ETSIT UPM

dit UPM

Blockchain: Desarrollo de Aplicaciones

Truffle - Contrato Contador Caso Estudio #1

BCDA 2018

Versión: 2018-11-02

Índice

- ~~Introducción, documentación.~~
- ~~Ganache~~
- Caso de Estudio #1: SC Contador
  - Crear un proyecto que usa un Smart Contract llamado Contador.
  - Usando Truffle, crear pruebas, scripts, cmds y una dapp.
- Caso de Estudio #2: Webpack
  - Añadir Webpack al caso de estudio #1
- Caso de Estudio #3: Introducción a React y Redux
  - Crear un ejemplo de una SPA con un nuevo contador.
    - El contador es diferente al de los casos de estudio anteriores: Sin Blockchain.
    - Partiendo de la configuración de Webpack del caso de estudio #2 y ampliarla para soportar react.
- Caso de Estudio #4: Smart Contract Contador con React y Redux. Y SIN Drizzle.
  - Unir todo lo visto en los casos de estudio anteriores.
- Caso de Estudio #5: SC Contador con React, Redux y CON Drizzle.
  - Añadir Drizzle al caso de estudio #4.

Objetivo

- Ilustrar el uso de Truffle en el proceso de desarrollo
  - Usando un Smart Contract sencillo: un Contador.
  - Crear un proyecto, hacer pruebas, crear aplicaciones y Dapps, ...

Instalaciones

- Instalar Truffle:
  - Instalación de sistema:

```
$ sudo npm install -g truffle
$ truffle version
Truffle v4.1.14 (core: 4.1.14)
Solidity v0.4.24 (solc-js)
```

  - Instalación en el directorio local (node_modules):

```
$ npm install truffle
$ npx truffle version
Truffle v4.1.14 (core: 4.1.14)
Solidity v0.4.24 (solc-js)
```

- Instalar Ganache.
  - Puede descargarse desde la página:

    https://trufflframework.com/ganache

Crear Proyecto y Contrato

- Creamos un proyecto nuevo en un directorio vacío:

```
$ mkdir contador
$ cd contador
$ truffle init
```

  - Examinar los ficheros y directorios creados: contracts, migrations, ...

- Creamos el fichero **`contracts/Contador.sol`** con el código del contrato inteligente.
  - Crear el fichero a mano o ejecutando:

```
$ truffle create contract Contador
```

  - Editar el contenido del fichero añadiendo el código del contrato.

- Compilar los contratros:

```
$ truffle compile
```

Smart Contract: **Contador**

contracts/Contador.sol

```
pragma solidity ^0.4.24;

contract Contador {

    uint8 public valor = 0;

    event Tic(string msg, uint8 out);

    function incr() public {
        valor++;
        emit Tic("Actualizado", valor);
    }

    function() public {
        revert();
    }
}
```

Migración para Desplegar

- Crear un fichero de migración para desplegar los contratos.
  - Crear el fichero de migración a mano o ejecutar:

```
$ truffle create migration DeployContador
```

  - Se creará el fichero:

```
migrations/1539598546_deploy_contador.js
```

  - Editar su contenido:

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

module.exports = function(deployer) {
  deployer.deploy(Contador);
};
```

- Añadir en la sección **networks** de **truffle.js** la configuración para desplegar en la red proporcionada por Ganache:

```
module.exports = {
    networks: {
        development: {
            host: "127.0.0.1",
            port: 7545,
            network_id: "*" // Match any network id
        }
    }
};
```

  - Ganache está corriendo en la máquina local y está escuchando las peticiones RPC en el puerto 7545.
  - Consultar en la documentación de Truffle las opciones de configuración.
  - En Windows hay que usar el fichero **truffle-config.js**.

- Ejecutar las migraciones:

```
$ truffle migrate
```

  - Se desplegará en la red Ganache una instancia de los contratos Migrations y Contador.

Testing

- Crear pruebas:
  - Usando **Solidity**.
    - Crear un fichero con un contrato de pruebas en **`test/TestContador.sol`**.
    - Ejecutar las pruebas de este fichero con:

```
$ truffle test test/TestContador.sol
```

  - Usando **Javascript**.
    - Crear un fichero con un contrato de pruebas en **`test/TestContador.js`**.
    - Ejecutar las pruebas de este fichero con:

```
$ truffle test test/TestContador.js
```

- Para ejecutar todas las pruebas existentes:

```
$ truffle test
```

- Consultar documentación para ver los detalles de probar con **Solidity**:
  - Cada fichero solidity se ejecuta como un conjunto de pruebas independientes,
    - creando en un entorno de despliegue nuevo (limpio) para la ejecución de cada fichero.
  - Truffle proporciona la librería **`truffle/Assert.sol`** que contiene numerosas funciones de comprobación: **fail**, **equal**, **notEqual**, **isEmpty**, **isNotEmpty**, **isZero**, **isNotZero**, **isTrue**, **isFalse**, **isAbove**, **isAtLeast**, **lengthEqual**, **balanceEqual**, ...
    - esta disponible en **`node_modules/truffle/build/Assert.sol`**
  - Truffle proporciona la librería **`truffle/DeployedAddresses.sol`** que permite acceder a la dirección donde se ha desplegado un contrato.
    - Para usar los contratos desplegados debe importarse el código de los contratos.
  - Los nombre de los contratos de prueba deben empezar con el prefijo **Test**.
  - Los nombres de las funciones de prueba deben empezar con el prefijo **test**.
    - Se ejecutan en el mismo orden en el que aparecen en el contrato.
  - Existen varios ganchos: **beforeAll**, **beforeEach**, **afterAll** and **afterEach**.
    - Son funciones que se ejecutan una vez antes de todos los test, antes de cada test, después de pasar todos los test, o después de pasar cada test.
    - Se usan para realizar acciones de configuración o limpieza.
  - ...

- Consultar documentación para ver los detalles de probar con **Javascript**:
  - Se basa en el framework de pruebas **Mocha**.
    - Los ficheros de prueba deben estar en el directorio **test** y deben tener la extensión **.js**.
    - Añade la función **contract()**, que es idéntica a **describe()** pero:
      - Se crea un nuevo entorno de ejecución limpio para cada contract() definido, es decir, se redespliegan los contratos otra vez para que empiecen en su estado inicial.
      - La función contract() proporciona la lista de cuentas disponibles en el cliente para que se usen en las pruebas.
  - Usa las assertions del paquete **Chai**.
  - Se usa el método **artifacts.require()** para cargar las abstracciones de los contratos que se necesiten en las pruebas.
    - Estas abstracciones se usan para interaccionar desde Javascript con las instancias de los contratos desplegadas.
  - Los ficheros de prueba tienen acceso a una instancia **web3** ya configurada con el proveedor correcto.
  - Puede cambiarse la configuración de Mocha para modificar su modo de funcionamiento.
  - ...

TestContador.sol

```
pragma solidity ^0.4.24;

import "truffle/Assert.sol";
import "truffle/DeployedAddresses.sol";
import "../contracts/Contador.sol";

contract TestContador {

    Contador contador = Contador(DeployedAddresses.Contador());

    // Testing valor inicial es 0
    function testZero() public {

        uint8 c =  contador.valor();

        uint expected = 0;

        Assert.equal(c, expected,
                     "El contador deberia ser 0 inicialmente.");
    }
```

```
    // Testing the incr() function
    function testIncr() public {

        uint8 c1 =  contador.valor();

        contador.incr();
        contador.incr();
        contador.incr();
        contador.incr();

        uint8 c2 =  contador.valor();

        uint res = c2 - c1;
        uint expected = 4;

        Assert.equal(res, expected,
                     "El contador deberia haberse incrementado en 4.");
    }

}
```

TestContador.js

```
var Contador = artifacts.require("./Contador.sol");

contract('Usamos un Contador:', accounts => {

  let contador;

  before(async () => {
    contador = await Contador.deployed();
  });
```

Nota:
La función **artifacts.require** se define en **trufflesuite/truffle-core**.
Es una función que se usa durante la fase de desarrollo para cargar los contratos cuando se ejecutan tests, migraciones o scripts desde el comando **truffle**.
Para cargar contratos en producción se usa **truffle-contract**.

```
                    // Es la misma prueba usando promesas o usando async/await

it("el valor inicial debe ser 0", () => {

  return contador.valor.call()
  .then(function(value) {

    assert.equal(value.valueOf(), 0,
                 "El valor inicial no es 0.");
  });
});


it("el valor inicial debe ser 0", async () => {

  const value = await contador.valor.call();

  assert.equal(value.valueOf(), 0,
               "El valor inicial no es 0.");
});
```

```
                                        // Esta prueba usa promesas
it("incrementar en uno el contador", () => {

  let c1, c2;

  return contador.valor.call()
  .then(value => {
    c1 = value;
    contador.incr();
  })
  .then(() => contador.valor.call() )
  .then(value => {
    c2 = value;

    const incr = c2 - c1;
    assert.equal(incr.valueOf(), 1,
                 "El incremento del valor no es 1.");
  });
});
```

// La misma prueba anterior usando async/await

```
  it("incrementa en cuatro el contador", async () => {

    let c1 = await contador.valor.call();
    await contador.incr();
    await contador.incr();
    await contador.incr();
    await contador.incr();
    let c2 = await contador.valor.call();

    const incr = c2 - c1;
    assert.equal(incr.valueOf(), 4,
                 "El incremento del valor no es 4.");
  });

});
```

Scripts Externos

- Truffle permite crear un entorno para ejecutar scripts externos que se conectan al nodo de la red Ethereum configurado en truffle, e interactúan con los contratos que hemos desplegado.
- Para ejecutar un script externo:+

```
$ truffle exec <path_fichero.js>
```

- El formato del fichero que contiene el script debe ser el siguiente:

```
module.exports = function(callback) {
  // perform actions
}
```

- Es un módulo que exporta una función que toma una callback como parámetro.
  - El script debe llamar a la callback cuando finalice.
  - La callback puede aceptar un error como su único parámetro.
    - Si se proporciona un error, la ejecución se detiene y el proceso devuelve un código de salida distinto de cero.

Script: scripts/ejemplo1.js

```
module.exports = callback => {

    var Contador = artifacts.require("./Contador.sol");

    let f = async () => {    // Definir la funcion f que devuelve promesas

        let contador = await Contador.deployed();
        let c1 = await contador.valor.call();

        await contador.incr();
        await contador.incr();
        await contador.incr();
        await contador.incr();

        let c2 = await contador.valor.call();

        const incr = c2 - c1;
        console.log("El incremento del valor es", incr);
    }

    f()                                                  // Invocar la funcion f
    .catch(err => console.log(`Error: ${err}`))          // Capturar errores
    .then(() => callback() );                            // Terminar
};
```

Nota: La función **artifacts.require** se define en **trufflesuite/truffle-core**.
Es una función que se usa durante la fase de desarrollo para cargar los contratos cuando se ejecutan tests, migraciones o scripts desde el comando **truffle**.
Para cargar contratos en producción se usar **truffle-contract**.

Script: scripts/ejemplo2.js

```
module.exports = async callback => {     // Usando async aqui

    try {
        const Contador = artifacts.require("./Contador.sol");

        let contador = await Contador.deployed();

        let c1 = await contador.valor.call();

        await contador.incr();
        await contador.incr();
        await contador.incr();
        await contador.incr();

        let c2 = await contador.valor.call();

        const incr = c2 - c1;
        console.log("El incremento del valor es", incr);
    } catch(err) {    // Capturar errores
        console.log(`Error: ${err}`);
    }

    callback();       // Terminar
};
```

Script: scripts/ejemplo3.js

```
module.exports = async callback => {
    const Contador = artifacts.require("./Contador.sol");
    let contador = await Contador.deployed();
    let c1 = await contador.valor.call();
    await contador.incr();
    await contador.incr();

    let result = await contador.incr();
    console.log(result);

    // Iterar por los eventos para ver si se disparo el evento Tic.
    result.logs.forEach(log => {
        if (log.event == "Tic") {
            const msg = log.args.msg;
            const out = log.args.out;
            console.log(`LOG: ${msg} >> ${out}`);
        }
    });

    let c2 = await contador.valor.call();
    const incr = c2 - c1;
    console.log("El incremento del valor es", incr);
    callback();
};
```

```
$ truffle exec scripts/ejemplo1.js
Using network 'development'.

El incremento del valor es 4
$ truffle exec scripts/ejemplo2.js
Using network 'development'.

El incremento del valor es 4
$ truffle exec scripts/ejemplo3.js
Using network 'development'.

{ tx: '0x1182c166a9c8477c62f4294f1a122d2db2cf6b78c984d12d5b12c563cecd6ffd',
  receipt:
   { transactionHash: '0x1182c166a9c8477c62f4294f1a122d2db2cf6b78c984d12d5b12c563cecd6ffd',
     transactionIndex: 0,
     blockHash: '0xb187d37f28fbfd95fb426764bbaaf4cd2540caba0ceccfc79c30a6445f383874',
     blockNumber: 95,
     gasUsed: 29087,
     cumulativeGasUsed: 29087,
     contractAddress: null,
     logs: [ [Object] ],
     status: '0x1',
     logsBloom:
'0x00000000000000000000000020000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000002000
00000000000000000000000000000000000000000000100000000000000000000800000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000020000000000000000000000000000000000000000000000000000000000000000000000
00004000000000000000000000000000000000000000000000000000000000000000000000000000' },
  logs:
   [ { logIndex: 0,
       transactionIndex: 0,
       transactionHash: '0x1182c166a9c8477c62f4294f1a122d2db2cf6b78c984d12d5b12c563cecd6ffd',
       blockHash: '0xb187d37f28fbfd95fb426764bbaaf4cd2540caba0ceccfc79c30a6445f383874',
       blockNumber: 95,
       address: '0xa7920759adba8a84447e30947fc1dce04e9a10f4',
       type: 'mined',
       event: 'Tic',
       args: [Object] } ] }
LOG: Actualizado >> 46
El incremento del valor es 3
```

Aplicaciones de Línea de Comandos

- Aplicaciones que se lanzan desde la línea de comandos y que se ejecutan independientemente sin usar el comando Truffle.
- El código de estas aplicaciones deben seguir los siguientes pasos:
  - Cargar los artefactos de los contratos necesarios.
    - Son los ficheros JSON creados en el directorio build/contracts.
  - Usar el módulo truffle-contract para transformar los artefactos en abstracciones de los contratos.
    - Las abstracciones permiten usar fácilmente los contratos.
  - Provisionar las abstracciones de los contratos con un proveedor de Web3.
    - Crear un proveedor que se conecta al nodo Ethereum.
  - Usar los contratos, ...

Ejemplo: app1/ejemplo.js

- Este ejemplo es una aplicación nodejs.
- Instalar previamente los paquetes web3 y truffle-contract.

```
$ npm init -y
$ npm install --save web3 truffle-contract
```

  - Nota: A fecha de hoy se ha instalado web3 versión 1 beta 36

- Para lanzar la aplicación:

```
$ node app1/ejemplo.js
```

- **Nota**: Hay una incompatibilidad entre las versiones web3@1 y truffle-contract@3.0.x. Sale un error al intentar acceder al despliegue de un contrato, quejándose de que no puede llamar a apply sobre undefined. Se arregla añadiendo el siguiente parche:

```
if (typeof Contador.currentProvider.sendAsync !== "function") {
    Contador.currentProvider.sendAsync = function() {
        return Contador.currentProvider.send.apply(
            Contador.currentProvider, arguments
        );
    };
}
```

app1/ejemplo.js

```
#! /usr/local/bin/node

//
// Ejemplo de una app que se lanza desde la linea de comando.
//

console.log("Ejemplo APP");

const Web3 = require("Web3");
const TruffleContract = require("truffle-contract");

// Cargar el artefacto del contrato Contador (json)
const json = require("../build/contracts/Contador.json");

// Crear la abstraccion del contrato Contador
const Contador = TruffleContract(json);

//  Ganache es el proveedor de Web3.
let web3Provider = new Web3.providers.HttpProvider('http://localhost:7545');

// Provisionar el contrato con el proveedor web3
Contador.setProvider(web3Provider);
```

```
// Instancia de Web3
let web3 = new Web3(web3Provider);

web3.eth.net.isListening()
.catch(() => {
    throw new Error("No puedo conectar con el nodo Ethereum.");
})
.then(async () => {
    console.log('Estoy conectado con el nodo Ethereum.');

    // Workaround for a compatibility issue between
    // web3@1 and truffle-contract@3.0.x
    if (typeof Contador.currentProvider.sendAsync !== "function") {
        Contador.currentProvider.sendAsync = function() {
            return Contador.currentProvider.send.apply(
                Contador.currentProvider, arguments
            );
        };
    }
```

```
// Usar la cuenta de usuario

// Usar la primera cuenta del usuario
const accounts = await web3.eth.getAccounts();
if (accounts.length == 0) {
    throw new Error("No hay cuentas.");
}
const account = accounts[0];
console.log("Cuenta de usuario =", account);
```

```
    // Usar el contrato

    // Obtener el contrato desplegado
    const contador = await Contador.deployed();
    console.log("Dirección del Contrato =", contador.address);

    const c1 = await contador.valor.call();
    await contador.incr({from: account});
    await contador.incr({from: account});
    await contador.incr({from: account});
    await contador.incr({from: account});
    const c2 = await contador.valor.call();
    console.log(c1.valueOf(), ">> ", c2.valueOf());
})
.catch(error => {
    console.log(error);
}).then(() => {
    console.log("FIN");
});
```

```
$ node app1/ejemplo.js
Ejemplo APP
Estoy conectado con el nodo Ethereum.
Cuenta de usuario =
0xcCdD01920308e39eb95af3dA86a9Dc454C0Cfe5F
Address del Contrato =
0xa7920759adba8a84447e30947fc1dce04e9a10f4
50 >>  54
FIN
```

app1/ejemplo2.js

```
#! /usr/local/bin/node

//
// Ejemplo de una app que se lanza desde la linea de comando.
//
// Observar el evento Tic.
//

console.log("Ejemplo APP");

const Web3 = require("Web3");
const TruffleContract = require("truffle-contract");

// Cargar el artefacto del contrato Contador (json)
const json = require("../build/contracts/Contador.json");

// Crear la abstraccion del contrato Contador
const Contador = TruffleContract(json);

//  Ganache es el proveedor de Web3.
let web3Provider = new Web3.providers.HttpProvider('http://localhost:7545');

// Provisionar el contrato con el proveedor web3
Contador.setProvider(web3Provider);
```

```
// Instancia de Web3
let web3 = new Web3(web3Provider);

web3.eth.net.isListening()
.catch(() => {
    throw new Error("No puedo conectar con el nodo Ethereum.");
})
.then(async () => {
    console.log('Estoy conectado con el nodo Ethereum.');

    // Workaround for a compatibility issue between
    // web3@1 and truffle-contract@3.0.x
    if (typeof Contador.currentProvider.sendAsync !== "function") {
        Contador.currentProvider.sendAsync = function() {
            return Contador.currentProvider.send.apply(
                Contador.currentProvider, arguments
            );
        };
    }
```

```
    // Obtener el contrato desplegado
    const contador = await Contador.deployed();
    console.log("Dirección del Contrato =", contador.address);

    // Observar evento Tic
    contador.Tic((err, event) => {
        console.log("Se ha producido un evento Tic:");
        if (err){
            console.log(err);
        } else {
            var msg = event.args.msg;
            var out = event.args.out;
            console.log(" * Msg =", msg);
            console.log(" * Out =", out.valueOf());
        }
    });
})
.catch(error => {
    console.log(error);
}).then(() => {
    console.log("FIN");
});
```

```
$ node app1/ejemplo2.js
Ejemplo APP
Estoy conectado con el nodo Ethereum.
Address del Contrato =
0xa7920759adba8a84447e30947fc1dce04e9a10f4
FIN
Se ha producido un evento Tic:
 * Msg = Actualizado
 * Out = 17
Se ha producido un evento Tic:
 * Msg = Actualizado
 * Out = 18
Se ha producido un evento Tic:
 * Msg = Actualizado
 * Out = 19
Se ha producido un evento Tic:
 * Msg = Actualizado
 * Out = 20
Se ha producido un evento Tic:
 * Msg = Actualizado
 * Out = 21
ETC...
```

Nota:
Cada vez que se actualiza el valor del contador, se genera un evento Tic, que se captura y se pinta una traza.

Dapp

- Dapp = Decentralised Applications
- Ejemplo de desarrollo de una aplicación web de cliente.
  - Crearemos un directorio para este desarrollo, por ejemplo **`app2`**.
  - Para esta aplicación crearemos una página HTML, un fichero CSS y un fichero Javascript con la lógica de la aplicación.
- Desarrollar un servidor Web para servir la aplicación web.
  - Crearemos un servidor con Nodejs y Express.
- Usar un navegador con la extensión MetaMask para gestionar las cuentas de usuario.

Servidor Nodejs

- Usar **Nodejs** para crear un servidor web de páginas estáticas.
- Ejecutar:

```
$ sudo npm install -g express-generator
$ express --ejs node_server
$ cd node_server
$ npm install
```

- Añadir en **`node_server/app.js`** las líneas:

```
app.use(express.static(path.join(__dirname, '../app2')));
app.use(express.static(path.join(__dirname,
                                 '../build/contracts')));
```

- Lanzaremos el servidor ejecutando:

```
$ npm start
```

- Lanzaremos varios navegadores (Chrome o firefox) con la extensión **MetaMask**,
  - para visitar la página **`http://localhost:3000`**

index.html

- El fichero **`app2/index.html`** es la página web de nuestra aplicación que es servida por el servidor Web.

- En el body:
  - Primero se crea el contenido de la página: un título, un párrafo y un elemento **span** para mostrar el valor del contador.
  - Al final, se cargan varios módulos Javascripts:
    - **jquery.js**
      - http://jquery.com
    - **web3.js**
      - https://github.com/ethereum/web3.js
    - **truffle-contract.js**
      - Descargado de la pagina **`https://github.com/trufflesuite/truffle`**.
      - Instalar el paquete **truffle-contract** con "**npm install truffle-contract**" y copiar el fichero minimizado **`dist/truffle-contract.min.js`** en **`app2/js`**.
    - **app.js**
      - Aquí es donde creamos el código de nuestra aplicación.

app2/index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="css/style.css">
    <title>Contador</title>
  </head>
  <body>
    <h1>Contador</h1>

    <p>
      Valor actual = <span id="valor"></span>
    </p>

    <button type="button" id="cincr">Incrementar</button>

    <script src="https://ajax.googleapis.com/ajax/libs/jquery/2.2.4/
jquery.min.js"></script>
    <script src="https://cdn.jsdelivr.net/gh/ethereum/web3.js/dist/
web3.min.js"></script>
    <script src="js/truffle-contract.min.js"></script>
    <script src="js/app.js"></script>
  </body>
</html>
```

style.css

app2/css/style.css

```
#valor {
   font-size: x-large;
   color: red;
}
```

app.js

- En **`app2/js/app.js`** escribimos la lógica de la aplicación.
  - Hemos creado un objeto App que encapsula todo el código desarrollado para no contaminar el entorno.
- Hay que seguir los mismos pasos que en una aplicación de línea de comandos:
  - Cargar los artefactos (json) de los contratos necesarios.
  - Usar truffle-contract para transformar los artefactos en abstracciones de los contratos.
  - Provisionar las abstracciones de los contratos con un proveedor de Web3.
    - En el navegador, este proveedor puede provenir de Metamask o Mist, o también podría ser un proveedor personalizado.
  - Usar los contratos, ...
  - Con Javascript crear funciones, programar eventos y manejadores, ...

app2/js/app.js

```
App = {
    web3Provider: null,
    Contador: null,  // Abstracción del contrato.
    contador: null,  // Instancia desplegada.

    init: function() {}, // Inicializar App.

    initWeb3: function() {}, // Inicializar web3.

    initContractAbstracts: function() {}, // Inicializar abstracción.

    initContractInstance: () => {}, // Obtener instancia desplegada.

    bindEvents: function() {}, // Configurar el botón.

    handleIncr: function(event) {}, // Manejador del botón.

    refreshContador: function() {}, // Refrescar el valor mostrado .
};

// Ejecutar cuando se ha terminado de cargar la pagina.
$(function() {
  $(window).load(function() {
    App.init();
  });
});
```

```
init: function() {
    console.log("Inicializando App.");

    App.initWeb3();
},

initWeb3: function() {
    console.log("Inicializando web3.");

    // Si hay inyectada una instancia de web3:
    if (typeof web3 !== 'undefined') {
        App.web3Provider = web3.currentProvider;
    } else {
        // Uso Ganache porque no hay una instancia de
        // web3 inyectada.
        App.web3Provider = new Web3.providers.HttpProvider(
                                        'http://localhost:7545');
    }
    web3 = new Web3(App.web3Provider);

    App.initContractAbstracts();
},
```

```
initContractAbstracts: function() {
    console.log("Inicializando abstracción del contrato.");

    // Cargar el artefacto del contrato Contador (json)
    $.getJSON('Contador.json', function(json) {

        // Crear la abstraccion del contrato Contador
        App.Contador = TruffleContract(json);

        // Provisionar el contrato con el proveedor web3
        App.Contador.setProvider(App.web3Provider);

        App.initContractInstance();
    });
},
```

```
initContractInstance: function() {
    console.log("Obtener instancia desplegada del contador.");

    App.Contador.deployed()
    .then(function(contador) {
        App.contador = contador;

        console.log("Configurar Vigilancia de los eventos del contador.");

        contador.Tic((err, event) => {
            console.log("Se ha producido un evento Tic:");
            if (err){
                console.log(err);
            } else {
                var msg = event.args.msg;
                var out = event.args.out;
                console.log(" * Msg =", msg);
                console.log(" * Out =", out.valueOf());
                $('#valor').text(out.valueOf());
            }
        });
        App.bindEvents();
    })
    .catch(function(err) {
        console.log(err.message);
    });
},
```

```
bindEvents: function() {
    console.log("Configurando manejador de eventos del boton.");

    $(document).on('click', '#cincr', App.handleIncr);
    App.refreshContador();
},

handleIncr: function(event) {
    console.log("Se ha hecho Click en el botón.");

    event.preventDefault();
    web3.eth.getAccounts(function(error, accounts) {
        if (error) {
            console.log(error);
        }
        const account = accounts[0];
        console.log("Cuenta =", account);
        // Ejecutar incr como una transacción desde la cuenta account.
        App.contador && App.contador.incr({from: account})
        .then(function() {
            App.refreshContador();
        })
        .catch(function(err) {
            console.log(err.message);
        });
    });
},
```

```
refreshContador: function() {
    console.log("Refrescando el valor mostrado del contador.");

    App.contador && App.contador.valor.call()
    .then(function(valor) {
        console.log("Valor =", valor.valueOf());
        $('#valor').text(valor);
    })
    .catch(function(err) {
        console.log(err.message);
    });
}
```

Pasos para usar la dapp

- Lanzar la red Ethereum: Ejecutar **Ganache**.
- Compilar y migrar los contratos:

```
$ truffle migrate --compile-all --reset
```

- Lanzar servidor web:

```
$ cd node_server
$ npm start
```

- Lanzar varios navegadores Chrome con MetaMask:
  - Abrir el panel de MetaMask y seleccionar la red "Custom RPC" que apunta a la red privada de Ganache (http://localhost:7545)
  - Meter el password en MetaMask para desbloquear las cuentas de usuario.
    - Si es necesario crear las cuentas de usuario creadas por Ganache copiando la seed phrase.
  - Conectarse a http://localhost:3000
  - Nota: Si se lanza un navegador sin MetaMask, en app.js se creará una instancia propia de web3 y usarán directamente las cuentas de Ganache.
- Usar la aplicación web pulsando el botón incrementar.
  - Todos los navegadores se refrescarán con la valor actualizado del contador desplegado.

- NOTA:
  - Al ejecutar las transacciones puede aparecer un error en la consola javascript informando de que el ***nonce*** de la red es incorrecto:

```
the tx doesn't have the correct nonce.
account has nonce of: 4 tx has nonce of: 10
```

  - Este error se debe a que MetaMask guarda en una cache información sobre las redes a las que se ha conectado. Estas redes las identifica por su NetworkId. Al relanzar Ganache, se está creando desde cero otra vez la misma red, con el mismo NetworkId, y MetaMask se queja porque el valor nonce usado por la red es menor que el esperado.
  - Este problema se soluciona entrando en el panel Setting de MetaMask y pulsando el botón "Reset Account".