Curso desarrollo Blockchain Ethereum con Solidity

Clase 7

Introducción al framework WEB3JS

Introducción al framework

- WEB3 es un **framework** especializado en la **interconexión** Frontend-Blockchain
- Existen diferentes versiones del mismo
- Es posible utilizar tanto la versión javascript (**web3js**) como la versión Phyton (**web3.py**)
- Provee una serie de métodos funciones y propiedades que simplifican la interacción con los contratos inteligentes liberados en la blockchain de Ethereum
- Toda la documentación asociada se la puede encontrar pública en https://web3js.readthedocs.io

Introducción al framework

Web3 se compone de los siguientes **submódulos** dentro de los cuales se podrán llevar a cabo diferentes tareas de interconexión frontend-blockchain

- **Eth:** el módulo eth permite interactuar con la red Ethereum
- **Net:** el módulo net permite interactuar con las propiedades de la red
- **Personal:** el módulo personal permite interactuar con las cuentas de Ethereum
- **Ssh:** el módulo ssh permite interactuar con el protocolo SSH
- **Bzz:** el módulo bzz permite interactuar con las swarm networks

```
var Web3 = require('web3');

// "Web3.providers.givenProvider" will be set if in an Ethereum supported browser.
var web3 = new Web3(Web3.givenProvider || 'ws://some.local-or-remote.node:8546');

> web3.eth
> web3.shh
> web3.bzz
> web3.utils
> web3.version
```

Introducción al framework

Web3.eth

Web3.eth se compone, a su vez de los siguientes **submódulos** dentro de los cuales se podrán llevar a cabo diferentes tareas

- **isMining:** Devuelve true si se está minando o false en caso contrario

```
web3.eth.isMining([callback])
```

- **getAccounts:** devuelve una lista de cuentas controladas por el nodo

```
web3.eth.getAccounts([callback])
```

- **getBalance:** devuelve el balance de una cuenta pudiendo especificar un bloque particular

```
web3.eth.getBalance(address [, defaultBlock] [, callback])
```

- **getTransaction:** devuelve información de la transacción cuyo hash se recibe por parámetro

```
web3.eth.getTransaction(transactionHash [, callback])
```

- **getBlockNumber:** devuelve el número del bloque más reciente

```
web3.eth.getBlockNumber([callback])
```

Introducción al framework

Web3.eth - sendTransaction

Uno de los métodos más importantes del subconjunto web3.eth es el de **sendTransaction** a traves del cual será posible enviar la transacción a la red.
La función recibe un **transactionObject** que se compone de

- **from:** la dirección (address) de quien envía la transacción
- **to:** la dirección (address) de quien recibe la transacción
- **value:** valor en Ether (expresado en WEI) que se envía junto con la transacción
- **gas:** unidades de gas que se espera gastar en esta transacción
- **gasPrice:** precio que el emisor está dispuesto a pagar por cada unidad de gas consumida
- **data:** Contiene información específica de la llamada a la función

```
// using the callback
web3.eth.sendTransaction({
    from: '0xde0B295669a9FD93d5F28D9Ec85E40f4cb697BAe',
    data: code // deploying a contracrt
}, function(error, hash){
    ...
});
```

Introducción al framework

Web3.net

Web3.net se compone a su vez de los siguientes **submódulos** dentro de los cuales se podrán llevar a cabo diferentes tareas

- **getId:** devuelve el id de la red actual (diferente en koban, ropsten, rinkeby)
- **isListening:** indica si el nodo está escuchando
- **getPeerCount:** indica la cantidad de peers conectados al nodo

```
var Net = require('web3-net');

// "Personal.providers.givenProvider" will be set if in an Ethereum supported browser.
var net = new Net(Net.givenProvider || 'ws://some.local-or-remote.node:8546');


// or using the web3 umbrella package

var Web3 = require('web3');
var web3 = new Web3(Web3.givenProvider || 'ws://some.local-or-remote.node:8546');

// -> web3.eth.net
// -> web3.bzz.net
// -> web3.shh.net
```

Introducción al framework

Web3.eth.personal

Web3.eth.personal se compone a su vez de los siguientes **submódulos** dentro de los cuales se podrán llevar a cabo diferentes tareas

- **newAccount:** permitirá la creación de una cuenta dentro de Ethereum

```
web3.eth.personal.newAccount(password, [callback])
```

- **unlockAccount:** permite desbloquear una cuenta por un tiempo determinado

```
web3.eth.personal.unlockAccount(address, password, unlockDuraction [, callback])
```

- **sign:** permite firmar data usando una cuenta específica

```
web3.eth.personal.sign(dataToSign, address, password [, callback])
```

- **ecRecover:** permite recuperar la dirección del firmante de una transacción

```
web3.eth.personal.ecRecover(dataThatWasSigned, signature [, callback])
```

Introducción al framework

Web3.ssh

Web3.ssh se compone a su vez de los siguientes **submódulos** dentro de los cuales se podrán llevar a cabo diferentes tareas

- **getInfo:** obtiene información sobre el nodo actual

```
web3.shh.getInfo([callback])
```

- **suscribe:** permite la suscripción a determinados mensajes

```
web3.shh.subscribe('messages', options [, callback])
```

- **clearSubscriptions:** elimina todas las suscripciones realizadas para escucha de mensajes

```
web3.shh.clearSubscriptions()
```

- **getFilterMessages:** recibe los mensajes que cumplen el criterio especificado

```
web3.shh.getFilterMessages(id)
```

Introducción al framework

Web3.bzz

Web3.bzz se compone a su vez de los siguientes **submódulos** dentro de los cuales se podrán llevar a cabo diferentes tareas

- **currentProvider:** devuelve la URL del provider actual o null

```
bzz.currentProvider
```

- **setProvider:** permite especificar el provider que se desea utilizar

```
web3.bzz.setProvider(myProvider)
```

- **pick:** permite abrir un browser para seleccionar archivos a subir

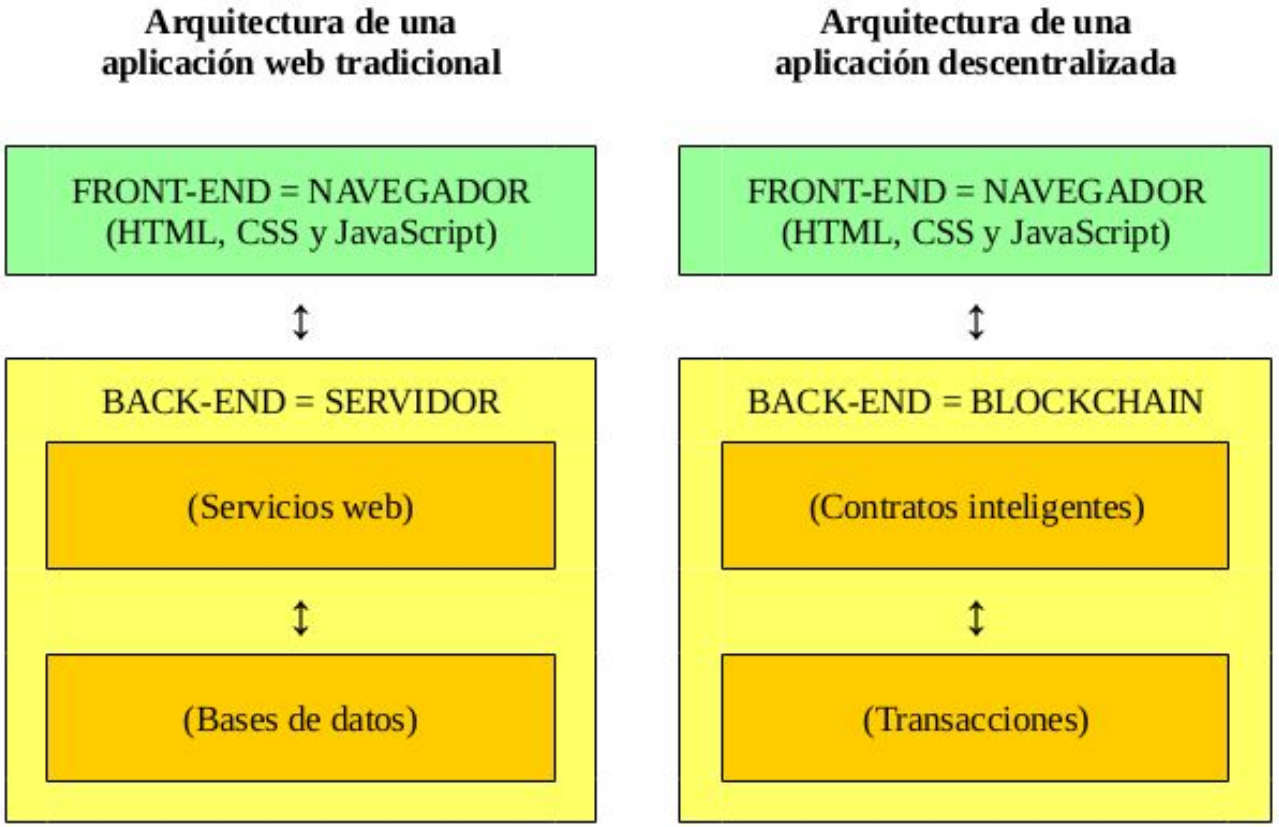
```
web3.bzz.pick.file()
web3.bzz.pick.directory()
web3.bzz.pick.data()
```

Asimismo, existen también métodos para la subida y descarga de los mismos

Arquitectura de una App Ethereum

Arquitectura de una App Ethereum

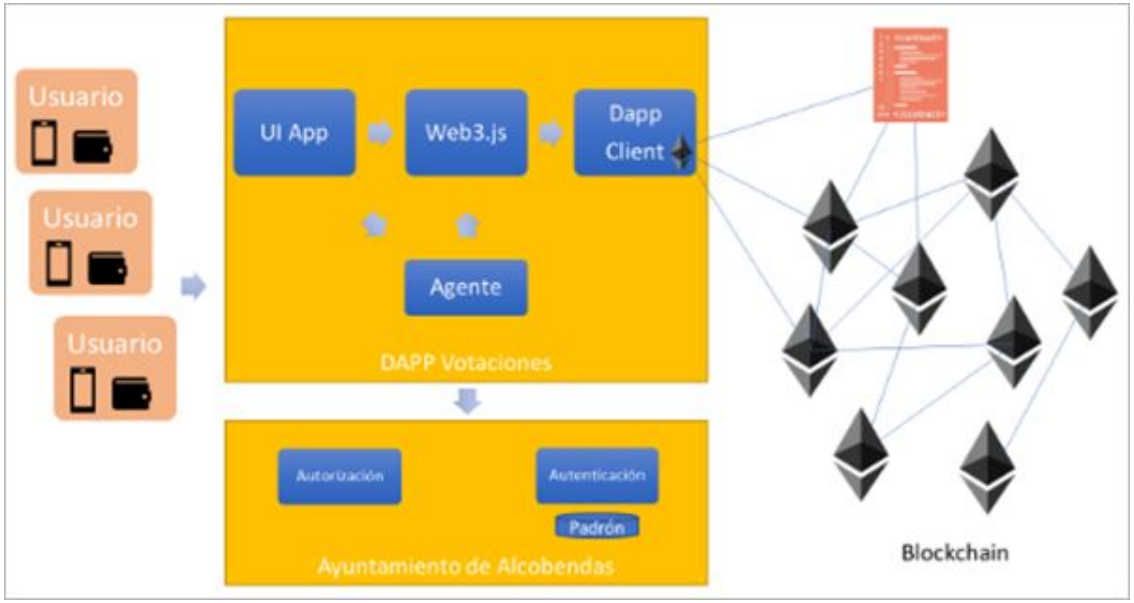
Para entender la arquitectura de una dApp sobre la Ethereum blockchain se debe comprender en que punto ingresa la blockchain respecto del modelo tradicional



Arquitectura de una App Ethereum

Es importante recordar que toda dApp ha de seguir al menos los siguientes principios

- Debe ser desentralizada
- Debe utilizar estándares preestablecidos
- Debe basarse en mecanismos públicos de consenso
- Debe comunicarse con la blockchain a traves de protocolos pre-establecidos



Instancias de Web3

Instancias de Web3

- Luego de instalar web3js se debe crear una instancia web3 y establecer un proveedor.
- Los navegadores compatibles con Ethereum como Mist o MetaMask tendrán un ethereumProvider o web3.currentProvider disponible.
- Para web3.js, verifique Web3.givenProvider.
- Si esta propiedad es nula, debe conectarse a un nodo remoto / local.

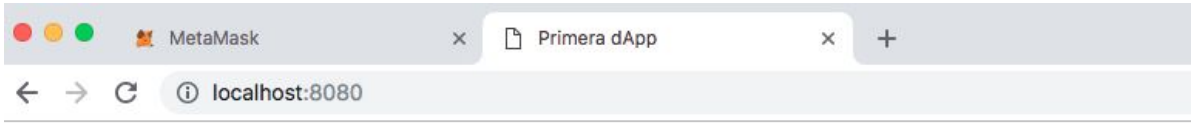
```
// in node.js use: var Web3 = require('web3');

var web3 = new Web3(Web3.givenProvider || "ws://localhost:8546");
```

Renderizado de información del Contrato

Renderizado de información del Contrato

Para el desarrollo y generación del frontend que consuma el contrato inteligente, se utilizará simplemente HTML y javascript básico. No obstante, es perfectamente posible (y esperado) la utilización de frameworks especializados como React, Angular o VUE.



Ejercicio Conección contra la Blockchain

Creando nuestra primera dApp!

Opciones	Votos del candidato
Carolina	
Mario	
Leonardo	
Joaquin	

Votar candidato

Formularios

Formularios

Mapearemos contra **HTML** y css básico toda la información row-a-row que será obtenida desde el contrato inteligente creado con **Solidity** y liberado en la **blockchain local** para poder armar el formulario que permita la generación de "votos" virtuales

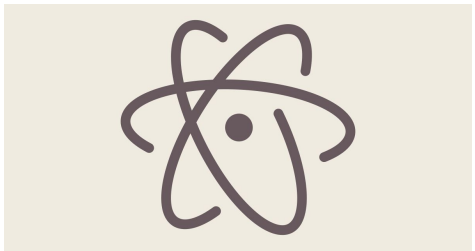
```
<body class="container">
  <h1>Ejercicio Conecci&oacute;n contra la Blockchain</h1>
  <h3>Creando nuestra primera dApp!</h3>
  <div class="table-responsive">
    <table class="table table-bordered">
      <thead>
        <tr>
          <th>Opciones</th>
          <th>Votos del candidato</th>
        </tr>
      </thead>
      <tbody>
        <tr>
          <td>Carolina</td>
          <td id="option-1"></td>
        </tr>
        <tr>
          <td>Mario</td>
          <td id="option-2"></td>
        </tr>
        <tr>
          <td>Leonardo</td>
          <td id="option-3"></td>
        </tr>
```

Realización del proyecto

Realización del proyecto

A continuación, desarrollaremos un ejemplo sumamente sencillo de una dApp. SI, una dApp completa.
Para ello utilizaremos las siguientes tecnologías, frameworks, librerías y demás

- Web3JS*
- HTML
- Javascript
- Jquery**
- Truffle
- Ganache (ex TestRPC)
- Atom



Realización del proyecto

El primer paso será navegar hasta la carpeta de destino y ejecutar el comando "**truffle unbox webpack**"

```
MacBook-Pro-de-mac:~ mac$ cd /Users/mac/Desktop/Ejercicios\ Simples\ Ethereum\ Blockchain/Web3JS_03
MacBook-Pro-de-mac:Web3JS_03 mac$ truffle unbox webpack
Downloading...
Unpacking...
Setting up...
```

Luego, se instalarán una serie de componentes que utilizaremos a lo largo del desarrollo

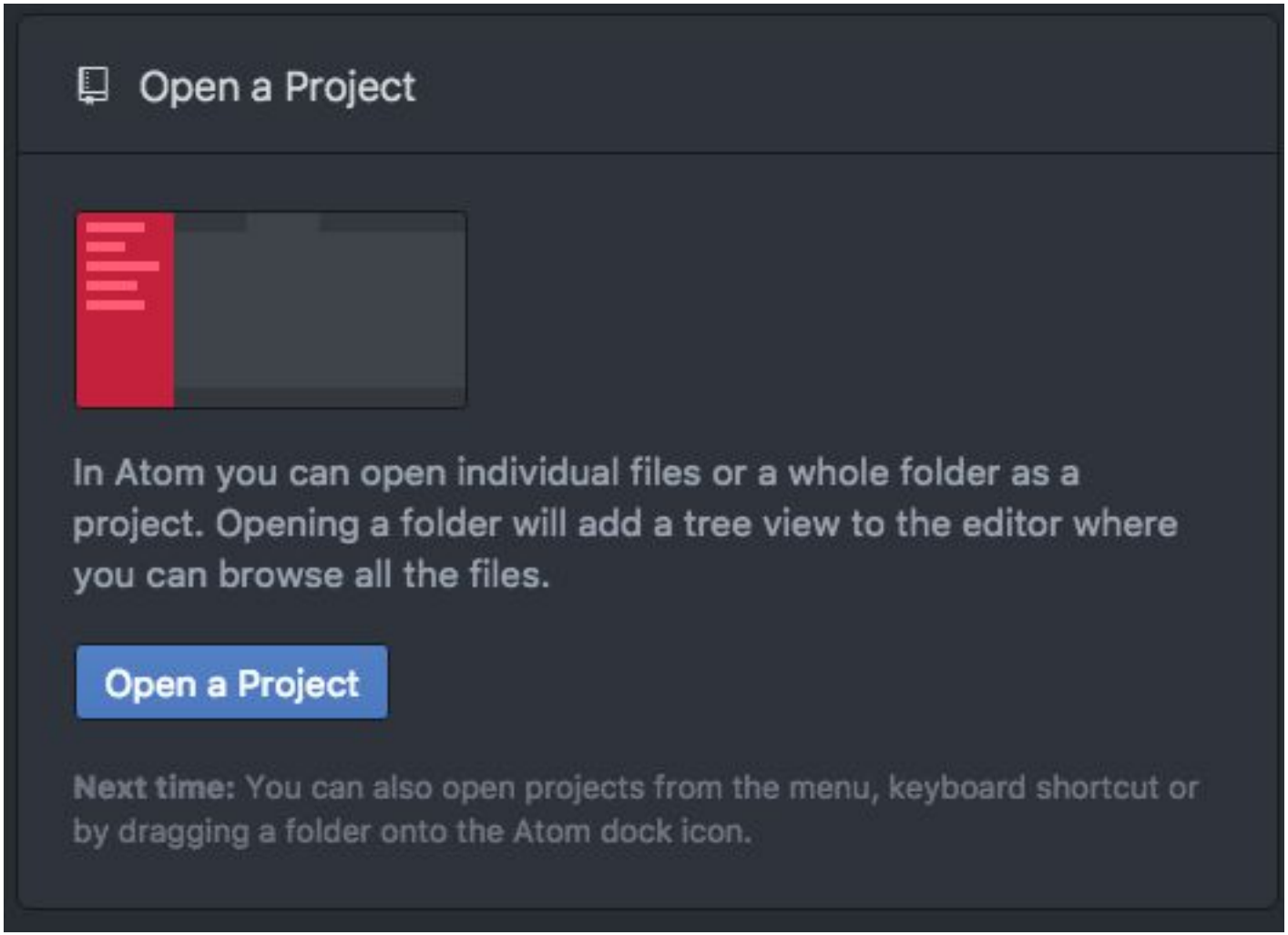
```
MacBook-Pro-de-mac:~ mac$ cd /Users/mac/Desktop/Ejercicios\ Simples\ Ethereum\ Blockchain/Web3JS_03
MacBook-Pro-de-mac:Web3JS_03 mac$ truffle unbox webpack
Downloading...
Unpacking...
Setting up...
Unbox successful. Sweet!

Commands:

  Compile:              truffle compile
  Migrate:              truffle migrate
  Test contracts:       truffle test
  Run linter:           npm run lint
  Run dev server:       npm run dev
  Build for production: npm run build
```

Realización del proyecto

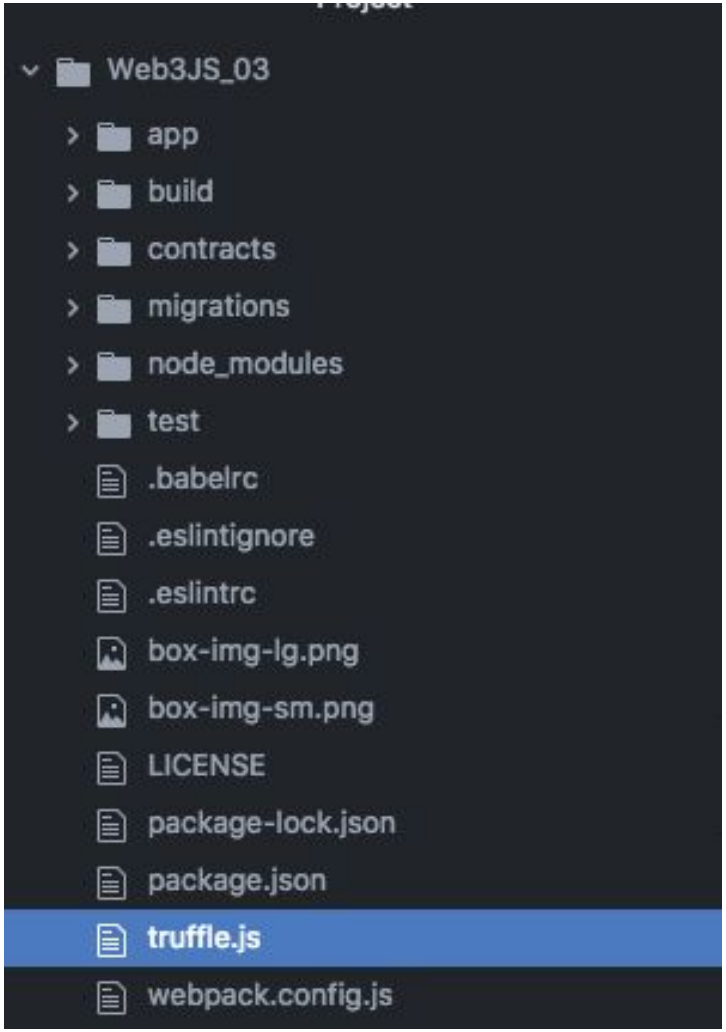
Abriremos el proyecto en Atom para comenzar el desarrollo



Realización del proyecto

Al abrirlo se encuentra la estructura básica del **proyecto** pre-configurado

- **app:** contiene los archivos de frontend que utilizaremos
- **build:** contiene la version buildeada
- **contracts:** contiene los contratos pre definidos
- **migrations:** contiene los archivos de migración
- **node_module:** contiene los módulos de nodeJS
- **test:** contiene los tests predefinidos



Realización del proyecto

En el archivo **truffle**.js indicar el **gas** y **gasprice** que estaremos dispuestos a pagar, de manera que el archivo que de de la siguiente manera

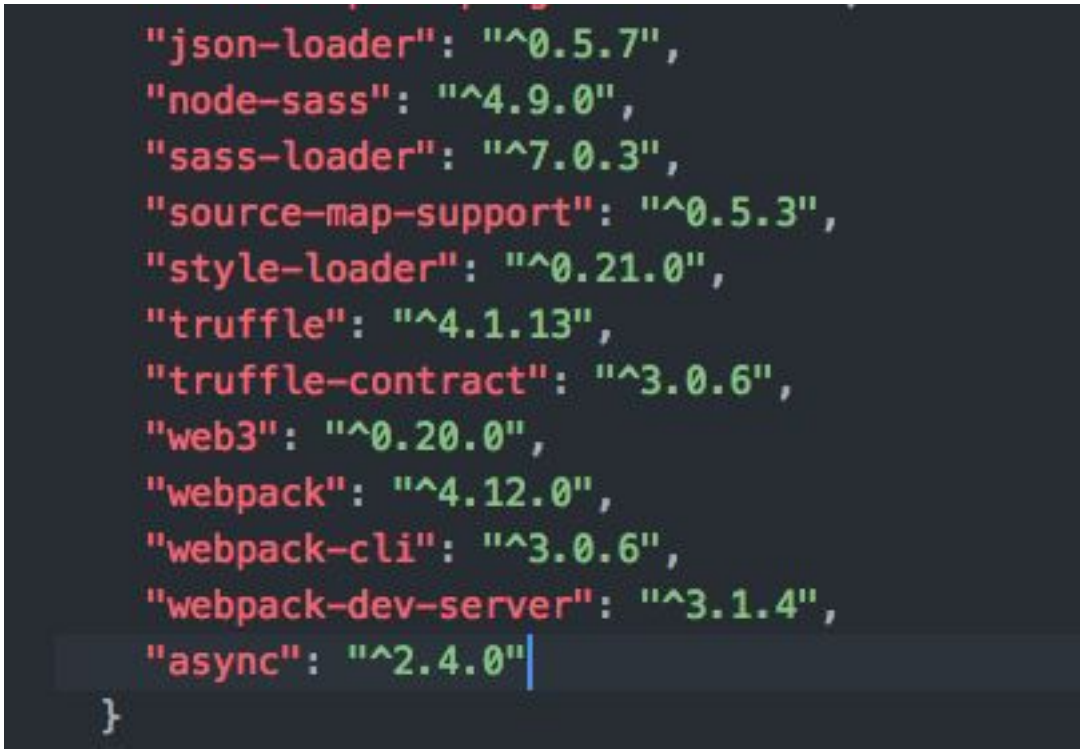
truffle.js

```
// Allows us to use ES6 in our migrations and tests.
//JDS developing with examples
require('babel-register')

module.exports = {
  networks: {
    ganache: {
      host: '127.0.0.1',
      port: 7545,
      gas: 4600000,
      gasPrice: 10e9,
      network_id: '*' // Match any network id
    }
  }
}
```

Realización del proyecto

Dentro del archivo **package**.json, deberemos agregar una referencia a la librería de **async** en su versión **^2.4.0**



La librería de **async** será luego utilizada para realizar las **llamadas** desde el fronted hacia el **contrato inteligente** de la blockchain. Luego de esto ejecutaremos **npm install** para descargar la librería.

Realización del proyecto

Crearemos un nuevo contrato llamado "**Votacion**.sol". Este contrato nos permitirá realizar una serie de "votaciones sobre candidatos que estableceremos al inicio.

```
pragma solidity ^0.4.24;

contract Votacion {
  address owner;
  mapping (bytes32 => uint8) public votesReceived;
  bytes32[] public candidateList;

  constructor(bytes32[] candidateNames) public {
    owner = msg.sender;
    candidateList = candidateNames;
  }

  function totalVotesFor(bytes32 candidate) public view returns (uint8) {
    require(validCandidate(candidate));
    return votesReceived[candidate];
  }

  function voteForCandidate(bytes32 candidate) public {
    require(validCandidate(candidate));
    votesReceived[candidate] += 1;
  }

  function validCandidate(bytes32 candidate) public view returns (bool) {
    for(uint i = 0; i < candidateList.length; i++) {
      if (candidateList[i] == candidate) {
        return true;
      }
    }
    return false;
  }
}
```

Realización del proyecto

Dentro del archivo "**2_deploy_contracts**.js", modificaremos la función de manera que quede de la siguiente manera

```
var Votacion = artifacts.require("Votacion");

module.exports = async function (deployer, network, accounts) {
  let futura = Votacion.new(['Carolina','Mario','Leonardo','Joaquín']);

  let instanciaVotacion = await futura;
  //printeamos la dirección
  console.log(instanciaVotacion.address);
}
```

Habremos modificado la función teniendo parámetros como **network** y **accounts**. De igual manera, loguearemos el **address** de la **instancia** creada dado que la necesitaremos en pasos próximos.

Realización del proyecto

Modificaremos el archivo agregando lo siguiente

```
development: {
  host: "localhost",
  port: 8545,
  network_id: "*" // Match any network id
}
```

Finalmente, ejecutaremos en una terminal "**ganache-cli**" y en otra "**truffle migrate –reset**"

```
[MacBook-Pro-de-mac:Web3JS_03 mac$ truffle migrate --reset
Using network 'development'.

Running migration: 1_initial_migration.js
  Deploying Migrations...
  ... 0xc97c5cc18df00b897d20d5033654cf7990db6b6f9cc3d4cbd39b1ff761267175
  Migrations: 0x3d9863a5114841e898464f5381cff0ff23d9136d
Saving successful migration to network...
  ... 0xd5a486a820a883307cbe49b2810fca0a3b1820bdd1413cc79f255dd97e1ffd32
Saving artifacts...
Running migration: 2_deploy_contracts.js
Saving successful migration to network...
  ... 0xf6c4cb11700966ff264e30119f61a44047df3ad911febb1f22399bc7559a3fa4
  ... 0x233cf08b980d33c47aaa966d2fac669a328504518a441c04067229ea34f2934d
Saving artifacts...
MacBook-Pro-de-mac:Web3JS_03 mac$
```

Realización del proyecto

Reemplazaremos el archivo **index**.html tomandolo desde el contenido extra del Alumni. La idea del **index**.html es la de poder permitirnos generar una interfaz sencilla que conectaremos via **WEB3** contra nuestro contrato inteligente

```
            <th>Votos del candidato</th>
          </tr>
        </thead>
        <tbody>
          <tr>
            <td>Carolina</td>
            <td id="option-1"></td>
          </tr>
          <tr>
            <td>Mario</td>
            <td id="option-2"></td>
          </tr>
          <tr>
            <td>Leonardo</td>
            <td id="option-3"></td>
          </tr>
          <tr>
            <td>Joaquin</td>
            <td id="option-4"></td>
          </tr>
        </tbody>
      </table>
    </div>
    <input type="text" id="candidato" />
    <a href="#" onclick="App.votarCandidato()" class="btn btn-primary">Vote</a>
</body>
<script src="https://cdn.rawgit.com/ethereum/web3.js/develop/dist/web3.js"></script>
<script src="https://code.jquery.com/jquery-3.1.1.slim.min.js"></script>
<script src="./app.js"></script>
</html>
```

Realización del proyecto

Modificaremos el archivo index.js de la siguiente manera

```
import "../stylesheets/app.css";

import { default as Web3} from 'web3';
import { default as contract } from 'truffle-contract'

//IMPORTANTE: el artefacto proviene del .json y no del .sol!
import votacion_artifacts from '../../build/contracts/Votacion.json'

var Votacion = contract(votacion_artifacts);

//NOTA: ejecutar contractInstance.address
// para obtener el address del contrato y pegarlo para reemplazar el siguiente
var contractInstance = Votacion.at('0x658146f01a63e1cd179a3dae99afd6e4007af51a');
var candidatos = {"Carolina": "option-1", "Mario": "option-2", "Leonardo": "option-3", "Joaquin": "option-4"};
```

En efecto, lo que estamos haciendo es importar **Web3**, importar el **contrato** desde truffle-contract y armar el **artefacto** que luego utilizaremos.
Se carga la **instancia** desde la dirección pre-generada y se arma un **array de candidatos** que luego utilizaremos.

Realización del proyecto

Agregaremos, dentro del archivo **index**.js las definiciones para window.**App** y para window.**addEventListener**

```
window.App = {

};

window.addEventListener('load', function(){

});
```

Luego, dentro de ambos, completaremos las **funciones** y **validaciones** necesarias para poder **ejecutar** tanto las conecciones como consutas desde nuestro frontend hacia nuestros **contratos inteligentes** deployados via truffle

Realización del proyecto

Completaremos el código del evento addEventListener de la siguiente manera

```
window.addEventListener('load', function(){
  if (typeof web3 !== 'undefined') {
    console.warn("Usando web3 desde origen externo")
    window.web3 = new Web3(web3.currentProvider);
  } else {
    console.warn("No se detectó un web3 provider");
    window.web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:8545"));
  }

  App.start();
});
```

Siempre será necesario validar la existencia o no de la variable global **web3** dado que de tener **Metamask** (u otro complemento que lo supla) ésta deberá existir para poder validar y firmar las transacciones que desde el frontend enviemos.

Realización del proyecto

Dentro del archivo **index**.js, dentro de window.App, generaremos la definición de las 3 funciones que estaremos utilizando para la interconección entre el frontend y el contrato

```
window.App = {
  start: function() {

  },
  votarCandidato: async function(message) {
  },

  actualizarVotosCandidato: function(nombreCandidato) {
  }
};
```

Luego incorporaremos la funcionalidad a cada función, las cuales se ejecutarán al **inicio**, al **votar** y al querer **actualizar** la información respectivamente.

Realización del proyecto

Completaremos el código de la función **start** de la siguiente manera

```
start: function() {
  var self = this;

  Votacion.setProvider(web3.currentProvider);
  Votacion.defaults({from: web3.eth.coinbase});

  //Primer paso, obtención de cuentas
  web3.eth.getAccounts(function(err, accs) {
      if (err != null) {
        alert("Error al querer obtener cuentas. No se podrá continuar");
        return;
      }
      if (accs.length == 0) {
        alert("No hay cuentas disponibles");
        return;
      }
    });

    var candidatosArray = Object.keys(candidatos);
    //Recorremos el array de candidatos
    for (var i = 0; i < candidatosArray.length; i++) {
      let nombreCandidato = candidatosArray[i];//tomamos el nombre
      //Refrescamos la UI del candidato
      //self.actualizarVotosCandidato(nombreCandidato);
    }
  },
```

Realización del proyecto

Completaremos el código de la función **actualizarVotosCandidato** de la siguiente manera

```
actualizarVotosCandidato: function(nombreCandidato) {
  let div_id = candidatos[nombreCandidato];
  //Aquí ya estamos llamando Al contrato!!
    var votosPromise = contractInstance.totalVotesFor.call(nombreCandidato);
    //Al ser una promesa, tendremos que aplicarle el then para poder hacer
    //lo que queremos con el resultado
    votosPromise.then(votos => $("#"+ div_id).html(votos.toString()));
  }
```

Ésta función actualizará la **cantidad de votos** de un candidato, consultando dicho valor directamente desde nuestro **contrato** inteligente deployado.
Para hacerlo, se deberá utilizar una promesa y mapear directo al **HTML** el valor resultante obtenido.

Realización del proyecto

Completaremos el código de la función votarCandidato de la siguiente manera

```
votarCandidato: async function(message) {
  var self = this;

  var nombreCandidato = $("#candidate").val();
  var address = web3.eth.coinbase;
  //Aquí estamos enviando nuestro voto a la blockchain
  var voto = await contractInstance.voteForCandidate(nombreCandidato, {from: address});

  //await self.actualizarVotosCandidato(candidateName);

  },
```

- Obtenemos el valor del candidato
- Tomamos una **dirección** de origen
- Generamos la transacción esperando al resultado mediante el uso de **await**

Realización del proyecto

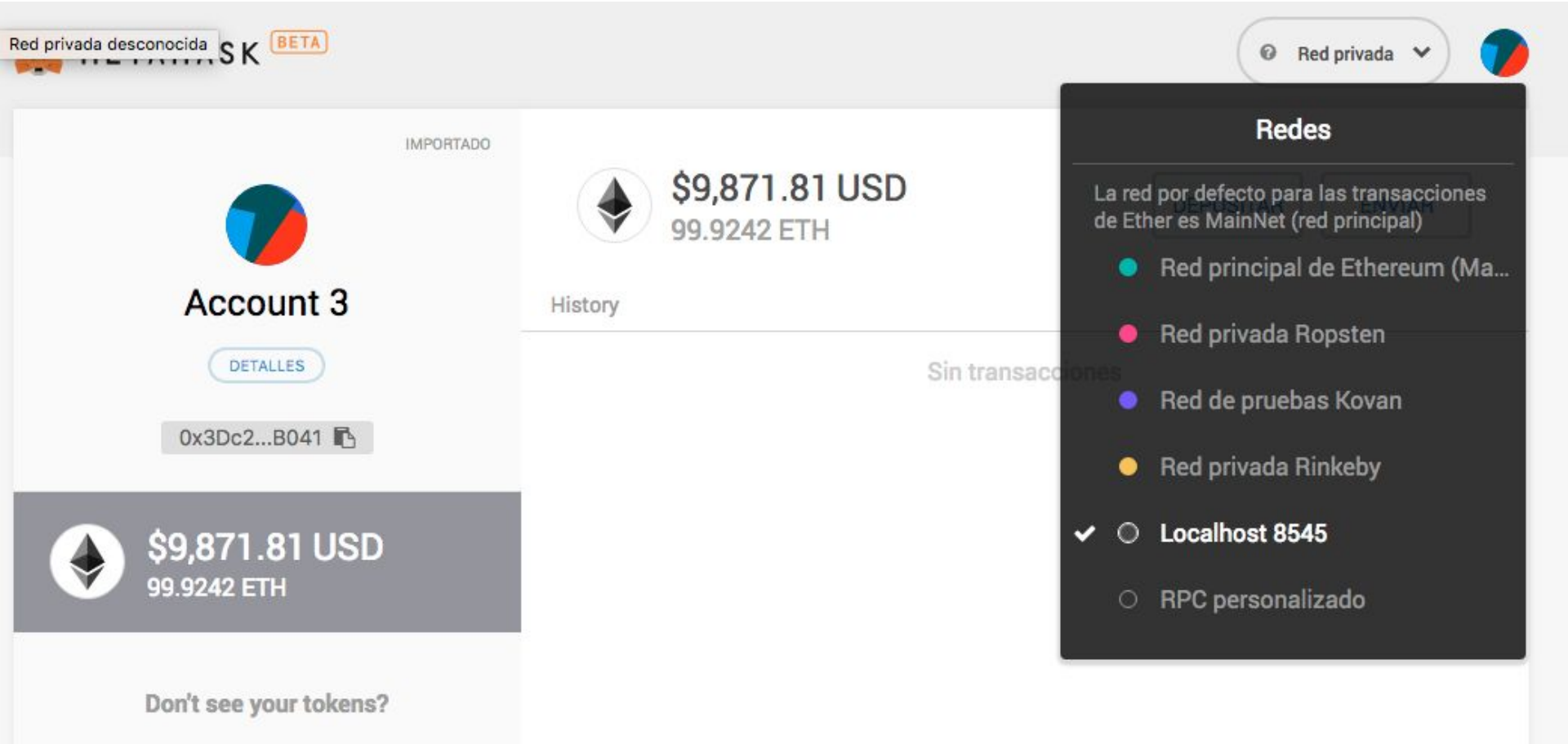
Ejecutamos el comando **npm run dev** desde la terminal

```
> webpack-dev-server

i ｢wds｣: Project is running at http://localhost:8080/
i ｢wds｣: webpack output is served from /
⚠ ｢wdm｣: Hash: 146fb35596bc71505b8c
Version: webpack 4.27.1
Time: 32949ms
Built at: 12/09/2018 5:08:53 PM
     Asset      Size  Chunks                    Chunk Names
    app.js   726 KiB       0  [emitted]  [big]  main
app.js.map  2.71 MiB       0  [emitted]         main
index.html  1.31 KiB          [emitted]
Entrypoint main [big] = app.js app.js.map
 [36] ./node_modules/url/url.js 22.8 KiB {0} [built]
[133] multi (webpack)-dev-server/client?http://localhost:8080 ./app/scripts/index.js 40 bytes {0} [built]
[134] (webpack)-dev-server/client?http://localhost:8080 7.78 KiB {0} [built]
[140] ./node_modules/strip-ansi/index.js 161 bytes {0} [built]
[142] ./node_modules/loglevel/lib/loglevel.js 7.68 KiB {0} [built]
[143] (webpack)-dev-server/client/socket.js 1.05 KiB {0} [built]
[145] (webpack)-dev-server/client/overlay.js 3.58 KiB {0} [built]
[150] (webpack)/hot sync nonrecursive ^\.\/log$ 170 bytes {0} [built]
[152] (webpack)/hot/emitter.js 75 bytes {0} [built]
[153] ./app/scripts/index.js 3.84 KiB {0} [built]
[154] ./node_modules/babel-runtime/regenerator/index.js 49 bytes {0} [built]
[157] ./node_modules/babel-runtime/helpers/asyncToGenerator.js 906 bytes {0} [built]
[192] ./node_modules/babel-runtime/core-js/object/keys.js 92 bytes {0} [built]
[196] ./app/styles/app.css 1.18 KiB {0} [built]
[201] ./node_modules/web3/index.js 193 bytes {0} [built]
    + 374 hidden modules
```

Realización del proyecto

Abriremos Chrome, importaremos la **clave privada** de la primer cuenta de ganache-cli para conectarlo



Realización del proyecto

Abriremos Chrome, importaremos la **clave privada** de la primer cuenta de ganache-cli para conectarlo. Navegamos hasta el sitio que hemos construído

Realización del proyecto

Finalmente, vemos como al presionar ”Votar candidato” Metamask intercede para firmar la tansacción