# Chapter 14 웹3 디앱

웹3 디앱은 웹에서 작동하는 블록체인 기반 애플리케이션이다. 블록체인에서 계정 주소와 잔고를 읽고, 이벤트를 통해 잔고의 변화를 관찰하는 작업을 웹에서 구현해보자. 웹지갑을 연결하여, 주소를 읽고 그 주소에서 원격에서 블록체인과 통신할 수 있게 한다. 또한 블록체인에서 투표하는 웹3 디앱을 구현하게 된다.

# 1. 웹에서 블록체인 연결하기

웹디앱 (Web DApp)은 웹에서 작동하는 블록체인 기반의 분산형 애플리케이션이다. 먼저 웹에 연결이 되어야겠다. 웹은 요청이 있으면, 중앙에서 처리하는 방식을 따른다. 데이터도 중앙에서 제공되고, 처리도 중앙에서 이루어진다. 이러한 중앙 중심적인 웹에 변화가 있다. 분산이 그것이다. 이른바 'Web 3.0' 또는 Web3으로 명명되고, 기존의 웹과는 달리 데이터, 처리능력을 어느 한 중앙에 두지 않고 분산하는 것이다.

더 알아보기: **Web3**

Web 3.0은 블록체인에 기반을 둔 차세대 월드와이드웹 World Wide Web을 말한다. Web 3.0이 아직 명확히 정의된 것은 물론 아니다. 웹을 처음 소개한 팀 버너스리는 시맨틱웹을 제안하기도 한다. Web 2.0은 중앙처리모델에 따라 작동되고, 사용자가 직접 참여하는 특징을 가진다. SNS에 글을 쓰고, YouTube에 영상을 올리고 하지만, 이런 서비스가 소수의 '빅 테크' 기업들에 의해 제공되고 장악되고 있다. Web3은 반대의 모델을 따르고 있다. 중앙에서 처리하는 그래서 누구의 것도 아닌 웹이 소수에 의해 독점되는 Web 2.0이 아니라 Web 3.0은 분산모델에 따라 작동한다. 우리가 곧 배우게 될 라이브러리도 명칭이 Web3이다. Web 3.0을 줄여서 Web3이라고 명명하고 웹 분산처리방식의 시작을 오래 전부터 알리고 있다.

## 1.1 웹 개발에 앞선 환경의 준비

지금까지 차근차근 배워왔다면 웹개발의 환경준비는 이미 되어 있을 것이다. 그래도 web3.js와 provider가 준비되었는지 확인하자.

### Web3.js 라이브러리를 준비

웹에서 블록체인에 접근하기 위해서는 라이브러리가 필요하다. 대표적인 라이브러리로는 Web3.js, Ethers.js를 꼽을 수 있는데, 여기서는 먼저 개발되어 널리 사용되고 있는 Web3.js를 사용한다.

### Web3 provider를 준비

Web3 provider는 이더리움 또는 다른 블록체인 네트워크와 상호 작용하기 위한 인터페이스를 제공하는 객체이다.

Web3 provider는 이들을 중간에서 이어주는 다리 역할을 한다. 웹3에서 블록체인에 데이터를 provider에게 요청해야 한다. 그러면 provider가 블록체인에 대신해서 요청을 전달한다. provider가 없으면, 이더리움 노드와 블록체인이 직접 상호작용할 수 없다.

Web3.js 라이브러리를 사용하려면, 먼저 provider 객체를 인식해야 한다. 이를 통해 원격 웹에서 블록체인의 계정 주소를 읽거나 스마트 컨트랙을 호출거나 거래를 전송하는 등의 여러 후속 작업을 수행할 수 있기 때문이다.

provider 객체는 로컬 또는 제3자에게서 제공될 수 있다.

* 자신이 준비한 로컬의 provider를 사용할 수 있다. 이를 위해서는 geth 네트워크를 실행하고, HTTP 또는 Websocket provider를 설정해야 한다. 자신이 사용하게 될 provider를 제공하는 노드 ganache@8345 또는 geth@8445를 띄어놓자.
* 가장 널리 사용되는 MetaMask 또는 ConsenSys에서 제공하는 Infura provider같은 제 3자 네트워크를 사용할 수 있다. Infura는 테스트망, 공중망에 대해 블록체인 네트워크 기능을 제공하고 있다.

## 1.2 웹서버

웹에서 블록체인을 사용하기 위해서는 웹서버가 필요하다. 혹시 의아하게 들릴 수 있다.

분명하게 알아두자. 웹페이지를 띄우기 위해 중앙집중형 웹서버가 필요한 것이고, 웹페이지에서 버튼을 클릭하는 등의 거래는 블록체인이 웹서버 역할을 하면서 분명 분산처리된다.

웹서버는 보통 80번 포트에서, 서비스의 요청 http requests을 받고, 이를 처리한 후 요청한 측에 responses를 돌려주는 기능을 한다. 여기서는 가벼운 Python의 웹서버를 띄워서 사용하기로 한다.

Python의 웹서버를 띄우기 위해서는 python -m http.server 명령을 한다.

명령어를 실행하는 위치는 프로젝트 디렉토리로 한다. 프로젝트 디렉토리 기준으로 상대 경로의 html을 읽기 위해서 그렇다.

* 기본 포트는 8000번을 사용하지만, 보안이 걱정되면 다른 포트를 사용해도 된다. 이더리움에서 사용하는 포트는 45로 끝나고 있으니 맞추어 8045를 사용해도 좋다.
* IP 주소는 127.0.0.1 또는 고정 IP를 쓴다.

명령창의 프로젝트 디렉토리로 이동해서 (pjt\_dir> 로 적고 있다), 다음과 같이 적어서 웹서버를 띄운다.

pjt\_dir> python -m http.server  
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/)  
  
또는 포트와 IP를 지정한 경우  
pjt\_dir> python -m http.server 8045 --bind 117.xxx.xx.xx  
Serving HTTP on 117.16.xx.xx port 8045 (http://117.xxx.xx.xx:8045/)  
... 서버 로그가 출력된다.

그리고 웹브라우저를 열고, http://127.0.0.1:8000/ 또는 http://localhost:8000으로 접속하면 된다. 포트를 8045로 변경했다면 8000 대신 http://localhost:8045를 사용한다.

## 1.3 index.html

웹 서버가 작동되는지 확인하기 위해 index.html 파일을 프로젝트 디렉토리 아래에 저장한다. 주의하자. 웹서버를 실행시킨 디렉토리에 저장되어야, index.html 파일이 인식된다.

[파일명 index.html]  
Hello Blockchain!

어떤 웹 브라우저라도, 선택하여 다음과 같이 URL을 입력한다. index.html 파일명은 생략되었지만 해당 URL에서는 그 파일을 찾기 때문에 우리가 방금 만든 그 파일을 열게 된다.

http://127.0.0.1:8000/ 이렇게 웹페이지를 열면 http://127.0.0.1:8000/index.html을 열게 된다.

## 1.4 웹에서 coinbase 출력

버튼을 누르면, 계정 주소를 읽어오도록 웹페이지를 제작해보자.

[파일명: scripts/testAccount.html]  
줄01 <!doctype html>  
줄02 <html>  
  
줄03 <head>  
줄04 <script src="https://cdn.jsdelivr.net/npm/web3@1.2.5/dist/web3.min.js"></script>  
줄05 <!-- script src="https://cdn.jsdelivr.net/npm/web3@0.20.5/dist/web3.min.js"></script -->  
줄06 </head>  
  
줄07 <body>  
줄08 <button onclick="myFunction()">Click me to see the coinbase.</button>  
줄09 <p id="coin"></p>  
  
줄10 <script>  
줄11 var web3 = new Web3(new Web3.providers.HttpProvider("http://127.0.0.1:8345"));  
  
줄12 async function myFunction() {  
줄13 const account = await web3.eth.getAccounts();  
줄14 document.getElementById("coin").innerHTML = "coinbase: " + account[0];  
줄15 }  
줄16 </script>  
  
줄17 </body>  
줄18 </html>

웹브라우저에 띄우려면, 입력할 url은 http://localhost:8045/scripts/testAccount.html 이다. 페이지를 만들고 나서는, 웹브라우에 url을 입력하여 올바르게 뜨는지 확인해야 한다.

위 코드에서 이해가 필요한 필요한 부분에 대해 설명을 해보자.

### 줄1:

모든 HTML 문서는 <!DOCTYPE html> 선언을 한다. 웹 브라우저에게 해당 문서가 HTML5 버전의 HTML 문서임을 알려주는 문서 형식 선언 (Document Type Declaration)이다.

<!DOCTYPE>은 HTML의 태그는 아니다. 웹 브라우저에서 인식해서 어떤 문서인지 파악하고 브라우저가 적절하게 맞춰 화면에 보이도록 만드는 태그라고 이해하자.

HTML5부터는 html문서는 <!DOCTYPE html>이라고 선언하며, 대소문자 무관하다.

더 알아보기: <!DOCTYPE html>

잠깐 <!DOCTYPE>이라고만 해주면, window.ethereum 객체가 생성이 되지 않는다! 이 간단한 오류를 바로 잡기 위해 많은 시간이 필요했다고 고백하니 주의한다! HTML 문서가 브라우저에서 제대로 작동하지 않거나 예상대로 표시되지 않는다면, 이 선언부를 누락한 것이 원인 중 하나일 수 있으니, HTML 문서의 첫 부분에 <!DOCTYPE html>을 추가하여 문제를 해결하자!

### 줄4: web3js 라이브러리

우리가 만드는 웹페이지는 블록체인과 연결하고 사용하기 위해 web3.js 라이브러리를 사용한다.

웹서버에서 띄우는 페이지에 require명령어를 사용하지 않도록 한다. require(Web3)라고 적으면 오류가 발생한다. require는 node.js에서 라이브러리를 호출할 때 사용하는 명령어이다.

웹브라우저에서 라이브러리를 사용할 경우, <script src"..."></script>이렇게 하면 된다. 이 때, web3.js가 설치된 상대 경로를 쓰거나 web3.js의 CDN 주소를 적어도 된다.

1. web3.js의 상대 경로를 적는 방법

웹페이지에서 javascript 라이브러리를 사용할 경우, 경로에 주의해서 포함해야 한다. web3.js 라이브러리는 **현재 HTML파일을 기준으로 상대 경로**를 적어준다. 예를 들어, testAccount.html을 기준으로 상대경로 src="../node\_modules/web3/dist/web3.js"를 적어준다.

<script type="text/javascript" src="../node\_modules/web3/dist/web3.js"></script>

아래 경로에서 보면 testAccount.html의 상위로 올라가고 거기서부터 web3.js가 있는 경로를 따라가 보자. 그 경로를 적어주면 되는 것이다.

프로젝트 디렉토리 pjt\_dir\  
 |  
 ---> index.html  
 |  
 ---> \_gethNow.sh  
 |  
 ---> node\_modules\web3\dist\web3.js  
 |  
 ---> scripts\  
 |  
 ---> testAccount.html  
 |  
 ---> src\  
 |  
 ---> Hello.sol

1. CDN을 적어도 된다.

web3.js를 직접 저장하지 않고, 외부 사이트로부터 가져와 사용할 수 있다. 자바스크립트의 CDN Content Delivery Network은 대표적으로 Google CDN, Cloudflare CDN, jsDelivr 등이 있으며, 버전은 정해서 적어주면 된다 (참조 https://github.com/ChainSafe/web3.js).

<script src="https://cdn.jsdelivr.net/npm/web3@0.20.5/dist/web3.min.js"></script>  
또는  
<script src="https://cdn.jsdelivr.net/gh/ethereum/web3.js@latest/dist/web3.min.js"></script>

Cloudflare에서 제공하는 아래 사이트를 사용할 수도 있으니, 편리한 것을 골라서 사용하면 된다. https://cdnjs.cloudflare.com/ajax/libs/web3/1.2.7/web3.min.js

web3.js는 1.0 이전과 그 이후로 API가 많이 변화했다. 줄5에 주석 처리 했지만, 옛 버전인 0.20을 사용하려면, <script src="https://cdn.jsdelivr.net/npm/web3@0.20.5/dist/web3.min.js"></script> 를 적는다.

현재는 웹3 버전 1.2.5을 사용한다. 버전을 교체하여 이전 0.20.x를 적용하려면 줄4를 주석으로 하고 대신 줄5의 주석을 제거하면 된다.

<script src="https://cdn.jsdelivr.net/npm/web3@1.2.5/dist/web3.min.js"></script>

### 줄11: 로컬 Web provider를 사용한다

HttpProvider("http://127.0.0.1:8345")를 사용한다. 이를 위해 ganache@8345를 미리 띄워놓아야 한다.

HTTP header의 설정을 아래와 같이 따로 해둘 수 있지만, 보통 생략하고 기본을 적용한다.

var options = {  
 keepAlive: true,  
 ...  
 timeout: 20000, // 밀리세컨드  
 headers: [  
 {  
 name: 'Access-Control-Allow-Origin',  
 value: '\*'  
 },  
 {  
 ...  
 }  
 ],  
 agent: {  
 http: http.Agent(...),  
 baseUrl: ''  
 }  
};

이런 설정을 넣어서 HttpProvider를 생성할 수 있다.

Web3.providers.HttpProvider("http://127.0.0.1:8345", options)

### 줄12 ~ 15: 비동기 함수로 myFunction()을 클릭하면 호출된다.

줄8의 버튼을 클릭하면 myFunction()을 호출한다.

web3.js 버전이 1.0으로 올라가면서, 명령어에 변화가 있다. 전과 달리, 명령어를 비동기적으로 처리해야 한다. 0.2 버전에서는 계정을 단순하게 호출하면 되었다.

const coinbase = web3.eth.accounts[0];

1.0이상에서는 비동기적으로 계정 주소를 읽어오기 위해 async, await 명령어를 사용해야 한다.

const account = await web3.eth.getAccounts();

### 자바스크립트 콘솔에서 디버깅

개발하면서 파일들을 웹브라우저에서 열어보면, 항상 제대로 동작하지 않을 수 있다. 그러면 디버깅을 하기 위해 자바스크립트 개발자콘솔을 열어볼 필요가 있다.

브라우저별로 개발자콘솔을 여는 방식이 다르다.

* 구글 크롬: 메뉴에서 'More Tools' > 'Developer Tools' 또는 단축키 "Shift+Ctrl+J"를 누르면 된다.
* Edge: 구글과 동일하게 More Tools > Developer Tools 또는 단축키 "Shift+Ctrl+I"를 누른다.
* Safari: 환경설정 > 메뉴막대에서 개발자용 메뉴 보기

열면 HTML페이지에서 오류가 발생했는지, 그 이유가 무엇인지 알아볼 수 있다. 상단의 탭 가운데 하나인 콘솔을 클릭하고 열면, > 프롬프트가 뜬다. 프롬프트에서 스크립트를 입력하면서 변수의 값을 확인하거나 디버깅할 수 있다.

javascript> console.log("hello");  
> hello console.log의 출력  
> document.body.style.backgroundColor = "lightblue"; 페이지 배경색을 변경한다

또한 콘솔에서 web3의 API를 테스트할 수 있다. testAccount.html 웹페이지가 열리면서, web3가 심어지기 때문에 자바스크립트 콘솔에서 이 명령어를 사용할 수 있게 된다. 키를 누르면서 web3에서 제공되는 함수들이 어떤 것이 있는지 알아 볼 수 있으며, 관련 명령어를 연습하기도 편리하다.

간단한 명령어 몇가지를 연습해보자.

javascript> window.web3 (또는 web3라고 입력해도 동일한 기능을 한다)  
> Proxy(Object) {currentProvider: Proxy(l), \_\_isMetaMaskShim\_\_: true}  
> web3.version (버전을 확인한다)  
1.2.5

계속해서, web3 함수를 연습해보자.

javascript> window.web3.eth.getAccounts() (web3.eth.getAccounts()로 해도 된다. Promise를 반환하고, 삼각형을 클릭하면 계정배열을 조회할 수 있다)  
[[PromiseResult]]: Array(5)  
0: "0xC8Ea4C4e655F8152aDC075a649AA7ec35564C7C0"  
1: "0x4fa2C7caac80A8518264d263BDB5ed74f1A6F398"  
2: "0xCE5Cd78E8d75236F101a9036BC81663D684166A6"  
3: "0x6F2F9f550A3ef6Ce75727D1aC180e699B8c3bB0f"  
4: "0x6F94A3529222ad7b67305c30928df0fd2D70966d"  
length: 5  
  
> web3.eth.getCoinbase().then(console.log)  
> Promise {<pending>}  
0x8078e6bc8e02e5853d3191f9b921c5aea8d7f631

더 알아보기: **window object**

window object는 웹 브라우저에서 전역 객체이고, 브라우저의 윈도우를 의미한다. 모든 자바 객체, 함수, 변수 모두 window객체의 멤버가 된다.예를 들어, console은 window에서 열린 console을 말하며, 'window.console' 또는 그냥 'console'이라고 해도 된다. 로그를 쓰려면window.console.log() 또는 `console.log()이라고 한다. HTML문서가 브라우저에서 열리면 document 객체 (DOM, Document Object Model)가 되는데, 이도 마찬가지이다. window.document 또는 그냥 document 같은 의미이다. window.document.getElementById("header")는 document.getElementById("header")와 같은 의미이다.

## 1.5 웹에서 잔고 출력

web3.js 라이브러리에서는 보통 Wei를 BigNumber 객체로 반환한다.

wei는 ether이다. 자바스크립트에서 wei를 표현하면서 문제가 발생할 수 있다.

예를 들어, web3.eth.getBalance 함수를 사용하여 계정의 잔고를 가져오는 경우, 해당 잔고는 BigNumber 객체로 반환된다. BigNumber 객체를 사용하여 원하는 형식으로 변환하거나 계산할 수 있다.

web3.eth.getBalance(coinbase)는 wei를 (eth가 아니다), 문자열로 출력한다. BigNumber로 출력하려면 toBN() 함수를 사용한다. 큰 숫자로 연산을 해보자. 100 ETH를 Wei로 변환하면 이므로 1e+20이다.

toWei(), fromWei() 함수는 String 또는 BN을 인자로 받지만, 숫자를 넘겨주면 pass numbers as strings or BN objects to avoid precision errors. 이런 오류를 만나게 된다.

> web3.utils.toWei(10, 'ether')  
Error: Please pass numbers as strings or BN objects to avoid precision errors.  
 at Object.toWei  
> web3.utils.toWei(String(10), 'ether') 문자열로 넘겨준다.  
'10000000000000000000' 문자열로 출력한다

아래 프로그램을 웹에서 로딩해 출력을 확인해보자.

[파일명: scripts/testBigNumber.html]  
줄01 <!doctype html>  
줄02 <html>  
  
줄03 <head>  
줄04 <script src="https://cdn.jsdelivr.net/npm/web3@1.2.5/dist/web3.min.js"></script>  
줄05 <!-- script type="text/javascript" src="node\_modules/bignumber.js/bignumber.min.js"></script -->  
줄06 <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/bignumber.js/9.0.1/bignumber.min.js">줄01 줄07 </script>  
줄08 </head>  
  
줄09 <body>  
줄10 <button onclick="getWEI()">Click me to see the balance.</button>  
줄11 <p id="coin"></p>  
줄12 <p id="coin0"></p>  
줄13 <p id="wei"></p>  
줄14 <p id="weiBN"></p>  
줄15 <p id="weiBigNumber"></p>  
줄16 <p id="weiString"></p>  
줄17 <p id="eth"></p>  
  
줄18 <script type="text/javascript">  
줄19 var web3 = new Web3(new Web3.providers.HttpProvider("http://127.0.0.1:8345"));  
  
줄20 async function getWEI() {  
줄21 var coinbase = await web3.eth.getCoinbase(); //getCoinbase  
줄22 document.getElementById("coin").innerHTML = "coinbase: " + coinbase;  
  
줄23 const account = await web3.eth.getAccounts();  
줄24 coinbase = account[0];  
줄25 document.getElementById("coin0").innerHTML = "coinbase now from account0: " + coinbase;  
  
줄26 var balanceWei = await web3.eth.getBalance(coinbase); //BigNumber 객체를 반환  
줄27 console.log(balanceWei \* (10\*\*10));  
줄28 var balanceWeiBN = web3.utils.toBN(balanceWei);  
줄29 document.getElementById("wei").innerHTML = "wei: " + balanceWei;  
줄30 document.getElementById("weiBN").innerHTML = "wei BN: " + balanceWeiBN\*(10\*\*100);  
줄31 document.getElementById("weiBigNumber").innerHTML = "wei BigNumber: " + BigNumber(balanceWeiBN\*(10\*\*100));  
줄32 document.getElementById("weiString").innerHTML = "wei in string: " + balanceWei.toString(); //no need toString  
줄33 var balanceEth = web3.utils.fromWei(balanceWei, 'ether');  
줄34 document.getElementById("eth").innerHTML = "eth: " + balanceEth;  
줄35 }  
줄36 </script>  
  
줄37 </script>  
줄38 </body>  
줄39 </html>

* 줄21 비동기적으로 실행되어 coinbase를 가져온다. coinbase는 geth@8445에서는 올바르게 설정되지만, gananche@8345에서는 0으로 설정된다.
* 줄23 ganache@8345는 10개의 계정주소를 제공한다. 줄24에서 그 배열의 0번째를 coinbase로 설정한다.
* 줄26 비동기적으로 실행되어 지갑의 잔액을 가져와 콘솔에 출력한다.
* 줄28 특정 주소(coinbase)의 잔고를 BigNumber 객체로 변환한다. 이렇게 하면 잔고를 안전하게 다룰 수 있다.
* 줄30 일부러 아주 큰 값, 또한 10의 100 제곱을 더 곱해주면 1e+120, 자릿수가 큰 숫자가 된다.

# 2. 메타마스크(MetaMask)

블록체인의 거래를 완성하기 위해서는 디지털사인이 필요하고, 사용자의 키 또한 필요하다. 원격 웹에서의 거래, 송금을 하거나 투표를 하는 경우에는 자신의 키를 어디에서 가져올 것인지 생각해봐야 한다. 지금까지 사용해오던 로컬지갑보다는 웹 지갑이 편리하고, 보안을 위해서는 마땅히 그래야 한다.

웹 지갑은 사용자가 편리하게 블록체인 거래를 할 수 있도록 도와주지만, 이 과정에서 보안 문제가 발생할 수 있다. 예를 들어, 웹브라우저에서 키를 읽어 온다고 하자. 웹서버가 온라인 상태라면 사용자의 개인 키가 서버에 저장되거나 노출될 수 있는 위험이 있다.

메타마스크와 같은 웹 지갑은 사용자의 개인 키를 안전하게 관리하고, 거래를 사인하고 암호화하는 과정은 사용자의 승인이 있어야 가능하고 로컬 컴퓨터에서만 이루어져서, 외부로 노출되는 위험이 줄어든다.

## 2.1 메타마스크 설치

메타마스크는 자신의 이더리움 노드가 필요없어 로컬에 이더리움을 설치하거나 동기화할 필요가 없다.

간단히 크롬이나 Edge 브라우저의 확장 프로그램으로 설치하여 사용할 수 있다.

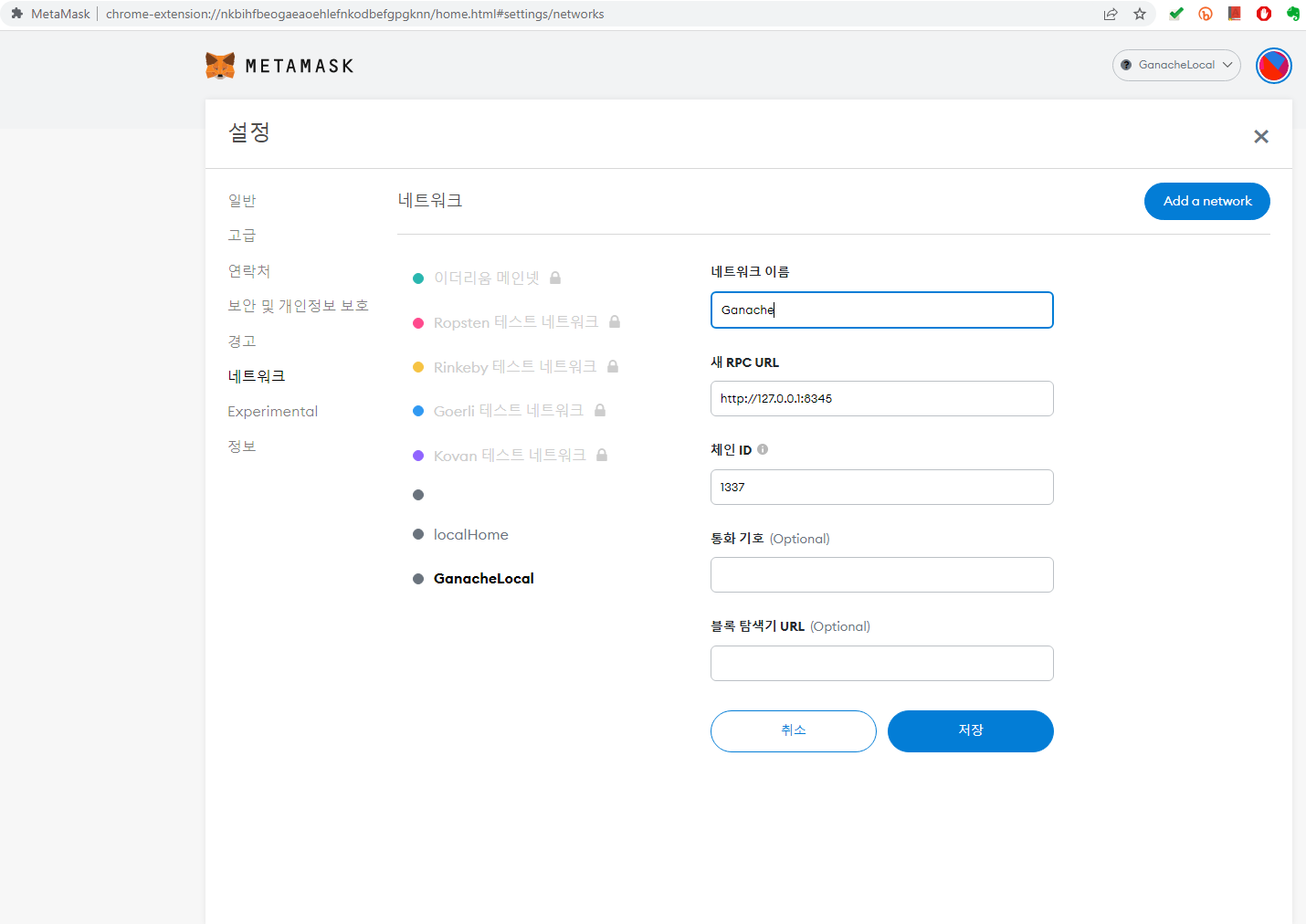
이 때 12 단어를 입력해야 한다. 이 구문은 시드(seed)로 이를 암호화하여 키를 생성하게 된다. 이 구문은 키 복구 구문(Secret Recovery Phrase)으로, 메타마스크를 삭제했거나 문제가 있어서 키를 복구할 때 사용될 수 있다.

## 2.2 메타마스크 연결

### 단계 1: 네트워크 추가

메타마스크에 자신의 Ganache 네트워크를 추가해보자. 에서 보듯이 네트워크 이름은 자신이 인식할 수 있는 명을 입력하고, URL은 앞에서 본 것처럼 localhost에 포트 번호를 추가해서 입력한다.

ganache 기본 Chain ID는 1337이다.



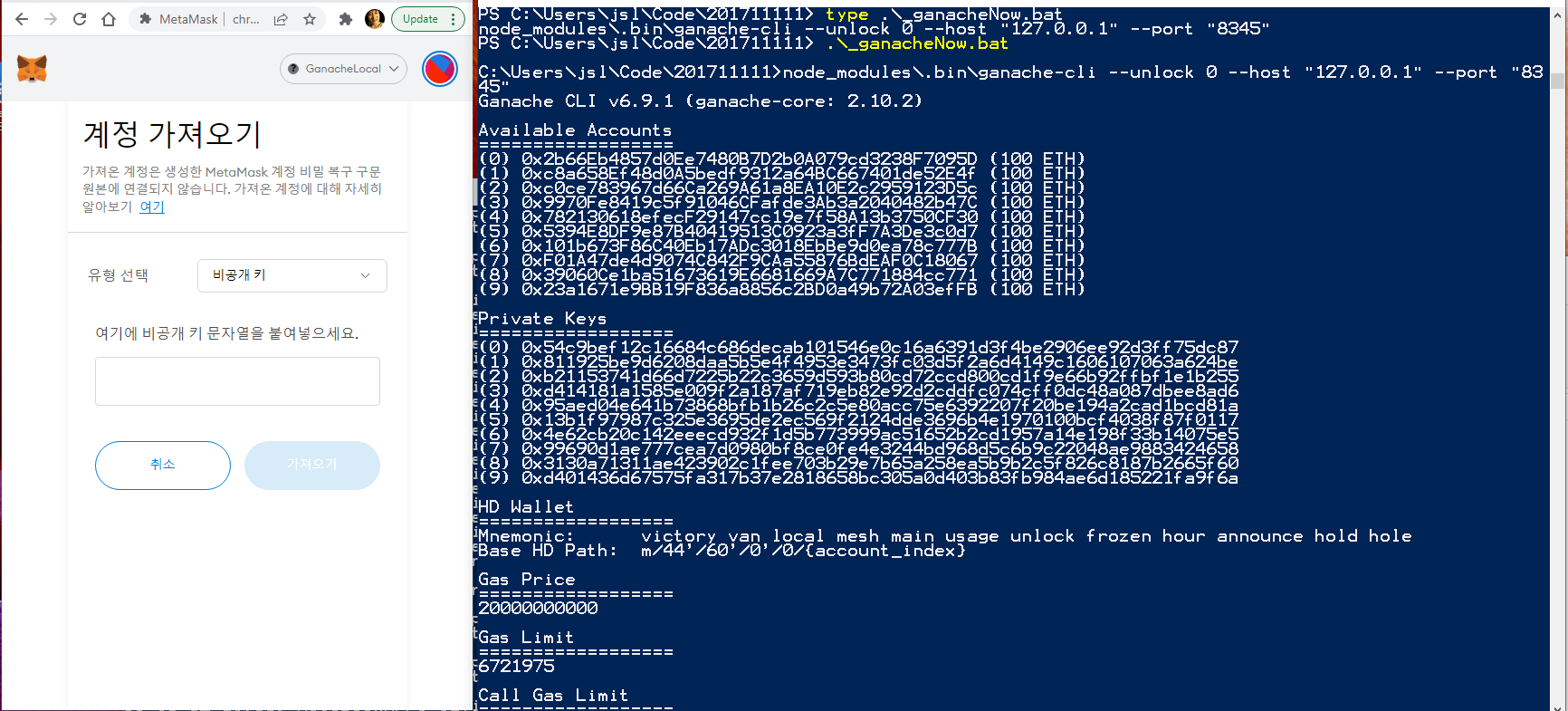
### 단계 2: 네트워크의 키에 등록, 연결하기

앞서 네트워크를 생성하였다고 하더라도, 상호작용할 계정이 아직 없다. 잔고가 있고 사용가능한 실제 키가 필요하다.

* 1. 생성한 네트워크 명칭 옆의 원판 아이콘을 클릭하고,
  2. '계정 가져오기'를 선택하면 창이 뜨고, '여기에 비공개 키 문자열을 붙여넣으세요'를 볼 수 있다.
  3. 그 아래 빈 칸에 키를 붙여 넣으면 된다. 이때 account키가 아니라, 사적 키(private key)를 복사해서 붙여넣어야 한다. 사적 키를 0x를 제외하고 입력해야 한다. 주의! ganache를 다시 생성한다면 키가 재설정되기 때문에 가져오기를 새로 해야 한다. 이 경우 불필요해진 이전 키를 삭제하는 편이 좋다. 삭제는 쉽다. 키의 우측편 세로로 점점점 기호를 마우스로 누르면 마지막 항목에 '계정 제거'가 있다.
  4. 가져온 키와 잔고가 그대로 적혀있는지 확인한다.

또는 반대로 메타마스크의 키를 자신의 geth로 가져올 수 있다. MetaMask에서 키를 선택 > 계정세부정보 > 비공개키 내보내기를 한 후, 그 키를 geth에서 가져오기를 하면 된다.

pjt\_dir> geth account import metakey.prv (MetaMask에서 내보낸 키)



### 단계 3: MetaMask 연결 요청

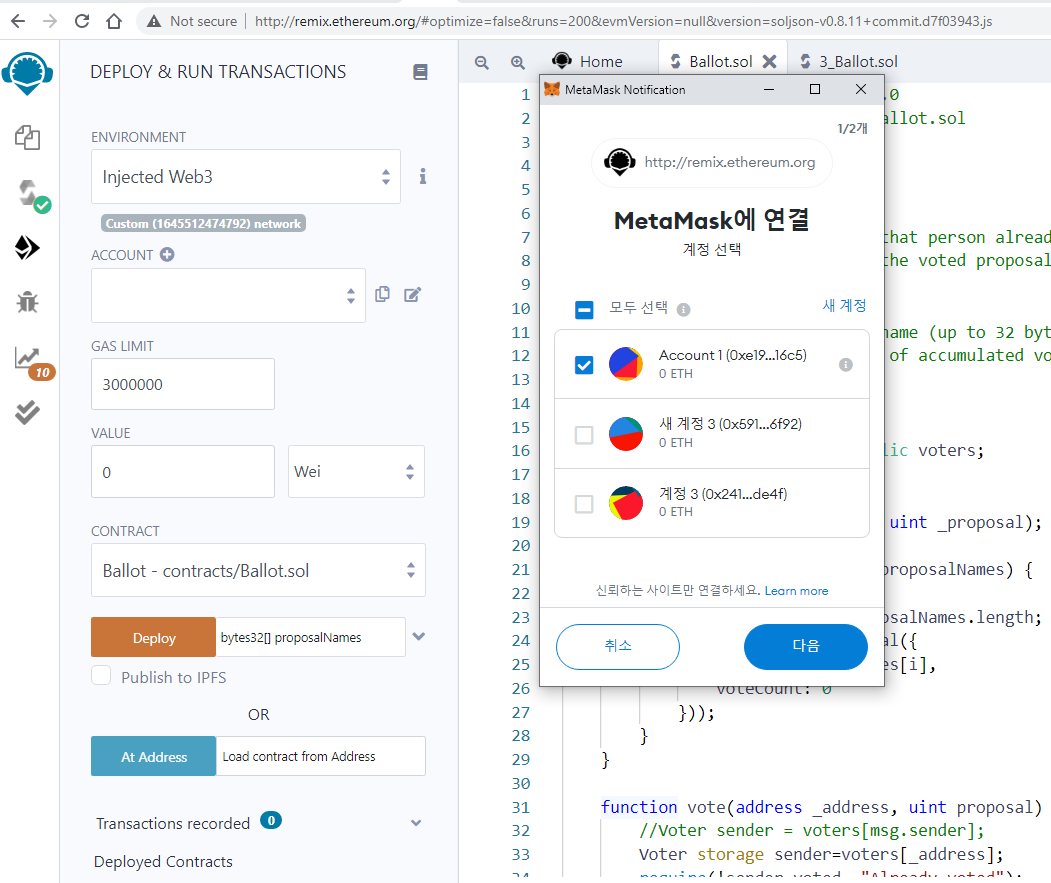
#### REMIX의 'Injected Web'

앞서 메타마스크와 로컬의 네트워크가 연결되었다.

또한 REMIX에 연결할 수도 있다. REMIX 배포창의 메뉴에서 Injected Web을 선택하여 (현재는 Javascript VM) 연결하면, 메타마스크를 통해 배포를 할 수 있다.

메타마스크와 REMIX를 연결하려면:

* 1. 앞서 로그인해서 메타마스크가 활성화되어 있어야 하겠다. 현재는 여우 아이콘 아래에 '연결되지 않음'이라고 표시되어 있다. 앞 단계에서 가져온 키를 선택한다.
  2. 다음 REMIX를 http:// 주소로 열어야 한다 (https:// 가 아니다!).
  3. REMIX의 배포창에서 Injected Web을 선택한다. 반응이 없으면 REMIX를 새로 열어야 할 수도 있다. 성공적으로 연결이 되면, 바로 밑의 주소가 앞서 선택한 네트워크의 계정으로 변경되는 것을 확인할 수 있다. 또한 메타마스크에서도 계정 주소 좌측에 '연결되지 않음' -> '연결'로 녹색 아이콘이 활성화된다.



#### 웹브라우저에서 메타마스크연결: ethereum 객체

웹브라우저에서 연결되면, ethereum 객체를 생성하게 된다. 실제 연결을 하면서 다음 장에서 이해하기로 하자.

## 2.3 메타마스크 설치 확인

### 메타마스크는 ethereum 객체를 만들어 준다

메타마스크가 설치되었으면, 웹페이지가 뜨면서, ethereum 객체를 생성해서 'window' 객체에 하위로 할당하여 window.ethereum을 생성하게 된다.

설치 여부를 확인하기 위해서는 if (ethereum) 또는 if (typeof window.ethereum !== 'undefined'), 즉 ethereum 객체가 이미 있는지 그렇다면 메타마스크가 설치되어 있다는 의미인 것이다.

ethereum.isMetaMask는 메타마스크가 떠 있는지 여부를 알려준다. 혹시 연결이 안되면 메타마스크 여우 아이콘에 마우스 오른쪽 버튼을 눌러 "This can read and change site data" > "On all sites"를 선택하여 권한을 해제한다.

주의: <p id="metamaskInstall">가 먼저 위치해야, 다음에 따라오는 script가 실행될 때 그 id명을 인식한다.

[파일명: scripts/metamaskEthereum.html]  
<!DOCTYPE html>  
<html>  
<head>  
 <title>Using web3 API with MetaMask</title>  
 <script src="https://cdn.jsdelivr.net/npm/web3@1.2.5/dist/web3.min.js"></script>  
  
</head>  
<body>  
 <p id="metamaskInstall"></p>  
 <script>  
 if (ethereum) { --> 설치되어 있다면  
 document.getElementById("metamaskInstall").innerHTML = "MetaMask is installed... " + ethereum.isMetaMask;  
 }  
 else { --> 설치되어 있지 않다면  
 document.getElementById("metamaskInstall").innerHTML = "MetaMask is not installed..." + ethereum.isMetaMask;  
 }  
 </script>  
</body>  
</html>

웹브라우저에 http://localhost:8000/scripts/metamaskEthereum.html을 입력하자. 출력에 "metamask is installed" 문장이 출력되면 설치가 되었다는 의미이다.

console 창을 보면 연결을 확인할 수 있다. You are accessing the MetaMask 출력을 볼 수 있고, ethereum이라는 명령어를 입력하면 객체의 내용이 출력된다.

javascript> ethereum  
Proxy {\_events: {…}, \_eventsCount: 1, \_maxListeners: 100, \_log: u, \_state: {…}, …}

### window.onload로 해도 된다

이번에는 윈도우가 뜨는 시점에 설치여부를 판단할 수 있다.

window.onload 이벤트 핸들러는 웹 페이지의 모든 리소스가 로드되고 브라우저가 준비되었을 때 실행되어, 이 시점에 메타마스크가 설치되었는지 확인한다.

[파일명: scripts/metamaskEthereumOnLoad.html]  
<!DOCTYPE html>  
<html>  
<head>  
 <title>Using web3 API with MetaMask</title>  
 <script src="https://cdn.jsdelivr.net/npm/web3@1.2.5/dist/web3.min.js"></script>  
 <script>  
 window.onload = function() {  
 if (ethereum) {  
 document.getElementById("metamaskInstall").innerHTML = "MetaMask is installed... " + ethereum.isMetaMask;  
 }  
 else {  
 document.getElementById("metamaskInstall").innerHTML = "MetaMask is not installed..." + ethereum.isMetaMask;  
 }  
 }  
 </script>  
</head>  
<body>  
 <p id="metamaskInstall"></p>  
</body>  
</html>

# 3. 웹지갑의 활용

## 3.1 provider 인식

### 이전에는 window.web3 객체가 자동으로 주입된다

이전에는 Web3 provider나 MetaMask를 통해 window.web3 객체가 자동으로 웹 페이지에 주입되었다.

if(window.web3) 명령어로 window.web3가 만들어졌는지에 따라 설치 여부를 확인하는 절차가 필요하다. 설치되었으면, 만들어진 window.web3를 새롭게 new Web3(window.web3.currentProvider)로 재설정한다.

이전에 사용되던 방식에 따라 window.web3 객체를 사용하는 코드는 다음과 같다.

const Web3 = require("web3");  
if (window.web3) { // window.web3가 자동으로 inject되었는지   
 window.web3 = new Web3(window.web3.currentProvider); //브라우저에서 자동 생성하는 currentProvider로 window.web3를 재설정  
 window.ethereum.enable(); //창이 뜨고 ethereum 연결  
}

### 현재는 window.web3 대신 window.ethereum가 주입된다.

EIP-1102, EIP-1193에 따라 MetaMask, Status 또는 Ethereum 호환 브라우저와 같은 Ethereum providers가 공통적으로 따라야 하는 Ethereum 계정에 대한 접근을 관리하는 방식을 표준화하여 편의성을 높이고 보안을 강화하고 있다.

최신의 방식은 window.web3가 주입되지 않고 window.ethereum를 주입해주어야 하고, window.ethereum이 provider가 되고 이로부터 사용자가 직접 web3 객체를 생성한다.

따라서 자동 주입되기 때문에 이더리움 연결을 활성화하기 위한 ethereum.enable()도 필요한 경우에만 호출한다. 사용자는 필요한 경우 MetaMask나 기타 Ethereum 지갑에서 직접 권한 부여를 승인한다.

currentProvider는 말 그대로 Web3.js의 초기화 과정에서 브라우저나 Node.js 환경에서 현재 설정된 프로바이더를 말하는데, 이도 더 이상 확인하지 않아도 된다.

아래 코드는 최신 표준을 준수하여, window.ethereum을 통해 이더리움 프로바이더에 접근하여 Web3 객체를 생성하고 있다.

const Web3 = require("web3");  
if (window.ethereum) { // window.ethereum이 provider이므로 currentProvider를 확인할 필요가 없다  
 window.web3 = new Web3(window.ethereum);  
}

더 알아보기: EIP-1102와 EIP-1193

이 두 EIP (이더리움 개선 제안)는 이더리움 애플리케이션과 브라우저 확장 프로그램 간의 통합을 향상시키기 위해 만들어진 표준화 개선 제안이다.

EIP-1102는 이제 window.ethereum을 통해 현재 연결된 Ethereum providers를 확인할 수 있게 된다. EIP-1193는 자바스크립트로 이더리움과 상호 작용하기 위한 표준 인터페이스를 정의한다. 따라서 EIP-1102와 EIP-1193는 Ethereum providers와 애플리케이션이 상호 작용하는 방식을 표준화하고 개선하기 위한 노력의 일환이고, 그에 따라서 currentProvider를 확인하는 것은 더 이상 필요하지 않을 수 있고 대신 window.ethereum을 통해 현재 연결된 Ethereum 프로바이더에 액세스할 수 있다.

## 3.2 provider에 따른 계정주소

다음 코드는 provider를 경우의 수에 따라 설정하고 있다.

* if(window.ethereum) window.ethereum이 생성되어 있는 경우: 즉 MetaMask가 설치되어 ethereum이 만들어져 있는 경우이다. window.ethereum에서 web3를 생성한다.
* else if(typeof window.web3 !== 'undefined') window.web3이 생성되어 있어 있는 경우: web3가 생성되어 밀어 넣어져 있는 상태로서 currentProvider가 이미 생성되어 있고 여기로부터 web3를 생성한다.
* 마지막으로 로컬 provider에서 web3를 생성한다.

myFunction이 실행되면, 계정의 주소를 출력한다.

* 메타마스크에 연결되어도, 앞서 Ganache 네트워크를 만들어 놓지 않으면, 다른 네트워크에 연결을 시도하게 된다.
* 메타마스크에 연결해서도 일관되게 web3를 사용하면 좋겠지만 그렇지 못하다. ethereum에서는 ethereum.request(({ method: 'eth\_requestAccounts' })라고 한다.
* 입력은 Private Keys를 붙여넣기 했었지만, 출력은 계정주소를 하고 있으니 확인해보자.

[파일명: scripts/testProvider.html]  
<!doctype html>  
<html>  
  
<head>  
<script src="https://cdn.jsdelivr.net/npm/web3@1.2.5/dist/web3.min.js"></script>  
<!-- script src="https://cdn.jsdelivr.net/npm/web3@0.20.5/dist/web3.min.js"></script -->  
</head>  
  
<body>  
<button onclick="myFunction()">Click me to see the coinbase.</button>  
<p id="conn"></p>  
<p id="coin"></p>  
  
<script>  
 // window.ethereum이 생성되어 있는 경우 (MetaMask가 설치되어 있는 경우)  
 if (window.ethereum) {  
 window.web3 = new Web3(window.ethereum); //window.ethereum is a provider itself  
 document.getElementById("conn").innerHTML = "connected to MetaMask!";  
 // window.web3가 생성되어 있는 경우 (기존에 web3가 밀어 넣어져 있는 경우)  
 } else if (typeof window.web3 !== 'undefined') {  
 window.web3 = new Web3(web3.currentProvider);  
 document.getElementById("conn").innerHTML = "connected to currentProvider";  
 // 로컬 provider에서 web3를 생성하는 경우  
 } else {  
 window.web3 = new Web3(new Web3.providers.HttpProvider("http://127.0.0.1:8345"));  
 document.getElementById("conn").innerHTML = "connected to Local Host";  
 }  
   
 //async function myFunction() { //not working when ethereum is connected  
 // const account = await web3.eth.getAccounts();  
 // document.getElementById("coin").innerHTML = "coinbase: " + account[0];  
 //}  
 async function myFunction() { // no need to be this way with ethereum.request  
 if(window.ethereum) {  
 accounts = await ethereum.request({ method: 'eth\_requestAccounts' });  
 } else {  
 accounts = await web3.eth.getAccounts();  
 }  
 document.getElementById("coin").innerHTML = "coinbase: " + accounts[0];  
 }  
  
</script>  
  
</body>  
</html>

연결하고 메타마스크에서 키를 읽어서 출력해 보자. 키는 함부로 노출해서는 안된다는 점에 유의하자.

if문으로 분기하는 경우를 하나씩 주석으로 가리면서 (comment out), 어느 경우에 어떻게 연결되는지와 연결된 계정주소를 web3에서 읽어낼 수 있는지 시간을 들여서 연습해보면 많은 도움이 될 것이다. ethereum에서 생성된 web3를 통해 web3.eth.getAccounts()로 계정을 읽어올 수 있다.

메타마스크에서 계정을 읽어오려면 request() 함수를 통해 요청해야 한다.

const accounts = await ethereum.request({ method: 'eth\_requestAccounts' });  
const account = accounts[0];

# 4. 웹에서 계정잔고 변경 이벤트

계정의 잔고에 변화가 있는지 관찰해보자. 그렇다면 뭔가 이벤트 프로그래밍을 해야 가능할 것이라고 판단된다.

## 4.1 웹소켓

웹소켓은 프로세스를 열어놓고, 양방향 통신을 가능하게 한다. http와 달리 request, response 메시지를 주고받지 않는다.

일단 통신이 연결되면 원하는대로 요청메시지를 주고 받을 수 있다. WebsocketProvider("ws://localhost:8345")) 코드와 같이 http:// 가 아니라 ws:// 로 적어주어야 한다.

const options = {  
 timeout: 30000, // ms  
 headers: {  
 authorization: 'Basic username:password'  
 },  
  
 clientConfig: {  
 ...  
 keepalive: true,  
 keepaliveInterval: 60000 // ms  
 },  
 reconnect: {  
 auto: true,  
 delay: 5000, // ms  
 maxAttempts: 5,  
 onTimeout: false  
 }  
};

이러한 설정을 넣어서, 물론 생략할 수도 있지만, 웹소켓을 연결할 수 있다.

const ws = new Web3.providers.WebsocketProvider('ws://localhost:8546', options);

## 4.2 이벤트

### 이벤트를 리스닝: web3.eth.subscribe

logs에서 이벤트를 읽을 때 web3 버전 1.x에서는 web3.eth.subscribe를 사용한다.

이전에는 이벤트가 발생했는지 계속 확인하는 방식이었으나, web3 1.x에서는 push방식으로 이벤트가 발생하면 구독한 채널에 알려주면 된다.

HttpProvider는 이벤트를 실시간으로 추적하는데 적합하지 않다. HTTP 프로토콜은 단방향 통신을 지원하기 때문에, 클라이언트는 서버에서 응답을 받는 것만 가능하다.

이와는 달리, 웹 소켓(WebSocket)은 이벤트를 실시간으로 감지하고 처리할 수 있다.

### newBlockHeaders를 리스닝

현재는 계정에 대해 발생하는 이벤트는 logs에서 구독하지 못한다. 따라서 newBlockHeaders에 대해 이벤트를 구독한다.

newBlockHeaders는 새로운 블록이 생성될 때마다 발생하는 이벤트이다. 새로운 블록의 헤더 정보를 포함하고 있으며, 블록의 높이, 해시, 시간 정보 등을 제공한다.

### 이벤트 발생

계정의 잔고에 변화가 있으면 출력 역시 수정된다.

* 'send 111' 버튼을 클릭해서 출금한다.
* 웹페이지에서 출금 후 잔고의 변화를 출력하는지 관찰한다. balanceSubscription.html이 열려 있는 상태에서 나란히 MetaMask를 열어 출금의 변화를 관찰하자.
* 메타마스크에서도 출금해도 그 변화를 출력하는지 관찰해보자. 메타마스크에서 출금 계좌를 선택하고 (웹페이지에 출력된 계좌와 동일), 소액이라도 좋으니 다른 계좌로 출금을 해보면, 잔고의 변화를 출력하게 된다.

[파일명: scripts/balanceSubscription.html]  
<!doctype html>  
<!-- @author: jsl @since: 20200428-->  
<html>  
  
<head>  
<script type="text/javascript" src="https://cdn.jsdelivr.net/npm/web3@1.2.5/dist/web3.min.js"></script>  
<script type="text/javascript">  
 //var web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:8345"));  
 var web3=new Web3(new Web3.providers.WebsocketProvider("ws://localhost:8345"));  
 var coinbase;  
 async function watchBalance() {  
 //var coinbase = web3.eth.coinbase; // old web3 0.2x  
 //const coinbase = await web3.eth.getCoinbase();  
 const accounts = await web3.eth.getAccounts();  
 coinbase = accounts[0];  
 console.log("Coinbase: " + coinbase);  
  
 var originalBalance = await web3.eth.getBalance(coinbase);  
 document.getElementById('coinbase').innerText = 'coinbase: ' + coinbase;  
 document.getElementById('original').innerText = ' original balance: ' + originalBalance + ' watching...';  
  
 //web3.eth.filter('latest').watch(function() { // old web3 0.2x  
   
 //let subscription = web3.eth.subscribe("logs", {  
 // fromBlock:0,  
 // toBlock:'latest',  
 // address: [coinbase], //Only get events from specific addresses  
 // topics: [web3.utils.sha3('Voted(address,uint256)')]  
 // }, function(err, success) {  
 //let subscription = web3.eth.subscribe('logs', options, function(err, success) {  
 let subscription = web3.eth.subscribe('newBlockHeaders', function(err, bheader) {  
 console.log("=====> subscribe fired!!");  
 if(!err) {  
 console.log("block header --> ", bheader);  
 console.log("header timestamp ---> ", bheader.timestamp);  
 //var currentBalance = await web3.eth.getBalance(coinbase);  
 web3.eth.getBalance(coinbase, function(err, bal) {  
 if(!err) {  
 //var weiBal = web3.fromWei(wei, 'ether');  
 document.getElementById("current").innerText = 'current: ' + bal + ' ETH';  
 document.getElementById("diff").innerText = 'diff: ' + (bal - originalBalance);  
 }  
 });  
 }  
 });  
 //subscription.on('data', function(blockHeader) { console.log(blockHeader) } );  
 //subscription.on('changed', changed => console.log(changed))  
 //subscription.on('error', err => { throw err })  
 subscription.on('connected', function(id) {  
 document.getElementById("conn").innerText = 'connected: ' + id;  
 });  
 }  
   
 async function send() {  
 await web3.eth.sendTransaction({  
 from: coinbase,  
 to: '0x11f4d0A3c12e86B4b5F39B213F7E19D048276DAe',  
 value: '111'  
 })  
 document.getElementById("send").innerHTML = "sent from: " + coinbase + "111";  
 }  
  
</script>  
</head>  
<body>  
 <h1>coinbase balance</h1>  
 <button type="button" onClick="watchBalance();">watch balance</button>  
 <div></div>  
 <div id="coinbase"></div>  
 <div id="original"></div>  
 <div id="current"></div>  
 <div id="diff"></div>  
 <div id="conn"></div>  
 <button onclick="send()">send 111.</button>  
 <div id="send"></div>  
</script>  
</body>  
</html>

웹서버 단말로 가서 balanceSubscription.html이 200 코드로 성공적으로 실행되었는지 확인한다.

pjt\_dir> python -m http.server 8000  
Serving HTTP on 0.0.0.0 port 8000 ...  
58.232.2.179 - - [03/Feb/2022 14:51:03] code 404, message File not found  
58.232.2.179 - - [03/Feb/2022 14:51:03] "GET /balance.html HTTP/1.1" 404 -  
58.232.2.179 - - [03/Feb/2022 14:51:09] code 404, message File not found  
58.232.2.179 - - [03/Feb/2022 14:51:09] "GET /balance.html HTTP/1.1" 404 -  
58.232.2.179 - - [03/Feb/2022 14:51:21] "GET /scripts/balanceSubscription.html HTTP/1.1" 200 -

그리고 웹브라우저에서 balanceSubscription.html을 호출한다.

url은

file:///home/jsl/Code/프로젝트경로/scripts/balanceSubscription.html  
또는   
http://localhost/scripts/balanceSubscription.html

버튼을 누르고, 잔고변경이 있으면 다음 결과를 볼 수 있다. 마이닝은 필요 없다. diff에 출금하고 나서의 차액이 출력되고 있는데 gas비를 포함한 금액이다. 마지막 항목은 구독채널번호로서, subscription.on('connected') 연결 이벤트에 따라 출력된다.

coinbase: 0x8078e6bc8e02e5853d3191f9b921c5aea8d7f631  
original balance: 100000000000000000000 watching...  
current: 99899580000000000000 ETH  
diff: -100420000000000000  
connected: 0x1

# 5. 투표 웹디앱

블록체인을 활용한 온라인 투표 시스템은 실제 투표 방식과 몇 가지 다른 점이 있다.

* 개인 키를 사용하여 투표를 인증하므로 투표자의 신원은 보호되고,
* 블록체인은 분산된 데이터베이스로 구성되어 있어 한 곳에 집중된 공격으로부터 안전하고,
* 거래 기록을 불변하게 기록하므로 데이터의 위변조가 거의 불가능하고,
* 투표 과정은 선관위 뿐만 아니라, 권한이 허용되면 후보자이든 참관인이든 공개될 수 있다.

블록체인을 적용한 투표가 시행되고 있고, 그런 사례가 이미 오래 전부터 여러 차례 보고되고 있다 (2016년 8월 30일 Forbes 기사 "Block The Vote: Could Blockchain Technology Cybersecure Elections?"). 한국에서도 정부가 2019년 시범서비스를 시행하고, 그 후 주민 투표 시스템을 도입하고 있다.

그럼에도 불구하고, 실물투표가 아닌 온라인 투표라는 점에서 '심리적 신뢰'를 완전하게 얻은 대안이 되려면 도전적인 시행과 평가가 선행되어야 한다.

## 5.1 투표 컨트랙 개발

투표와 관련 ERC1202이라는 표준이 제안되고 있지만, 여기서는 Solidity의 샘플 코드를 수정해서 제작해보자.

투표 컨트랙에는 다음 함수를 구현한다.

* 생성자에서는 투표의 개시에 필요한 투표안을 설정하게 된다. 투표자는 자신이 투표안 가운데 하나를 선택하게 된다. 투표안은 단순 번호이거나 설명으로 표현될 수 있다.
* vote(): 투표자의 계정주소와 찬성안을 입력받아, 투표하는 함수
* winningProposal(): 최다득표한 안을 선별하는 함수
* winnerName(): 최다 득표안을 출력하는 함수
* getNVoted(): 총 투표수를 출력하는 함수

블록체인을 적용한 웹 투표가 매력적이지만, 구현에 앞서 몇 가지 문제를 생각해 볼 필요가 있다.

* 투표 및 투표 결과 기록에는 gas비용이 발생하고, 투표자 또는 투표 시스템 운영자 누구의 부담으로 할지 기준이 필요하다. 일반적으로는 투표 시스템 운영자가 gas 비용을 부담하는 경우가 많다.
* 온라인 투표 과정을 사진으로 찍고 공유하는 문제에 대한 기준이 필요하다.
* 1인 1표의 원칙을 준수하는 방안이 있어야 한다. 유권자의 신원확인을 위한 고유한 식별자가 필요하고 또한 복수 계정을 발급받아 투표하는 것을 막을 수 있어야 한다.
* 유권자 검증을 위해, 개인정보, 시간 및 위치정보 등을 저장할 필요할 수 있다. 단 개인정보보호를 위한 암호화 및 보안이 필요하다.

[파일명: src/Ballot.sol]  
줄01 // SPDX-License-Identifier: GPL-3.0  
줄02 // modified version of Ethereum Ballot.sol  
줄03 pragma solidity >=0.7.0 <0.9.0;  
  
줄04 contract Ballot {  
줄05 struct Voter {  
줄06 bool voted; // if true, that person already voted  
줄07 uint vote; // index of the voted proposal  
줄08 }  
줄09 struct Proposal {  
줄10 bytes3 name; // short name (up to 3 bytes)  
줄11 uint voteCount; // number of accumulated votes  
줄12 }  
  
줄13 address public chairperson;  
줄14 mapping(address => Voter) public voters;  
줄15 Proposal[] public proposals;  
줄16 uint nVoted; //counted in function vote  
줄17 event Voted(address \_address, uint \_proposal);  
   
줄18 constructor(bytes3[] memory proposalNames) {  
줄19 nVoted=0;  
줄20 chairperson = msg.sender;  
줄21 for (uint i = 0; i < proposalNames.length; i++) {  
줄22 proposals.push(Proposal({  
줄23 name: proposalNames[i],  
줄24 voteCount: 0  
줄25 }));  
줄26 }  
줄27 }  
  
줄28 function vote(address \_address, uint proposal) public {  
줄29 //Voter sender = voters[\_address];  
줄30 Voter storage sender=voters[\_address];  
줄31 require(!sender.voted, "Already voted");  
줄32 sender.voted = true;  
줄33 sender.vote = proposal;  
줄34 proposals[proposal].voteCount += 1;  
줄35 nVoted += 1; //add 1  
줄36 emit Voted(\_address, proposal);  
줄37 }  
  
줄38 function winningProposal() public view returns (uint winningProposal\_) {  
줄39 uint winningVoteCount = 0;  
줄40 for (uint p = 0; p < proposals.length; p++) {  
줄41 if (proposals[p].voteCount > winningVoteCount) {  
줄42 winningVoteCount = proposals[p].voteCount;  
줄43 winningProposal\_ = p;  
줄44 }  
줄45 }  
줄46 }  
  
줄47 function winnerName() public view returns (bytes3 winnerName\_) {  
줄48 winnerName\_ = proposals[winningProposal()].name;  
줄49 }  
  
줄50 function getNVoted() public view returns (uint) {  
줄51 return nVoted;  
줄52 }  
줄53 }

* 줄3 Solidity 버전은 0.7 ~ 0.9로 넓게 설정하고 있다.
* 줄5~8에서 (1) 투표 여부, (2) 찬성안의 번호를 저장하기 위해 Voter struct을 다음과 같이 선언한다.

struct Voter {  
 bool voted; // 투표했으면 true, 하지 않았으면 false  
 uint vote; // 몇 번 투표안에 기표했는지  
}

* 줄14는 Voter를 계정 주소와 매핑하고 있다. 즉, 투표자의 계정 주소에 대해 (1) 투표 여부와 (2) 찬성안을 알 수 있게 한다.
* 줄9~12에서 투표안의 (1) 명칭 (편의상 세 글자로 제한하고, bytes3로 설정), (2) 득표수를 저장하기 위해 Proposal struct을 다음과 같이 정의한다.

struct Proposal {  
 bytes3 name; // 편의상 3 bytes로 제한  
 uint voteCount; // 득표수  
}

* 줄15 여기서는 투표안을 3개 만들 것이므로, 배열로 선언한다. 배열에 입력할 때는 push함수를 사용한다.

Proposal[] proposals;  
proposals.push(Proposal({ name: proposalNames[0], voteCount: 0 }));

* 줄17 이벤트

event Voted(address \_address, uint \_proposal);

이벤트가 발생하면, 투표 시점의 투표자의 주소와 투표안을 출력하게 된다.

* 줄18~27 생성자

배포하기 위해서는 객체를 생성해야 한다. 이 때 생성자 constructor(bytes3[] memory proposalNames)에 투표안을 인자로 입력해야 한다.

투표안은 bytes3으로 설정하고 3글자로 제한하고 있다. 예를 들면, 'kim', 'lee', 'lim' 3글자의 bytes3이다. string으로 하면 이진수를 사용하지 않아도 되겠지만 gas가 더 필요하겠다. bytes를 입력을 할 때는 16진수로 6자로 한다.

["0x6b696d", "0x6c6565", "0x6c696d"]

* 줄28~37 투표 vote() 함수에 2개의 인자 (1) 계정주소, (2) 찬성안을 입력한다. 또한 득표수 .voteCount, 총투표 nVoted를 계산한다.
* 줄38~46 winningProposal() 함수에서는 최대로 득표한 안을 선택한다.
* 줄47~49 winnerName()에서는 방금 설명한 winningProposal() 함수를 호출해서 최대득표안을 출력한다. 주의, return은 꼭 쓰지 않아도 된다. 그러면 마지막 명령문이 반환이 된다. 속성으로 선언되어 있고, 함수의 인자로 동일한 명칭을 사용하면 원래의 선언이 가리워지게 된다 (shadow). 기능명도 마찬가지이다. winnerName() 함수명을 인자명으로 사용하면, 인자명이 함수명을 가리기 때문에 winnerName\_이라고 마지막에 밑줄을 넣는다. 인자를 동일한 명칭으로 사용하려면 원래의 선언을 사용하지 않으면 된다.
* 줄50~52

총 투표수 nVoted를 계산하려면 다음과 같이 반복문을 적용할 수 있지만, 비용이 증가한다.

for (uint p = 0; p < proposals.length; p++) {  
 nVoted+=proposals[p].voteCount;  
}

이렇게 하기 보다는 vote() 함수에서 nVoted를 매 투표마다 추적하면 편리하다.

## 5.3 컴파일

컴파일할 때 "--combined-json abi,bin"이라고 적어야 한다. 사이에 공백을 띄워서 "abi, bin"라고 입력하면 오류가 발생한다.

pjt\_dir> solc-windows.exe --optimize --combined-json abi,bin src/Ballot.sol > src/Ballot.json

더 진행되기 전, 생성된 Ballot.json을 열어서, abi와 바이트 코드가 저장되었는지 확인하자.

pjt\_dir> type src\\Ballot.json

type 명령어로 출력을 해보는데, 그 명령어를 입력할 때 디렉토리 구분자에 주의한다. 운영체제에 따라 디렉토리 구분자가 서로 상이하고 여기서는 '\'로 적었다.

## 5.4 컨트랙 배포

### ABI, 바이트 코드를 파일에서 읽기

앞서 컴파일하고 출력된 ABI, 바이트 코드를 파일에서 읽어내어 보자. 줄2의 Object.keys(\_abiJson.contracts) 함수로 JSON으로부터 키를 읽어내서 컨트랙명을 인식한다. 그리고 점연산자 .abi와 .bin을 통해 ABI, 바이트 코드를 읽어온다.

[파일명: src/Ballot.js]  
var \_abiJson = require('./Ballot.json');  
contractName=Object.keys(\_abiJson.contracts); // ['src/Ballot.sol:Ballot']  
console.log("- contract name: ", contractName);  
  
\_abi=\_abiJson.contracts[contractName].abi  
\_bin=\_abiJson.contracts[contractName].bin  
console.log("- ABI: ", \_abi);  
console.log("- Bytecode: ", \_bin);

위 파일을 실행해보자.

pjt\_dir> node src/Ballot.js

abi, bin를 읽어서, 출력이 되고 있는지 확인해보자. abi 출력을 잘 살펴보면, [Object]가 많이 보인다. 자바스크립트 Object을 말한다. 컴파일하고 저장한 json의 abi 출력과는 다른 것을 눈치챌 수 있다. JSON.stringify(\_abi)라고 해주면 물론 문자열을 볼 수 있다.

### 배포

new web3.eth.Contract(\_abiArray) 개체를 생성하고, deploy() 함수를 통해 배포를 한다. geth@8445에서는 별도의 마이닝과 계정의 지급해제, 즉 .unlockAccount()이 필요하다.

아래 .deploy() 함수를 호출할 때 생성자 인자를 넘겨주어야 하는 부분을 주의하자. ["0x6b696d", "0x6c6565", "0x6c696d"] 이렇게 배열을 넘겨주어야 하지만 그렇게 하면 1 개만 넘겨주라고 오류가 발생한다. 그럴 경우에는 다음과 같이 해준다. [["0x6b696d", "0x6c6565", "0x6c696d"]]

[파일명: src/BallotDeployAbiBinFromFile.js]  
var Web3=require('web3');  
var \_abiBinJson = require('./Ballot.json');  
  
var web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:8345"));  
  
contractName=Object.keys(\_abiBinJson.contracts); // reading ['src/Ballot.sol:Ballot']  
console.log("- contract name: ", contractName); //or console.log(contractName);  
  
\_abiArray=JSON.parse(JSON.stringify(\_abiBinJson.contracts[contractName].abi));  
\_bin="0x"+\_abiBinJson.contracts[contractName].bin;  
  
async function deploy() {  
 const accounts = await web3.eth.getAccounts();  
 console.log("Deploying the contract from " + accounts[0]);  
 var deployed = await new web3.eth.Contract(\_abiArray)  
 .deploy({data: \_bin, arguments: [["0x6b696d", "0x6c6565", "0x6c696d"]]})  
 .send({from: accounts[0], gas: 1000000}, function(err, transactionHash) {  
 if(!err) console.log("hash: " + transactionHash);   
 })  
 console.log("---> The contract deployed to: " + deployed.options.address)  
}  
deploy()

코드를 작성하면서 생성자에 투표안도 잊지 않고 입력하였고, 이제 파일을 실행해보자.

pjt\_dir> node src\BallotDeployAbiBinFromFile.js  
  
- contract name: [ 'src/Ballot.sol:Ballot' ]  
Deploying the contract from 0x24f6A17559eC4c8c5aF91c8803ee0f9c048655eC  
hash: 0x0c1f073b4357d8f7c5bb3ef24228d837047a2b4adea3316926d81a8b52c81450  
---> The contract deployed to: 0x88f94583066fca67f0B9b46C1d89ef69288cbd22 ---> BallotUse1.js의 (A)로 복사 붙여넣기 한다.

## 5.5 컨트랙 사용

앞서 만든 컨트랙을 배포하고, 이제 그 API를 호출해서 출력을 확인해보자.

줄2~3 이벤트를 포착하려면 웹소켓이 필요하다.

* WebSocket은 ws://로 해야 한다. http:// 로 하지 않도록 한다.
* 웹소켓 연결은 이벤트를 무한 대기하기 때문에, 단말이 흡사 작동되지 않는 것처럼 보일 수 있다. 불편하면 http provider를 사용하면서 출력을 확인해도 좋다. 이벤트 발생은 다음 웹 앱에서는 페이지에 출력이 잘 되고 있다.

줄9~14는 이벤트가 발생하면 Event fired 라는 출력을 하고 있다.

줄8 배포주소는 챙겨서, 아래 (A)에 붙여넣기 하는 것을 잊지 말자.

vote() 호출하면서 발생하는 gas, 이를 적게 산정하면 'out of gas' 오류가 발생할 수 있다. REMIX를 사용해서 vote()를 실행하면 gas가 80000000이 산정된다.

Ballot.sol은 Voter, Proposal, chairperson과 같은 속성이 있으나, 이들을 바로 읽지 않도록 한다.속성은 public으로 (값을 확인하기 위해) 바로 읽을 수 없다.

동일 계정의 재투표, 같은 계정에서 2회 투표를 하면 해시는 생성되지만, 거래는 완료되지 못한다. gas는 사용되는지 확인해보자.

[파일명: src/BallotUse1.js]  
var Web3=require('web3');  
//var web3=new Web3(new Web3.providers.HttpProvider("http://localhost:8345"));  
var web3=new Web3(new Web3.providers.WebsocketProvider("ws://localhost:8345"));  
var \_abiBinJson = require('./Ballot.json');  
contractName=Object.keys(\_abiBinJson.contracts); // reading ['src/Ballot.sol:Ballot']  
console.log("- contract name: ", contractName); //or console.log(contractName);  
\_abiArray=JSON.parse(JSON.stringify(\_abiBinJson.contracts[contractName].abi)); //JSON parsing needed!!  
var ballot = new web3.eth.Contract(\_abiArray, '0x88f94583066fca67f0B9b46C1d89ef69288cbd22'); ---> (A) 위 배포주소를 여기에 붙여넣기  
var event = ballot.events.Voted({fromBlock: 0}, function (error, result) {  
 if (!error) {  
 console.log("Event fired: " + JSON.stringify(result.returnValues));  
 console.log("address: "+result.returnValues.\_address+" proposal: "+result.returnValues.\_proposal);  
 }  
});  
  
async function doIt() {  
 const accounts = await web3.eth.getAccounts();  
 console.log("Account0: " + accounts[0]);  
 const balance0Before = await web3.eth.getBalance(accounts[0]);  
 console.log("(1) Balance0 before: " + balance0Before);  
 const balance1Before = await web3.eth.getBalance(accounts[1]);  
 console.log("(2) Balance1 before: " + balance1Before);  
  
 ballot.methods.winningProposal().call().then(function(p) {  
 console.log("(3) wining proposal before: " + p);  
 });  
 ballot.methods.winnerName().call().then(function(n) {  
 console.log("(4) winner before: " + n);  
 });   
 //await ballot.methods.vote(accounts[3], 0).send({from: accounts[0],gas:800000});  
 //await ballot.methods.vote(accounts[4], 1).send({from: accounts[1],gas:800000});  
 //await ballot.methods.vote(accounts[5], 1).send({from: accounts[2],gas:800000});  
 //await ballot.methods.vote(accounts[6], 1).send({from: accounts[6], gas:800000}, function (error, tranHash) {  
 // console.log("tranHash: " + tranHash);  
 //});  
 //ballot.methods.vote("0x18",1).send({from: accounts[0], gas:800000},function (error, tranHash) {  
 ballot.methods.vote("0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2",1).send({from: accounts[0], gas:800000},function (error, tranHash) {  
 if (!error) {  
 console.log("(5) voted. transaction hash ===> ",tranHash);  
 }  
 });  
 ballot.methods.getNVoted().call().then(function(n) {  
 console.log("(6) how many voted: " + n);  
 });  
 ballot.methods.winningProposal().call().then(function(p) {  
 console.log("(7) wining proposal after: " + p);  
 });  
 ballot.methods.winnerName().call().then(function(n) {  
 console.log("(8) winner after: " + n);  
 });  
 ballot.methods.getNVoted().call().then(function(n) {  
 console.log("(9) how many voted: " + n);  
 });  
 const balance0After = await web3.eth.getBalance(accounts[0]);  
 console.log("(10) Balance0 after: " + balance0After);  
 console.log("(11) Balance0 diff: " + (balance0After - balance0Before));  
 const balance1After = await web3.eth.getBalance(accounts[1]);  
 console.log("(12) Balance1 after: " + balance1After);  
 console.log("(13) Balance1 diff: " + (balance1After - balance1Before));  
}  
  
doIt()

### 마이닝이 끝나면 결과가 출력된다.

ganache가 아니면, 마이닝이 반드시 필요하다. 트랜잭션이 3회 발생하므로, 계정 해제가 건별로 이루어진 경우 오류가 발생할 수 있다.

httpprovider로 실행하면 이벤트는 발생하지 않는다.

pjt\_dir> node src/BallotUse1.js  
  
- contract name: [ 'src/Ballot.sol:Ballot' ]  
Account0: 0x0dF1De1d4C0b9eDCed552594335c804c3Af74E98  
Balance0 before: 99938915360000000000  
Balance1 before: 99996131300000000000  
wining proposal before: 1  
winner before: 0x6c6565  
how many voted: 11  
wining proposal after: 1  
winner after: 0x6c6565  
how many voted: 11  
Balance0 after: 99938915360000000000  
Balance0 diff: 0  
Balance1 after: 99996131300000000000  
Balance1 diff: 0

websocket provider 실행하면 이벤트가 발생하지만, 무한대기에 빠져 올바르게 작동하지 않을 수 있다.

voted. transaction hash ===> 0xcb62e44e66e7f2ed2a80ef663bd179735c4f8054a8976072ef656f76d94a8ff2  
Event fired: {"0":"0x241A076eE90E5703952C70c7cCe43d4388C8De4f","1":"1","\_address":"0x241A076eE90E5703952C70c7cCe43d4388C8De4f","\_proposal":"1"}  
address: 0x241A076eE90E5703952C70c7cCe43d4388C8De4f proposal: 1

## 5.6 웹 DApp

웹페이지 화면은 테스트 수준으로 제작하기로 하고, 투표자 입력과 버튼으로 구성한다.

### 입력 필드

화면에는 2 개의 입력필드가 제공되고 있다.

* 1. Address 필드에는 투표자 자신의 계정 주소를 입력한다. 개인 정보를 보호하기 위해 감추는 것이 좋겠지만, 테스트하기 용이하도록 화면에서 입력받고 있다.
  2. Proposal 필드에는 찬성안을 자유롭게 입력한다. 투표안이 아닌 번호를 입력하지 않도록 설계해야 한다.

### 요청 버튼

컨트랙의 함수는 모두 하나의 웹페이지에 버튼을 통해서 호출하고 있다.

* 1. who votes? 버튼은 getVoter()를 호출,
  2. vote 버튼은 vote()를 호출,
  3. who win? 버튼은 최다 득표안을 조회하기 위해 winning(), printNVoted() 함수를 호출,
  4. watch vote 버튼은 투표 로그를 출력하기 위해 watchVote() 함수를 호출한다.

### 페이지 분리

실제로는 투표의 과정을 적절하게 웹페이지에 나누어 설계해야 좋겠지만, 테스트하기 용이하도록 한 페이지에 모든 기능을 포함하고 있다.

권한에 따라 투표자에게 당연히 허용되지 않아야 할 투표 결과도 웹페이지 하단에 표 형식으로 실시간 확인이 가능하도록 만들었다.

### 출력

제공되는 버튼을 누르면 실행 결과가 바로 출력되고 있다. 예를 들어 투표 버튼을 누르면, 그 내용이 연달아 출력되고 있다. 실제로는 권한에 따라 기능을 사용하도록 설계되어야 한다.

[파일명: scripts/ballot.html]  
<!doctype html>  
<html>  
  
<head>  
<script src="https://cdn.jsdelivr.net/npm/web3@1.2.5/dist/web3.min.js"></script>  
<script type="text/javascript">  
 //var web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:8345"));  
 var web3 = new Web3(new Web3.providers.WebsocketProvider("ws://localhost:8345")); //(A)  
  
 \_abiArrayFromSolc = [{"inputs":[{"internalType":"bytes3[]","name":"proposalNames","type":"bytes3[]"}],"stateMutability":"nonpayable","type":"constructor"},{"anonymous":false,"inputs":[{"indexed":false,"internalType":"address","name":"\_address","type":"address"},{"indexed":false,"internalType":"uint256","name":"\_proposal","type":"uint256"}],"name":"Voted","type":"event"},{"inputs":[],"name":"chairperson","outputs":[{"internalType":"address","name":"","type":"address"}],"stateMutability":"view","type":"function"},{"inputs":[],"name":"getNVoted","outputs":[{"internalType":"uint256","name":"","type":"uint256"}],"stateMutability":"view","type":"function"},{"inputs":[{"internalType":"uint256","name":"","type":"uint256"}],"name":"proposals","outputs":[{"internalType":"bytes3","name":"name","type":"bytes3"},{"internalType":"uint256","name":"voteCount","type":"uint256"}],"stateMutability":"view","type":"function"},{"inputs":[{"internalType":"address","name":"\_address","type":"address"},{"internalType":"uint256","name":"proposal","type":"uint256"}],"name":"vote","outputs":[],"stateMutability":"nonpayable","type":"function"},{"inputs":[{"internalType":"address","name":"","type":"address"}],"name":"voters","outputs":[{"internalType":"bool","name":"voted","type":"bool"},{"internalType":"uint256","name":"vote","type":"uint256"}],"stateMutability":"view","type":"function"},{"inputs":[],"name":"winnerName","outputs":[{"internalType":"bytes3","name":"winnerName\_","type":"bytes3"}],"stateMutability":"view","type":"function"},{"inputs":[],"name":"winningProposal","outputs":[{"internalType":"uint256","name":"winningProposal\_","type":"uint256"}],"stateMutability":"view","type":"function"}]; //(B)  
 var ballot = new web3.eth.Contract(\_abiArrayFromSolc, '0x88f94583066fca67f0B9b46C1d89ef69288cbd22'); //(C)  
 async function getVoter() { //(D)  
 var accounts;  
 if(window.ethereum) {  
 console.log("reading from metamask");  
 accounts = await ethereum.request({ method: 'eth\_requestAccounts' });  
 } else {  
 console.log("reading not from metamask");  
 accounts = await web3.eth.getAccounts();  
 }  
 var originalBalance = await web3.eth.getBalance(accounts[0]);  
 document.getElementById('coinbase').innerText = 'voter: ' + accounts[0];  
 document.getElementById('original').innerText = 'voter balance: ' + originalBalance;   
 }  
 async function vote() { //(E)  
 var \_a = document.getElementById("voteForm").address.value;  
 var \_p = document.getElementById("voteForm").proposal.value;  
 var \_p=parseInt(\_p);  
 //console.log("FORM VALUE is address? "+ web3.utils.isAddress(\_a));  
  
 var accounts;  
 if(window.ethereum) {  
 console.log("reading from metamask");  
 accounts = await ethereum.request({ method: 'eth\_requestAccounts' });  
 } else {  
 console.log("reading not from metamask");  
 accounts = await web3.eth.getAccounts();  
 }  
 console.log(accounts);  
 //ballot.vote.sendTransaction(\_a,\_p,{from:web3.eth.accounts[0]},function (error, result)  
 //console.log("\_a: "+ \_a + "\_p: " + \_p + "accounts0: " + accounts[0]);  
 // ballot.methods.vote(\_a,\_p).send({from: accounts[0], gas:800000});  
 await ballot.methods.vote(\_a,\_p).send({from: accounts[0], gas:800000}, function (error, tranHash) { //(F)  
 if (!error) {  
 document.getElementById("nowVoted").innerHTML = "voted ===> "+\_a+"::"+\_p+"::"+tranHash;  
 // var currentBalance = web3.eth.getBalance(coinbase).toNumber();  
 // document.getElementById("current").innerText = 'current: ' + currentBalance;  
 // document.getElementById("diff").innerText = 'diff: ' + (currentBalance - originalBalance);  
 }  
 });  
 printNVoted();  
 }  
 function printNVoted() {  
 ballot.methods.getNVoted().call().then(function(n) {  
 document.getElementById("nvoted").innerText = 'how many voted: '+n;  
 });  
 }  
 function winning() {  
 //var win=ballot.winningProposal.call();  
 ballot.methods.winningProposal().call().then(function(p) {  
 document.getElementById("winning").innerText = 'winning: '+p;  
 });  
 }  
 function watchVote() { //(G)  
 //ballot.Voted().watch(function(error, result) {  
 //var event = \_test.events.MyLog({fromBlock: 0}, function (error, result) {  
 ballot.events.Voted({fromBlock: 0}, function (error, result) {  
 if (!error) {  
 console.log("Event fired: " + JSON.stringify(result.returnValues));  
 var table = document.getElementById("voteTable");  
 var row = table.insertRow(0);  
 var cell1 = row.insertCell(0);  
 var cell2 = row.insertCell(1);  
 var cell3 = row.insertCell(2);  
 var cell4 = row.insertCell(3);  
 var cell5 = row.insertCell(4);  
 cell1.innerHTML = result.address;  
 cell2.innerHTML = result.blockNumber;  
 cell3.innerHTML = result.transactionHash;  
 cell4.innerHTML = result.returnValues.\_address;  
 cell5.innerHTML = result.returnValues.\_proposal;  
 }  
 });  
 }  
  
</script>  
<style>  
 table, td {  
 border: 1px solid black;  
 border-collapse: collapse;  
 }  
</style>  
</head>  
<body>  
 <h1>Vote</h1>  
 <form id="voteForm">  
 Address: <input type="text" name="address" value="0x"><br>  
 Proposal: <input type="text" name="proposal" value="1"><br>  
 </form>  
 <button type="button" onClick="getVoter();">who votes?</button>  
 <button type="button" onClick="vote();">vote</button>  
 <button type="button" onClick="winning();">who wins?</button>  
 <button type="button" onClick="watchVote();">watch vote</button>  
 <div></div>  
 <div id="coinbase"></div>  
 <div id="original"></div>  
 <div id="current"></div>  
 <div id="diff"></div>  
 <div id="nowVoted"></div>  
 <div id="winning"></div>  
 <div id="nvoted"></div>  
 <table id="voteTable">  
 <caption>vote logs</caption>  
 <tr>  
 <th>address from</th>  
 <th>blockNumber</th>  
 <th>transactionHash</th>  
 <th>address</th>  
 <th>proposal</th>  
 </tr>  
 </table>  
 <br>  
</body>  
</html>

코드의 우측에 붙은 주석의 알파벳 순서로 설명을 해보자.

* 1. 이벤트가 발생하는 페이지이므로 웹소켓으로 .WebsocketProvider() 연결한다.
  2. ABI를 파일에서 읽어오지 않고 직접 붙여넣기 하고 있다. 혹시 아래에서 보는 것 처럼 자바스크립트의 [Object]가 포함되어 있는지, 오류의 원인이 될 수 있으니 주의하자. 잘못된 ABI (src/Ballot.js의 출력을 복사)는 다음과 같다. 코드에 포함된 올바른 값과 비교해보자.

\_abiArray=[  
 ...  
 { inputs: [ [Object], [Object] ],  
 name: 'vote',  
 outputs: [],  
 stateMutability: 'nonpayable',  
 type: 'function' },  
 ...  
]

* 1. 배포하고 획득한 컨트랙 주소를 적어주고 있다. geth@8445와 달리, ganache@8345는 연결이 일단 끊어지면 초기화되므로, 배포 주소 역시 변경 된다. 그럴 경우에는 다시 배포하고 주소를 갱신해서 적어야 한다.
  2. 투표자의 주소 투표는 원격에서 발생한다. 따라서 원격의 주소를 읽어야 한다. 이 경우 MetaMask가 쓰임새가 있다. MetaMask가 연결되고 ethereum으로부터 계정 주소를 읽지 않아도 된다. if문으로 ethereum으로 연결되면 ethereum.request({ method: 'eth\_requestAccounts' })에서 읽어낼 수 있다. 그냥 보통 해왔던 것처럼 web3.eth.getAccounts()에서 읽는다.
  3. 투표에 필요한 주소는 웹지갑 또는 로컬 지갑에서 읽고 있지만, 실제 투표에서는 실제 투표자의 웹지갑에서 읽어야 한다. 누가 어느 투표안에 투표하는지는 폼의 필드에서 읽고 있다. Solidity에서 주소는 16진수 값이고, 투표안은 uint이다. FORM에서 읽은 값은 자체가 문자열이고, 투표안은 parseInt() 함수로 형변환을 한다. 주소는 문자열로 넘겨주어도 되고, 형변환을 하지 않아도 된다. web3.utils.isAddress() 함수로 주소인지 확인하는 편이 바람직하다.
  4. gas 비용 vote(\_a,\_p).send({from: accounts[0], gas:800000})에서 보듯이, accounts[0]에서 거래가 발생하고 비용도 그 계정에서 부담하도록 되어있다. 실제 투표에서는 비용을 유권자에게 부담지워야 하는 것인지 논의가 필요할 수 있겠다.
  5. 이벤트 관련 코드는 함수 안에 넣는다. 버튼을 누르면 함수가 실행이 되면서 이벤트 추적을 시작하게 된다.

### 웹 페이지 실행

웹페이지를 실행하기 전 아래가 준비되었는지 확인하자.

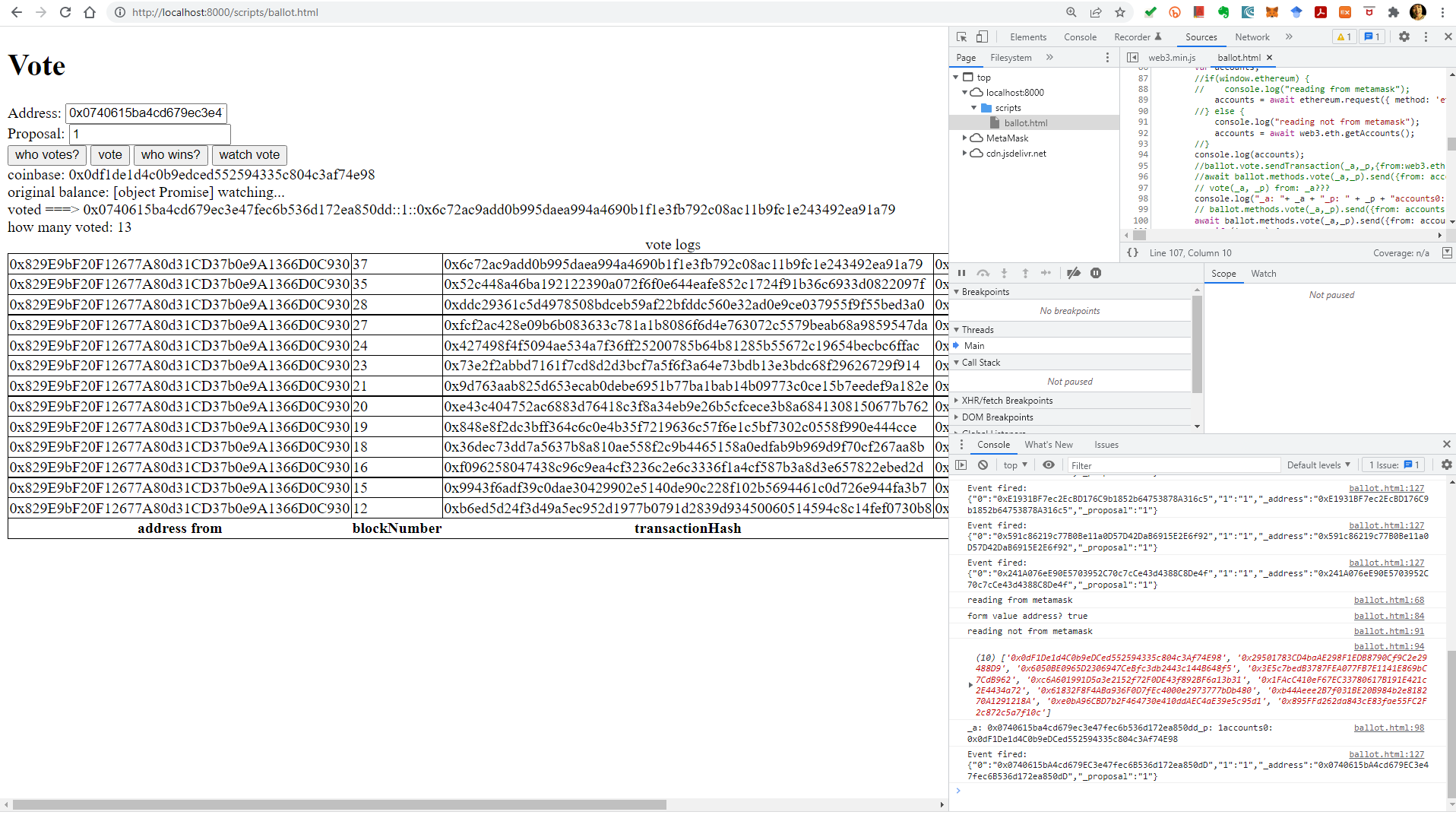
* ganache@8345 (또는 배포한 네트워크에 따라 geth@8445),
* http server,
* 메타마스크(MetaMask,)
* 또한 로그의 출력을 확인하기 위해, 자바스크립트 콘솔도 열어 놓자.

웹브라우저 주소창에 http://localhost:8000/scripts/ballot.html 을 입력한다.

그림에서 보듯이, 화면에는 입력 필드 2 개와 4개 의 버튼이 있다.

* 'who votes' 버튼은 coinbase와 잔고를 출력한다. 이 때 MetaMask가 설치되었으면 그 계정을 연결하는 팝업창이 뜬다. 연결할 계정을 선택하면 여기로부터 주소와 잔고를 읽어낸다.
* 'vote' 버튼은 주소와 투표안을 입력하고 클릭하면 블록체인에 저장된다. gas는 현재 특정 계좌에서 부담하는 것으로 되어 있는 것에 주의한다. 투표를 완료하면, 동일한 계정 주소의 재투표는 불가능하다. 버튼을 누른다 해도 아무 반응도 하지 않는다. 번거롭지만 MetaMask에 다른 키를 가져와서 테스트해야 한다. 실제로 누가 gas를 부담할지, 어느 키를 적용할 것인지 정책적 기준에 따라 코드를 수정할 필요가 있다.
* 'who wins' 버튼은 어떤 투표안이 가장 많은 득표를 하였는지 출력한다.
* 'watch vote' 버튼은 좌하단의 표에 vote가 실행되면서 발생한 이벤트의 내역을 출력한다. 여러 번 버튼을 누르면 이벤트가 중복되어 출력되니 주의하자. 그 내역은 발생한 이벤트가 하나씩 쌓인 것을 보여주고 있다.

우측 화면은 웹페이지 디버깅을 위해 열어놓고 있고, 우하단은 console.log()로 출력한 메시지를 출력하고 있다.



alt text

## 연습문제

1. 웹디앱은 기존의 웹과는 달리, 데이터와 처리 능력을 어느 한 중앙에 두지 않고 분산하는 것이다. OX로 답하시오.
2. web3.js는 로컬의 라이브러리로 설치되어 있지 않으면 적용할 수 없다. OX로 답하시오.
3. 원격 웹에서 블록체인의 거래, 송금을 하거나 투표를 하는 경우 웹 지갑을 사용할 수 있다. OX로 답하시오.
4. REMIX에서는 Injected web을 선택하면 웹지갑에 연결할 수 없다. OX로 답하시오.
5. 웹지갑 메타마스크에 연결하고, 설치되어 있으면 MetaMask is installed를 출력하는 코드들 작성하시오.
6. 웹지갑 메타마스크에 연결된 후, account0을 읽는 코드를 작성하시오.
7. 현재는 계정에 대해 발생하는 이벤트는 logs에서 구독하지 못한다. 잔고의 변화를 웹에서 추적하려면 어디를 관찰해야 하는지 적으시오.
8. 컨트랙의 생성자 constructor(bytes3[] memory proposalNames)를 호출하기 위해 자바스크립트에서 객체를 생성한다고 하고, lim, kim, lee 문자열을 인자로 입력한다고 하자. 아래 .deploy() 함수의 (1)을 채워보시오.

new web3.eth.Contract(\_abiArray)  
 .deploy({data: \_bin, (1)})

1. 웹페이지의 버튼 <button type="button" onClick="getVoter();">who votes?</button>이 있다고 하자. 버튼을 클릭하면 투표자의 계정 주소를 읽어오도록 아래 자바스크립트 코드의 (1)에 필요한 코드를 작성하시오.

async function getVoter() {   
 var accounts;  
 (1)  
 document.getElementById('coinbase').innerText = 'voter: ' + accounts[0];   
}

1. 웹페이지의 버튼 <button type="button" onClick="watchVote();">watch vote</button>이 있다고 하자. 버튼을 클릭하면 투표 이벤트 event Voted(address \_address, uint \_proposal)가 발생하고, 투표자의 첫 번째 계정 주소와 두 번째 찬성안을 읽어오도록 아래 자바스크립트 코드의 (1)에 필요한 코드를 작성하시오.

function watchVote() {   
 ballot.events.Voted({fromBlock: 0}, function (error, result) {  
 if (!error) {  
 console.log("Event fired: " + JSON.stringify(result.returnValues));  
 var table = document.getElementById("voteTable");  
 var row = table.insertRow(0);  
 (1)  
 }  
 });  
}

1. 온라인 경매를 웹디앱으로 제작해보자. 온라인 경매는 중앙에서 처리하면, 무엇인가 조작이 혹시 있을 수 있다. 블록체인을 적용하면 투명하고 수정 불가능하게 처리될 수 있다는 장점이 있다. 다음 기능을 요구사항을 구현한다.

* 경매의 등록
  + 등록자 인증 (이메일, 전번)
  + 제품등록: 제품명, 분류, 이미지
  + 경매내용
    - 경매기간
    - 제품시작가
    - 입찰단위 - 입찰할 경우, 최저가격차
    - 판매가능최저금액
    - 입찰단위
* 입찰의 규칙
  + 동일금액인 경우, 먼저 입찰한 사람에게 낙찰
  + 1회 응찰
  + 낙찰가 산정
    - 낙찰 수수료 15%
    - 부가세 10%
    - 5000만원 낙찰 + 750만원(수수료) + 75만원 (10%부가세) = 5825
  + 7일 이내 지불
* 입찰 후, 소유권은 즉시 이전