

Rendering



Inicialización de la aplicación

```
<div id="root"></div> //index.html
```

```
const element = <h1>Hello, world</h1>;
```

```
ReactDOM.render(element, document.getElementById('root'));
```

Inmutabilidad



Los elementos React son Inmutables. Una vez creado no puedes cambiar sus hijos o atributos. Son fotogramas en el tiempo. Para actualizarlo solo conocemos `ReactDOM.render()`

```
function tick() {  
  const element = (  
    <div>  
      <h1>Hello, world!</h1>  
      <h2>It is {new Date().toLocaleTimeString()}.</h2>  
    </div>  
  );  
  ReactDOM.render(element, document.getElementById('root'));  
  
  setInterval(tick, 1000);
```

Diff

React compara cada render con el anterior y sólo aplica las modificaciones que son necesarias

[https://es.reactjs.org/redirect-to-codepen/
rendering-elements/update-rendered-element](https://es.reactjs.org/redirect-to-codepen/rendering-elements/update-rendered-element)

Componentes

- Representan partes de la interfaz de usuario
<https://bradfrost.com/blog/post/atomic-web-design>
- Pueden ser Stateless (Sin estado) o Statefull (Con estado)
- Hay dos formas de crear componentes (funcionales y clases)

Funcionales

```
function Welcome(props) {  
  return <h1>Hello, {props.name}</h1>;  
}
```

Clases

Extender `React.Component` e implementar `render()`

```
class Welcome extends React.Component {  
  render() {  
    return <h1 className="name-text">Hello, {this.props.name}</h1>;  
  }  
}
```

Ojo con los nombres de los atributos como `class => className` => JSX es JS

Como se usan

```
// Paso de parametros como props  
function Welcome(props) {  
  return <h1>Hello, {props.name}</h1>;  
}
```

```
ReactDOM.render(  
  <Welcome name="Sara" />,  
  document.getElementById('root')  
) ;
```

Qué ha pasado?

1. Llamamos a ReactDOM.render() con el elemento
`<Welcome name="Sara" />`.
2. React llama al componente Welcome con `{name: 'Sara'}` como “props”.
3. Nuestro componente Welcome devuelve un elemento
`<h1>Hello, Sara</h1>` como resultado.
4. React DOM actualiza eficientemente el DOM para que coincida con `<h1>Hello, Sara</h1>`.

Dividir componentes

```
function Comment(props) {  
  return (  
    <div className="Comment">  
      <div className="UserInfo">  
        <img className="Avatar"  
          src={props.author.avatarUrl}  
          alt={props.author.name}  
        />  
        <div className="UserInfo-name">{props.author.name}</div>  
      </div>  
      <div className="Comment-text">{props.text}</div>  
      <div className="Comment-date">{formatDate(props.date)}</div>  
    </div>  
  );  
}
```

Avatar

```
function Avatar(props) {  
  return (  
    <img className="Avatar"  
      src={props.user.avatarUrl}  
      alt={props.user.name}  
    />  
  );  
}
```

UserInfo

```
function UserInfo(props) {  
  return (  
    <div className="UserInfo">  
      <Avatar user={props.user} />  
      <div className="UserInfo-name">  
        {props.user.name}  
      </div>  
    </div>  
  );  
}
```

Comment

```
function Comment(props) {  
  return (  
    <div className="Comment">  
      <div className="UserInfo">  
        <UserInfo user={props.author} />  
      </div>  
      <div className="Comment-text">  
        {props.text}  
      </div>  
      <div className="Comment-date">  
        {formatDate(props.date)}  
      </div>  
    </div>  
  );  
}
```

Todos los componentes de
React deben actuar como
funciones puras con respecto
a sus props



Funciones puras

- Los componentes de React deben ser tratados como **funciones puras**. Para los mismos parámetros de entrada obtenemos siempre el mismo resultado visual
- **No podemos cambiar las props que nos llegan como parámetro** (De hecho es una mala práctica en general) porque en ese caso no se obtendrían los resultados esperados.

Comparación

```
// Función pura
function pureSum( a, b ) {
    return a + b;
}

// Impura
function nonPureSum(a) {
    const inputValue = document.querySelector('.inputVal').value;
    a = a + inputValue;
    return result;
}
```

defaultProps

Puedo aplicar valores por defecto a las props en un componente

```
MyComponent.defaultProps = {  
  name: 'Alex',  
  age: 25  
}
```

propTypes

Puedo aplicar ciertas reglas de tipos en tiempo de desarrollo que deben cumplir los componentes

```
ChangeMobile.propTypes = {  
  isAppLoading: PropTypes.bool,  
  initialize: PropTypes.func,  
  translate: PropTypes.func  
}
```

<https://es.reactjs.org/docs/typechecking-with-proptypes.html>