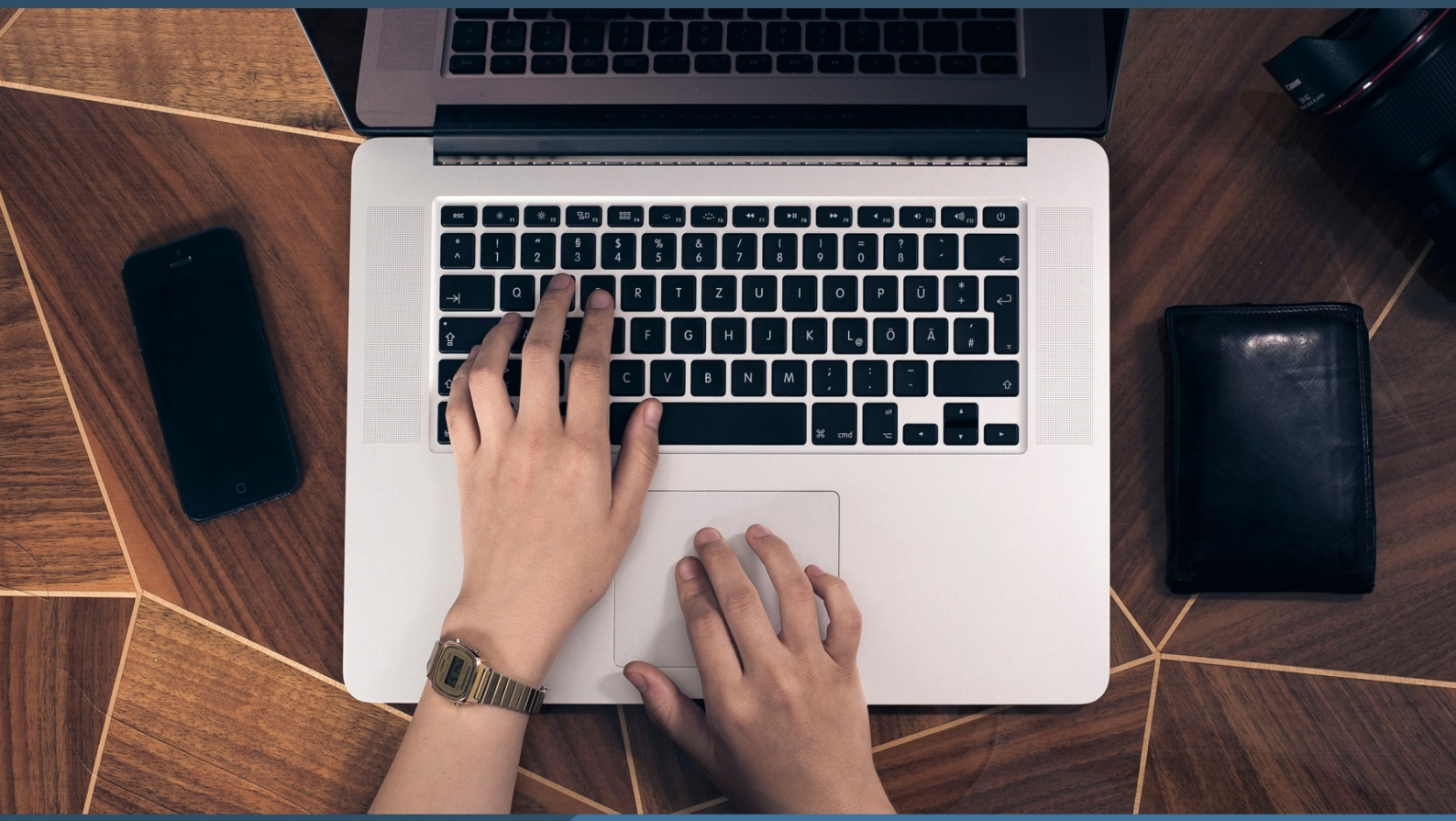


TU PRIMERA APP CON REACT.JS EN 10 MINUTOS

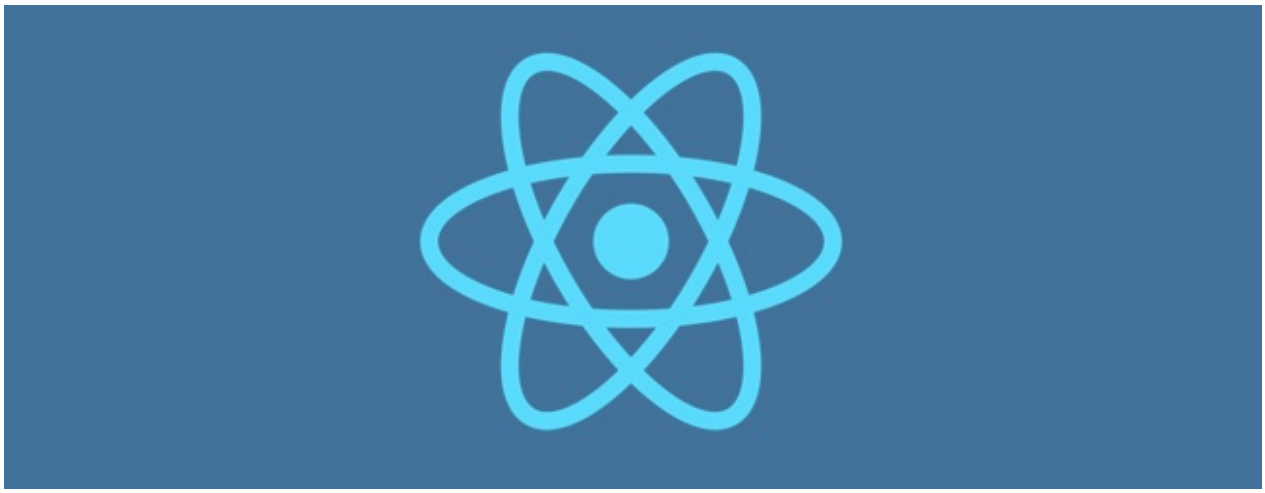
CARLOSAZAUSTRE.ES

Table of Contents

Tu primera App con React.js en 10 minutos

Hola! Mi nombre es Carlos Azaustre y soy el autor de esta guía que acabas de descargar.

Con esta guía vas a aprender desde cero, y sin necesidad de complicados flujos de trabajo y herramientas, a crear una aplicación con React.js en 10 minutos para comprender su funcionamiento y propiedades básicas.



De esta manera vas a tener en muy poco tiempo una aplicación React funcional que será tu punto de partida para adentrarte en el ecosistema React.

> Última actualización: 21 de Octubre 2016

Creando nuestra App

Página HTML

Lo primero que tenemos que hacer es crear un fichero HTML que será dónde React inserte los componentes que crearemos. Lo más importante en éste fichero será el elemento `<div>` donde se insertará la App y los `<script>` donde enlazaremos a la librerías que necesitemos.

Las librerías que emplearemos son `react.js` obviamente, `react-dom` que nos permite manejar el DOM del navegador, ya que React tambien puede ser usado en el servidor. Y por último `babel-standalone` que nos va ha permitir utilizar toda la nueva sintaxis de JavaScript (La especificación ECMAScript 6 o ES2015) aunque los navegadores para los que trabajemos no implementen todo.

Si no lo conoces, Babel es un *transpilador* que nos permite usar la nueva sinxtaxis y se ocupará de traducirlo a la versión de JavaScript que entienden los navegadores. Además incluye la transformación de JSX a JavaScript, ya que para que nos sea más cómodo crear componentes, utilizaremos éste suplemento.

Por tanto, nuestro `index.html` mínimo será

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Mi primera App React</title>
</head>
<body>
  <div id="app"></div>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/react/15.3.2/react.min.js"></sc
  <script src="https://cdnjs.cloudflare.com/ajax/libs/react/15.3.2/react-dom.min.js">
  <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-standalone/6.17.0/babel.m
  <script type="text/babel" src="app.js"></script>
</body>
</html>
```

A destacar el elemento `<div>` con el `id="app"` que utilizaremos posteriormente y el `script` `app.js` que es dónde vamos a tener el código fuente de nuestra aplicación que además le especificamos que sea de tipo `text/babel` en lugar de `text/javascript` para que *Babel* funcione

Componente principal

Ahora vamos a crear nuestro primer componente, le llamaremos `App` y es el que contendrá toda la funcionalidad y componentes de nuestra app.

Como tenemos *Babel* enlazado como `script`, podemos utilizar ECMAScript 6 para el código de nuestra App. Los componentes de React se pueden crear con el método `React.createClass()` o con la sintaxis de `class` que nos proporciona ES2015 y permitiendo que *herede* de `React.Component`.

Por tanto vamos a crear un fichero `app.js` con el siguiente contenido

```
class App extends React.Component {
  render () {
    return (<h1>Mi primera App</h1>);
  }
}

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

`App` será una clase de JS que hereda de `Component` de React.js y en su método `render()` devovelmos un componente que contiene un `<h1>` con el texto `Mi primera App`. Ésta sintáxis aunque parezca HTML, no lo es, es sintáxis JSX y será traducida a lo siguiente:

```
class App extends React.Component {
  render () {
    return React.createElement("h1", null, "Mi primera App");
  }
}
```

Como puedes ver, JSX lo hace más intuitivo y manejable.

Añadiendo un componente Hijo

No vamos a meter toda la lógica de nuestra app en un sólo componente, entonces la programación de interfaces basada en componentes perdería su "gracia" ya que nos permite modularizar el UI.

Por tanto vamos a crear más componente y anidarlos unos dentro de otros.

Nuestra app va mostrar un listado de empleados, y para cada uno de ellos utilizaremos un componente que lo represente.

```
class Empleado extends React.Component {
  render () {
    return (
      <li>
        <img src={this.props.imagen} />
        {this.props.nombre} - {this.props.email}
      </li>
    );
  }
}
```

El marcado de nuestro componente es un elemento `<li>` que contiene una imagen y un texto, y para ello utilizamos un objeto llamado `this.props` ¿Esto qué es? el objeto `props` es el que nos permite pasar información de componentes padres a hijos.

Por tanto, vamos a crear un componente padre `ListaEmpleados` que se encargue de renderizar tantos componentes `Empleado` sean como objetos empleado les pasemos.

Creamos el componente `ListaEmpleados` que recibirá un array de objetos y lo recorrerá llamando a un `Empleado` por cada uno. Veámoslo:

```
class ListaEmpleados extends React.Component {
  renderEmpleado () {
    return
  }
  render () {
    return (
      <ul>
        {
          this.props.empleados.map(empleado => {
            return (<Empleado
              nombre={empleado.nombre}
              email={empleado.email}
              imagen={empleado.imagen}
            />);
          });
        }
      </ul>
    );
  }
}
```

Como ves, estamos recorriendo con la función `map` un array que nos llega como propiedad `empleados` de un elemento padre y con una función *arrow* devolvemos un componente `Empleado` dónde le pasamos las propiedades que definimos: "nombre", "email" e "imagen"

Ya sólo nos queda modificar el componente App, para que renderice `ListaEmpleados` y pasarle el array de objetos:

```
var empleados = [
  { nombre: "Pepe", email: "pepe@correo.com", imagen: "foto1.png" },
  { nombre: "Paco", email: "paco@correo.com", imagen: "foto2.png" },
  { nombre: "Manolo", email: "manolo@correo.com", imagen: "foto3.png" }
];

class App extends React.Component {
  render () {
    return <ListaEmpleados empleados={empleados} />;
  }
}
```

Manejando el estado

Los componentes en React, además de tener propiedades que heredan de sus "padres", pueden tener estado. El estado es un objeto inmutable que tiene la peculariedad de que si es modificado, el componente llamará al método `render()` y hará que se *re-renderice* con la nueva información que reciba.

Por tanto, vamos a hacer unas modificaciones en el componente `App` para que tenga estado y su estado inicial contenga el array de objetos empleado.

Para ello, cómo estamos utilizando ES2015, las clases en JavaScript tienen un método `constructor()` y ahí es donde inicializaremos el estado.

```
class App extends React.Component {
  constructor () {
    super();
    this.state = {
      empleados: [
        { nombre: "Pepe", email: "pepe@correo.com", imagen: "foto1.png" },
        { nombre: "Paco", email: "paco@correo.com", imagen: "foto2.png" },
        { nombre: "Manolo", email: "manolo@correo.com", imagen: "foto3.png" }
      ];
    }
  }
}
```

Cómo `App` hereda de `React.Component` en el constructor debemos llamar al método `super()` para que llame también al constructor padre. Y el estado lo inicializamos a través del objeto `this.state`

Para que el estado tenga repercusión en el renderizado, también tenemos que modificar un poco el método render pasando a la propiedad `empleados` el valor del estado con `this.state.empleados`:

```
render () {
  return <ListaEmpleados empleados={this.state.empleados} />
}
```

¿Para que nos sirve esto? Imagina que quieres añadir nuevos "empleados" a la lista, al tener vinculado el estado al componente, cada vez que se modifique el estado (añadamos un nuevo empleado) el componente se va a "repintar" y aparecerá los datos del nuevo empleado en el navegador.

Propagación de eventos

Así pues, vamos a implementar esa funcionalidad paso a paso. Primero tendremos que modificar el componente `ListaEmpleados` para añadirle un formulario que recoja estos datos, tal que así:

```
class ListaEmpleados extends React.Component {
  render () {
    return (
      <div>
        <ul>
          {
            this.props.empleados.map(empleado => {
              return (<Empleado nombre={empleado.nombre} email={empleado.email} image
            })
          }
        </ul>
        <form onSubmit={this.props.onAddEmployee}>
          <input type="text" placeholder="Nombre" name="nombre"/>
          <input type="email" placeholder="Email" name="email" />
          <button type="submit">Añadir</button>
        </form>
      </div>
    )
  }
}
```

Lo que hemos hecho es añadir un elemento `<form>` que contiene dos `<input>` uno para recoger el nombre y otro para recoger el email. Por último tiene un botón de tipo `submit` que disparará el evento `onSubmit` que tiene el `<form>` y que nos proporciona React.js.

Si te fijas, `onSubmit` llama a una función `this.props.onAddEmployee`. Esto quiere decir que la función que realmente manejará este evento se la estamos proporcionando a través de las propiedades del componente, y por tanto será el componente padre el que la gestione.

Al contrario que los datos, que fluyen de padres a hijos a través de las propiedades, los eventos fluyen hacia arriba. Son disparados por un hijo y el padre puede recogerlo y así modificar lo que sea necesario.

Hacemos esto porque el formulario se encuentra en `ListaEmpleados` pero el array de todos ellos se encuentra en `App` y es de esta manera con la que podemos comunicarnos.

Por tanto, lo que nos toca ahora es modificar el componente `App` para que recoja ese evento, lo maneje y cambie el estado:

```
class App extends React.Component {
  constructor () { ... }
  render () {
    return (
      <ListaEmpleados
        empleados={this.state.empleados}
        onAddEmployee={this.handleOnAddEmployee.bind(this)}
      />
    )
  }
}
```

He añadido una nueva propiedad `onAddEmployee` que es la que recibe el hijo, y ésta propiedad llama a una función de `App` que ahora crearemos que se llama `handleOnAddEmployee`. Además ésta función le añadimos el método `bind` y le pasamos el objeto `this`.

Esto lo hacemos porque dentro de `handleOnAddEmployee` vamos a usar `this` y si no está *bindeando* no podremos usarlo correctamente.

Pasamos a desarrollar la función `handleOnAddEmployee`. Ésta función va a recibir por parámetro un objeto `event` el cuál trae la información que tengan los *input* del formulario.

Por tanto, primero usaremos el método `preventDefault()` para evitar que la página se recargue por ser el comportamiento por defecto de un formulario.

Despues recogeremos el valor del campo `nombre` y del `email`, y actualizaremos el estado.

Para actualizar el estado, puedes estar tentado de hacer un `push` al array de empleados, pero recordemos que el estado debe ser inmutable y la función `push` modifica el array desde el que se invoca, por tanto vamos a usar la función `concat` que no modifica el array si no que devuelve otro. De esta manera nuestro estado no *muta* y es más funcional, testeable y fácil de controlar. Veamos el código:

```
handleOnAddEmployee (event) {
  event.preventDefault()
  let empleado = {
    nombre: event.target.nombre.value,
    email: event.target.email.value
  }

  this.setState({
    empleados: this.state.empleados.concat([empleado])
  })
}
```

Como ves, para actualizar el estado usamos el método `setState` y al objeto "empleados", le añadimos el nuevo "empleado".

Cuando esto ocurra, como hemos dicho, llamará al método render y volverá a *renderizar* el componente `ListaEmpleados` con el nuevo empleado.

Para que no tengas dudas, te dejo a continuación el código completo de `app.js` para que lo pruebes:

```
class Empleado extends React.Component {
  render () {
    return (
      <li>
        <img src={this.props.imagen} />
        {this.props.nombre} - {this.props.email}
      </li>
    )
  }
}

class ListaEmpleados extends React.Component {
  render () {
    return (
      <div>
        <ul>
          {
            this.props.empleados.map(empleado => {
              return (<Empleado nombre={empleado.nombre} email={empleado.email} image
            })
          }
        </ul>
        <form onSubmit={this.props.onAddEmployee}>
          <input type="text" placeholder="Nombre" name="nombre"/>
          <input type="email" placeholder="Email" name="email" />
          <button type="submit">Añadir</button>
        </form>
```

```
      </div>
    )
  }
}

class App extends React.Component {
  constructor () {
    super()
    this.state = {
      empleados: [
        { nombre: "Pepe", email: "pepe@correo.com", imagen: "foto1.png" },
        { nombre: "Paco", email: "paco@correo.com", imagen: "foto2.png" },
        { nombre: "Manolo", email: "manolo@correo.com", imagen: "foto3.png" }
      ]
    }
  }


  handleOnAddEmployee (event) {
    event.preventDefault()
    let empleado = {
      nombre: event.target.nombre.value,
      email: event.target.email.value
    }

    this.setState({
      empleados: this.state.empleados.concat([empleado])
    })
  }

  render () {
    return <ListaEmpleados
      empleados={this.state.empleados}
      onAddEmployee={this.handleOnAddEmployee.bind(this)}
    />
  }
}


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

Conclusión

Con sólo esto ya conoces el funcionamiento básico de React y te sirve como punto de partida para empezar a profundizar con ésta librería.

Ten en cuenta que React es mucho más y se pueden hacer muchas más cosas. Ésta guía es el inicio para que le pierdas el miedo a la librería y te animes a probarla.

¿Quieres más? No te preocupes, **muy pronto lanzaré un curso online completo de React**, dónde veremos en profundidad cómo funciona la librería, buenas prácticas, integración con otras librerías, cómo crear varias vistas y rutas, etc...

Como ya formas parte de mi lista de correo, serás de los/as primeros/as en enterarte cuando el curso esté listo :)

¿Te ha gustado la guía? Devuélveme el favor con un Tweet

Si te ha gustado esta guía, te voy a pedir el favor de compartirlo con tus seguidores en Twitter y recomendarlo a tus amigos por email.

> Gracias por la guía sobre #React @carlosazaustre. Consigue la tuya y crea tu primera app aquí: https://carlosazaustre.es/unete

¡Muchas gracias por ayudarme a difundirlo!

Twitteálo!

Sobre el autor de ésta guía

Me llamo Carlos Azaustre y soy desarrollador web. Soy un amante de JavaScript y actualmente soy CTO en Chefly a la par que comparto conocimiento con la comunidad a través de mi blog y mi canal de YouTube

Nos vemos por las redes!

- Blog: http://carlosazaustre.es/blog
- Twitter: http://twitter.com/carlosazaustre
- Facebook http://facebook.com/carlosazaustre
- Instagram http://instagram.com/carlosazaustre
- GitHub http://github.com/carlosazaustre
- YouTube http://youtube.com/carlosazaustre