

HENRY

A bright yellow beam of light originates from the left edge of the frame and points directly at the letter 'R' in the word 'HENRY'. The beam is wider on the left and tapers as it approaches the letter. The word 'HENRY' is rendered in a bold, black, sans-serif font.



Estilos en React



Estilos en React (Legacy)

```
1 import React from 'react';
2 import './App.css';
3
4 class App extends React.Component {
5   render() {
6     return (
7       <div className="App">
8         <h1>Título</h1>
9       </div>
10     );
11   }
12 }
13
14 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:

```
1
2 // // $ npm install --save-dev css-loader style-loader
3
4 module.exports = {
5   ....
6   ....
7   module:{
8     rules:[
9       {
10         test:/\.css$/,
11         use:['style-loader','css-loader']
12       }
13     ]
14   },
15   ....
16   ....
17 }
```



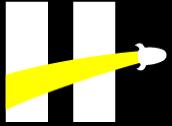
Estilos en React

Pros:

- Compatibilidad: Se da estilos igual que antes, se pueden 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.



< Demo />



INLINE STYLING (CSS-in-JS)

```
1
2 const divStyle = {
3   color: 'blue',
4   backgroundImage: 'url(' + imgUrl + ')',
5 };
6
7 function HelloWorldComponent() {
8   return <div style={divStyle}>Hello World!</div>;
9 }
```

Podemos escribir CSS en JS!

Hacemos un objeto que tenga 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



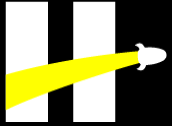
INLINE STYLING (CSS-in-JS)

Pros:

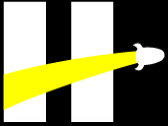
- Menos configuración: no necesitamos ningún loader.
- Estilos locales: no puede haber colisiones.

Contras:

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



< Demo />



CSS MODULES

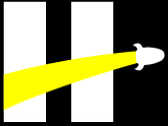


```
1 import React from 'react';
2 import style from './Product.css';
3
4 function Product(props) {
5   return (
6     <div className={` ${style.producto} ${s.hola}`>
7       <h3 className={style.hola}>{props.title}</h3>
8       <p>{props.price}</p>
9     </div>
10  );
11 }
12
13 export default Product;
14
15
```



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

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



CSS MODULES

```
1 module.exports = {
2   ...
3   {
4     test: /\.css$/,
5     use: ['style-loader', {
6       loader: 'css-loader',
7       options: {
8         modules: true,
9         localIdentName: '[path][name]__[local]--[hash:base64:5]',
10        camelCase: true,
11        ignore: '/node_modules/',
12      },
13    }],
14  },
15  ...
16 };
```

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



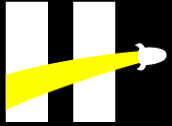
CSS MODULES

Pros:

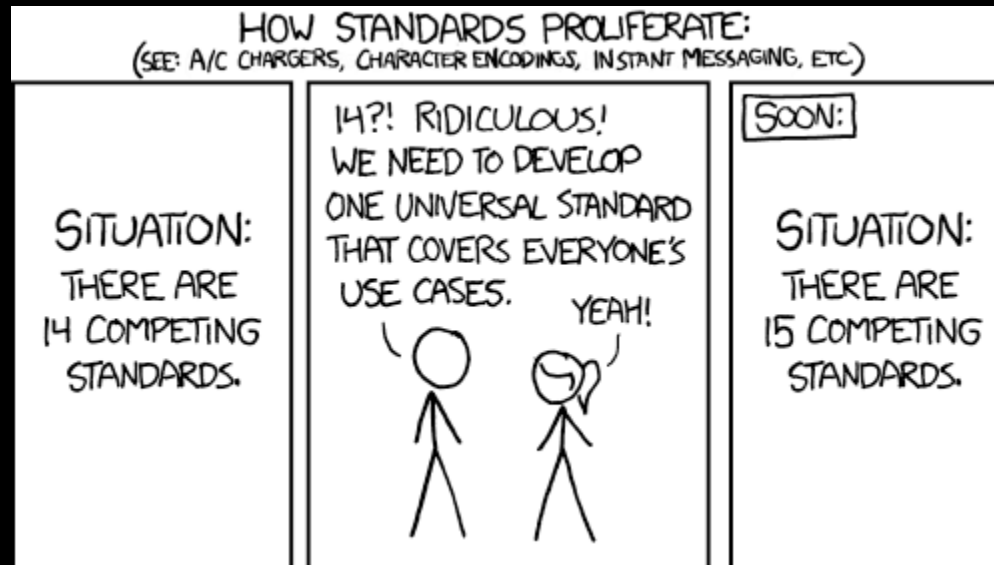
- **Componentizado:** Los estilos son locales, no puede haber colisiones.
- **Estilos locales:** Los estilos son locales para cada componente.

Contras:

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



< Demo />





Styled Components

```
1 import styled from 'styled-components';
2
3 const DivWrapper = styled.div`
4   width: 50%;
5   border: 2px solid black;
6   ${props => (props.color === 'blue') ? `background-color: blue`: null}
7   ${props => (props.color === 'red' ? `background-color: red`: null)}
8 `;
9
10 export default function Component() {
11   return (
12     <DivWrapper>
13       <p>Hello Styled component</p>
14     </DivWrapper>
15   )
16 }
```

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



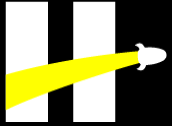
Styled Components

Pros:

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

Contras:

- Nuevo paradigma, hay que acostumbrarse a usarlo.



< Demo />