Blockchain: Desarrollo de Aplicaciones

Contador Blockchain con React y Redux (Sin Drizzle) Caso de Estudio #4

BCDA 2018

Versión: 2018-12-05

Índice

Paso a Paso

- Copiar el directorio **`paso3`** donde se desarrollo en el caso de **estudio #3** a un directorio nuevo llamado **`caso4`**.
- Añadir la parte de blockchain Ethereum del caso de estudio #1:
  - Copiar los directorios con los contratos (**`contracts`**) y las migraciones (**`migrations`**).
- El servidor node que sirve la dapp no lo copio de momento.
  - En desarrollo usaré "**`npm run dev`**" para ejecutar el servidor que me proporciona webpack.
  - Crear un enlace simbólico a build/contracts desde src/js para que se recargue el servidor cada vez que recompile o migre los smart contracts.

```
$ cd caso4/src/js
$ ln -s ../../build/contracts
```

- **`package.json`** no se modifica porque ya están metidas las dependencias de **truffle-contract** y **web3**.
- Copiar el fichero **`truffle.json`** con la configuración de Truffe.
- continúa...

- En el caso de estudio #1, el fichero **`app2/app.js`** inicializaba web3, cargaba la abstracción del contrato Contador, obtenía la instancia desplegada del contrato Contador, configuraba el botón HTML usado para incrementar el contador, refrescaba la etiqueta que muestra el valor del contador, incluía la lógica de la aplicación.
  - Inspirándonos en este fichero **`app2/app.js`**, crearemos un nuevo fichero **`src/js/blockchain.js`** que contendrá solo lo necesario para acceder al contrato contador, eliminando todo lo relativo al interface de usuario, React, Redux, ...
  - El fichero **`src/js/blockchain.js`**:
    - inicializará web3,
    - cargará la abstracción del contrato Contador,
    - obtendrá e inicializará la instancia desplegada,
    - proporcionará un método **`incr`** para incrementar el contador,
    - proporcionará un método **`get`** para obtener su valor.
    - atenderá los eventos **`Tic`**.

```
import Web3 from "web3";
import TruffleContract from "truffle-contract";

import cJson from "./contracts/Contador.json";

let web3 = window.web3;

// Instancia desplegada en la red blockchain:
let contador;
```

src/js/blockchain.js (I)

```
// Inicializa el módulo.
// Devuelve una promesa que se cumple cuando ha finalizado la inicializacion.
export const init = ticCB => {
    console.log("Inicializando web3.");
    let web3Provider;
    if (typeof web3 !== 'undefined') { // Si ya hay inyectada una instancia de web3.
        web3Provider = web3.currentProvider;
    } else { // Uso Ganache porque no hay una instancia de web3 inyectada.
        web3Provider = new Web3.providers.HttpProvider('http://localhost:7545');
        window.web3 = new Web3(web3Provider);
    }

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

    Contador.setProvider(web3Provider); // Provisionar el contrato

    console.log("Obtener instancia desplegada del contador.");
    return Contador.deployed()
    .then(instance => {
        contador = instance;
        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 {
                const {msg,out} = event.args;
                console.log(" * Msg =", msg);
                console.log(" * Out =", out.valueOf());
                ticCB(+out.valueOf()); // ¡¡¡ Ejecutar callback pasada como parametro !!!
            }
        });
    });
};
```

src/js/blockchain.js (II)

```
// Crea una transaccion para incrementar contador.
// Devuelve una promesa que se cumple cuando ha finalizado la transaccion.
export const incr = () => {
    console.log("Crear transaccion para incrementar contador.");

    return new Promise((resolve, reject) => {

        if (!contador) {
            throw new Error('No existe la instancia contador.');
        }

        web3.eth.getAccounts((error, accounts) => {
            if (error) {
                console.log(error);
                return reject(error);
            }

            const account = accounts[0];
            console.log("Cuenta =", account);

            // Ejecutar incr como una transacción desde la cuenta account.
            resolve(contador.incr({from: account}));
        });
    });
};
```

src/js/blockchain.js (III)

```
// Consultar el valor del contador.
// Devuelve una promesa que se cumple devolviendo el valor del contador.
export const get = async () => {
    console.log("Consultando el valor del Contador.");

    if (!contador) {
        throw new Error('No existe la instancia contador.');
    }

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

    return value.valueOf();
};
```

src/js/blockchain.js (IV)

- Cambios en **`src/js/redux/actions.js`**:
  - Importar el módulo **`blockchain.js`**.
  - Las acciones asíncronas **`getCounterAction`** y **`incrCounterAction`** ya no hacen una petición AJAX a un servidor web.
    - Ahora invocan los métodos **`get`** e **`incr`** proporcionados por **`blockchain.js`**.
- Podemos eliminar el **`import`** de **`axios`** de **`actions.js`**, y la dependencia de **`axios`** de **`package.json`**.

```
import * as CTES from './constants';

import * as blockchain from "../blockchain";

// Accion asincrona a la red blockchain para obtener el valor del contador.

export const getCounterRequestAction = () => {
    return {
        type: CTES.COUNTER_GET_REQUEST,
        payload: { requesting: true,
                   error: '',
        }
    }
};

export const getCounterCompletedAction = counter => {
    return {
        type: CTES.COUNTER_GET_COMPLETED,
        payload: { counter,
                   requesting: false,
                   error: '',
        }
    }
};

export const getCounterErrorAction = error => {
    return {
        type: CTES.COUNTER_GET_ERROR,
        payload: { requesting: false,
                   error,
        }
    }
};
```

src/js/redux/actions.js (I)

```
export const getCounterAction = () => (dispatch, getState) => {

    dispatch(getCounterRequestAction());

    return blockchain.get()
    .then((counter) => {
        dispatch(getCounterCompletedAction(+counter));
    })
    .catch(error => {
        const emsg = error.message || error;
        dispatch(getCounterErrorAction(emsg))

    });
};
```

src/js/redux/actions.js (II)

```
// Accion asincrona a la red blockchain para incrementar el valor del contador.

export const incrCounterRequestAction = () => {
    return {
        type: CTES.COUNTER_INCR_REQUEST,
        payload: { requesting: true,
                   error: '',
        }
    }
};

export const incrCounterCompletedAction = counter => {
    return {
        type: CTES.COUNTER_INCR_COMPLETED,
        payload: { counter,
                   requesting: false,
                   error: '',
        }
    }
};

export const incrCounterErrorAction = error => {
    return {
        type: CTES.COUNTER_INCR_ERROR,
        payload: { requesting: false,
                   error,
        }
    }
};
```

src/js/redux/actions.js (III)

```
export const incrCounterAction = () => (dispatch, getState) => {

    dispatch(incrCounterRequestAction());

    // Nota: la accion incrCounterCompletedAction se dispara cuando se
    // produce un evento Tic.

    return blockchain.incr()
    .catch(error => {
        const emsg = error.message || error;
        dispatch(incrCounterErrorAction(emsg))

    });
};
```

src/js/redux/actions.js (IV)

- Cambios en **`src/js/components/AppCounter.jsx`**:
  - Importar el módulo **`blockchain.js`**.
  - Modificar el método **`componentDidMount`** para inicializar el módulo blockchain llamando al método **`blockchain.init(callback)`**.
    - La callback pasada como parámetro es una función que lanza la acción **`incrCounterCompletedAction`**.
      - La acción ha sido ligada con dispatch, usando **`bindActionCreators`**, para que blockchain no dependa de nada de Redux.
    - Esta callback la invoca blockchain cuando llegan eventos **`Tic`**.
  - La llamada a **`blockchain.init`** devuelve una promesa que cuando se cumple, se lanza la acción **`getCounterAction`** para obtener el estado actual del contador en la cadena de bloques..
- Nota: **`componentDidMount`** es uno de los métodos del ciclo de vida de los componentes React.

```
import React from 'react';

import { connect } from 'react-redux';

import { bindActionCreators } from 'redux';

import * as blockchain from "../blockchain";

import AppHeader from './AppHeader';
import AppData from './AppData';
import AppControl from './AppControl';

import {getCounterAction,
        incrCounterAction,
        incrCounterCompletedAction } from "../actions";
```

src/js/components/AppCounter.jsx (I)

```
class AppCounter extends React.Component {

    componentDidMount() {

        // Quiero que el modulo Blockchain no sepa nada de Redux,
        // ni del Store. Como dentro de Blockchain se necesita
        // lanzar la accion incrCounterCompletedAction cuando llega
        // un evento Tic de la red Blockchain, lo que hago es pasar
        // esta accion al modulo Blockchain, enlazandola previamente
        // con el store. Asi, dentro de Blockchain no hay que importar
        // acciones, ni el store, ni nada relacionado con Redux.
        //
        // De no hacerlo asi, tendria que pasar como parametro la
        // funcion dispatch, y el modulo Blockchain tendria que
        // importar las acciones a usar.
        const boundIncrCounterCompletedAction =
bindActionCreators(incrCounterCompletedAction, this.props.dispatch);

        blockchain.init(boundIncrCounterCompletedAction)
        .then(() => {
            this.props.dispatch(getCounterAction());
        })
        .catch((error) => {
            console.log('ERROR en componentDidMount:', error);
        });
    }
```

src/js/components/AppCounter.jsx (II)

```
    render() {
        return (
            <div className="appCounter">
                <AppHeader/>
                <AppData counter={this.props.counter}
                         updating={this.props.updating ? 'Updating' : ''}
                         error={this.props.error}/>
                <AppControl
                    onClick={() => { this.props.dispatch(incrCounterAction()); }}
                    disabled={this.props.updating}
                />
            </div>
        );
    }
}
```

src/js/components/AppCounter.jsx (III)

```
const mapsStateToProps = (state) => ({
    counter: state.counterReducer.counter,
    updating: state.counterReducer.requesting,
    error: state.counterReducer.error
});


export default connect(mapsStateToProps)(AppCounter);
```

src/js/components/AppCounter.jsx (IV)

Ejecutar

- Ejecutamos:
  - $ npm install
  - Lanzamos Ganache.
  - $ truffle migrate --reset --compile-all
  - $ npm run dev
  - Lanzamos Chrome
    - y desbloqueamos MetaMask (password).
    - Puede ser necesario resetear la cuenta para limpiar la cache que recuerda el valor del ultimo nonce usado.

Pendiente

- El mecanismo de requesting /updating ahora falla porque los eventos Tic pueden llegar en cualquier momento reflejando el estado del contador.
- Quitar la parte de deshabilitar el botón de incrementar.
- Mejorar el tratamiento de los casos de error.
- Eliminar las dependencias de jquery. No se usa.

create-react-app

create-react-app

- Es una aplicación que crea un proyecto (boilerplate) para desarrollar aplicaciones React,
  - Instala WebPack

Repetir caso4 -> caso4.1

- Pasos para hacer el caso de estudio 4 usando create-react-app:

```
$ sudo npm install -g create-react-app
$ create-react-app caso4.1
$ cd caso4.1
$ cp -r ../caso4/contracts .
$ cp -r ../caso4/migrations .
$ cp ../caso4/truffle.js .
$ truffle compile
$ truffle migrate
$ mv src src.pslm
$ mkdir src ; cd src
$ ln -s ../build/contracts
$ cp -r ../../caso4/src/css .
$ cp -r ../../caso4/src/js/* .
$ mv app.jsx index.js
Editar index.js:
  cambiar import ../css/style.css a ./css/style.css
  cambiar document.getElementById('app') por document.getElementById('root')
$ cd ..
$ npm install --save-dev web3 truffle-contract
$ npm install --save redux react-redux redux-thunk
```

- Lanzar el servidor de desarrollo con:

```
$ npm start
```