

Estilos en React



Estilos en React (Legacy)

```
1 import React from 'react';
  import './App.css';
   class App extends React.Component {
     render() {
       return (
         <div className="App">
           <h1>Título</h1>
10
       );
11
12 }
  export default App;
```

Vemos que estamos usando 'import' con un archivo .css! Esto sucede gracias a webpack.



Estilos en React

Necesitamos un loader nuevo para poder importar archivos css:

```
module.exports = {
       module:{
           rules:[
                   test:/\.css$/,
10
11
                   use:['style-loader','css-loader']
12
13
14
15
16
17 }
```



Estilos en React

Pros:

- Compatibilidad: Se da estilos igual que antes, se puden reusar los css que ya teníamos!
- No hay que aprender nada nuevo, es el mismo paradigma que antes.

Contras:

- Los estilos son globales. Va en contra de la filosofía de los componentes.
- Tenemos los mismos problemas de organización de CSS que antes.



INLINE STYLING (CSS-in-JS)

```
const divStyle = {
   color: 'blue',
   backgroundImage: 'url(' + imgUrl + ')',
  };

function HelloWorldComponent() {
   return <div style={divStyle}>Hello World!</div>;
}
```

Podemos escribir CSS en JS!

Hacemos un objeto que tena las reglas css, y se lo pasamos al atributo style de un tag.

Esta es la forma de dar estilos que muestra React en su documentación



INLINE STYLING (CSS-in-JS)

Pros:

- Menos configuración: no necesitamos ningún loader.
- Estilos locales: no puede haber colisiones.

Contras:

- Perdemos los pseudoSelectores (hover, etc..)
- La sintaxis es un poco rara!



CSS MODULES

```
1 import React from 'react';
 2 import s from './Product.css';
   function Product(props) {
     return (
      <div className={`${s.producto} ${s.hola}`}>
        <h3 className={s.hola}>{props.title}</h3>
        {props.price}
                                                   10
     );
11
12
13
   export default Product;
14
15
```

La idea atrás de CSS modules es tener lo mejor de los estilos anteriores: Escribir en css propiamente dicho, y mantener scopes locales.

```
1 .producto h3 {
2   background-color: SpringGreen;
3 }
4
5 .producto {
6   color: salmon;
7 }
8
9 .hola {
10   font-size: 30px;
11 }
```



CSS MODULES

```
module.exports = {
           test: /\.css$/,
           use: ['style-loader', {
             loader: 'css-loader',
             options: {
               modules: true,
               localIdentName: '[path][name] [local]--[hash:base64:5]',
10
               camelCase: true,
               ignore: '/node modules/',
11
12
13
14
15
16 };
```

Para implementar CSS-Modules tenemos que agregar este loader, configurado de esta manera.



CSS MODULES

Pros:

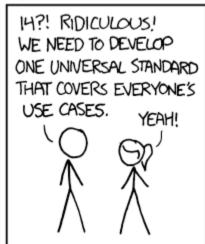
- Componentizado: Los estilos son locales, no puede haber colisiones.
- Estilos locales: Los estilos son locales para cada componente.

Contras:

- Perdemos los estilos globales, de todos modos
- podriamos combinarlo con la primera forma quevimos.

HOW STANDARDS PROLIFERATE: (SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC.)

SITUATION: THERE ARE 14 COMPETING STANDARDS.



500N:

SITUATION: THERE ARE 15 COMPETING STANDARDS.



Styled Components

"The basic idea of styled-components is to enforce best practices by removing the mapping between styles and components."



Pros:

- Componentizado: Creamos Componentes con estilos.
- Reutilizacion: Podemos Reutilizar componentes en vez de estilos.

Contras:

• Nuevo paradigma, hay que acostumbrarse a usarlo.