Taller de React, ES6 y webpack

Este documento tiene como objetivo servir de guía al taller que se realizará en el Codemotion 20015 el próximo 27 de Noviembre de 2015 y que tendrá una duración de 90 minutos.

Puesta en marcha del entorno

Como requisito antes de comenzar, es necesario tener instalado NodeJS. Para ello, recomendamos el uso de NVM (Node Version Manager), con el cual podemos instalar y utilizar varias versiones de NodeJS en nuestro entorno.

NOTA: Tenéis más información de cómo funciona y algunos ejemplos de uso en este artículo.

En definitiva, si ya tenéis NVM instalado, sólo tenéis que ejecutar:

```
$ nvm install v5.0.0
```

Con lo que al ejecutar los siguientes comandos, deberiais ver sin problema las versiones correctas de NodeJS y NPM:

```
$ node -v
v5.0.0

$ npm -v
3.3.6
```

Creación del proyecto "weather"

Nuestro objetivo en este taller va a ser migrar a React una apliación desarrollada en jQuery que permite consultar el tiempo indicando una localidad o obteniendola de nuestra geolocalización.

El código original que vamos a migrar lo podéis descargar aquí. Como la idea es partir de la aplicación jQuery e ir haciendo cambios poco a poco, descargad ya el ejemplo para tenerlo preparado.

Listos para comenzar!! Vamos pues a crear un directorio para el proyecto y le daremos el nombre de weather-react:

```
$ mkdir weather-react
$ cd weather-react
```

Dentro del directorio del proyecto, inicializamos NPM para poder gestionar sus dependencias:

```
$ npm init -y
```

Como vamos a ir modificando el proyecto original jQuery para ir haciendolo más "React", copiaremos todos los ficheros que habíamos descargado del proyecto original, dentro de este nuevo directorio (esto incluye los ficheros `index.html`, `weather.` y `weahter.js`).

Integración con webpack

Antes de migrar a React el proyecto, vamos primero a poner a punto nuestro workflow de trabajo

con webpack. Para ello necesitamos instalarlo:

```
$ npm install --global webpack
```

Si todo ha ido bien, deberíais poder ejecutar:

```
$ webpack -h
webpack 1.12.4
Usage: https://webpack.github.io/docs/cli.html
```

Empaquetado de ficheros JavaScript

Para comenzar a integrar nuestros ficheros JavaScript en la build, es necesario crear el fichero de configuración de webpack `webpack.config.js`. Partiremos de la configuración más simple posible:

```
module.exports = {
    entry: './weather.js',
    output: {
        filename: 'bundle.js',
        path: __dirname
    }
}
```

Si ahora ejecutamos webpack, se generará el fichero de salida `bundle.js` con el resultado de empaquetar todos los recursos:

```
$ webpack
Hash: 298c1b69b8649145efa7
Version: webpack 1.12.4
Time: 71ms
    Asset     Size  Chunks             Chunk Names
bundle.js  6.47 kB       0  [emitted]  main
   [0] ./weather.js 4.93 kB {0} [built]
```

Ya sólo nos queda sustituir la siguiente linea en el `index.html`:

```
<script src="weather.js"></script>
```

Por la del bundle generado por webpack:

```
<script src="bundle.js"></script>
```

Si no hemos metido la pata, todo debería seguir funcionando de la misma forma :)

Recuerda que si quieres generar la versión de producción del bundle, sólo debes ejecutar webpack con el siguiente flag:

```
webpack -p
```

Empaquetado de ficheros CSS

Webpack permite empaquetar y gestionar múltiples recursos al margen de los de tipo JavaScript, ofreciendo soporte también para CSS, SASS o mucho muchos más.

Como en el proyecto ya tenemos nuestros estilos en un fichero CSS, vamos a incluir `weather.css` en la build.

Para ello, debemos indicar a webpack que procese estos nuevos recursos modificando su fichero de configuración:

```
module.exports = {
    entry: './weather.js',
    output: {
        filename: 'bundle.js',
        path: __dirname
    },
    module: {
        loaders: [
            {
                test: /\.css$/,
                loader: 'style!css'
            }
        ]
    }
}
```

Esta nueva definición require de que los loaders `style` y `css` sean instalados previamente mediante NPM:

```
npm install --save-dev style-loader css-loader
```

Sólo nos queda pues modificar el fichero `weather.js`y hacer en su primera línea, un `require` del `weather.css`:

```
var styles = require('./weather.css');

var g, GLoc = {
    settings: {
       ...
```

Para finalizar, ya podemos eliminar la linea del `index.html` donde se incluye el fichero de estilos al estar ya empaquetado en el `bundle.js`:

```
<link rel="stylesheet" type="text/css" href="weather.css">
```

De nuevo, todo debería seguir funcionado.

Migrando a React

React con webpack

Para poder comenzar a utilizar React en este proyecto, necesitamos que webpack sea capaz de procesar JSX (lenguaje con el que definiremos declarativamente las vistas en React). Esto se consigue, como en todos los casos previos, mediante la inclusión de un loader (`babel-loader` en este caso). Es importante tener en cuenta que gracias a este loader, vamos a tener acceso también a todas las novedades de ES6 (aka ES2015).

Instalemos pues las dependencias de React en primer lugar:

```
npm install --save react react-dom
```

Y posteriormente, los loaders que webpack necesitará para procesar el código React:

```
npm install --save-dev babel-loader babel-preset-es2015 babel-preset-react
```

En este caso, también es necesaria una pequeña modificación del `webpack.config.js` para que estos nuevos loaders sean cargados:

```
module.exports = {
    entry: './weather.js',
    output: {
        filename: 'bundle.js',
        path: __dirname
    },
    module: {
        loaders: [
           {
               test: /\.css$/,
               loader: 'style!css'
           },
           {
               test: /\.jsx?$/,
               exclude: /node_modules/,
               loader: 'babel',
               query: {
                   presets:['es2015','react']
               }
           }
       ]
    },
    resolve: {
        extensions: ["", ".js", ".jsx", ".css"]
    }
}
```

Definición de widgets

La idea es ir migrando el código HTML/JS basado en jQuery a React de forma progresiva. Para ello vamos a definir un `div` que va a contener todo el código renderizado por React y que será incluido en el `index.html`:

```
    ...
    <body id="weather-background" class="default-weather">
      <div class="page-wrap">
        <!-- Nuevo div para el output de react -->
        <div id="react-output"></div>
    ...
```

En el `weather.js`, crearé mi primera clase React que va a representar la aplicación del tiempo y comprobaré que webpack no se queja al procesar estos nuevos elementos:

```
import React from 'react';
import ReactDOM from 'react-dom';
import styles from './weather.css';

class WeatherApp extends React.Component {
    render() {
        return (
            <div/>
        );
    }
}
```

```
ReactDOM.render(<WeatherApp />, document.getElementById("react-output"));
```

A partir de ahora, podemos dejar a webpack en modo `watch` para que vaya procesando los cambios y los podamos ir viendo al momento en el navegador:

```
webpack -w
```

Para iniciar la transformación, el primer paso será mover todo el marcado HTML que tenemos en `index.html` dentro del bloque etiquetado con el atributo `class="page-wrap"` al método render de Nuestra clase `WatherApp`:

```
class WeatherApp extends React.Component {
    render() {
        return (
            <div>
              <header className="search-bar">
                <p className="search-text">
                  <span className="search-location-text">What's the weather like
in
                    <input id="search-location-input" className="search-
location-input" type="text" placeholder="City" /> ?
                  </span>
                </p>

                <div className="search-location-button-group">
                  <button id="search-location-button"
                          className="fa fa-search search-location-button search-
button"></button>
                  <button id="geo-button" className="geo-button fa fa-location-
arrow search-button"></button>
                </div>
              </header>

              <div id="front-page-description" className="front-page-description
middle">
                <h1>Blank Canvas Weather</h1>
                <h2>An Obligatory Weather App</h2>
              </div>

              <div id="weather" className="weather middle hide">
                <div className="location" id="location"></div>
                <div className="weather-container">
                  <div id="temperature-info" className="temperature-info">
                    <div className="temperature" id="temperature"></div>
                    <div className="weather-description" id="weather-
description"></div>
                  </div>
                  <div className="weather-box">
                    <ul className="weather-info" id="weather-info">
                      <li className="weather-item humidity">Humidity: <span
id="humidity"></span>%</li>
                      <li className="weather-item wind">
                        Wind: <span id="wind-direction"></span>
                        <span id="wind"></span>
                        <span id="speed-unit"></span></li>
                    </ul>
                  </div>
                </div>
              </div>
            </div>
        );
    }
```

```
}
```

Como podemos comprobar, todo sigue funcionando despues de este cambio.

Modularización

Aunque ya tenemos todo el marcado en una vista React, vamos a intentar partir el ejemplo en componentes más pequeños para ir aislando su funcionalidad.

Este es el resultado de la modularización:

```
class SearchBar extends React.Component {
    render() {
        return (
            <header className="search-bar">
              <p className="search-text">
                <span className="search-location-text">What's the weather like
in
                <input id="search-location-input"
                       className="search-location-input"
                       type="text" placeholder="City"/> ?</span>
              </p>

              <div className="search-location-button-group">
                <button id="search-location-button"
                        className="fa fa-search search-location-button search-
button"></button>
                <button id="geo-button"
                        className="geo-button fa fa-location-arrow search-
button"></button>
              </div>
             </header>
        );
    }
}

class Wellcome extends React.Component {
    render() {
        return (
            <div id="front-page-description" className="front-page-description
middle">
              <h1>Blank Canvas Weather</h1>
              <h2>An Obligatory Weather App</h2>
            </div>
        );
    }
}

class Info extends React.Component {
    render() {
        return (
          <div id="weather" className="weather middle hide">
            <div className="location" id="location"></div>
            <div className="weather-container">
              <div id="temperature-info" className="temperature-info">
                <div className="temperature" id="temperature"></div>
                <div className="weather-description" id="weather-
description"></div>
              </div>
              <div className="weather-box">
                <ul className="weather-info" id="weather-info">
                  <li className="weather-item humidity">Humidity: <span
```

```
id="humidity"></span>%</li>
                  <li className="weather-item wind">
                    Wind: <span id="wind-direction"></span>
                    <span id="wind"></span>
                    <span id="speed-unit"></span></li>
                </ul>
              </div>
            </div>
          </div>
      );
    }
}

class WeatherApp extends React.Component {
   render() {
       return (
           <div>
             <SearchBar />
             <Wellcome />
             <Info />
           </div>
       );
    }
}
```

Por facilidad a la hora de seguir el ejemplo, vamos a dejar todo el código en el mismo fichero, pero la práctica habitual es separa los componentes en distintos ficheros e importarlos cuando sea necesario mediante `import`. Gracias a webpack, todo acabará siendo incluido en nuestro `bundle.js` de salida.

Eventos

Llegados a este punto, sigue siendo jQuery el que gestiona los eventos que se producen en la aplicación. Vamos pues a migrar esta parte a eventos gestionado por React.

Eventos que se producen en la aplicación:

- `keypress` cuando escribimos el nombre de la ciudad a buscar (si es un enter, entonces se ejecuta la búsqueda).
- `click` en el botón de buscar.
- `click` en el botón de geolocalización.

Vamos a comenzar por el primero y vamos a hacernos cargo de la gestión del evento de `keypress`. Pare ello deberemos borrar el código jQuery que se ocupa de actuar sobre este evento y modifcar la clase `SearchBar` para tratar este evento, recuperar los datos del tiempo y enviar el resultado a `WeatherApp`:

```
const API_TOKEN = "0596fe0573fa9daa94c2912e5e383ed3";

class SearchBar extends React.Component {
    selectLocation(event) {
        if (event.keyCode !== 13) return;
        this.showWeather();
    }

    showWeather() {
        let location = this.refs["search-location-input"].value;
```

```
        let url= `http://api.openweathermap.org/data/2.5/weather?q=${location}&appid=${API_TOKEN}`;

        $.getJSON(url, (data) => {
            this.props.onData(data);
        });
    }

    render() {
        return (
            <header className="search-bar">
              <p className="search-text">
                <span className="search-location-text">What's the weather like in
                  <input id="search-location-input"
                         ref="search-location-input"
                         className="search-location-input" type="text"
                         placeholder="City"
                         onKeyDown={this.selectLocation.bind(this)} /> ?
                </span>
              </p>

              <div className="search-location-button-group">
                <button id="search-location-button"
                        className="fa fa-search search-location-button search-button"></button>
                <button id="geo-button" className="geo-button fa fa-location-arrow search-button"></button>
              </div>
            </header>
        );
    }
}
```

Quedando `WeatherApp` de la siguiente forma:

```
class WeatherApp extends React.Component {
    showWeatherData(data) {
        console.log(data);
    }

    render() {
        return (
            <div>
              <SearchBar onData={this.showWeatherData} />
              <Wellcome />
              <Info />
            </div>
        );
    }
}
```

Ahora que tenemos el evento controlado, vamos a ocultar el mensaje de bienvenida y mostrar el bloque de info del tiempo. Para ello haremos uso de la gestión del estado de los componentes en React y modificaremos `WeatherApp`:

```
class WeatherApp extends React.Component {
    constructor(props) {
        super(props);
        this.state = { ready : false };
    }
```

```
    showWeatherData(data) {
        this.setState({ ready : true });
    }

    render() {
        return (
            <div>
              <SearchBar onData={this.showWeatherData.bind(this)} />
              { (this.state.ready) ? <Info ref="info" /> : <Wellcome /> }
            </div>
        );
    }
}
```

Como podemos ver, el panel cambia, pero la info del tiempo no se muestra. Vayamos pues a completar esta parte!!

Para que la informacióńc correcta se muestre, añadiremos un nuevo método al componente `Info` para que se pueda realizar la carga y lo invocaremos desde `WeatherApp`. Podemos ver como otros métodos de cálculo son necesarios en `Info`, pero no son más que una copia del
código jQuery que ya teníamos:

```
class Info extends React.Component {
    constructor(props) {
        super(props);

        this.state = {
            loaded: false,
            location: '',
            humidity: '',
            description: '',
            temperature: '',
            windSpeed: 0,
            windDegree: '',
            windDirection: ''
        };
    }

    loadWeatherData(data) {
        this.setState({
            loaded: true,
            location: data.name + ', ' + data.sys.country,
            humidity: data.main.humidity,
            description: data.weather[0].description,
            windDirection: this.getWindDirection(data.wind.deg),
            temperature: Math.round(data.main.temp - 273.15),
            windSpeed: Math.round(data.wind.speed * 3.6)
        });
    }

    getWindDirection(degree) {
        if (degree > 337.5 || degree <= 22.5) {
            return 'N';
        } else if (22.5 < degree <= 67.5) {
            return 'NE';
        } else if (67.5 < degree <= 112.5) {
            return 'E';
        } else if (112.5 < degree <= 157.5) {
            return 'SE';
        } else if (157.5 < degree <= 202.5) {
```

```
            return 'S';
        } else if (202.5 < degree <= 247.5) {
            return 'SW';
        } else if (247.5 < degree <= 292.5) {
            return 'W';
        } else if (292.5 < degree <= 337.5) {
            return 'NW';
        }
    }

    render() {
        return (
            <div id="weather" className="weather middle" style={styles.info}>
              <div className="location"
id="location">{this.state.location}</div>

              <div className="weather-container">
                <div id="temperature-info" className="temperature-info">
                  <div className="temperature"
id="temperature">{this.state.temperature}</div>
                  <div className="weather-description" id="weather-
description">{this.state.description}</div>
                </div>
                <div className="weather-box">
                  <ul className="weather-info" id="weather-info">
                    <li className="weather-item humidity">Humidity: <span
id="humidity">{this.state.humidity}</span>%</li>
                    <li className="weather-item wind">
                      Wind: <span id="wind-
direction">{this.state.windDirection}</span> <span
id="wind">{this.state.windSpeed}</span> <span id="speed-
unit">{this.state.speedUnit}</span>
                    </li>
                  </ul>
                </div>
              </div>
             </div>
        );
    }
}

class WeatherApp extends React.Component {
    constructor(props) {
        super(props);
        this.state = { ready : false };
    }

    showWeatherData(data) {
        this.setState({ ready : true });
        this.refs["info"].loadWeatherData(data);
    }

    render() {
        return (
            <div>
              <SearchBar onData={this.showWeatherData.bind(this)} />
              { (this.state.ready) ? <Info ref="info" /> : <Wellcome /> }
            </div>
        );
    }
}
```

Ya sólo nos queda que pueda funcionar el botón de buscar y el de geoposicionar:

```
class SearchBar extends React.Component {
    selectLocation(event) {
        if (event.keyCode !== 13) return;
        this.showWeather();
    }

    selectCurrentLocation() {
        navigator.geolocation.getCurrentPosition((position) => {
            this.showWeatherByLatitude(position.coords.longitude,
position.coords.latitude);
        });
    }

    showWeather() {
        let location = this.refs["search-location-input"].value;
        let url= `http://api.openweathermap.org/data/2.5/weather?q=$
{location}&appid=${API_TOKEN}`;

        $.getJSON(url, (data) => {
            this.props.onData(data);
        });
    }

    showWeatherByLatitude(longitude, latitude) {
        let url= `http://api.openweathermap.org/data/2.5/weather?lat=$
{latitude}&lon=${longitude}&appid=${API_TOKEN}`;

        $.getJSON(url, (data) => {
            this.props.onData(data);
        });
    }

    render() {
        return (
             <header className="search-bar">
               <p className="search-text">
                 <span className="search-location-text">What's the weather like
in
                   <input id="search-location-input"
                          ref="search-location-input"
                          className="search-location-input" type="text"
                          placeholder="City"
                          onKeyDown={this.selectLocation.bind(this)} /> ?
                 </span>
               </p>

               <div className="search-location-button-group">
                 <button id="search-location-button"
                         className="fa fa-search search-location-button search-
button"
                         onClick={this.showWeather.bind(this)}></button>
                 <button id="geo-button"
                         className="geo-button fa fa-location-arrow search-
button"

onClick={this.selectCurrentLocation.bind(this)}></button>
               </div>
             </header>
        );
    }
}
```

Wrap-up y algunas mejoras

Con todo el marcado migrado, los componentes React extraidos y los eventos definidos, ya sólo nos queda borrar todo el código jQuery de `weather.js` y comprobar que la aplicación sigue siendo funcional.

NOTA: Si has llegado hasta aquí y algo no te funciona, revisa tu código comparándolo con el que he publicado en el repo de la charla en Github.

A partir de aquí, si queremos prescindir totalmente de jQuery, podemos usar `fetch` para realizar las peticiones AJAX al servicio del tiempo que se usa en la aplicación. Como fetch aún no está implementado en todos los navegadores, podemos usar una librería externa que actua como polyfill. En nuestro caso vamos a utilizar `isomorphic-fetch`

Para ello, instalamos como siempre las dependencias necesarias:

```
npm install --save isomorphic-fetch es6-promise
```

Y sustituiremos la parte de `$.getJSON` que obtiene los datos en JSON mediante jQuery, por la parte de `fetch`:

```
import es6promise from 'es6-promise';
import fetch from 'isomorphic-fetch';
es6promise.polyfill();

const API_TOKEN = "0596fe0573fa9daa94c2912e5e383ed3";

class SearchBar extends React.Component {

    ...

    showWeather() {
        let location = this.refs["search-location-input"].value;
        let url = `http://api.openweathermap.org/data/2.5/weather?q=${location}&appid=${API_TOKEN}`;

        fetch(url)
            .then(function (response) {
                return response.json();
            })
            .then((data) => {
                this.props.onData(data);
            });
    }

    showWeatherByLatitude(longitude, latitude) {
        let url = `http://api.openweathermap.org/data/2.5/weather?lat=${latitude}&lon=${longitude}&appid=${API_TOKEN}`;

        fetch(url)
            .then(function (response) {
                return response.json();
            })
            .then((data) => {
                this.props.onData(data);
            });
    }

    render() {
        ...
```

```
    }
}
```

Si hemos integrado `fetch` en la aplicación, ya podemos quitar tranquilamente la dependencia de jQuery del `index.html` y seguir funcionando con normalidad.