Blockchain: Desarrollo de Aplicaciones

Introducción a React y Redux
Caso de Estudio #3
¡¡¡ CONTADOR SIN BLOCKCHAIN !!!

BCDA 2018

Versión: 2018-12-05

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.

Paso 0

Crear Servidor Web para el valor del Contador

¿Qué haremos en el paso 0?

- En este paso haremos un servidor web con node y express para gestionar el valor de un contador.

- En los siguientes pasos desarrollaremos una aplicación web cliente del servidor contador anterior.
  - Usando React y Redux

Servidor Contador Backend

- El servidor lo desarrollaremos en el directorio caso3/paso0.
- Hacer un servidor node que implemente el backend donde se guarda el valor del contador.

```
$ express contador_server
```

- Modificar el script `start` de `package.json` para que el servidor atienda en el puerto **5000**.

```
"scripts": {
  "start": "PORT=5000 node ./bin/www"
},
```

- Añadir en `routes/index.js` las rutas necesarias para acceder al contador.

```
let counter = 0;

router.get('/counter', (req, res, next) => {
  res.json({counter});
});

router.post('/counter/incr', (req, res, next) => {
    counter++;
    res.json({counter});
});
```

- Añadir en app.js, antes de las rutas:

```
// Control de Acceso HTTP (CORS) - Author: Sonsoles
app.use((req, res, next) => {
    res.header("Access-Control-Allow-Origin", "*");
    res.header("Access-Control-Allow-Headers",
               "Origin, X-Requested-With, Content-Type, Accept");
    next();
});
```

  - CORS: Cross-origin resource sharing / Intercambio de Recursos de Origen Cruzado.
    - Este estándar añade encabezados HTTP para que los servidores describan cuales son los orígenes que tienen permiso acceso a la información usando un navegador web.

Probar el servidor

- Lanzar el servidor:

```
$ cd caso3/paso/contador_server
```

```
$ npm start
```

- Hacer peticiones:

```
$ curl -X POST http://localhost:5000/counter/incr
{"counter":4}
```

```
$ curl -X GET http://localhost:5000/counter
{"counter":4}
```

```
$ curl -X POST http://localhost:5000/counter/incr
{"counter":5}
```

```
$ curl -X GET http://localhost:5000/counter
{"counter":5}
```

Paso 1

React

¿Qué haremos en el paso 1?

- Modificar la aplicación web Contador desarrollada en el tema de WebPack para crear la interface de usuario con componentes React.
- También:
  - Eliminaremos la parte del contrato Contador en Blockchain.
  - Usaremos el servidor Web Contador del paso 0.

Referencias

- Existen muchas sitios en internet que explican cómo puede configurarse un proyecto React con WebPack 4.
- Por ejemplo:
  - The minimal React + Webpack 4 + Babel Setup

    https://www.robinwieruch.de/minimal-react-webpack-babel-setup/
  - How to Create a React app from scratch using Webpack 4

    https://medium.freecodecamp.org/part-1-react-app-from-scratch-using-webpack-4-562b1d231e75
  - Tutorial: How to set up React, webpack 4, and Babel (2018)

    https://www.valentinog.com/blog/react-webpack-babel/

Crear un proyecto nuevo

- Este desarrollo lo haremos en el directorio caso3/paso1.
- Copiar la configuración de Webpack del caso de estudio #2:
  - Copiar el fichero `package.json`.
  - Copiar los ficheros de configuración de Webpack:
    - `webpack.const.js`,
    - `webpack.common.js`,
    - `webpack.dev.js` y
    - `webpack.prod.js`.

```
const path = require('path');

const SRC_DIR_PATH = path.resolve(__dirname, './src');
const SRC_APP_PATH = path.resolve(SRC_DIR_PATH, './js/app.js');
const SRC_INDEX_PATH = path.resolve(SRC_DIR_PATH, './index.html');

const DIST_DIR_NAME = 'dist';
const DIST_DIR_PATH = path.resolve(__dirname, DIST_DIR_NAME);
const DIST_BUNDLE_FILENAME = 'main.js';
const DIST_INDEX_FILENAME = 'index.html';

module.exports = {
  SRC_DIR_PATH,
  SRC_APP_PATH,
  SRC_INDEX_PATH,
  DIST_DIR_NAME,
  DIST_DIR_PATH,
  DIST_BUNDLE_FILENAME,
  DIST_INDEX_FILENAME
};
```

webpack.const.js

```
const CopyWebpackPlugin = require('copy-webpack-plugin');
const CleanWebpackPlugin = require('clean-webpack-plugin');

const {
  SRC_APP_PATH, SRC_INDEX_PATH,
  DIST_DIR_NAME, DIST_DIR_PATH,
  DIST_BUNDLE_FILENAME, DIST_INDEX_FILENAME } = require('./webpack.const');

module.exports = {
  entry: SRC_APP_PATH,
  output: {
    filename: DIST_BUNDLE_FILENAME,
    path: DIST_DIR_PATH
  },
  plugins: [
    new CleanWebpackPlugin([DIST_DIR_NAME]),
    new CopyWebpackPlugin([
      { from: SRC_INDEX_PATH, to: DIST_INDEX_FILENAME }
    ])
  ],
  module: {
    rules: [
      {
       test: /\.css$/,
       use: [ 'style-loader', 'css-loader' ]
      }
    ]
  }
};
```

webpack.common.js

```
const webpack = require('webpack');
const merge = require('webpack-merge');
const common = require('./webpack.common.js');

const {DIST_DIR_PATH} = require('./webpack.const');

module.exports = merge(common, {
    mode: 'development',
    devtool: 'inline-source-map',
    devServer: {
        contentBase: DIST_DIR_PATH,
        port: 3000
    },
    plugins: [
        new webpack.DefinePlugin({
            'process.env.NODE_ENV': JSON.stringify('development')
        })
    ]
});
```

webpack.dev.js

```
const webpack = require('webpack');
const merge = require('webpack-merge');
const UglifyJSPlugin = require('uglifyjs-webpack-plugin');

const common = require('./webpack.common.js');

module.exports = merge(common, {
    mode: "production",
    devtool: 'source-map',
    plugins: [
        new UglifyJSPlugin({
            sourceMap: true
        }),
        new webpack.DefinePlugin({
            'process.env.NODE_ENV': JSON.stringify('production')
        })
    ]
});
```

webpack.prod.js

Instalar React

- Al usar React vamos a escribir código ES6 y JSX que hay que transpilar a ES5 para que lo entiendan los navegadores.
  - Usamos Babel para transpilar.
- Instalación de Babel:
  - Para transpilar de ES6 a ES5:

```
$ npm install --save-dev babel-core
            babel-loader@7 babel-preset-env
```

  - Para soportar object spread, ...

```
$ npm install --save-dev babel-preset-stage-3
```

  - Para usar React y transpilar jsx a js:

```
$ npm install --save-dev babel-preset-react
$ npm install --save react react-dom
```

- Añadir en package.json la sección "babel" con la configuración de Babel:

```
"babel": {
  "presets": [
     ["env",
     {
       "targets": {
         "browsers": ["last 2 Chrome versions"]
       }
     }
   ],
    "react",
    "stage-3"
  ]
},
```

  - Nota: esta configuración también se podría haber creado en un fichero llamado `.babelrc`.
  - El target requerido en preset+env es para asegurarnos de que la versión del navegador soporta async/await.
    - Alternativas: Usar babel-polyfill, babel-plugin-transform-runtime, ...

- En `webpack.common.js`:
  - Añadir una nueva entrada en **module.rules** para indicar cómo se cargan los **jsx.**
  - Añadir la entrada `resolve` para indicar cuales son las extensiones de los ficheros a cargar.

```
const CopyWebpackPlugin = require('copy-webpack-plugin');
const CleanWebpackPlugin = require('clean-webpack-plugin');

const {
  SRC_APP_PATH, SRC_INDEX_PATH,
  DIST_DIR_NAME, DIST_DIR_PATH,
  DIST_BUNDLE_FILENAME, DIST_INDEX_FILENAME } = require('./webpack.const');


module.exports = {
  entry: SRC_APP_PATH,
  output: {
    filename: DIST_BUNDLE_FILENAME,
    path: DIST_DIR_PATH
  },
  plugins: [
    new CleanWebpackPlugin([DIST_DIR_NAME]),
    new CopyWebpackPlugin([
      { from: SRC_INDEX_PATH, to: DIST_INDEX_FILENAME }
    ])
  ],
   module: {
     rules: [
       {
         test: /\.(js|jsx)$/,
         exclude: /node_modules/,
         use: ['babel-loader']
       },
       {
        test: /\.css$/,
        use: [ 'style-loader', 'css-loader' ]
       }
     ]
   },
   resolve: {
     extensions: ['*', '.js', '.jsx']
   }
};
```

webpack.common.js

Adaptar Página HTML

- Copiar los siguiente ficheros del caso de estudio #2 de Webpack:
  - `src/index.html`
  - `src/css/style.css`

src/css/style.css

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

src/index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <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="main.js"></script>
  </body>
</html>
```

Retocar src/index.html

- Eliminar del `body` de `index.html` todas la etiquetas que crean la pagina actual.
  - La etiqueta **`script`** que carga **`main.js`** debe mantenerse.
- Añadir un elemento `div` donde se insertará el componente raíz hecho con React:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Contador</title>
  </head>
  <body>
    <div id="app"></div>

    <script src="main.js"></script>
  </body>
</html>
```

src/index.html

Renderizar el componente raíz de React

- Crear el fichero `src/js/app.jsx`.
- Este fichero renderiza el componente raíz de React dentro de la etiqueta `div` con el `id="app"` de la página HTML.
  - De momento solo uso una cabecera `h1` para probar.
  - **ATENCIÓN:** Hay que actualizar la variable **`SRC_APP_PATH`** de `webpack.const.js` para que la extensión sea **`jsx`**.

```
import React from 'react';
import ReactDOM from 'react-dom';

import "../css/style.css";

ReactDOM.render(
    <h1>Contador 1</h1>,
    document.getElementById('app')
);
```

src/js/app.jsx

Probarlo

- Instalar dependencias:

```
$ npm install
```

- Ejecutar:

```
$ npm run dev
```

  - Y lanzar un navegador para ver la página `http://localhost:3000`.

Componentes React

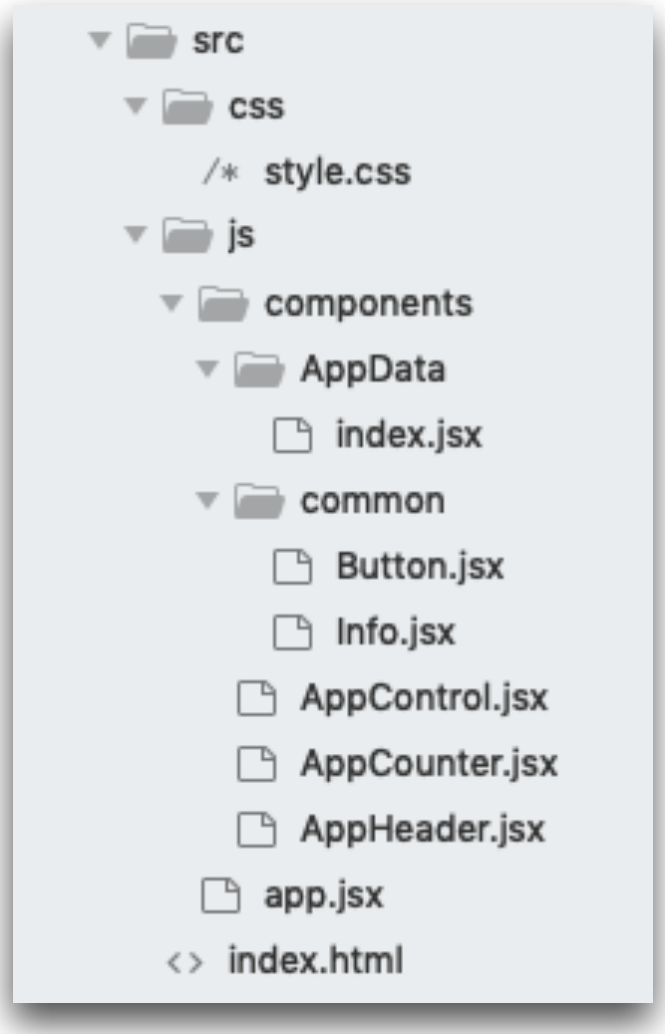
- Cambiar `src/js/app.jsx` para que renderice un component React llamado `App` en vez de la cabecera h1.

- Nota:
  - Si el servidor lanzado anteriormente con "npm run dev" sigue ejecutándose, entonces debería refrescarse automáticamente la página mostrada en el navegador al modificar el fichero app.jsx.

src/js/app.jsx

```
import React from 'react';
import ReactDOM from 'react-dom';

import "../css/style.css";

class App extends React.Component {
    constructor(props) {
        super(props);
    }

    render() {
        return (
            <div className="counter">
                <h1>Contador 2</h1>
            </div>
        );
    }
}

ReactDOM.render(<App />, document.getElementById('app'));
```

- Modificar el componente App para que esté formado por varios subcomponentes.
  - Creo los subcomponentes en el subdirectorio `src/js/components`.



src/js/app.jsx

```
import React from 'react';
import ReactDOM from 'react-dom';

import "../css/style.css";

import AppCounter from './components/AppCounter';

const App = () => (
    <AppCounter />
);

ReactDOM.render(<App />, document.getElementById('app'));
```

src/js/components/AppCounter.jsx

```
import React from 'react';

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

class AppCounter extends React.Component {
    render() {
        return (
            <div className="appCounter">
                <AppHeader />
                <AppData counter={666}
                         updating={'Actualizando'}
                         error={'Algo está mal'} />
                <AppControl onClick={() => false}
                            disabled={false} />
            </div>
        );
    }
}

export default AppCounter;
```

src/js/components/AppHeader.jsx

```
import React from 'react';

const AppHeader = () => (
    <div className="appCounter-header">
        <h1 className="appCounter-header-title">Contador</h1>
        <p className="appCounter-header-version">Versión: 1</p>
    </div>
);

export default AppHeader;
```

```
import React from 'react';

import PropTypes from 'prop-types';

import Updating from '../common/Info';
import Error from '../common/Info';

const AppData = props => (
    <div className="appCounter-data">
        <p>
            Valor =
            <span className="appCounter-data-value">
                {props.counter}
            </span>
        </p>
        <Updating className="appCounter-data-updating"
                  msg={props.updating}
                  visible={!!props.updating} />
        <Error className="appCounter-data-error"
               msg={props.error}
               visible={!!props.error} />
    </div>
);
```

src/js/components/AppData/index.jsx (I)

```
// Valores por defecto en caso de que no se pase alguna propiedad.
AppData.defaultProps = {
    counter: '-1',
    updating: '',
    error: ''
};

// Validar tipo de las propiedades.
// Indicar que propiedades son obligatorias.
// ...
AppData.propTypes = {
    counter: PropTypes.number,
    updating: PropTypes.string,
    error: PropTypes.string
};

export default AppData;
```

src/js/components/appData/index.jsx (II)

src/js/components/appControl.jsx

```
import React from 'react';
import PropTypes from 'prop-types';

import IncrControl from './common/Button';

const AppControl = ({ disabled, onClick }) => (
    <IncrControl className="appCounter-control"
                 text="Incrementar"
                 onClick={onClick}
                 disabled={disabled} />
);

// Validar tipo de las propiedades.
// Indicar que propiedades son obligatorias.
// ...
AppControl.propTypes = {
    onClick: PropTypes.func.isRequired,
    disabled: PropTypes.bool,
}

export default AppControl;
```

```jsx
import React from 'react';

import PropTypes from 'prop-types';

const Button = ({className, text, disabled, onClick }) => (
    <button disabled={disabled}
            className={`button-common ${className}`}
            onClick={onClick} >
        {text}
    </button>
);


// Valores por defecto en caso de que no se pase alguna propiedad.
Button.defaultProps = {
    className: ''
};

// Validar tipo de las propiedades.
// Indicar que propiedades son obligatorias.
// ...
Button.propTypes = {
    className: PropTypes.string,
    onClick: PropTypes.func.isRequired,
    value: PropTypes.string
}

export default Button;
```

src/js/components/common/Button.jsx

```
import React from 'react';

import PropTypes from 'prop-types';

const Info = ({className, msg, visible}) => visible && (
    <span className={ `${className} info-common` }>
        { msg }
    </span>
);

// Valores por defecto en caso de que no se pase alguna propiedad.
Info.defaultProps = {
    className: '',
    msg: ''
};

// Validar tipo de las propiedades.
// Indicar que propiedades son obligatorias.
// ...
Info.propTypes = {
    className: PropTypes.string,
    visible: PropTypes.bool,
    msg: PropTypes.string
};

export default Info;
```

src/js/components/common/Info.jsx

src/css/style.css

```
appCounter {
}

.appCounter-header {
     font-size: x-large;
     color: mediumblue;
}

.appCounter-header-title {
    font-size: x-large;
    color: darkblue;
}

.appCounter-header-version {
    font-size: x-small;
    color: lightblue;
    position: absolute;
    top: 10px;
    right: 10px;
}

.appCounter-data {
     color: darkgreen;
     border: 1px solid green;
     margin: 10px;
    padding: 10px;
}

.appCounter-data-value {
    font-size: x-large;
    color: green;
    padding: 5px;
}

.appCounter-data-updating {
    display: block;
    font-size: smaller;
    color: lightgreen;
}

.appCounter-data-error {
    display: block;
    font-size: smaller;
    color: red;
}

.appCounter-control {
     font-size: small;
}

.button-common {
     color: blue;
}

.button-common:disabled {
    color: lightgrey;
}

.info-common {
}
```

Probarlo

- Supongo que aun se está ejecutando el servidor que lanzamos antes.
  - Si no es así, volver a arrancarlo:

    ```
    $ npm run dev
    ```

- Lanzar si es necesario un navegador visualizando la página:

  ```
  http://localhost:3000
  ```

- Nota:
  - Si el servidor lanzado anteriormente con "npm run dev" sigue ejecutándose, entonces debería refrescarse automáticamente la página mostrada en el navegador al modificar el fichero app.jsx.

Paso 2

Redux

Redux

- Para gestionar el estado de la aplicación Contador de nuestro caso de estudio usaremos **Redux**.
- En este paso, desarrollaremos solo la parte de Redux (constantes, acciones, reducers, middlewares y el store) independientemente de la interface de usuario (componentes de React).
  - En el siguiente paso uniremos los desarrollos de React y Redux.
- Copiar el directorio paso1 en un nuevo directorio paso2.
  - En `paso2/src/js/redux` crearemos las constantes, acciones, reducers, middlewares y el store de Redux.

- El estado de la aplicación Contador mantiene tres valores:
  - **counter**: número con el valor del contador.
  - **requesting**: booleano que indica si estamos en mitad de un acceso (asíncrono) al backend.
  - **error**: guardar un mensaje explicativo del último error que se haya producido.
- Temario nuevo:
  - Acciones asíncronas.
    - redux-thunk
  - Middlewares.

- Instalaciones necesarias:
  - Redux:
    ```
    $ npm install --save redux
    ```
  - Acciones asíncronas:
    ```
    $ npm install --save redux-thunk
    ```
  - Utilidades usadas por la aplicación:
    ```
    $ npm install --save lodash
    ```
  - Peticiones Ajax:
    ```
    $ npm install --save axios
    ```

src/js/redux/constants.js

```
// Thunk: DETALLES PARA CONSTRUIR LAS ACCIONES ASINCRONAS:
//   *_REQUEST -> Acabo de hacer una peticion HTTP al servidor backend.
//   *_COMPLETED -> El servidor backend me devuelve una respuesta.
//   *_ERROR -> El servidor backend me devuelve un error.


// Acciones para obtener el valor del contador del servidor.

export const COUNTER_GET_REQUEST   = 'counter::get::request';
export const COUNTER_GET_COMPLETED = 'counter::get::completed';
export const COUNTER_GET_ERROR     = 'counter::get::error';

// Acciones para incrementar el valor del contador del servidor.

export const COUNTER_INCR_REQUEST   = 'counter::incr::request';
export const COUNTER_INCR_COMPLETED = 'counter::incr::completed';
export const COUNTER_INCR_ERROR     = 'counter::incr::error';

// Backend: Server API
export const COUNTER_SERVER_URL  = 'http://localhost:5000'
```

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

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

// Accion Asincrona al servidor para obtener el valor del contador.
// Se espera a que el servidor responda o de un error.

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 }
    }
};
export const getCounterAction = () => (dispatch, getState) => {
    dispatch(getCounterRequestAction());

    return axios.get(`${CTES.COUNTER_SERVER_URL}/counter`)
    .then(({data}) => dispatch(getCounterCompletedAction(data.counter)))
    .catch(error => { const emsg = error.message || error;
                      dispatch(getCounterErrorAction(emsg))
    });
};
```

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

```
// Accion Asincrona al servidor para incrementar el valor del contador.
// Se espera a que el servidor responda o de un error.

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 }
    }
};
export const incrCounterAction = () => (dispatch, getState) => {
    dispatch(incrCounterRequestAction());

    return axios.post(`${CTES.COUNTER_SERVER_URL}/counter/incr`)
    .then(({data}) => dispatch(incrCounterCompletedAction(data.counter)))
    .catch(error => { const emsg = error.message || error;
                      dispatch(incrCounterErrorAction(emsg))
    });
};
```

src/js/redux/reducers.js

```
import { combineReducers } from 'redux';
import cloneDeep from 'lodash/cloneDeep';
import * as CTES from './constants';

const INITIAL_STATE = {
    counter: 0,
    requesting: false,
    error: '',
};
const counterReducer = (state = cloneDeep(INITIAL_STATE), action) => {
    const {type, payload} = action;
    switch (type) {
        case CTES.COUNTER_GET_REQUEST:
        case CTES.COUNTER_GET_COMPLETED:
        case CTES.COUNTER_GET_ERROR:
        case CTES.COUNTER_INCR_REQUEST:
        case CTES.COUNTER_INCR_COMPLETED:
        case CTES.COUNTER_INCR_ERROR:
            return { ...state,
                     ...payload,
                   };
        default:
            return state;
    }
};
export default combineReducers({
    counterReducer,
});
```

src/js/redux/middlewares.js

```
export const holaMiddleware = store => next => action => {

    console.log('hola', action.type || 'ASYNC');

    return next(action);
};

export const adiosMiddleware = store => next => action => {

    console.log('adios', action.type || 'ASYNC');

    return next(action);
};
```

src/js/redux/store.js

```
import { createStore, applyMiddleware, compose } from 'redux';

import thunk from 'redux-thunk';

import reducers from './reducers';

import { holaMiddleware, adiosMiddleware } from './middlewares';


const createStoreWithMiddleware = compose(
    applyMiddleware(holaMiddleware, thunk, adiosMiddleware),
)(createStore);


export default createStoreWithMiddleware(reducers);
```

Programa de Prueba

- Este programa de prueba solo se ha creado para ver que la parte que hemos desarrollado de Redux funciona.
  - No es parte de la aplicación que estamos desarrollando.
- Creamos un fichero con un programa de prueba:
  `src/js/redux/pruebaredux.js`
- Instalamos babel-cli

```
$ npm install --save-dev babel-cli
```

- Ejecutamos el programa de prueba con:

```
$ npx babel-node src/js/redux/pruebaredux.js
```

src/js/redux/pruebaredux.js

```
import store from './store';

import {
    getCounterAction,
    incrCounterAction
} from './actions';

const dumpStore = () => {
    console.log('*** STATE:', store.getState(), '***');
};

dumpStore();

const subscribeId = store.subscribe(dumpStore);

(async () => {
    await store.dispatch(getCounterAction());
    await store.dispatch(incrCounterAction());
    await store.dispatch(getCounterAction());
})();
```

```
$ npx babel-node src/js/redux/pruebaredux.js
*** STATE: { counterReducer: { counter: 0, requesting: false, error: '' } } ***
hola ASYNC
hola COUNTER::GET::REQUEST
adios COUNTER::GET::REQUEST
*** STATE: { counterReducer: { counter: 0, requesting: true, error: '' } } ***
hola COUNTER::GET::COMPLETED
adios COUNTER::GET::COMPLETED
*** STATE: { counterReducer: { counter: 1, requesting: false, error: '' } } ***
hola ASYNC
hola COUNTER::INCR::REQUEST
adios COUNTER::INCR::REQUEST
*** STATE: { counterReducer: { counter: 1, requesting: true, error: '' } } ***
hola COUNTER::INCR::COMPLETED
adios COUNTER::INCR::COMPLETED
*** STATE: { counterReducer: { counter: 2, requesting: false, error: '' } } ***
hola ASYNC
hola COUNTER::GET::REQUEST
adios COUNTER::GET::REQUEST
*** STATE: { counterReducer: { counter: 2, requesting: true, error: '' } } ***
hola COUNTER::GET::COMPLETED
adios COUNTER::GET::COMPLETED
*** STATE: { counterReducer: { counter: 2, requesting: false, error: '' } } ***
```

Paso 3

React + Redux

Unir React y Redux

- Ahora integraremos los desarrollos de los pasos 1 y 2:
  - Componentes React y
  - Estado Redux.

Extensión del Navegador: Redux DevTools

- La extensión `Redux DevTools` de los navegadores se usa en **desarrollo** para permitir depurar los cambios que se producen en el estado de la aplicación (el store).
  - Para Chrome: Instalarla desde la tienda de Chrome.

    `https://chrome.google.com/webstore`

  - Tras relanzar Chrome debería verse este botón en la barra de herramientas:

- Esta extensión define una nueva función **`compose`** que es la que hay que usar en vez del **`compose`** de **`Redux`**.
- Esta nueva función **`compose`** se deja accesible en:

  `window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__`

- El fichero store.js con la nueva función compose quedaría así:

src/js/redux/store.js

```
import { createStore, applyMiddleware, compose } from 'redux';

import thunk from 'redux-thunk';

import reducers from './reducers';

import { holaMiddleware, adiosMiddleware } from './middlewares';


// Extension ReduxDevTools de Chrome.
// Elegir la funcion compose a usar en modo desarrollo o produccion.
const composeEnhancers = process.env.NODE_ENV === 'production'
              ? compose
              : window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__ || compose;

const createStoreWithMiddleware = composeEnhancers(
    applyMiddleware(holaMiddleware, thunk, adiosMiddleware),
)(createStore);


export default createStoreWithMiddleware(reducers);
```

Usar el store

- Debemos hacer que el objeto store esté disponible para todos los componentes de la aplicación.
  - Los componentes necesitarán consultar el estado, disparar acciones, ...
- Esto se hace cómodamente usando el componente **`Provider`** proporcionado por el módulo **`react-redux`**.
  - Provider se coloca como componente raíz.
  - Toma store como propiedad y se encarga de que todos los subcomponentes puedan acceder al store sin que tengamos que pasarlo nosotros explícitamente.
- Para instalar react-redux:

```
$ npm install --save react-redux
```

src/js/app.jsx

```
import React from 'react';
import ReactDOM from 'react-dom';

import { Provider } from 'react-redux';

import "../css/style.css";

import store from './redux/store';

import AppCounter from './components/AppCounter';

const App = () => (
    <Provider store={store}>
        <AppCounter/>
    </Provider>
);

ReactDOM.render(<App />, document.getElementById('app'));
```

Conectar Componentes al Store

- El objetivo es conectar los componentes con el Store para que se actualicen automáticamente si cambia el estado de Store.
  - Un componente conectado se convierte en lo que se llama un Smart Component.
  - Se hace usando el método **`connect`** proporcionado por **`react-redux`**, que transforma un componente en un smart componente.
  - Esto se *suele* hacer solo con los componentes raíz por motivos de prestaciones,
    - dejando los subcomponentes como componentes sin estado.
- Es necesario tener un mecanismo para que dado un estado, se obtengan las propiedades que necesita el smart componente para pintarse.
  - Se hace escribiendo un método, típicamente llamado **mapStateToProps**, que toma el estado como parámetro, y devuelve un objeto con las propiedades que necesita el smart componente.

- Modificaciones a realizar en AppCounter.jsx:
  - Importamos la función connect de react-redux.

```
import { connect } from 'react-redux';
```

  - Definimos una función llamada **`mapStateToProps`** que calcula las propiedades que necesita `AppCounter` para pintarse.

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

    - Devuelve un objeto con la props que necesita el smart component.
    - El valor de cada propiedad se obtiene del estado (**state**); seguido del reducer **counterReducer**; y seguido del valor del estado que nos interese, **counter**, **requesting** o **error** .
  - El componente se conecta, creando el smart component, que es lo que se exporta:

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

  - Finalmente, en render se accede al valor de las propiedades (this.props.XXX) que se habrán creado a partir del estado:

```
render() {
    . . .
    <AppData counter={ this.props.counter }
             updating={this.props.updating ? 'Updating' : ''}
             error={this.props.error} />
    <AppControl onClick={ . . . }
                disabled={this.props.updating} \>
    . . .
}
```

- Para invocar las acciones:
  - Importar las acciones definidas es **`actions.js`**.

```
import { incrCounterAction } from "../redux/actions";
```

  - Las acciones se invocan usando el método **`dispatch`** de **`props`**.
    - El método **`dispatch`** se añadió a **`props`** cuando se conecto el componente.
      - El método **`connect`** tiene otros argumentos para implementar cosas más avanzadas.

```
<AppControl
   onClick={() => {
      this.props.dispatch(incrCounterAction());
   }} . . .
```

src/js/components/AppCounter.jsx

```
import React from 'react';

import { connect } from 'react-redux';

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

import {incrCounterAction} from "../redux/actions";

class AppCounter extends React.Component {
  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>
    );
  }
}

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

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

Ciclo de vida: Inicialización

- Cuando lanzamos la aplicación queremos consultar el valor actual del contador y presentar su valor.
- Modificamos **AppCounter.jsx** para importar la acción **getCounterAction**, y redefinimos el método **componentDidMount** para obtener el valor inicial del contador

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

class AppCounter extends React.Component {
    . . .
    componentDidMount() {
        this.props.dispatch(getCounterAction())
    }
. . .
```

src/js/components/AppCounter.jsx

```
import React from 'react';

import { connect } from 'react-redux';

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

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

class AppCounter extends React.Component {
    componentDidMount() {
        this.props.dispatch(getCounterAction())
    }
    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>
        );
    }
}
const mapsStateToProps = (state) => ({
    counter: state.counterReducer.counter,
    updating: state.counterReducer.requesting,
    error: state.counterReducer.error
});
export default connect(mapsStateToProps)(AppCounter);
```

Probarlo

- Lanzar el (backend) servidor node que almacena el valor del contador.

```
$ cd paso0/contador_server
$ npm start
```

- Lanzar el servidor que sirve la aplicación web:

```
$ npm run dev
```

- Usar un navegador web y conectarse a la página:

```
http://localhost:3000
```