],
...
```

계정별 히스토리 보기

```
const { eos } = require('./config');

async function get_history_action (account, pos, offset ) {
    const result = await eos.rpc.history_get_actions(account, pos, offset);
    console.log(result);
}
```

계정별 히스토리 보기 결과


```
"act": {
    "account": "eoscointoken",
    "name": "transfer",
    "authorization": [
        {
            "actor": "lazylion5555",
            "permission": "active"
        }
    ],
    "data": {
        "from": "lazylion5555",
        "to": "lazylion1231",
        "quantity": "1.0000 CLUB",
        "memo": "memo"
    },
    "hex_data": "504a2993bae8bf8910860893bae8bf89102700000000000004434c5542000000046d656d6f"
},
"elapsed": 5,
"cpu_usage": 0,
"console": "",
"total_cpu_usage": 0,
"trx_id": "a29a17d2807368c5a905ff84c99e34d0f413f23c71c55d19536032430bdbfb42",
"inline_traces": []
```

거래소는 입/출금을 어떻게 처리 할까요?

1. 이오스 계정 생성비용은 대략 3천원정도 필요합니다.

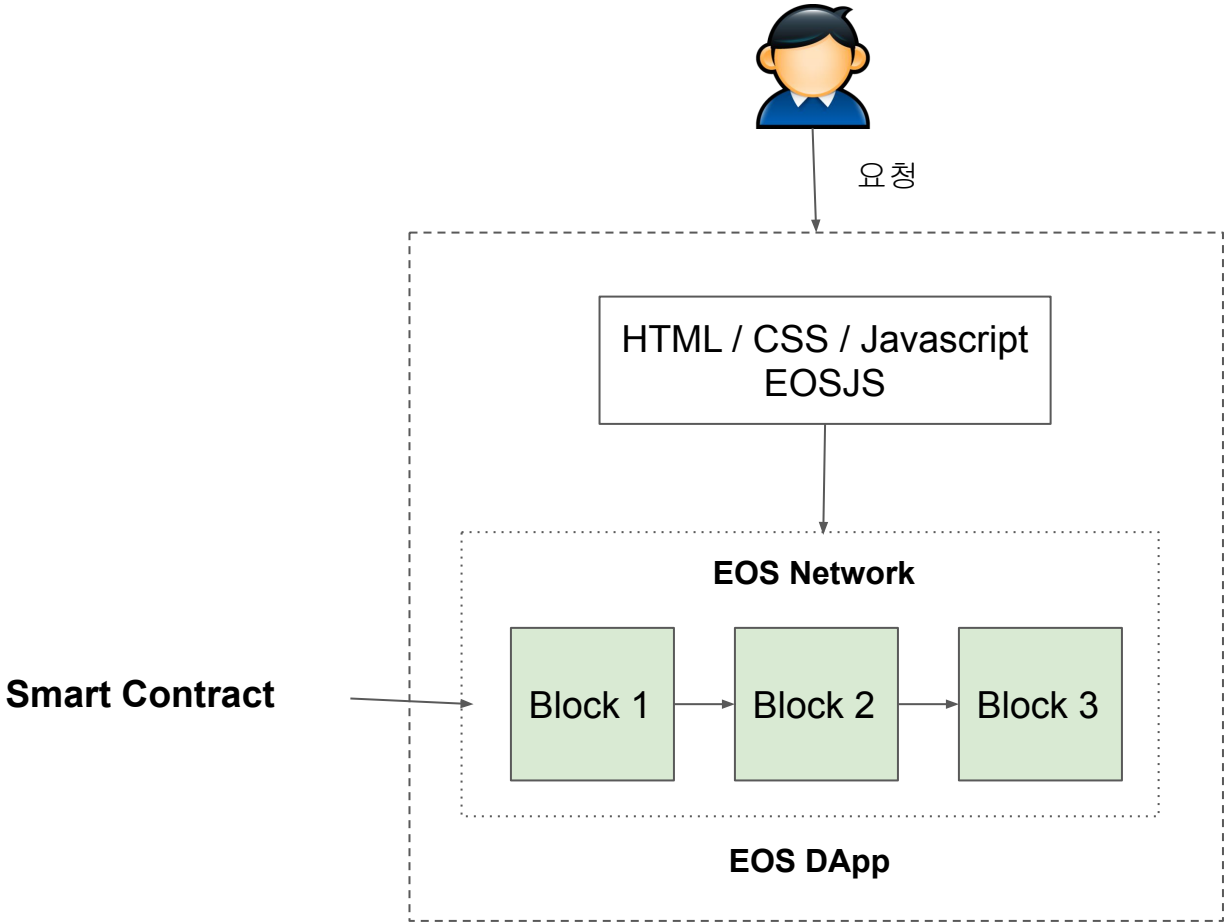
2. 거래소들은 유저마다 계정을 생성 할 수 없는 상황입니다.
   계정 생성비용에 대한 문제를 해결하기 위하여 전송시에 memo 를 이용하여
   입금 처리를 하고 있습니다.
   ex) https://eosflare.io/account/tokenbankpro

   - 이와 비슷한 사례로는 리플이 있습니다. 리플 또한 계정 생성비용에 20 리플이 필요합니다.
   이오스와 비슷하게 destination tag 를 이용하여 유저를 구분합니다.

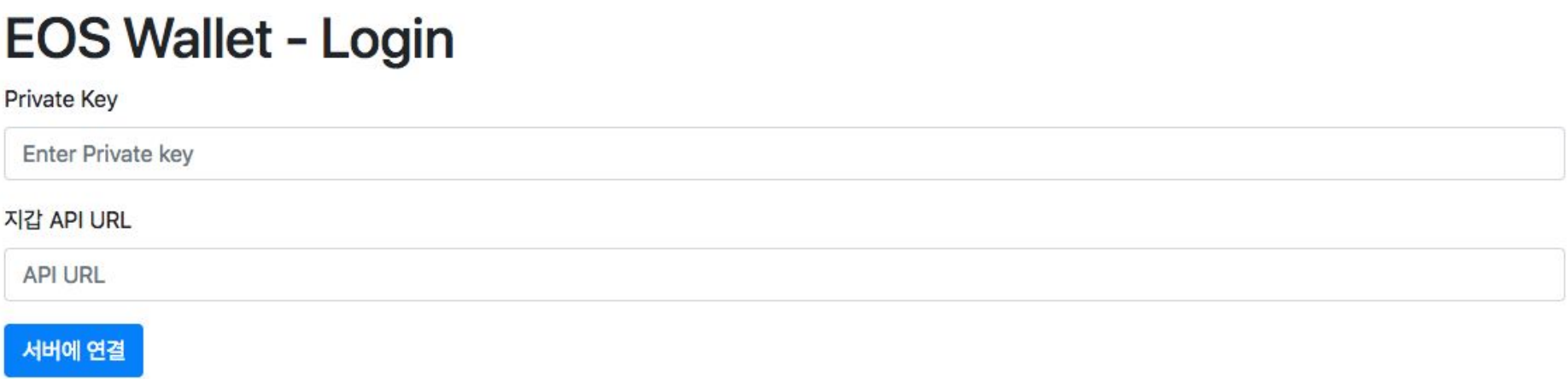
3. 이더리움과 달리 이오스는 계정별로 트랜잭션 히스토리 API 를 지원하기 때문에 히스토리 API 를 통해 계정별로 입금처리를 수월하게 진행 할수 있습니다.

Q. DApp과 SmartContract의
차이점은 무엇일까요?

DApp?


요청
HTML / CSS / Javascript
EOSJS
EOS Network
Smart Contract
Block 1
Block 2
Block 3
EOS DApp

실습 - 이오스 지갑 만들어보기




```
npm install
sudo npm install -g nodemon
```