

www.cas-training.com

C/ de la Basílica, 19. 5ª y 8ª planta. 28020 Madrid
Tfno. 91 553 61 62 - Fax 91 598 0346
info@cas-training.com



React



Accredited Partner



Accredited Partner



- Básico
 - HTML y DOM
 - Jerarquía versus Composición de objetos
 - Clases versus funciones
 - Funcion versus [lambda \(función anónima o función flecha\)](#)
- Deseable
 - Modelo de eventos, tópicos y suscriptores
 - En [JS las funciones SON objetos](#)
 - [Javascript Mixins](#)
 - [Javascript pure function](#)
 - Historia de Netscape, IE, JS, EcmaScript, TypeScript,...
 - [La mutación](#), en general, plantea problemas

Que es React



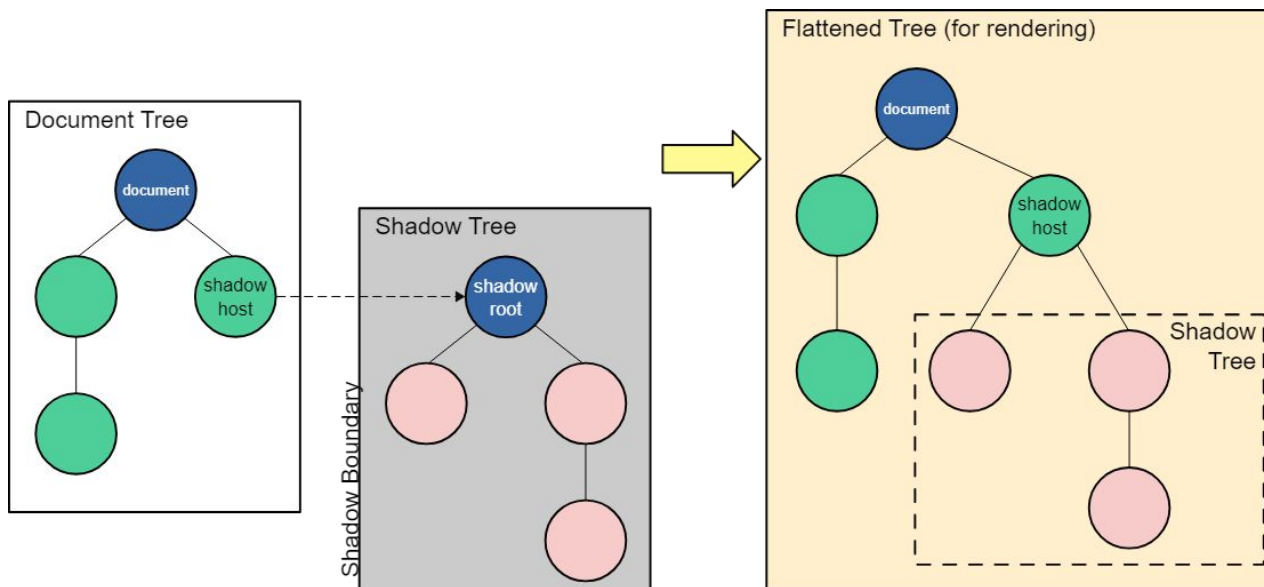
- React es parecido a Angular pero al principio cuesta menos porque tiene menos “de serie”
- React es parecido a librerías MVC (Spring MVC, Django, Laravel, ASPNET MVC) pero en el lado del cliente y la lógica de render va en la vista
- Básicamente intenta hacer que la programación JS+HTML no sea caótica, favoreciendo la estructura y la reutilización
- Está muy enfocado a presentación y la operativa JS tradicional no asociada al DOM no se ve muy afectada
 - En este sentido es diferente a Angular
- Sigue un enfoque simplista y minimalista ([estilo scaffolding](#) automático)

React mantiene un DOM “virtual”

[Link](#)



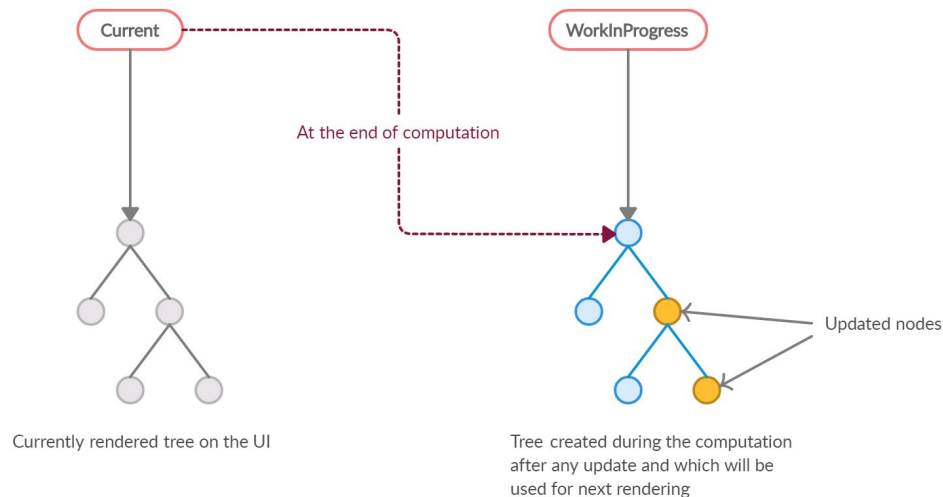
- El DOM Virtual no es lo mismo que el Shadow DOM
 - El Shadow es una [tecnología concreta](#) subyacente a la librería JS que uses
 - El Virtual es una [forma de diseñar una API](#), un patrón de diseño



Que hace React por debajo



- Virtual DOM: Representación ideal en memoria del DOM
- Mount / Unmount: Poner y quitar componentes HTML del DOM
- Reconciliación: Combinar el DOM virtual con el real para visualizar algo
- Render: Pintar la página
- Esto es un rollo y en general no hace falta saber [cómo funciona internamente](#)



Render en React



1. Llamamos a `root.render()` con el elemento `<Welcome name="Sara" />`.
2. React llama al componente de bienvenida con `{nombre: 'Sara'}` como accesorios.
3. Nuestro componente de bienvenida devuelve un elemento `<h1>Hola, Sara</h1>` como resultado.
4. React DOM actualiza eficientemente el DOM para que coincida con `<h1>Hola, Sara</h1>`.

```
function Welcome(props) {  
  return <h1>Hello, {props.name}</h1>;  
}  
  
const root = ReactDOM.createRoot(document.getElementById('root'));  
const element = <Welcome name="Sara" />;  
root.render(element);
```

props



- En general “props” se refiere a las entradas que admiten nuestros elementos de programación en React
- Pueden contener propiedades predefinidas o definidas por el usuario
- Los props son de solo lectura
- Veremos en ejercicios que [se pueden pasar lambdas como props](#)
- En general tendremos en
 - Funciones ([recomendado](#))
 - Clases

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

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

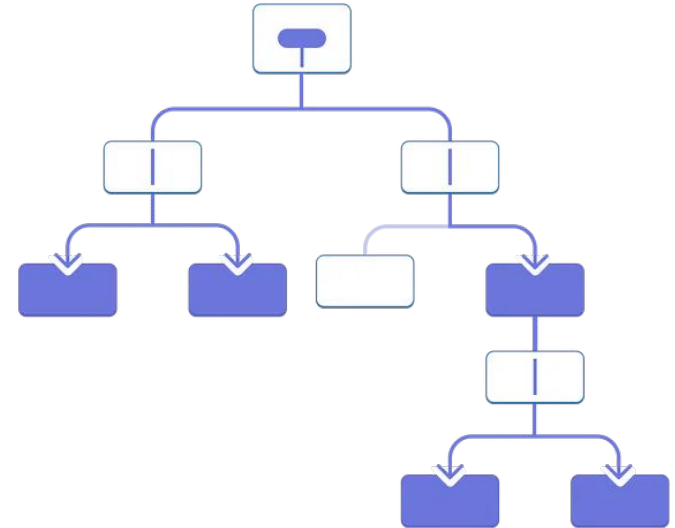
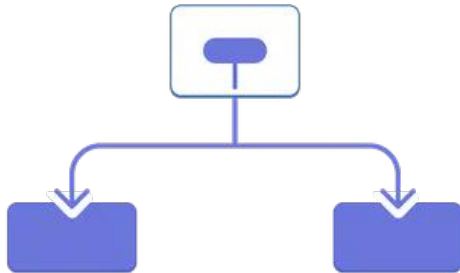
```
function Welcome(props) {  
  return <h1>Hello, {props.name}</h1>;  
}  
  
const root = ReactDOM.createRoot(document.getElementById('root'))  
const element = <Welcome name="Sara" />;  
root.render(element);
```

Props y “pasar props”

Ejemplo



- Es habitual tener que pasar props de abajo hacia arriba (en la composición de objetos) o al revés
- Si esto se hace pesado (mucha profundidad por ejemplo) podemos usar `createContext` y pasarlo “lateralmente”



Components



- React te permite definir componentes como clases o funciones
 - Clases como en P.O.O.
 - Funciones como en programación tradicional o en programación funcional (ambos casos)
- Los componentes definidos como clases proporcionan más funcionalidad
- Para definir una “Component class” de React, extendemos React.Component

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

```
class ShoppingList extends React.Component {  
  render() {  
    return (  
      <div className="shopping-list">  
        <h1>Lista de compras para {this.props.name}</h1>  
        <ul>  
          <li>Instagram</li>  
          <li>WhatsApp</li>  
          <li>Oculus</li>  
        </ul>  
      </div>  
    );  
  }  
}  
  
// Uso de ejemplo: <ShoppingList name="Mark" />
```

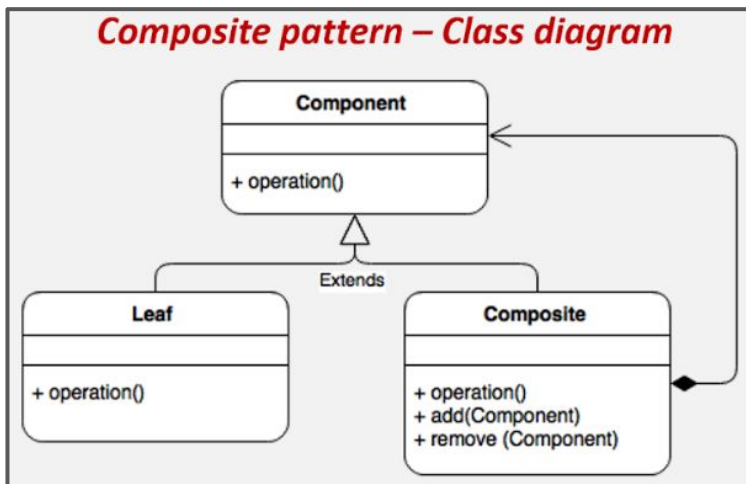
So What About Inheritance?

At Facebook, we use React in thousands of components, and we haven't found any use cases where we would recommend creating component inheritance hierarchies.



Components y jerarquía

- React recomienda no crear tus propias clases de componentes base
 - Si quieres funcionalidad no visual lo extraes a módulos JS aparte
- En los componentes de React, la reutilización de “fragmentos” se logra principalmente mediante la composición en lugar de la herencia



Components y composición



- La composición es más dinámica que la jerarquía
- En este ejemplo
 - WelcomeDialog
 - wrapea (envuelve) a
 - FancyBorder
- Un objeto (visual) dentro de un objeto (visual)
- Vemos una correlación entre
 - etiqueta HTML
 - Funcion / Clase

```
function FancyBorder(props) {
  return (
    <div className={'FancyBorder FancyBorder-' + props.color}>
      {props.children}
    </div>
  );
}

function WelcomeDialog() {
  return (
    <FancyBorder color="blue">
      <h1 className="Dialog-title">
        Welcome
      </h1>
      <p className="Dialog-message">
        Thank you for visiting our spacecraft!
      </p>
    </FancyBorder>
  );
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<WelcomeDialog />);
```

Components como funciones y objetos de funciones



- Los Components pueden ser funciones
 - Piensa en programación funcional (lambdas)
- En este caso
 - `<SplitPane />` es un objeto
 - `<Contacts />` debe ser otro (no declarado en pantalla)
 - `<Chat />` debe ser otro



how is an object of a function implemented

- La forma más sencilla de entender esto es pensar en una clase con un solo método
 - Internamente es más “ligero” pero más complicado de entender si miras las tripas

```
function SplitPane(props) {  
  return (  
    <div className="SplitPane">  
      <div className="SplitPane-left">  
        {props.left}  
      </div>  
      <div className="SplitPane-right">  
        {props.right}  
      </div>  
    </div>  
  );  
}  
  
function App() {  
  return (  
    <SplitPane  
      left={  
        <Contacts />  
      }  
      right={  
        <Chat />  
      } />  
  );  
}
```

Composición de objetos clase con objetos función



- En este caso hacemos un objeto clase que encapsula un objeto Dialog
- Puede ser de cualquier forma
 - Funcion → Clase
 - Clase → Funcion
 - O entre elementos del mismo tipo

```
function Dialog(props) {  
  return (  
    <FancyBorder color="blue">  
      <h1 className="Dialog-title">  
        {props.title}  
      </h1>  
      <p className="Dialog-message">  
        {props.message}  
      </p>  
      {props.children}  
    </FancyBorder>  
  );  
}
```

```
class SignUpDialog extends React.Component {  
  constructor(props) {  
    super(props);  
    this.handleChange = this.handleChange.bind(this);  
    this.handleSignUp = this.handleSignUp.bind(this);  
    this.state = {login: ''};  
  }  
  
  render() {  
    return (  
      <Dialog title="Mars Exploration Program"  
        message="How should we refer to you?">  
        <input value={this.state.login}  
          onChange={this.handleChange} />  
        <button onClick={this.handleSignUp}>  
          Sign Me Up!  
        </button>  
      </Dialog>  
    );  
  }  
  
  handleChange(e) {  
    this.setState({login: e.target.value});  
  }  
}
```

Components → Fragments



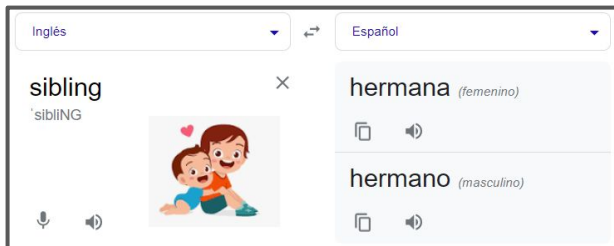
- Fragmento es un mecanismo en React que te permite recuperar varios componentes tirando de uno solo sin añadir “nada más” al DOM que los elementos concretos de cada elemento
- Puede parecer verbose pero está comprobado que mejora la reutilización
- Se entiende mejor con un ejemplo o con un símil con el DIV

```
return (  
  <>  
    <Post  
    <Post  
  </>  
)
```

A black arrow points from the right towards the opening fragment tag '<>' in the code snippet, highlighting its use to group multiple components.

Components → Fragments

Fragments are useful because grouping elements with a Fragment has no effect on layout or styles, unlike if you wrapped the elements in another container like a DOM element. If you inspect this example with the browser tools, you'll see that all `<h1>` and `<article>` DOM nodes appear as siblings without wrappers around them:



```
▼ <div id="root"> == $0
  <h1>An update</h1>
  ▼ <article>
    <p>It's been a while since I posted...</p>
  </article>
  <h1>My new blog</h1>
  ▼ <article>
    <p>I am starting a new blog!</p>
  </article>
</div>
```

```
1 export default function Blog() {
2   return (
3     <>
4       <Post title="An update" body=
5       <Post title="My new blog" bo
6     </>
7   )
8 }
9
10 function Post({ title, body }) {
11   return (
12     <>
13       <PostTitle title={title} />
14       <PostBody body={body} />
15     </>
16   );
17 }
18
19 function PostTitle({ title }) {
20   return <h1>{title}</h1>
21 }
22
23 function PostBody({ body }) {
24   return (
25     <article>
26       <p>{body}</p>
27     </article>
28   );
29 }
```

Components → Fragments



- La sintaxis de un fragment es sencillamente `<>` y `</>`

```
return (  
  <>  
  <Post  
  <Post  
  </>  
)
```

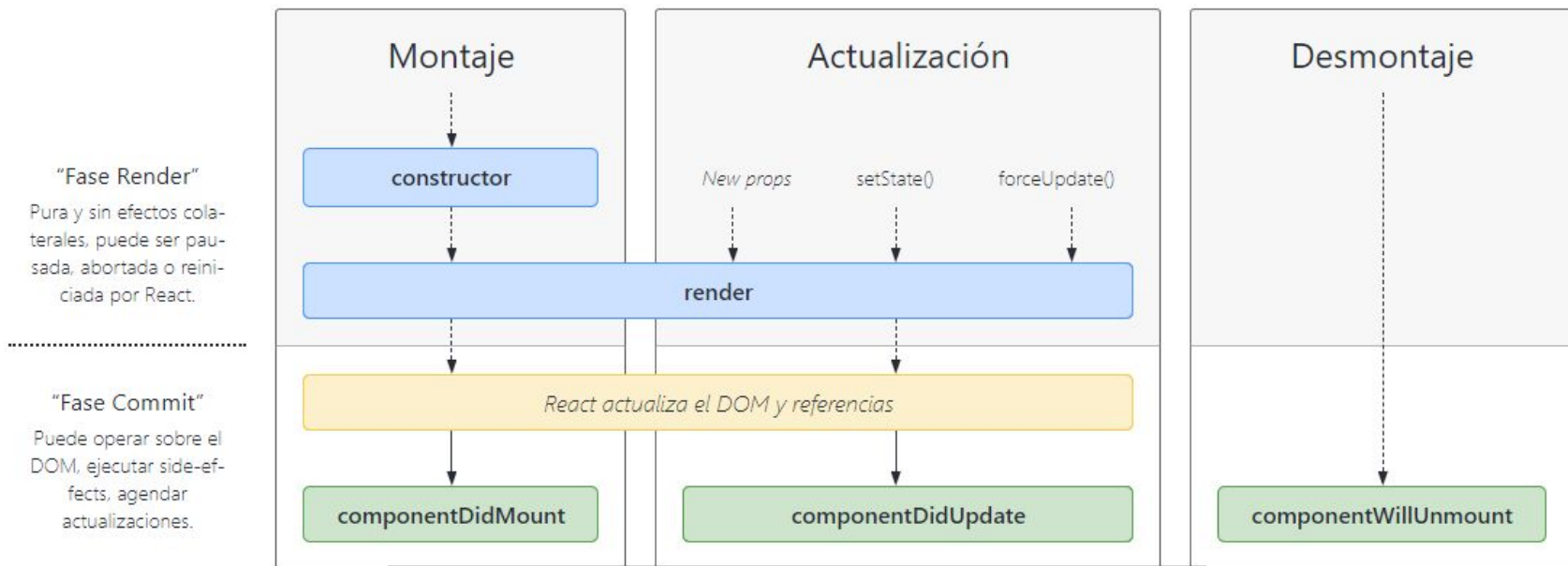
- Si usas un `div`, el `div` se añade al DOM, con un fragmento no

```
function TableData () {  
  return (  
    <div>  
      <td>Eat</td>  
      <td>Learn</td>  
      <td>Code</td>  
    </div>  
  );  
}
```

```
function Table () {  
  return (  
    <table>  
      <tr>  
        <TableData />  
      </tr>  
    </table>  
  );  
}
```


Component lifecycle

Link



what is mounting in react



For a component to be mounted in ReactJS it means to be initialized and inserted in the DOM. Mounting is the initial phase in which the instance of the component is created and inserted into the DOM. When the component gets successfully inserted into the DOM, the component is said to be mounted. 2 nov 2023

Component lifecycle

- **componentDidMount()** se invoca inmediatamente después de insertar un componente en el árbol.
- La inicialización que **requiere nodos DOM** debe ir aquí.
- Si necesita **cargar datos desde un punto final remoto**, este es un buen lugar para crear una instancia de la solicitud de red.
- Este método es un buen lugar para configurar suscripciones.
- Puede llamar a `setState()` inmediatamente en `componentDidMount()`. Activará un `render()` adicional, pero sucederá antes de que el navegador actualice la pantalla. Esto garantiza que aunque se llamará `render()` dos veces en este caso, el usuario no verá el estado intermedio.

El ciclo de vida es complicado cuando buceas en él

```
class Example extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0
    };
  }

  componentDidMount() {
    document.title = `You clicked ${this.state.count} times`;
  }

  componentDidUpdate() {
    document.title = `You clicked ${this.state.count} times`;
  }

  render() {
    return (
      <div>
        <p>You clicked {this.state.count} times</p>
        <button onClick={() => this.setState({ count: this.state.count + 1 })}>
          Click me
        </button>
      </div>
    );
  }
}
```

Component lifecycle



- `componentDidUpdate()` debe comprobar si hubo cambios en el retorno al llamar al servidor o podría entrar en bucle infinito al actualizar el DOM

```
componentDidUpdate(prevProps) {  
  // Typical usage (don't forget to compare props):  
  if (this.props.userID !== prevProps.userID) {  
    this.fetchData(this.props.userID);  
  }  
}
```

Ejemplo de
problema con
AJAX

Montaje

constructor

"Fase Render"

Pura y sin efectos colaterales, puede ser pausada, abortada o reiniciada por React.

"Fase Pre-commit"

Puede leer el DOM.

"Fase Commit"

Puede operar sobre el DOM, ejecutar side-effects, agendar actualizaciones.

componentDidMount

Actualización

New props

setState()

forceUpdate()

static getDerivedStateFromProps

shouldComponentUpdate

render

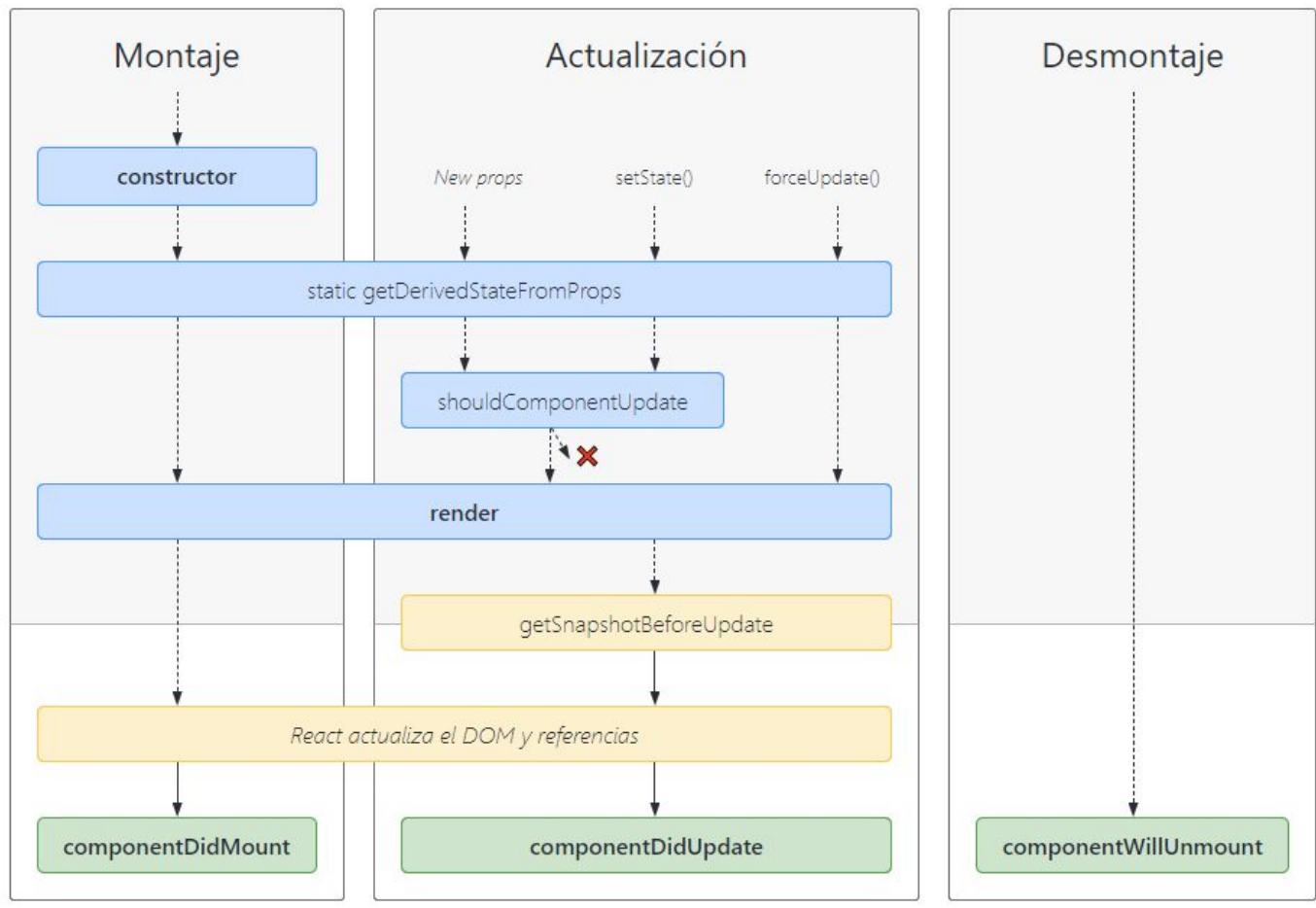
getSnapshotBeforeUpdate

React actualiza el DOM y referencias

componentDidUpdate

Desmontaje

componentWillUnmount



Lógica stateless

- Si la lógica no tiene estado está recomendado usar funciones “normales”
- Este código contiene una llamada AJAX async-await ([¿sync o async?](#))

```
import { useEffect, useState } from 'react';
import './App.css';
function App() {
  const makeAPICall = async () => {
    try {
      const response = await fetch('http://localhost:8080/', {mode: 'cors'});
      const data = await response.json();
      console.log({ data })
    }
    catch (e) {
      console.log(e)
    }
  }
  useEffect(() => {
    makeAPICall();
  }, [])
  return (
    <div className="App">
      <h1>React Cors Guide</h1>
    </div>
```

[Link](#)

Hooks y estado



- A veces creamos una clase con la única finalidad de contener un dato o un par de datos
 - Con una función sólo no tendríamos estado
- Podemos usar hooks para esto

```
class Example extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = {  
      count: 0  
    };  
  }  
}
```

```
import React, { useState } from 'react';  
  
function Example() {  
  // Declare a new state variable, which we'll call "count"  
  const [count, setCount] = useState(0);  
}
```

Hooks - useState



- Los hooks permiten introducir un comportamiento que es todavía más minimalista que una función y necesita guardar estado entre llamadas

```
import React, { useState } from 'react';

function Example() {
  // Declara una nueva variable de estado, la cual llamaremos "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

```
// Declara una nueva variable de estado, que llamaremos "count".
const [count, setCount] = useState(0);
```

useState (un hook) devuelve una tupla de variable/función. Técnicamente es un [destructuring assignment](#)



Hooks - useState

- Se pueden crear múltiples hooks en una función
- El tipo puede ser simple o complejo
- Se usan fuera de clases (los objetos tienen su propio estado más complejo)

```
function ExampleWithManyStates() {  
  // Declarar múltiples variables de estado!  
  const [age, setAge] = useState(42);  
  const [fruit, setFruit] = useState('banana');  
  const [todos, setTodos] = useState([ { text: 'Learn Hooks' } ]);  
  // ...  
}
```

Hooks are functions that let you “hook into” React state and lifecycle features from function components. Hooks don’t work inside classes — they let you use React without classes.

Hooks - useState



Equivalencia hook - clase

```
import React, { useState } from 'react';

function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);


  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

```
class Example extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0
    };
  }

  render() {
    return (
      <div>
        <p>You clicked {this.state.count} times</p>
        <button onClick={() => this.setState({ count: this.state.count + 1 })}>
          Click me
        </button>
      </div>
    );
  }
}
```

Hooks y efectos

- `useEffect()` es similar a los métodos de ciclo de vida del componente “tipo clase” pero para metodos
 - Más simple
 - Requiere más código procedimental para distinguir qué lo disparó
- En este ejemplo actualizamos el `<title>` de la página además del fragmento de la página



```
import React, { useState, useEffect } from 'react';

function Example() {
  const [count, setCount] = useState(0);

  // Similar to componentDidMount and componentDidUpdate:
  useEffect(() => {
    // Update the document title using the browser API
    document.title = `You clicked ${count} times`;
  });

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

Hooks - Effects

- Se puede hacer la liberación de recursos (cleanUp) programáticamente

```
import React, { useState, useEffect } from 'react';

function FriendStatus(props) {
  const [isOnline, setIsOnline] = useState(null);

  useEffect(() => {
    function handleStatusChange(status) {
      setIsOnline(status.isOnline);
    }
    ChatAPI.subscribeToFriendStatus(props.friend.id, handleStatusChange);
    // Specify how to clean up after this effect:
    return function cleanup() {
      ChatAPI.unsubscribeFromFriendStatus(props.friend.id, handleStatusChange);
    };
  });

  if (isOnline === null) {
    return 'Loading...';
  }
  return isOnline ? 'Online' : 'Offline';
}
```



Hooks - Effects



Vamos a ver [un caso completo](#) y a diagramarlo

```
function useChatRoom({ serverUrl, roomId }) {  
  useEffect(() => {  
    const options = {  
      serverUrl: serverUrl,  
      roomId: roomId  
    };  
    const connection = createConnection(options);  
    connection.connect();  
    return () => connection.disconnect();  
  }, [roomId, serverUrl]);  
}
```

Custom Hooks



- Un custom hook usa lo que hemos visto con `useState()` y/o `useEffect()` para facilitar su reutilización

```
import React, { useState, useEffect } from 'react';

function useFriendStatus(friendID) {
  const [isOnline, setIsOnline] = useState(null);

  function handleStatusChange(status) {
    setIsOnline(status.isOnline);
  }

  useEffect(() => {
    ChatAPI.subscribeToFriendStatus(friendID, handleStatusChange);
    return () => {
      ChatAPI.unsubscribeFromFriendStatus(friendID, handleStatusChange);
    };
  });

  return isOnline;
}
```

Hooks - Limitaciones



- Solo se puede llamar a Hooks en el nivel superior
 - No dentro de bucles o ifs
- Solo se puede hacer Hooks desde los componentes de React

JSX



- Opcionalmente podemos usar una sintaxis especial de JS llamada JSX que facilita la legibilidad y programación de código con React
- Usa notación JS para acceder al DOM (en vez de propiedades HTML)
 - `className="..."` en vez de `class="..."`
- Requiere de un proceso de compilación a JS hecho con [Babel](#)

```
function getGreeting(user) {  
  if (user) {  
    return <h1>Hello, {formatName(user)}!</h1>;  
  }  
  return <h1>Hello, Stranger.</h1>;  
}
```

- React rompe la separación de lógicas de negocio y de render
 - React embraces the fact that rendering logic is inherently coupled with other UI logic: how events are handled, how the state changes over time, and how the data is prepared for display.
 - Instead of artificially separating technologies by putting markup and logic in separate files, React separates concerns with loosely coupled units called “components” that contain both.
- ¿Cómo cuadra esto con
 - JS a mano
 - Java y JSP
 - C# y ASP.NET
 - PHP a mano
 - Laravel
 - Django en Python?

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


JSX



- Entre { y } ponemos expressions
- Esto recuerda mucho a los mixins de JS de toda la vida [pero tiene ventajas](#)

```
const name = 'Josh Perez';  
const element = <h1>Hello, {name}</h1>;
```

```
function formatName(user) {  
  return user.firstName + ' ' + user.lastName;  
}  
  
const user = {  
  firstName: 'Harper',  
  lastName: 'Perez'  
};  
  
const element = (  
  <h1>  
    Hello, {formatName(user)}!  
  </h1>  
)
```

Tipos en React con JSX



- La sintaxis de React es tan intuitiva como es posible

```
const element = (  
  <h1 className="greeting">  
    Hello, world!  
  </h1>  
);
```


```
const element = React.createElement(  
  'h1',  
  {className: 'greeting'},  
  'Hello, world!'  
);
```

- Vamos a ver dos tutoriales que explican cómo se programa esto:
 - [Clock como ejemplo de comportamiento estático y pasar a gestionar estado](#) (old)
 - [Thinking in React como ejemplo de composition tipada](#) (ultima versión de la doc)
- Analicemos ambos ejemplos y enlacemos el comportamiento global

Binds y eventos

- El bind hace que el callback sepa contra quien tiene que actuar
 - Es un poco verbose pero necesario
- Este tipo de operativa se puede usar para sincronizar objetos compuestos en un objeto envolvente
 - No es herencia

```
class Toggle extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = {isToggleOn: true};  
  
    // Este enlace es necesario para hacer que `this` funcione en el callback  
    this.handleClick = this.handleClick.bind(this);  
  }  
  
  handleClick() {  
    this.setState(prevState => ({  
      isToggleOn: !prevState.isToggleOn  
    }));  
  }  
  
  render() {  
    return (  
      <button onClick={this.handleClick}>  
        {this.state.isToggleOn ? 'ON' : 'OFF'}  
      </button>  
    );  
  }  
}
```



¿Qué hace este código?

[Link](#)



```
function NumberList(props) {  
  const numbers = props.numbers;  
  const listItems = numbers.map((number) =>  
    <li>{number}</li>  
  );  
  return (  
    <ul>{listItems}</ul>  
  );  
}  
  
const numbers = [1, 2, 3, 4, 5];  
const root = ReactDOM.createRoot(document.getElementById('root'));  
root.render(<NumberList numbers={numbers} />);
```

Prevención de envío de formularios



```
handleChange(event) {  
  this.setState({value: event.target.value});  
}  
  
handleSubmit(event) {  
  alert('A name was submitted: ' + this.state.value);  
  event.preventDefault();  
}  
  
render() {  
  return (  
    <form onSubmit={this.handleSubmit}>  
      <label>  
        Name:  
        <input type="text" value={this.state.value} onChange={this.handleChange} />  
      </label>  
      <input type="submit" value="Submit" />  
    </form>  
  );  
}
```

Link forwarding



- Este snippet permite que componentes que usen FancyButton puedan acceder al anchor del button subyacente
 - Por ejemplo para modificarlo

```
const FancyButton = React.forwardRef((props, ref) => (  
  <button ref={ref} className="FancyButton">  
    {props.children}  
  </button>  
));  
  
// You can now get a ref directly to the DOM button:  
const ref = React.createRef();  
<FancyButton ref={ref}>Click me!</FancyButton>;
```

Portals



- Permiten componer elementos hijos en su propio contexto visual (ajeno al del padre)
- Requiere un poco de CSS

App.js NoPortalExample.js PortalExample.js ModalContent.js

```
1 import NoPortalExample from './NoPortalExample';
2 import PortalExample from './PortalExample';
3
4 export default function App() {
5   return (
6     <>
7       <div className="clipping-container">
8         <NoPortalExample />
9       </div>
10      <div className="clipping-container">
11        <PortalExample />
12      </div>
13    </>
14  );
15 }
```



```
.modal {
  display: flex;
  justify-content: space-between;
  align-items: center;
  box-shadow: 0 10px 10px rgba(100, 100, 100, 0.5);
  background-color: white;
  border: 2px solid #007bff;
  border-radius: 12px;
  position: absolute;
  width: 250px;
  top: 70px;
  left: calc(50% - 125px);
  bottom: 70px;
}
```

Best practices con JSX



- Se consciente de si tus funciones son “puras” o tienen estado

- Mejor si no lo tienen, el estado es un rollo

- Usa operadores ternarios si es apropiado

```
return role === ADMIN ? <AdminUser /> : <NormalUser />
```

- Se consistente en el uso de funciones

- Funciones normales
- Funciones flecha
- Funciones expresión

```
1
2 // arrow function
3
4 const Test = () => {
5   | return (<h1> This is a test component </h1>)
6 }
7
8 export default Test
```

```
2
3
4 function Test(){
5   | return (<h1> This is a test component </h1>)
6 }
7
8 export default Test
```

```
1
2 // function expression
3
4 const Test = function () {
5   | return (<h1> This is a test component </h1>)
6 }
7
8 export default Test
```


Testing



- Disponemos de diferentes métodos de `setup()` y `tearDown()`
- La librería más usada para testing es Jest
 - Se suele lanzar desde `npm test`

```
PS npm test

> learning-jest@1.0.0 test
> jest

PASS ./index.test.js
  FizzBuzz
    ✓ [3] should result in "fizz" (2 ms)
    ✓ [5] should result in "buzz" (1 ms)
    ✓ [15] should result in "fizzbuzz" (1 ms)
    ✓ [1,2,3] should result in "1, 2, fizz" (1 ms)

Test Suites: 1 passed, 1 total
Tests:      4 passed, 4 total
Snapshots:  0 total
Time:       0.586 s, estimated 1 s
Ran all test suites.
```

Testing



```
// hello.test.js

import React from "react";
import { render, unmountComponentAtNode } from "react-dom";
import { act } from "react-dom/test-utils";

import Hello from "../hello";

let container = null;
beforeEach(() => {
  // setup a DOM element as a render target
  container = document.createElement("div");
  document.body.appendChild(container);
});

afterEach(() => {
  // cleanup on exiting
  unmountComponentAtNode(container);
  container.remove();
  container = null;
});
```

```
it("renders with or without a name", () => {
  act(() => {
    render(<Hello />, container);
  });
  expect(container.textContent).toBe("Hey, stranger");

  act(() => {
    render(<Hello name="Jenny" />, container);
  });
  expect(container.textContent).toBe("Hello, Jenny!");

  act(() => {
    render(<Hello name="Margaret" />, container);
  });
  expect(container.textContent).toBe("Hello, Margaret!");
});
```

Testing y Mocking



- Una parte importante en testing es el mocking
- El mocking más específico de React es el mocking del navegador
 - Jest usa [jsdom](#) para simular el navegador sin arrancar uno de verdad

```
const dom = new JSDOM(`<!DOCTYPE html><p>Hello world</p>`);  
console.log(dom.window.document.querySelector("p").textContent); // "Hello world"
```

- Hay [algunas críticas](#) sobre [las limitaciones de este approach](#) de testing y se suele recomendar un head-less browser en su lugar
 - Una cosa es testear motor JS, otra cosa testear el DOM entero + JS



dom versus js engine

Error boundaries

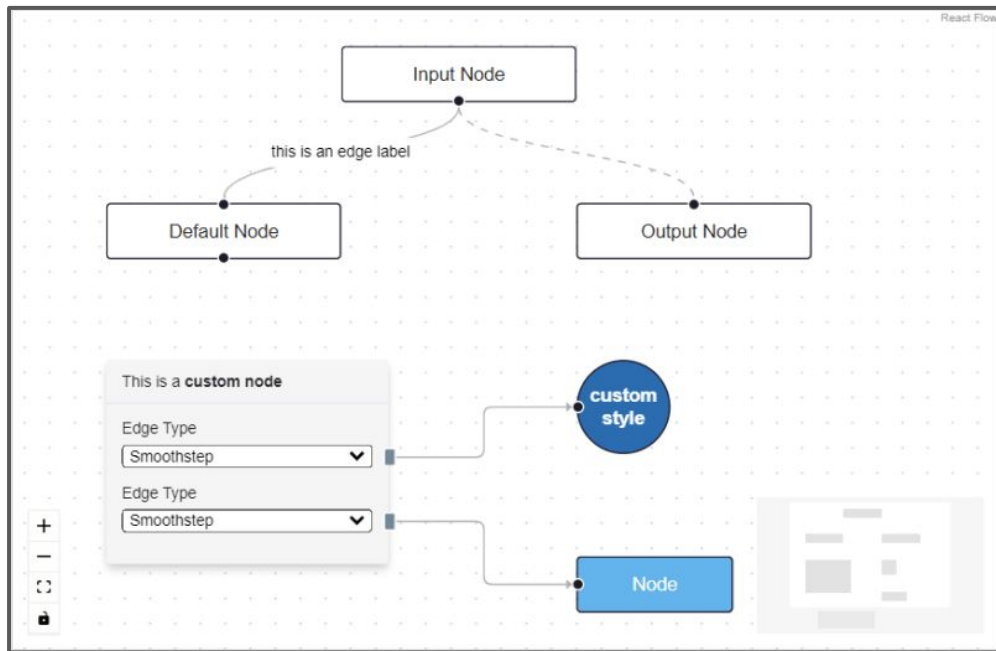
- Si una parte de la página no renderiza bien por problemas de programación, React no la visualiza
- Podemos rodear nuestros components con “error boundaries”
- Se recomienda no abusarlo

```
class ErrorBoundary extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = { hasError: false };  
  }  
  
  static getDerivedStateFromError(error) {  
    // Update state so the next render will show the fallback UI.  
    return { hasError: true };  
  }  
  
  componentDidCatch(error, info) {  
    // Example "componentStack":  
    //   in ComponentThatThrows (created by App)  
    //   in ErrorBoundary (created by App)  
    //   in div (created by App)  
    //   in App  
    logErrorToMyService(error, info.componentStack);  
  }  
}
```


Diagramado



- Hay varias librerías que permiten diagramar proyectos basados en React
 - [react-diagrams](#) y [react-hooks](#)
 - [ReactFlow](#)
 - [projectstorm/react-diagrams](#)
- Ninguna hace descubrimiento de código (AFAIK), siempre es aparte del código



Temas adicionales

- [Strict Mode](#)
- [Puedes usar el AJAX que quieras](#)
 - Y puedes tenerlo con o sin estado obviamente 
- No hay un criterio estricto de [organización de ficheros](#)
- ¿Como continuar?
 - [Iterar sobre arrays en JSX](#)
 - [React-router](#)
 - [React-native \(moviles\)](#)
 - [Añadir atributos condicionalmente](#)
 - ...

```
componentDidMount() {  
  fetch("https://api.example.com/items")  
    .then(res => res.json())  
    .then(  
      (result) => {  
        this.setState({  
          isLoading: true,  
          items: result.items  
        });  
      },  
      // Note: it's important to handle errors here  
      // instead of a catch() block so that we don't swallow  
      // exceptions from actual bugs in components.  
      (error) => {  
        this.setState({  
          isLoading: true,  
          error  
        });  
      }  
    )  
}
```