

REACT.JS

Curso v.2019

Alfonso Tienda

INTRODUCCIÓN

¿QUÉ ES REACT Y CUÁL ES SU PROPÓSITO?

- React es una librería de javascript
- Su propósito es renderizar HTML para permitir crear la interacción con el usuario en múltiples formatos (Web, Móvil, PWA)
- En este curso nos centraremos en Web, pero los conocimientos son la base para otros sistemas.

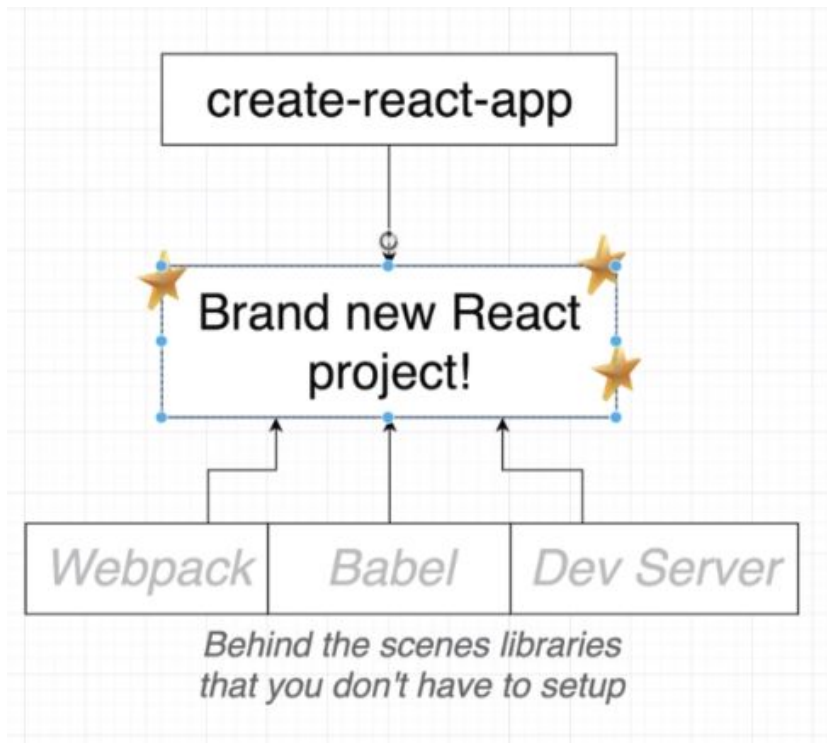
MÁS COSAS SOBRE REACT

- React puede funcionar por sí solo.
- Se programa en javascript
- Puede usar múltiples librerías, como Redux o axios, que son de propósito general de javascript.
- Un componente se puede escribir como una clase ES6 o como una función.
- JSX es un lenguaje con una sintaxis similar a HTML, pero son objetos Javascript
- Los eventos los capturan los manejadores de eventos

INSTALAR REACT

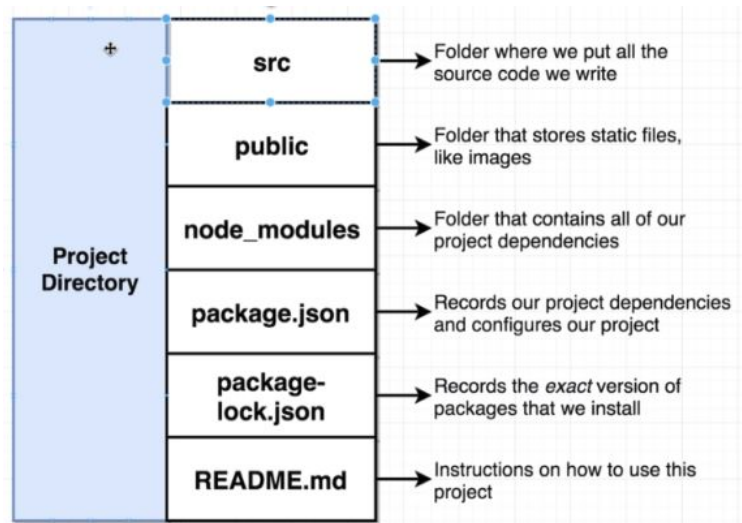
- Instalar NodeJS
- Instalar create-react-app
 - `npm install -g create-react-app`
- Generar un proyecto
 - `create-react-app helloworld`
- Build!
 - `cd helloworld`
 - `npm start`

¿POR QUÉ CREATE-REACT-APP?

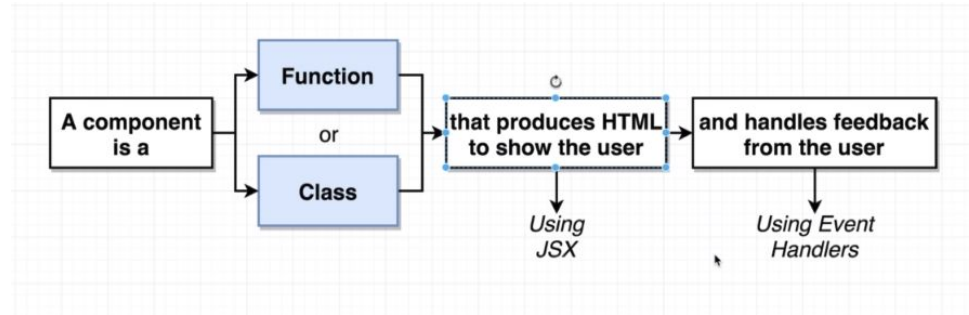


CREEMOS NUESTRA APP JSX

create-react- app jsx



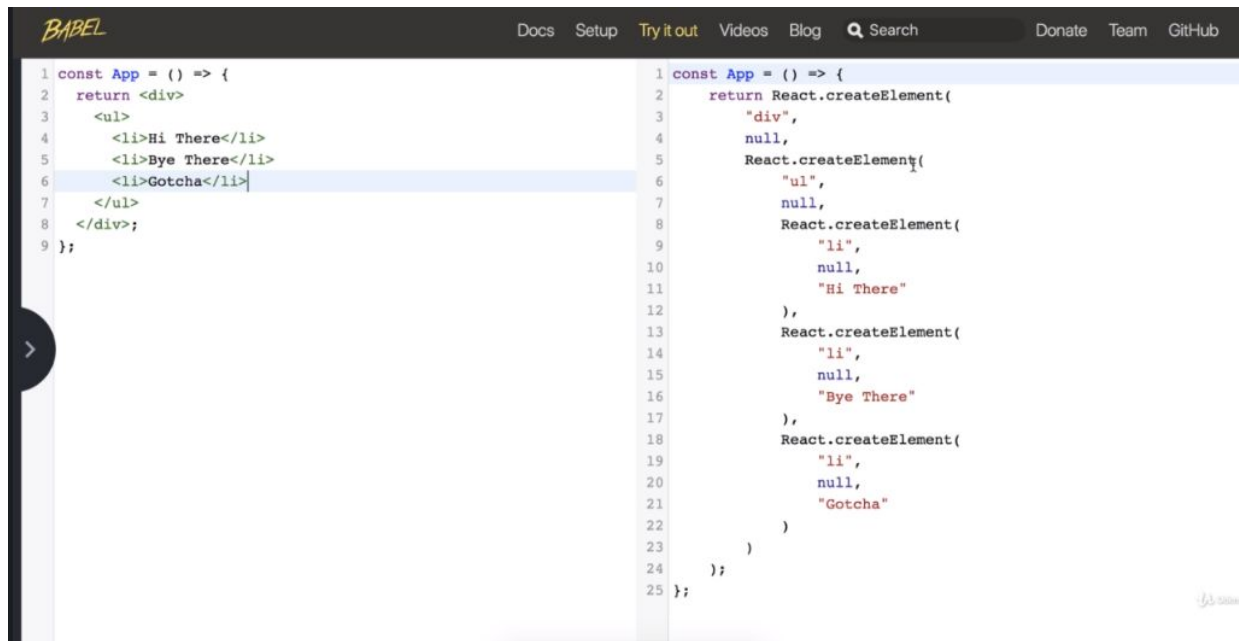
COMPONENTES



JSX

PLAYGROUND

babeljs.io



The screenshot shows the Babel Playground interface with two panels. The left panel displays the input code, and the right panel displays the transformed code.

Input Code (Left Panel):

```
1 const App = () => {  
2   return <div>  
3     <ul>  
4       <li>Hi There</li>  
5       <li>Bye There</li>  
6       <li>Gotcha</li>  
7     </ul>  
8   </div>;  
9 };
```

Transformed Code (Right Panel):

```
1 const App = () => {  
2   return React.createElement(  
3     "div",  
4     null,  
5     React.createElement(  
6       "ul",  
7       null,  
8       React.createElement(  
9         "li",  
10        null,  
11        "Hi There"  
12      ),  
13      React.createElement(  
14        "li",  
15        null,  
16        "Bye There"  
17      ),  
18      React.createElement(  
19        "li",  
20        null,  
21        "Gotcha"  
22      )  
23    )  
24  );  
25 };
```

The Babel logo is in the top left corner of the interface. The top navigation bar includes links for Docs, Setup, Try it out, Videos, Blog, Search, Donate, Team, and GitHub. A small Babel logo is also visible in the bottom right corner.

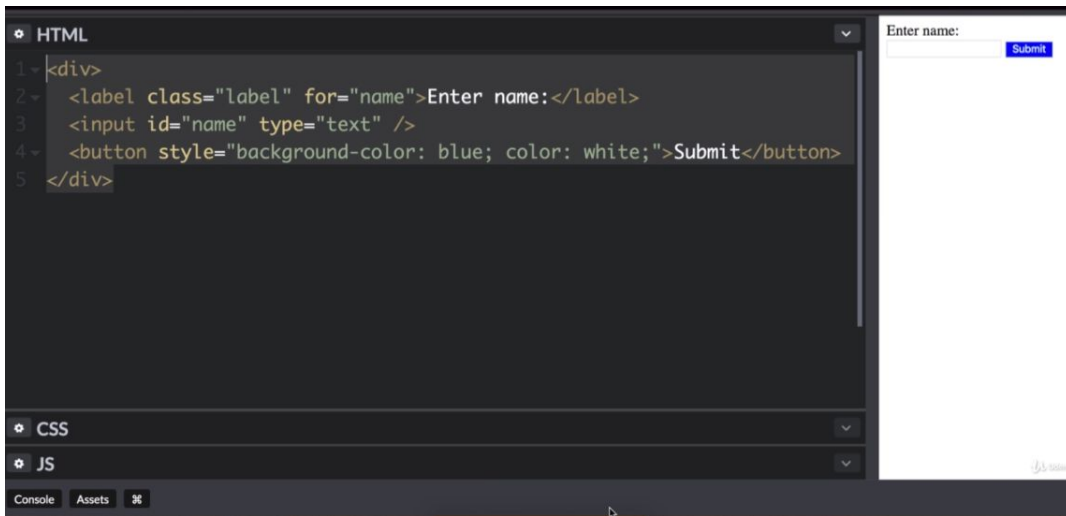
DE HTML A JSX

- Dialecto especial, no es exactamente HTML
- Los browsers no entienden JSX. Escribimos JSX y la librería 'react' lo traduce a Javascript común
- Muy similar a HTML con un par de diferencias.

PLAYGROUND

Codepen.io

creamos y probamos el
elemento y lo probamos
como JSX



JSSX INLINE STYLING

The `style`` prop expects a mapping from style properties to values, not a string. ➤
For example, `style={{marginRight: spacing + 'em'}}` when using JSX.

in button (at src/index.js:13)

in div (at src/index.js:8)

in App (at src/index.js:19)

► 25 stack frames were collapsed.

Module../src/index.js
src/index.js:19

```
16 | };  
17 |  
18 | // Take the react component and show it on the screen  
> 19 | ReactDOM.render(<App />, document.querySelector('#root'));  
20 |
```

JSX VS HTML

- Anadir 'inline styling' se realiza a través de objetos
- Añadir una clase a un elemento usa sintaxis diferente
- JSX puede referenciar variables JS (incluyendo funciones o ejecuciones de las mismas), no objetos enteros por ejemplo.

EJERCICIO

Mostrar la hora con la función:

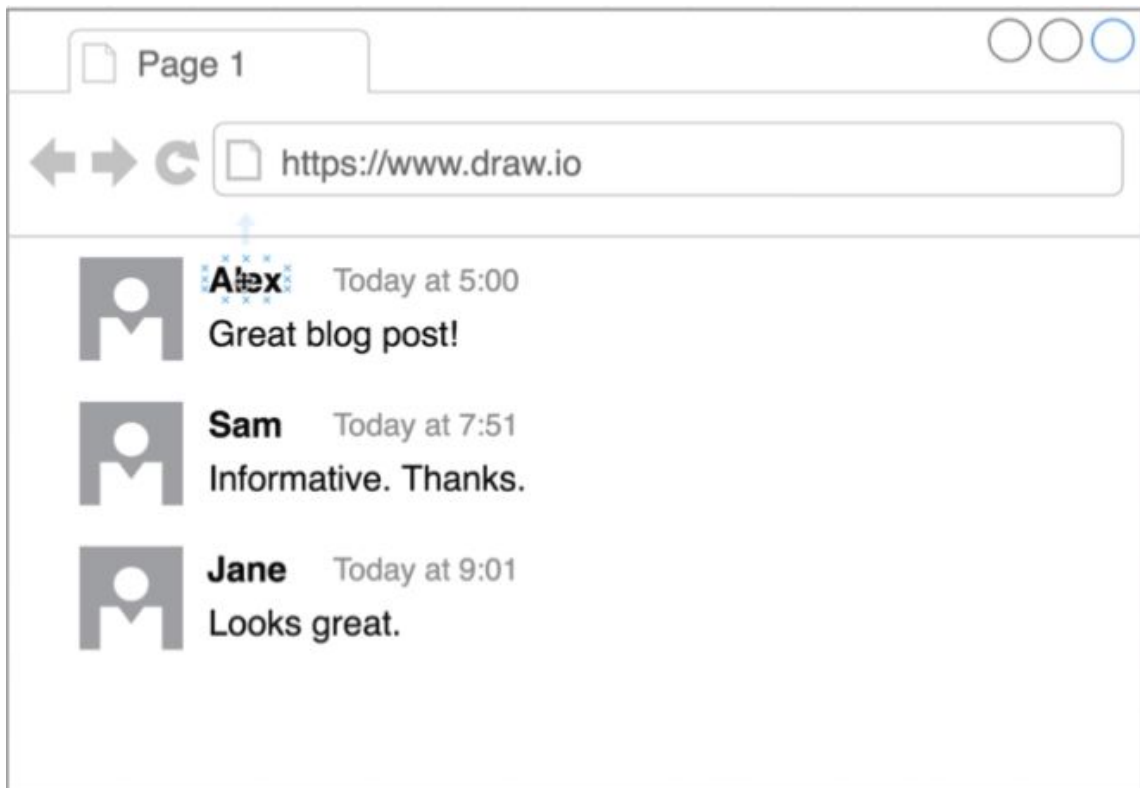
```
function getTime() {  
    return (new Date()).toLocaleTimeString()  
}
```

PROPS

OBJETIVOS DE LOS COMPONENTES

- Anidamiento. Un componente debe poder ser mostrado dentro de otro componente
- Reusabilidad: Queremos que sean reutilizados de forma sencilla
- Configuración: Debemos poder configurarlo cuando se crea

NUESTRA PRIMERA APP : COMPONENTS



ENLACES IMPORTANTES

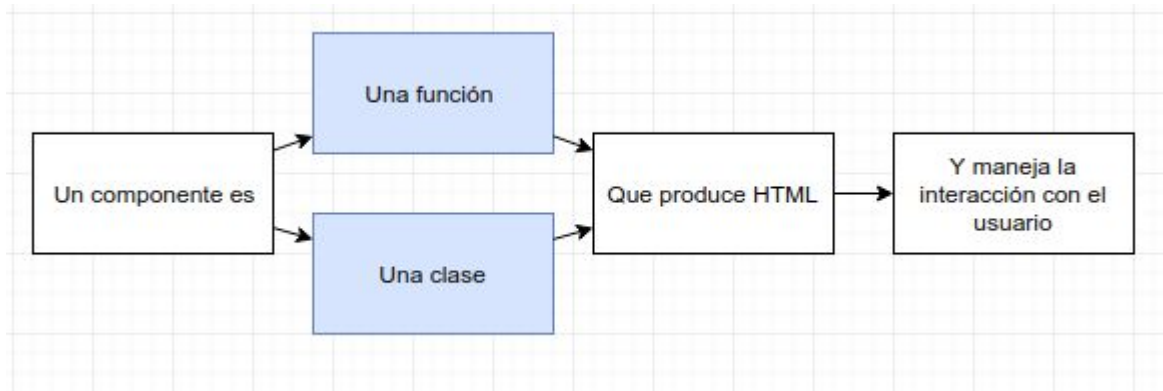
Faker: <https://github.com/marak/Faker.js/>

Semantic UI: <https://semantic-ui.com/>

Semantic UI CDN: <https://cdnjs.com/libraries/semantic-ui>

CLASS BASED COMPONENTS

REPASO DEL CONCEPTO DE COMPONENTE



¿CUANDO UTILIZAMOS UNO U OTRO?

Los componentes funcionales son útiles para componentes simples que se basan fundamentalmente en renderizar un aspecto

Los componentes basados en clases se utilizan cuando son más complejos, formularios o bien utilizar elementos del ciclo de vida

VENTAJAS DE LOS COMPONENTES BASADOS EN CLASES

Mejor organización de código

Puede usar el estado (otro sistema de React), lo que hace más sencillo manejar la entrada del usuario.

Más fácil usar los eventos del ciclo de vida.

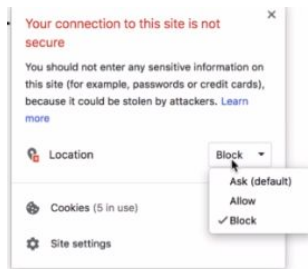
NUEVA APP



Nos dirá si es invierno o verano según el mes que estamos y dónde estamos.

Geolocation API: <https://developer.mozilla.org/es/docs/Web/API/Geolocation/getCurrentPosition>

Reseteo de posición:



ESTADO

ESTADO

Guardemos y cambiemos el estado en la aplicación actual.

Renderización del estado.

Recuperaremos posts:

<https://jsonplaceholder.typicode.com/>

CICLO DE VIDA

CICLO DE VIDA

`constructor()`: este ciclo de vida se ejecuta cuando el componente es instanciado. Aquí podemos definir su configuración inicial. Por ejemplo, configurar el estado o crear conexiones con nuestras funciones.

`componentWillMount()`: las modificaciones en este ciclo de vida no causan actualizaciones en el componente, y se corre justo antes de montar o renderizar el componente. Por ejemplo, un cambio condicional en el estado.

CICLO DE VIDA 2

`render()`: este método, en esta etapa, genera la interfaz gráfica inicial. Modificar el estado puede causar un ciclo infinito. Por esta razón este método debe ser puro.

`componentDidMount()`: el código que retorna nuestra función ya ha sido renderizado en el DOM y nuestra interfaz, hemos llegado al final de la etapa de montaje. Este método solo se ejecuta una única vez. Es perfecto para trabajar con código asíncrono, llamadas a APIs, y código retrasado con `setTimeout`.

CICLO DE VIDA 3

Los componentes pueden o no actualizarse, y lo pueden hacer más de una vez. Los cambios en el estado o en las propiedades, son los causantes de estas actualizaciones, generando una interfaz con los nuevos valores. Los ciclos es esta etapa son:

`componentWillReceiveProps(nextProps)`: el primer ciclo en la etapa de actualización. Nos permite hacer cambios en el componente basados en un cambio en las propiedades. La razón por la que este método recibe el parámetro `nextProps`, es para permitirnos validar el cambio en las propiedades. `nextProps` debe ser diferente a `this.props`.

CICLO DE VIDA 4

`shouldComponentUpdate(nextProps, nextState)`: nos permite validar un cambio en el estado o en las propiedades del componente por medio de `nextProps`, `this.props`, `nextState`, y `this.state` y retornar verdadero o falso para renderizar nuevamente o no el componente, respectivamente. Por defecto, siempre retorna `true`.

`componentWillUpdate(nextProps, nextState)`: se ejecuta cuando `shouldComponentUpdate()` retorna verdadero. Se hacen los últimos cambios antes de renderizar nuevamente el componente.

CICLO DE VIDA 5

`componentDidUpdate(prevProps, prevState)`: este es el último método de esta etapa. El componente se ha renderizado con los nuevos valores. Es perfecto para interactuar con el DOM

Los componentes son desmontados

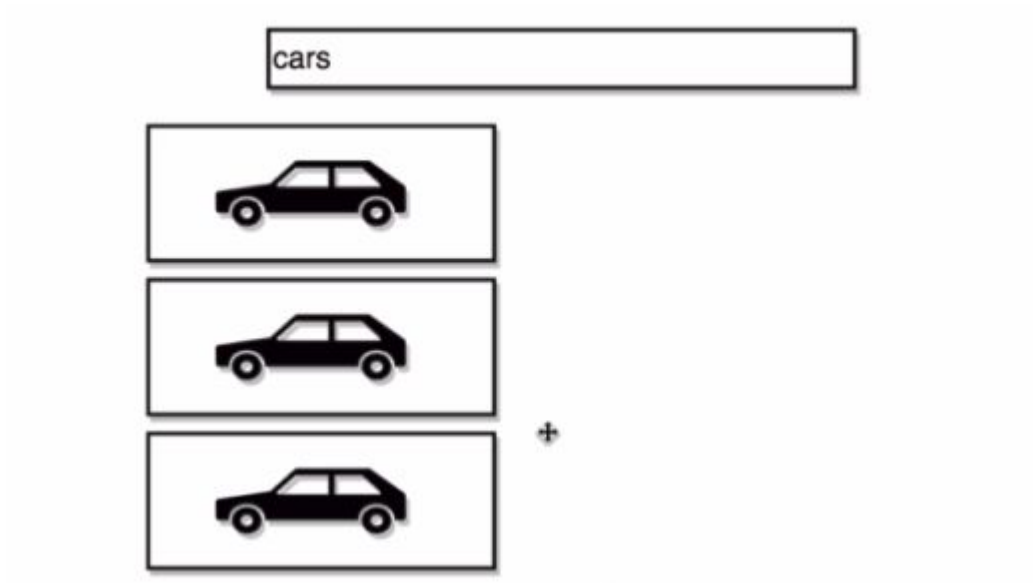
Está es la última fase de los componentes. Consta de un único método que es invocado justo antes de que el componente sea destruido o sea sacado de nuestra interfaz.

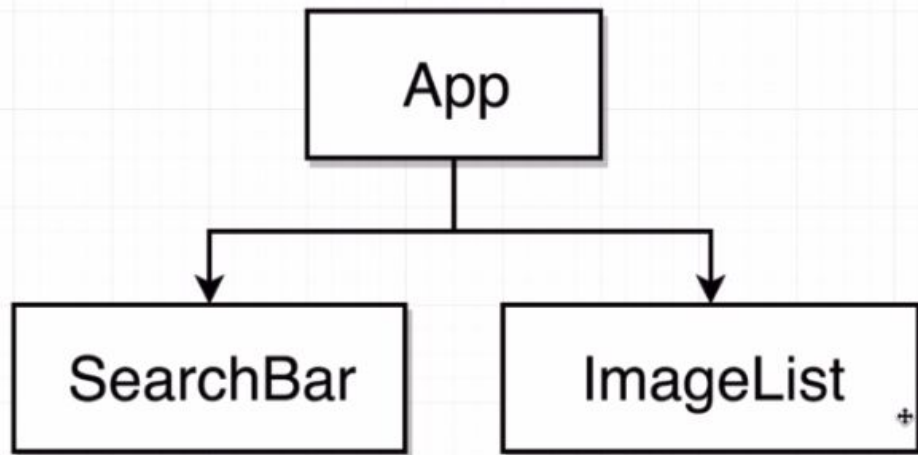
`componentWillUnmount()`: su principal funcionalidad es limpiar nuestro componente. Por ejemplo, dejar de escuchar eventos o cancelar peticiones HTTP pendientes.

USER INPUT:
FORMULARIOS

NUEVA APP: IMÁGENES

Utilizando unsplash: <https://unsplash.com/developers>





ROUTER

INSTALACIÓN

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

```
import { BrowserRouter as Router, Route, Link } from  
'react-router-dom'
```

Uso

```
const App = () => (  
  <BrowserRouter>  
    <div>  
      <ul>  
        <li><Link to="/">Home</Link></li>  
        <li><Link to="/about">About</Link></li>  
      </ul>  
      <hr/>  
      <Route path="/" component={Home}/>  
      <Route path="/about" component={About}/>  
    </div>  
  </BrowserRouter>  
)  
export default App
```

ROUTE

Ruta exacta a componente:

```
<Route exact path="/home" component={Home} />
```

Ruta con match (this.props.match)

```
<Route exact path="/detail/:id" component={Detail} />
```

Atributo 'exact':

```
<Route exact path="/users" component={Users} />
```

```
<Route path="/users/create" component={CreateUser} />
```

JSON - SERVER

INSTALACIÓN

```
npm install -g json-server
```

Crear db.json

```
json-server --watch db.json
```

DB.JSON

```
{ "medicamentos": [  
    { "id": 1, "name": "json-server", "author": "typicode" }  
],  
  "recetas": [  
    { "id": 1, "quantity": 24, "patient": "juan",  
      "medicamentoId": 1 }  
],  
}
```

ROUTERS

MemoryRouter

No cambia la URL en el browser, la mantiene en memoria.

HashRouter

Usa el client-side hash routing (routing de almohadilla)

La parte de la almohadilla no llegará al servidor, el servidor siempre envía el index.html. Funciona en navegadores antiguos.

BrowserRouter

El más usado. Usa la API pushState de los navegadores modernos. Son direcciones completas. Soporta history.

PORTALS

PORTALS SON ELEMENTOS MODALES

`ReactDOM.createPortal(child, container)`

Pruebas en codepen: <https://codepen.io/gaearon/pen/yzMaBd>

PORTALES: EJEMPLO

```
import React from 'react';
import ReactDOM from 'react-dom'
const Modal = props => {
  return ReactDOM.createPortal(
    <div className="ui dimmer modals visible active" onClick={props.onDismiss}>
      <div className="ui standard modal visible active" onClick={e=>e.stopPropagation()}>
        <div className="header">{props.header}</div>
        <div className="content">{props.content}</div>
        <div className="actions">
          {props.actions}
        </div>
      </div>
    </div>
    ,document.querySelector("#modal")
  )
}
export default Modal;
```

PORTALES: EJEMPLO

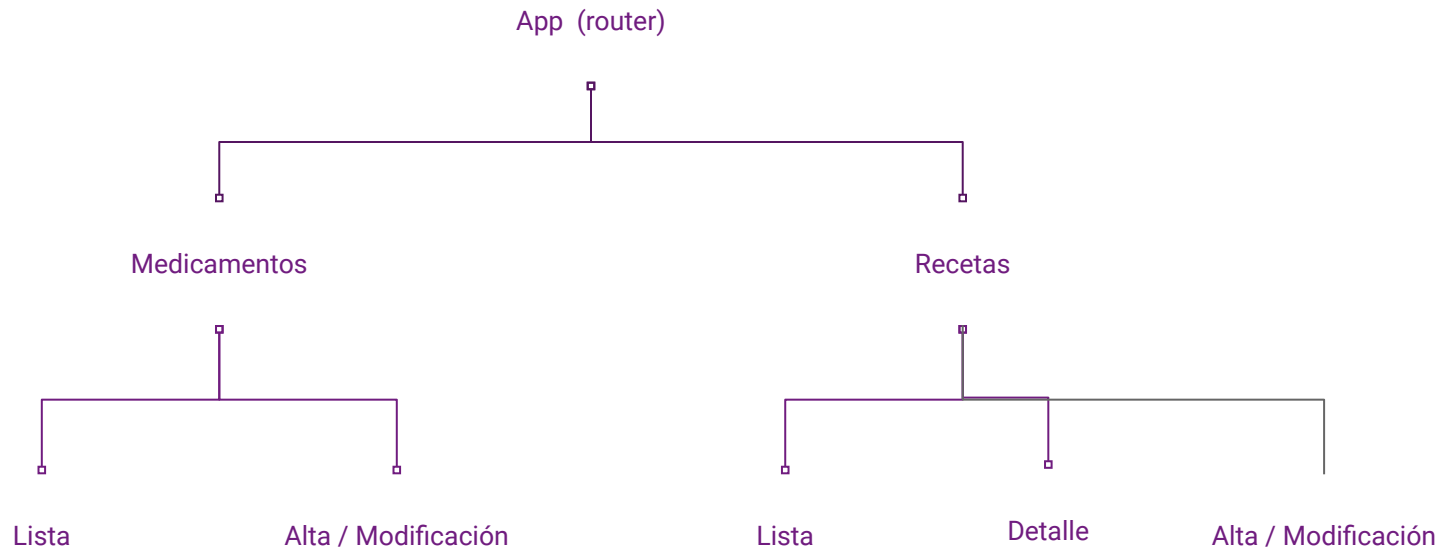
```
<Modal header={header}    content={`¿Seguro que quieres borrar ...?`}
      onDismiss={this.dismiss()}
      actions={<>
        <button className="ui negative button" onClick={this.delete}>Delete</button>
        <button className="ui button" onClick={this.dismiss()}>Cancel</button>
      </>
    }/>
```

```
dismiss = () => {
  history.push("/")
}
```

```
deleteStream = () => {
  this.props.delete(this.props.match.params.id);
}
```

RECETAS APP

RECETAS APP



CONTEXTO

CONTEXTO

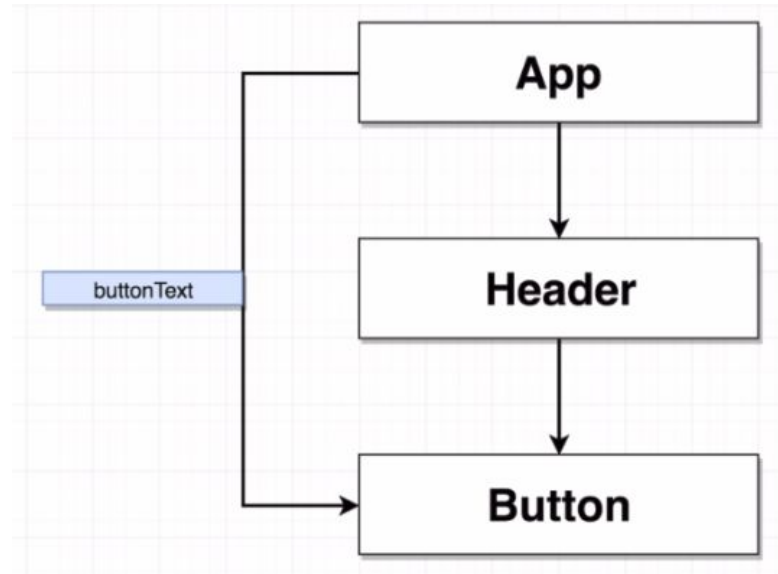
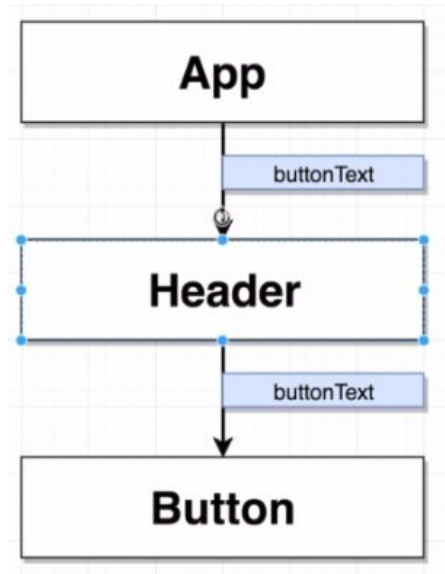
Propiedades:

- Pasa datos de un componente padre a cualquier hijo directo

Contexto:

- Pasa datos de un componente padre a cualquier descendiente

CONTEXTO



CREATECONTEXT

```
import React, { createContext } from 'react'
```

```
const AplicacionContext = createContext({ data... }) // Creamos el contexto
```

```
const { Provider, Consumer } = AplicacionContext // Obtenemos el provider y el consumer
```

CONTEXT PROVIDER: PROVEE DE LA INFORMACIÓN

```
import React, { Component, createContext } from 'react'

const { Provider, Consumer } = createContext()
const initialState = { count: 0 } // definimos un estado inicial

class App extends Component {
  render() {
    return (
      <Provider value={ initialState }> // Se lo pasamos al provider
        ...
      </Provider>
    )
  }
}
```

CONTEXT CONSUMER: CONSUME LA INFORMACIÓN

```
import React, { Component, createContext } from 'react'
const { Provider, Consumer } = createContext()
const initialState = { count: 0 }
```

```
export const MyComponent = props => (
  <Consumer>
    { context => { // context = { count: 0 }
      return (
        <div>
          | Count: { context.count }
        </div>
      )
    }
  </Consumer>
)
```

```
class App extends Component {
  render() {
    return (
      <Provider value={ initialState }>
        <MyComponent />
      </Provider>
    )
  }
}
```

CREANDO UN CONTEXTO: LANGUAGECONTEXT.JS

```
import React from 'react';

const Context = React.createContext('en');

export class LanguageStore extends React.Component {
  state = {
    language: 'en'
  };

  onLanguageChange = language => {
    this.setState({language});
  }

  render() {
    return (
      <Context.Provider value={{...this.state, onLanguageChange : this.onLanguageChange}}>
        {this.props.children}
      </Context.Provider>
    )
  }
}

export default Context;
```


CONSUMIENDO UN CONTEXTO

```
import React from 'react';
import LanguageContext from '../contexts/LanguageContext'

class Button extends React.Component {
  render() {
    return (
      <LanguageContext.Consumer >
        {value => value.language==='en' ? "Submit" : "Enviar"}
      </LanguageContext.Consumer>
    )
  }
}

export default Button;
```

ÁMBITO DEL CONTEXTO

```
import React from 'react';

import UserCreate from './UserCreate';
import {LanguageStore} from '../contexts/LanguageContext'
import LanguageSelector from './LanguageSelector'
```

```
class App extends React.Component {

  render() {
    return (
      <div className="ui container">
        <div>
          <LanguageStore>
            <LanguageSelector />
            <UserCreate />
          </LanguageStore>
        </div>
      </div>
    )
  }
}

export default App;
```

CAMBIANDO EL VALOR

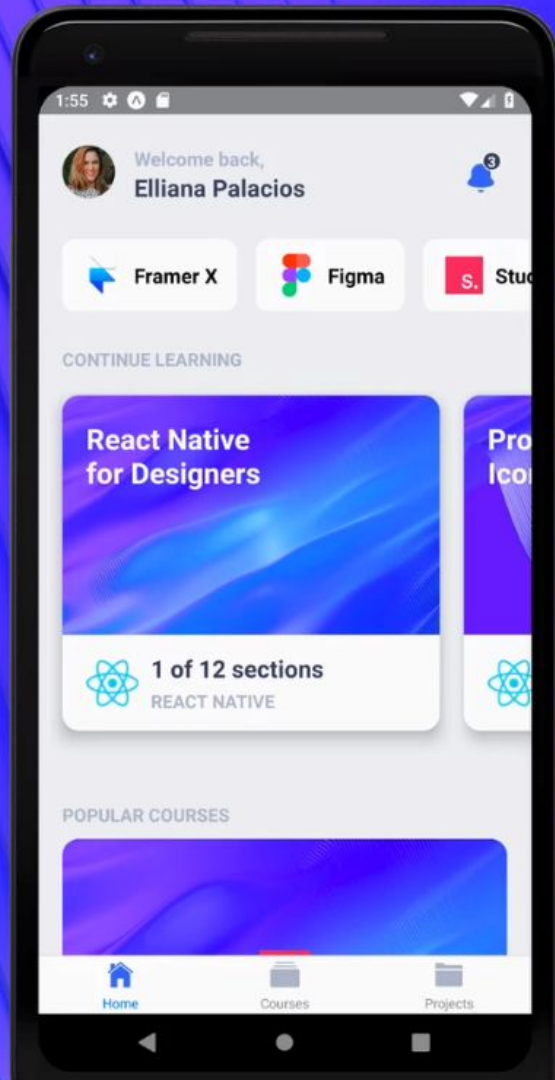
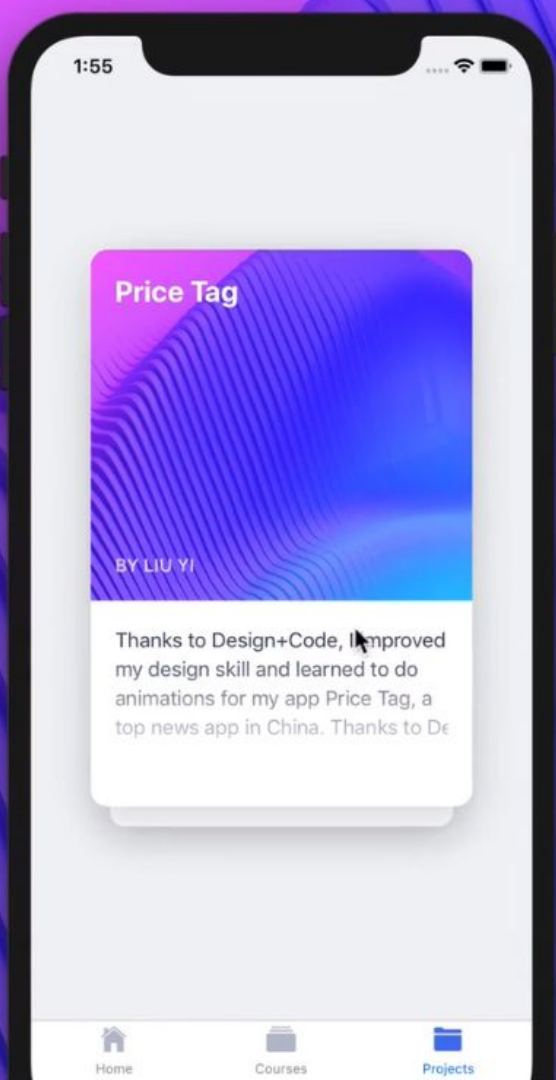
```
import React from 'react';
import LanguageContext from '../contexts/LanguageContext'

class LanguageSelector extends React.Component {
  render() {
    return (
      <div>
        <LanguageContext.Consumer>
          {ctx => {
            const aLabel = ctx.language==='en' ? <span>Select Language: </span>
              : <span>Seleccionar idioma: </span>
            return (
              <>
                {aLabel}
                <i className="flag us" onClick={() => ctx.onLanguageChange('en')} />
                <i className="flag es" onClick={() => ctx.onLanguageChange('es')} />
              </>
            )
          }}
        </LanguageContext.Consumer>
      </div>
    )
  }
}

export default LanguageSelector;
```

REACT ECOSYSTEM

REACT NATIVE



REACT 360 (VR)

