Webpack.

Webpack es un módulo creado para empaquetar las aplicaciones modernas desarrolladas con JavaScript, siendo posible usarlo desde un API como también desde la línea de comandos (cli).

La función principal de webpack es poder crear un único paquete con toda nuestra app, unificando el código en un único archivo .js (bundle), .css e imágenes.
Es realmente muy simple de implementar, y lo único que requiere es del archivo de configuración en el cwd de nuestro proyecto.

Instalando webpack.

```
$ npm install webpack --save
```

Ejemplo de archivo de configuración (webpack.config.js) :

```
const webpack = require('webpack');
const path = require('path');
const BUILD_DIR = path.resolve(__dirname, 'src/public');
const APP_DIR = path.resolve(__dirname, 'src/app');

const config = {
  entry: APP_DIR + '/index.jsx',
  output: {
    path: BUILD_DIR,
    filename: 'bundle.js'
  },
  module : {
    loaders : [
      {
        test : /\.js*?/,
        include : APP_DIR,
        loader : 'babel'
```

```
      }
    ]
  }
};

module.exports = config;
```

En el ejemplo de configuración hemos seteado que el archivo principal es index.jsx, y que el directorio con el código fuente es src/app.
Por otro lado, en la constante BUILD_DIR hemos indicado que el directorio de salida (en donde se guardará el archivo bundle.js) es src/public.

Ejecutando webpack.

$ ./node_modules/webpack/bin/webpack.js --watch

Redux.

Redux es un contenedor de estados predecibles para aplicaciones desarrolladas en JavaScript. Básicamente aporta un mecanismo de muy simple uso para poder mantener los estados de una aplicación, y gracias a su método observe podemos 'reaccionar' a cada cambio.

Los estados en Redux son controlados dentro de un store, el cual recibe como único argumento una función (reducer) que es la que tendrá las reglas de negocio para cada una de las actions.

Todos los reduces deben cumplir con una estructura mínima, ser funciones puras y recibir 2 argumentos, el primero es el estado actual, el segundo es el action.

Por ejemplo:

```
function counter(state = 0, action) {
  switch (action.type) {
    case 'INCREMENT':
    return state + 1;
    case 'DECREMENT':
    return state - 1;
    default:
    return state;
  }
}
```

Como vemos, el reducer counter tiene un estado por default que es cero y un action. El action es un objeto que debe tener la property 'type' la cual se utiliza para saber que tipo de action se desea realizar sobre el estado.

Una vez que tenemos definido nuestro reducer lo que debemos hacer es pasarlo como argumento al método createStore.

```
let store = createStore(counter);

store.subscribe(() => console.log(store.getState()))

store.dispatch({ type: 'INCREMENT' });
store.dispatch({ type: 'INCREMENT' });
store.dispatch({ type: 'DECREMENT' });
```

Como vemos, el método createStore retorna un store el cual nos ofrecer 2 métodos, el primero es subscribe que permite agregar un callback que se disparará cada vez que un estado cambia, el segundo es dispatch, que justamente se utiliza para cambiar un estado.

Primer app.

En el directorio example se encontrará un scaffolding para poder empezar con nuestras primeras aplicaciones en Reac.
Para poder instalar simplemente debemos ejecutar

```
$ npm install
```

La app en si está basada en Express para servir el contenido estático, el cual se encuentra dentro de ./src/public.
Todo el código que desarrollemos se encontrará dentro de ./src/app y para poder compilar debemos usar webpack.

Es necesario lanzar 2 consolas, en la primer dejaremos en bind a express:
$ PORT=5000 node app

En la segunda consola dejaremos a webpack en modo 'watch'.

```
$ ./node_modules/webpack/bin/webpack.js --watch
```

Una vez que dejamos ambas apps ejecutando (express y webpack) ya podemos comenzar a trabajar sin problemas, teniendo en cuenta que cada cambio a nivel código simplemente requiere un refresh en el navegador (F5).