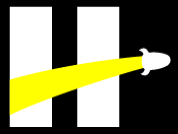
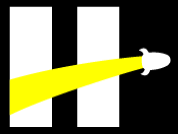

HENRY



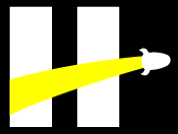
Estilos en React



Estilos en React (Legacy)

```jsx
import React from 'react';
import './App.css';

class App extends React.Component {
  render() {
    return (
      <div className="App">
        <h1>Título</h1>
      </div>
    );
  }
}

export default App;
```

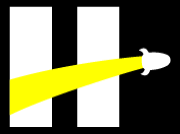
Vemos que estamos usando `import` con un archivo .css! Esto sucede gracias a webpack.



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 }
```



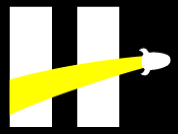
Estilos en React

Pros:

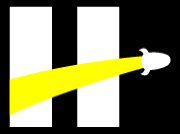
- Compatibilidad: Se da estilos igual que antes, se puden reusar los css que ya teníamos!
- No hay que aprender nada nuevo, es el mismo paradigma que antes.

Contras:

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



< Demo />



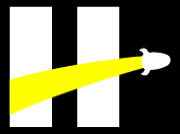
INLINE STYLING (CSS-in-JS)

```
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 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



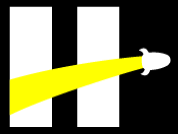
INLINE STYLING (CSS-in-JS)

Pros:

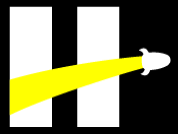
- Menos configuración: no necesitamos ningún loader.
- Estilos locales: no puede haber colisiones.

Contras:

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



< Demo />

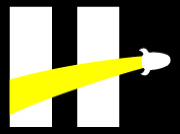


CSS MODULES

```jsx
import React from 'react';
import s from  './Product.css';

function Product(props) {
  return (
    <div className={`${s.producto} ${s.hola}`}>
      <h3 className={s.hola}>{props.title}</h3>
      <p>{props.price}</p>
    </div>
  );
}

export default Product;
```

```css
.producto h3 {
  background-color: SpringGreen;
}

.producto {
  color: salmon;
}

.hola {
  font-size: 30px;
}
```

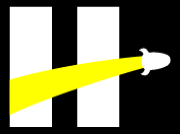
La idea atrás de CSS modules es tener lo mejor de los estilos anteriores: Escribir en css propiamente dicho, y mantener scopes locales.



CSS MODULES

```
module.exports = {
  ...
      {
        test: /\.css$/,
        use: ['style-loader', {
          loader: 'css-loader',
          options: {
            modules: true,
            localIdentName: '[path][name]__[local]--[hash:base64:5]',
            camelCase: true,
            ignore: '/node_modules/',
          },
        }],
      },
  ...
};
```

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



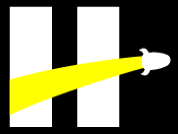
CSS MODULES

Pros:

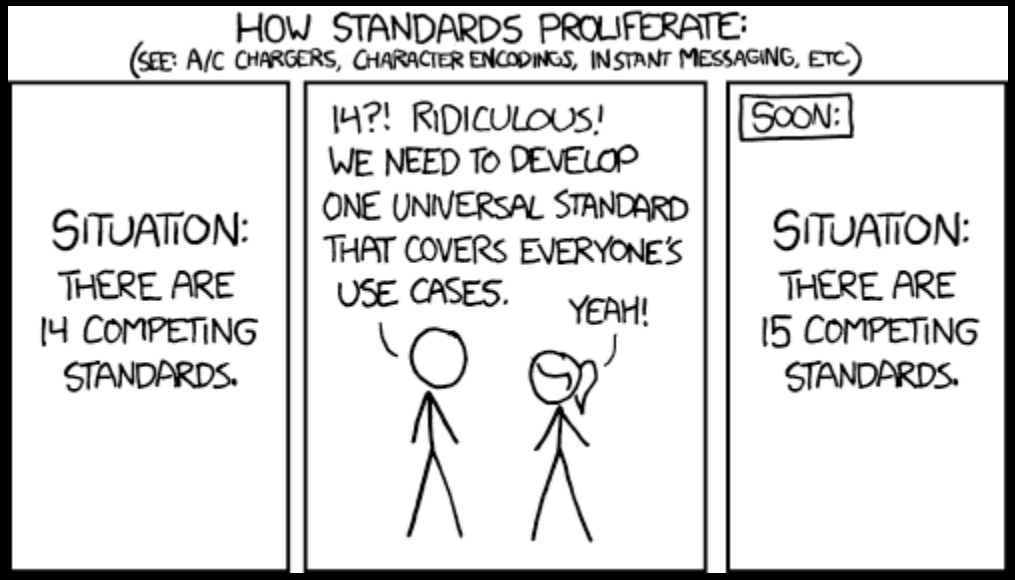
- **Componentizado**: Los estilos son locales, no puede haber colisiones.
- **Estilos locales:** Los estilos son locales para cada componente.

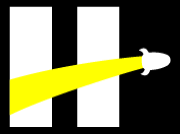
Contras:

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



< Demo />

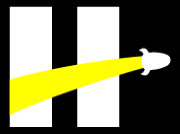

HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.



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.”*



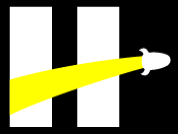
Styled Components

Pros:

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

Contras:

- Nuevo paradigma, hay que acostumbrarse a usarlo.



< Demo />