




REACT





Props vs. State

La principal diferencia entre el *state* y las *props* es que las *props* son inmutables.



Esta es la razón por la cual el componente del contenedor debe definir el *state* que se puede actualizar y cambiar, mientras que los componentes secundarios solo deben pasar datos del estado usando *props*.

Uso de props

App.js

```
import React from 'react';
class App extends React.Component {
  render() { return (
    <div>
      <h1>{this.props.headerProp}</h1>
      <h2>{this.props.contentProp}</h2>
    </div> );
  }
}
export default App;
```

index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App.jsx';
ReactDOM.render(<App headerProp = "Header from props..." contentProp = "Content from props..." />,
  document.getElementById('app')
);
export default App;
```

Default Props

App.js

```
import React from 'react';
class App extends React.Component {
  render() { return (
    <div> <h1>{this.props.headerProp}</h1> <h2>{this.props.contentProp}</h2> </div> );
  }
}
App.defaultProps = {
  headerProp: "Header from props...",
  contentProp: "Content from props..." }
export default App;
```

index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App.jsx';
ReactDOM.render(<App/>, document.getElementById('app'));
```

Combinación (child)

```
import React from 'react';
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = { header: "Header from props...",
                  content: "Content from props..."
                }
  }
  render() { return ( <div> <Header headerProp = {this.state.header}/>
    <Content contentProp = {this.state.content}/> </div> ); } }
class Header extends React.Component {
  render() { return ( <div> <h1>{this.props.headerProp}</h1> </div> );
  }
}
class Content extends React.Component {
  render() { return ( <div> <h2>{this.props.contentProp}</h2> </div> );
  }
}
export default App;
```



LifeCycle



Métodos (parte 1)

- **componentWillMount:** es ejecutado antes del render, tanto en el cliente como en el servidor.
- **componentDidMount:** es ejecutado después del primer render solo del lado del cliente. (se utiliza para actualizar *state*).
- **componentWillReceiveProps:** se invoca tan pronto como se actualizan las props antes de que se llame a otro render.

Project

The-Iron-Store

- .git
- node_modules
- public
- src
 - components
 - admin
 - checkout
 - reviews
 - shop
 - slider
 - styles
 - shop.css
 - Cart.js
 - ProductCard.js
 - Shop.js
 - splash
 - Background.js
 - Splash.css
 - Splash.js
 - SplashButtonOne.js
 - SplashButtonTwo.js
 - TempHeader
 - ProductMenu.js
 - data
 - styles
 - App.test.js
 - index.js

Background.js

```
1 import React, { Component } from 'react';
2 import './Splash.css';
3
4
5 class Background extends Component {
6   constructor() {
7     super();
8     this.state = {
9       pictures: [],
10     };
11   }
12
13   componentDidMount() {
14     fetch('https://randomuser.me/api/?results=500')
15       .then(results => {
16         return results.json();
17       }).then(data => {
18         let pictures = data.results.map((pic) => {
19           return (
20             <div key={pic.results}>
21               <img src={pic.picture.medium} />
22             </div>
23           );
24         });
25         this.setState({pictures: pictures});
26         console.log("state", this.state.pictures);
27       })
28   }
29
30   render() {
31     return (
32       <div className="container2">
33         <div className="container1">
34           {this.state.pictures}
35         </div>
36       </div>
37     )
38   }
39 }
```

constructor

super

set initial state

lifecycle method: fetch + api call

results (usually JSON)

map over data

return:

set key

select what data to return

set the state with this.setState

render the data

src/components/splash/Background.js 1:1

LF UTF-8 JavaScript (JSX) ethanbranch 1 file 2 updates

Peticiones fetch

El uso más simple de `fetch()` toma un argumento — la ruta del recurso que se quiera traer — y devuelve una promesa que contiene la respuesta (un objeto `Response`).

```
var miImagen = document.querySelector('img');
fetch('flores.jpg')
  .then(function(response) {
    return response.blob(); })
  .then(function(myBlob) {
    var objectURL = URL.createObjectURL(myBlob);
    miImagen.src = objectURL;
  });
```