

#>/<>

HACK A BOSS

<CODE YOUR TALENT>

HISTORIA 01

#>/<>

HACK A BOSS

<CODE YOUR TALENT>

WEB



¿Qué es un servidor? ¿Y un servicio?



¿Qué es un cliente?



¿Qué es Hipertexto?



¿Qué es HTML?



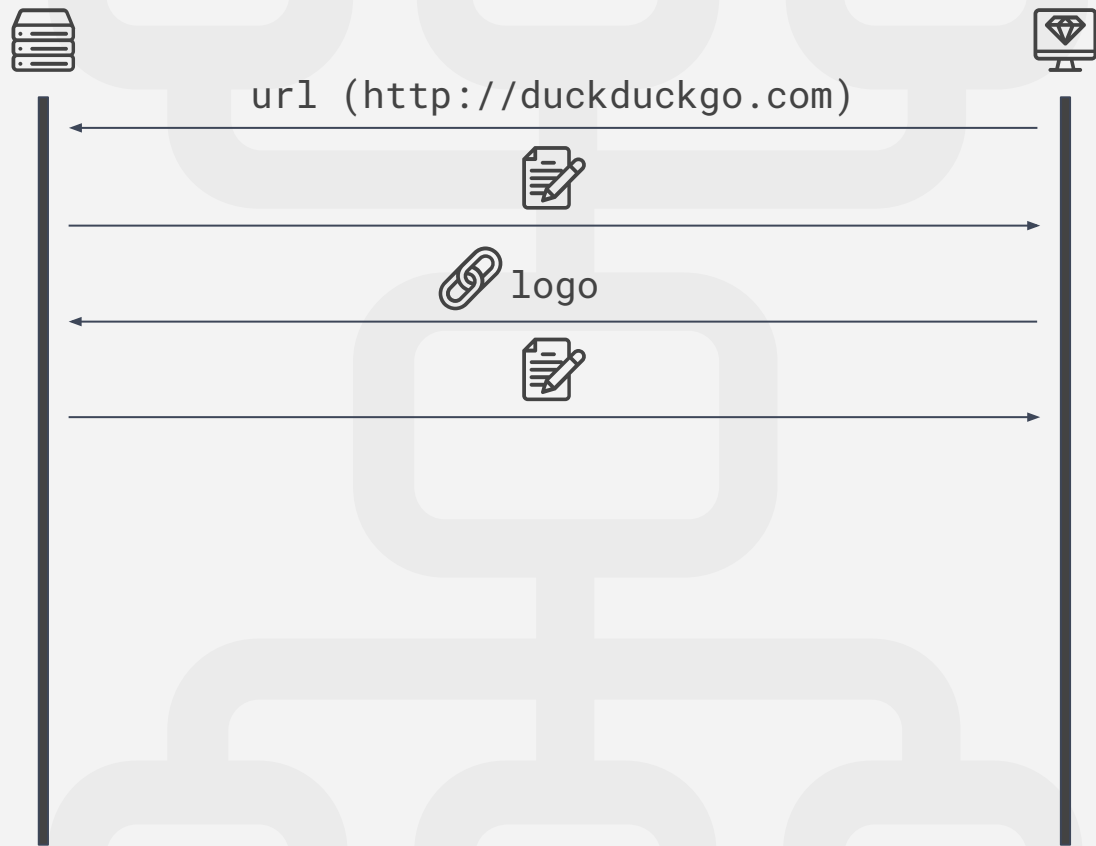
¿Qué es HTTP?

#>/<>

HACK A BOSS

<CODE YOUR TALENT>

WEB



#>/<>

HACK A BOSS

<CODE YOUR TALENT>

¿Qué sucede al hacer click en un enlace?

- Se realiza una nueva petición
- Se cargan todos los documentos enlazados
 - Otra vez!
- Se refresca la página

¿Es esto rápido?

- No demasiado
- Caché del navegador

HISTORIA DE JAVASCRIPT

Inicios

- Netscape (1995)
 - Objetivo: Añadir interactividad
- ECMAScript Standard (1996)



- ECMAScript 5 (2009)

HISTORIA DE JS

Browser Wars



VS



#>/<>

HACK A BOSS

<CODE YOUR TALENT>

HISTORIA DE JS

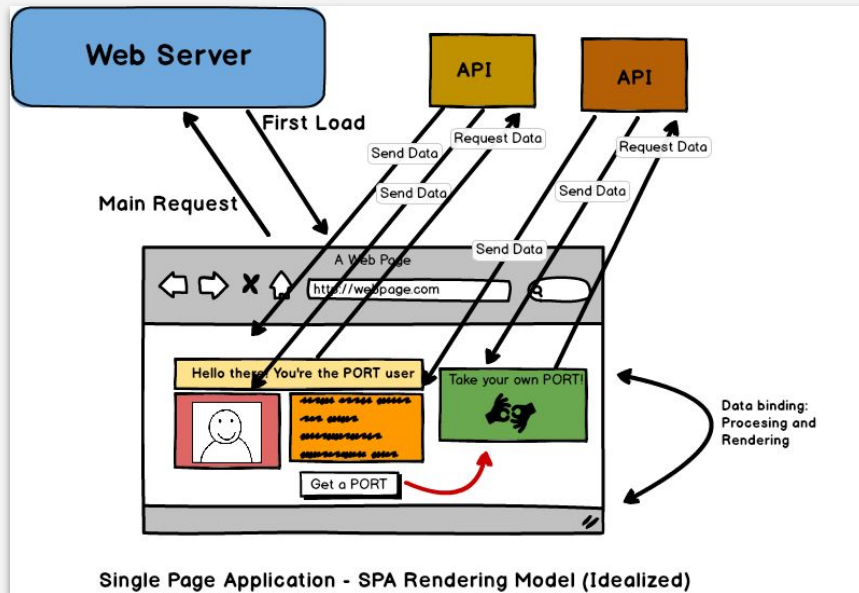
Evolución

- XMLHttpRequest (2000)
 - AJAX (Asynchronous JS and XML)
- jQuery (2006)
- Google Chrome (2008)
- Motor V8 (Open Source)
- Node.js (2009)

HISTORIA DE JS

SPA: Single Page Application

- Precarga o carga dinámica
- Nunca se recarga
- Todavía teórico (2000 - 2010)



#>/<>

HACK A BOSS
<CODE YOUR TALENT>

HISTORIA DE JS

Frameworks

- Frameworks: backbone, ember
- angularJS (2010):
 - "declarative", decoupled
 - two-way binding
- MEAN stack
 - Como contraposición a LAMP

#>/<>

HACK A BOSS

<CODE YOUR TALENT>

HISTORIA DE JS

Frameworks “modernos”

- Angular (2016) [Google]
- Vue (2014) [usado por Alibaba]
- React (2013) [Facebook]

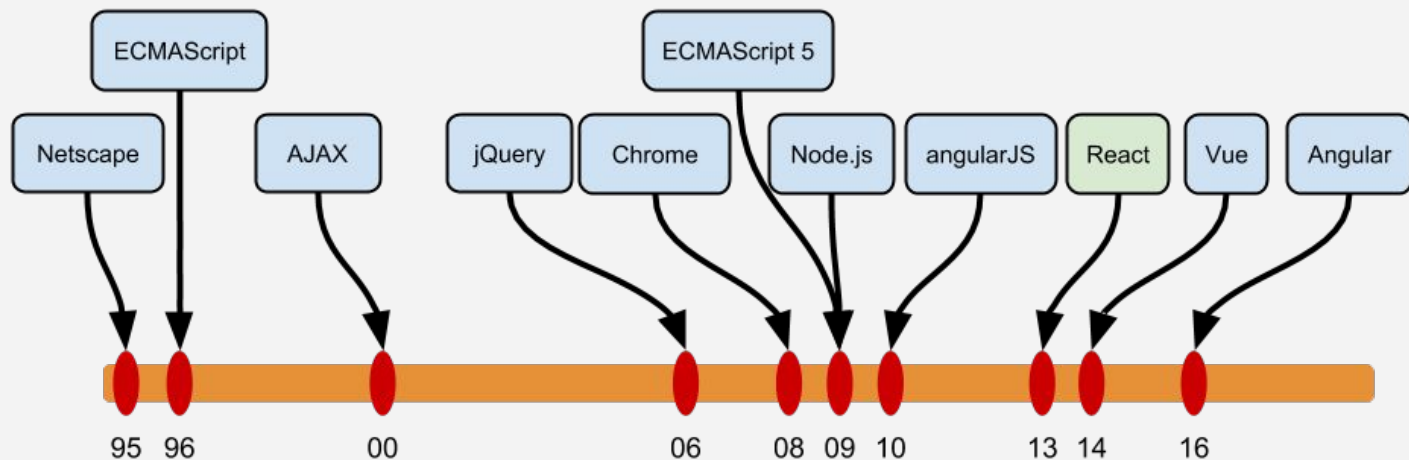
#>/<>

HACK A BOSS

<CODE YOUR TALENT>

HISTORIA DE JS

Timeline



#>/<>

HACK A BOSS

<CODE YOUR TALENT>

02

PREVIO

#>/<>

HACK A BOSS

<CODE YOUR TALENT>

JAVASCRIPT NECESARIO

ES6

- Template Strings
- Shorthand property names
- Arrow Functions
- Parameter defaults
- spread & rest operators
- destructuring assignment
- ES Modules
- Promesas y async/await

#>/<>

HACK A BOSS

<CODE YOUR TALENT>

JAVASCRIPT NECESARIO

ES6 (II)

- Operador ternario
- Métodos de array
 - map
 - filter
 - reduce
 - find
 - includes
- Nullish coalescing operator
- Optional chaining

#>/<>

HACK A BOSS

<CODE YOUR TALENT>

Template Strings

- Strings normales, con superpoderes

```
const greeting = 'Hello'
const subject = 'World'
console.log(`${greeting} ${subject}!`) // Hello World!
// lo mismo que:
console.log(greeting + ' ' + subject + '!')
```


Shorthand Property names

- Al crear un objeto, solamente si una propiedad y su valor tienen el mismo nombre, se puede omitir el valor

```
const a = 'hello'  
const b = 42  
const c = {d: [true, false]}  
console.log({a, b, c})  
// es lo mismo que:  
console.log({a: a, b: b, c: c})
```

Arrow Functions

- (param) => valor
- (param) => { bloque con return }

```
let lista = [1, 2, 3];
```

```
function doubleA(a){  
  return 2 * a;  
}  
  
let doubleB = function(a){  
  return 2 * a;  
}  
  
let listaDoble = lista.map(  
  function(a){  
    return 2 * a;  
  }); // [2, 4, 6]
```

```
let doubleA = (a) => {  
  return 2 * a;  
}  
  
let doubleB = a => 2 * a;  
  
let listaDoble = lista  
  .map(a => 2 * a);  
// [2, 4, 6]
```

Arrow Functions

- Atención a las llaves al devolver objetos!

Objeto

```
let multiplos = (a) => ({  
  double: 2 * a,  
  triple: 3 * a,  
})
```

Bloque con return

```
var doubleA = (a) => {  
  return 2 * a;  
}
```

Parameter defaults

- Valores por defecto para los parámetros

```
function suma(a, b = 0) {  
  return a + b  
}  
  
// lo mismo que  
const suma = (a, b = 0) => a + b  
  
// antes se hacía así  
function suma(a, b) {  
  b = b === undefined ? 0 : b  
  return a + b  
}
```

Rest

- lista con el resto de parámetros recibidos
- solo puede ir de último

```
function multiply(multiplier, ...theArgs) {  
  return theArgs.map(function(element) {  
    return multiplier * element;  
  });  
}  
  
var arr = multiply(2, 1, 2, 3);  
console.log(arr); // [2, 4, 6]  
// con arrow functions  
const multiply = (multiplier, ...theArgs) =>  
  theArgs.map(element => multiplier * element);
```

Spread

- Distribuye los elementos de un array/objeto como si se pasasen uno a uno

```
let arr = [3, 5, 1];  
let arr2 = [8, 9, 15];  
  
let merged = [...arr, ...arr2]; // [3,5,1,8,9,15]
```

Asignación por desestructuración

- Asigna variables a partir de elementos de un array o propiedades de un objeto

Array

```
let arr = [2, 1, 2, 3];  
let [a, b, c, d] = arr;  
// a = 2, b = 1, c = 2, d = 3  
  
let [first, ...rest] = arr;  
// first = 2, rest = [1, 2, 3]
```

Objeto

```
let props = { a: 2, b: 1, c: 2, d: 3 };  
let {a, b, c, d} = props;  
// a = 2, b = 1, c = 2, d = 3  
  
let {a: firstVal} = props;  
// firstVal = 2 (renombrado)
```

Asignación por desestructuración

- Se puede acceder a propiedades anidadas

```
let user = {
  displayName: "jdoe",
  fullName: {
    firstName: "John",
    lastName: "Doe"
  }
};

let {fullName: {firstName: name}} = user;
//name = "John"
```


Operador ternario

- Son expresiones (*one-liners*)

```
const message = bottle.fullOfWater
  ? 'The bottle has water!'
  : 'The bottle may not have water :-( '
// es lo mismo que
let message
if (bottle.fullOfWater) {
  message = 'The bottle has water!'
} else {
  message = 'The bottle may not have water :-( '
}
```

Null coalescing operator

- Valor por defecto, en caso de `null` o `undefined`

```
// Se hacía así
x = x || 'some default'

// pero daba problemas con números o booleanos, donde "0" o "false"
// son valores válidos. Entonces, si queríamos soportar esto:
suma(null, 3)
```

```
// hacíamos así:
function suma(a, b) {
  a = a == null ? 0 : a
  b = b == null ? 0 : b
  return a + b
}
```

```
// ahora podemos hacer así:
function suma(a, b) {
  a = a ?? 0
  b = b ?? 0
  return a + b
}
```

Optional chaining

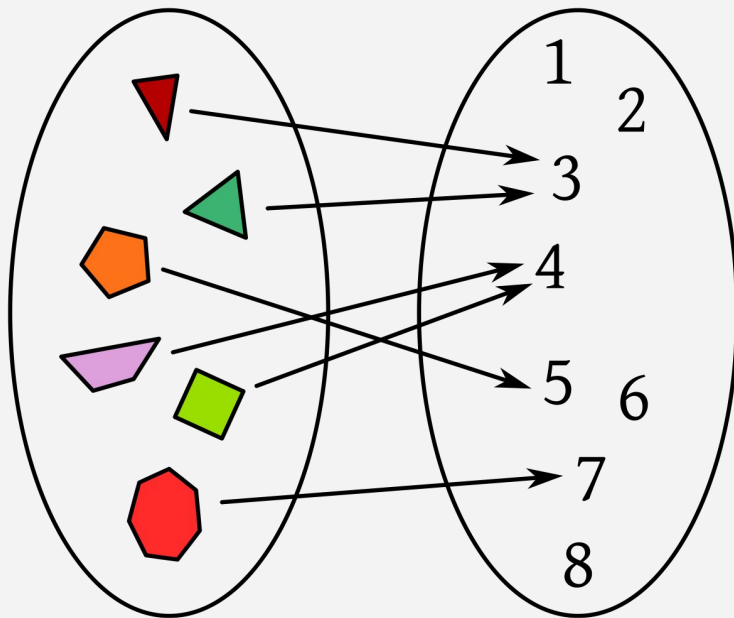
- También conocido como el operador Elvis
- Permite acceder a propiedades o invocar funciones que pueden no existir

```
// what we did before optional chaining:  
const streetName = user && user.address && user.address.street.name  
  
// what we can do now:  
const streetName = user?.address?.street?.name  
  
// esto funciona incluso si onSuccess es undefined  
// (en tal caso, no se ejecutará ninguna función)  
onSuccess?.({data: 'yay'})
```

Programación funcional

- Funciones matemáticas
- Parámetros → Resultado
- Funciones:

first-class citizen



Programación funcional: Estado

- Estado
 - Stateful vs Stateless
- Estado global vs Estado local
 - Ámbito
- Concepto de “declarativo”
- Concepto de “mutabilidad”

Funciones puras

- Mismos parámetros → Mismo resultado
- Sin efectos secundarios
- $(x) \rightarrow y$ **y nada más**

```
let lista = [1, 2, 5, 7, 11]

function duplicarLista(l){
  for (i = 0; i < l.length; i++){
    lista[i] = 2*l[i];
  }
  console.log(l);
}
```

```
let lista = [1, 2, 5, 7, 11]

function duplicarListaF(l){
  let resultado = [];
  for (i = 0; i < l.length; i++){
    resultado[i] = 2*l[i];
  }
  return (resultado);
}

console.log(duplicarListaF(lista));
```

Programación declarativa

- Ej: Duplicar los elementos de una lista

```
let lista = [1, 2, 5, 7, 11]
```

Imperativo

```
let listaFor = [];  
for (i = 0; i < lista.length; i++){  
    listaFor[i] = 2*lista[i];  
}  
  
console.log(listaFor);
```

Declarativo

```
function duplicar(x){  
    return 2*x;  
}  
  
let listaMap = lista.map(duplicar);  
  
console.log(listaMap);
```

Programación declarativa

```
function duplicar(x){  
    return 2*x;  
}  
  
let listaMap = lista.map(duplicar);
```

```
let listaMap = lista.map(  
    function(x){  
        return 2*x;  
    }  
);
```

```
let listaMap = lista.map(x => 2*x);
```

- Expresiones Lambda
- Funciones Lambda
- Funciones anónimas



FUNCIONAL

Programación declarativa

- Ejercicios: <http://reactivex.io/learnrx/>

#>/<>

HACK A BOSS

<CODE YOUR TALENT>

Funciones de orden superior

- Funciones que devuelven funciones
- Aplicación parcial

```
function generadorPotencia(exponente){  
  return function(base){  
    return Math.pow(base, exponente);  
  }  
}  
  
let cuadrado = generadorPotencia(2);  
let cubo = generadorPotencia(3);  
  
console.log(cuadrado(9));
```

Funciones de orden superior

- Sintaxis mucho más corta con arrow functions

```
const generadorPotencia = (exp) => (base) => Math.pow(base, exp);  
  
let cuadrado = generadorPotencia(2);  
  
console.log(cuadrado(9)); //81
```

Curry / Uncurry

- Aridad de una función: cantidad de parámetros
- Currificar: convertir en unaria

```
function suma(a, b){  
  return a + b;  
}
```

```
let resultado = suma(2,3); //5
```

```
function suma(a){  
  return function(b){  
    return a + b;  
  }  
}
```

```
let resultado = suma(2)(3); //5
```

Composición de funciones

- Sintaxis mucho más corta con arrow functions

```
const generadorPotencia = (exp) => (base) => Math.pow(base, exp);  
let cuadrado = generadorPotencia(2);
```

```
const generadorSuma = (sumando1) => (sumando2) => sumando1 + sumando2;  
let suma7 = generadorSuma(7);
```

```
console.log(suma7(cuadrado(9))); //88  
console.log(generadorSuma(7)(3)); //10  
console.log(generadorSuma(7)(generadorPotencia(3)(9))); //736
```

Práctica: añadir elemento al DOM mediante JS

```
<body>
  <div id="root"></div>
  <script type="module">
    </script>
  </body>
```

- Crear un div
- El textContent del div debe ser 'Hello World'
- El className debe ser 'container'
- Añadir el div al DOM colgando de 'root' (append)

Práctica: añadir elemento al DOM mediante JS

```
<body>
  <div id="root"></div>
  <script type="module">
    const rootElement = document.getElementById('root')
    const element = document.createElement('div')
    element.textContent = 'Hello World'
    element.className = 'container'
    rootElement.append(element)
  </script>
</body>
```



REACT

03

The React logo, a stylized atom with three intersecting elliptical orbits and a central circle, is rendered in a dark gray color on a black background.

REACT

REACT

- Es una librería, no un framework
 - Pintar interfaces de usuario
 - Gestionar el estado
- DOM
- VirtualDOM
- Elementos React
 - *children*

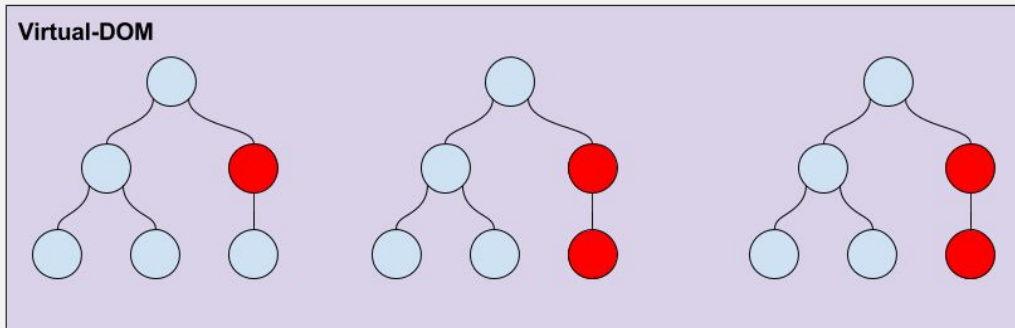
#>/<>

HACK A BOSS

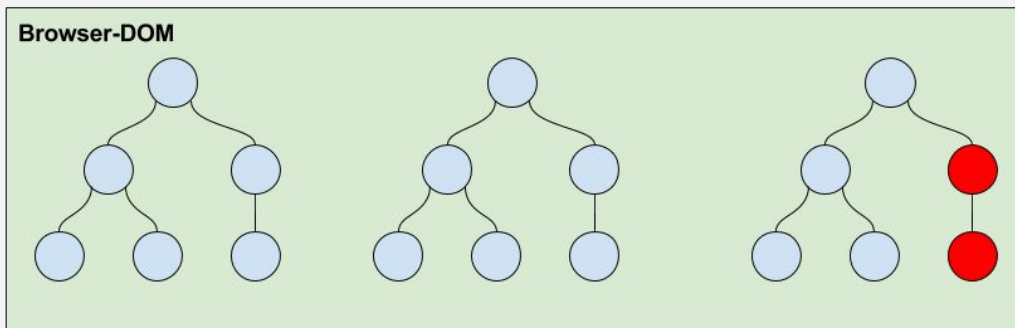
<CODE YOUR TALENT>

VirtualDOM

- Cálculos más rápidos en VirtualDOM



Cambio de estado → Calcular diff → Repintar



VirtualDOM

- Elemento del DOM que hará de contenedor

```
<html>
  <body>
    <div id="root"></div>
  </body>
</html>
```

- VirtualDOM de ***elementos React***

```
let paragraph = React.createElement(
  'p',
  {'className': 'red_border'},
  'Lorem ipsum dolor sit amet'
);
ReactDOM.render(
  paragraph,
  document.getElementById('root')
);
```

Práctica: añadir elemento al DOM mediante React

- Modificar el código anterior para usar React (RAW) en lugar de Javascript
- Añadir estas líneas para cargar React desde internet

```
<script  
src="https://unpkg.com/react@17.0.2/umd/react.development.js"></script>  
<script  
src="https://unpkg.com/react-dom@17.0.2/umd/react-dom.development.js">  
</script>
```



REACT

Ejemplo: añadir elemento al DOM mediante React

```
<body>
  <div id="root"></div>
  <script src="https://unpkg.com/react@17.0.2/[cortado]..."></script>
  <script src="https://unpkg.com/react-dom@17.0.2/[cortado]..."></script>
  <script type="module">
    const rootElement = document.getElementById('root')
    const element = React.createElement('div', {
      className: 'container',
      children: 'Hello World',
    })
    ReactDOM.render(element, rootElement)
  </script>
</body>
```

#></>

HACK A BOSS

<CODE YOUR TALENT>

Práctica: añadir elementos anidados al DOM mediante React

- Modificar el código anterior para conseguir el siguiente HTML (desde React)

```
<body>
  <div id="root">
    <div class="container">
      <span>Hello</span>
      <span>World</span>
    </div>
  </div>
</body>
```

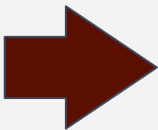
Solución: añadir elementos anidados al DOM mediante React

```
<script type="module">
  const rootElement = document.getElementById('root')
  const element = React.createElement('div', {
    className: 'container',
    children: [
      React.createElement('span', null, 'Hello'),
      ' ',
      React.createElement('span', null, 'World'),
    ],
  })
  ReactDOM.render(element, rootElement)
</script>
```

JSX

- Javascript XML
- Elementos React
- Notación similar a HTML
- Se compila a objetos Javascript (Babel)

```
<div color="blue" shadowSize={2}>  
  Click Me  
</div>
```



```
React.createElement(  
  'div',  
  {  
    color: 'blue',  
    shadowSize: 2  
  },  
  'Click Me'  
);
```


Expresiones en JSX

- Se usan llaves: {}

```
<div name="Pedro" surname={calculateSurname(user)}>  
  2 + 2 = {2 + 2}  
</div>
```

- JSX también es una expresión

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

Práctica: añadir elemento al DOM mediante JSX

- Añadir esta línea para cargar Babel

```
<script src="https://unpkg.com/@babel/standalone@7.12.4/babel.js">
</script>
```

- Modificar script type="module" para que sea procesado por Babel

```
<script type="text/babel">
</script>
```

Práctica: añadir elemento al DOM mediante JSX

- Modificar el código anterior para pintar “Hello World” usando JSX en lugar de React (RAW)

```
<body>
  <div id="root"></div>
  <script src="https://unpkg.com/react@17.0.2/[cortado]..."></script>
  <script src="https://unpkg.com/react-dom@17.0.2/[cortado]..."></script>
  <script src="https://unpkg.com/@babel/standalone@7.12.4/babel.js" />
  <script type="text/babel">
    // código aquí
  </script>
</body>
```

Solución: añadir elemento al DOM mediante JSX

```
<body>
  <div id="root"></div>
  <script src="https://unpkg.com/react@17.0.2/[cortado]..."></script>
  <script src="https://unpkg.com/react-dom@17.0.2/[cortado]..."></script>
  <script src="https://unpkg.com/@babel/standalone@7.12.4/babel.js"></...
  <script type="text/babel">
    const element = <div className="container">Hello World</div>
    ReactDOM.render(element, document.getElementById('root'))
  </script>
</body>
```

Práctica: JSX

- Interpolar valores
- Pintar mediante JSX el div usando estas variables

```
<body>
  <div id="root"></div>
  <script src="https://unpkg.com/react@17.0.2/[cortado]..."></script>
  <script src="https://unpkg.com/react-dom@17.0.2/[cortado]..."></script>
  <script src="https://unpkg.com/@babel/standalone@7.12.4/babel.js" />
  <script type="text/babel">
    const children = 'Hello World'
    const className = 'container'
    // continuar aquí
  </script>
</body>
```

Solución: JSX

- Interpolar valores

```
<script type="text/babel">
  const children = 'Hello World'
  const className = 'container'
  const element = <div className={className}>{children}</div>
  ReactDOM.render(element, document.getElementById('root'))
</script>
```

Práctica: JSX

- Spread de un objeto con las 'props'

```
<body>
  <div id="root"></div>
  <script src="https://unpkg.com/react@17.0.2/[cortado]..."></script>
  <script src="https://unpkg.com/react-dom@17.0.2/[cortado]..."></script>
  <script src="https://unpkg.com/@babel/standalone@7.12.4/babel.js" />
  <script type="text/babel">
    const children = 'Hello World'
    const className = 'container'
    const props = {children, className}
    // continuar aquí
  </script>
</body>
```

Solución: JSX

- Spread un objeto con las 'props'

```
<script type="text/babel">
  const children = 'Hello World'
  const className = 'container'
  const props = {children, className}
  const element = <div {...props} />
  ReactDOM.render(element, document.getElementById('root'))
</script>
```

- Children es también una prop



HERRAMIENTAS

REACT

- Webpack
- Create React App
- Extensiones
- npm
- git
- IDE

#>/<>

HACK A BOSS

<CODE YOUR TALENT>

Webpack

- empaquetador
- integra otras herramientas
- live reload, babel, sourcemaps
- development web server

CRA

- Create React App
- bootstrapper
- buenas prácticas
- buena [documentación](#)
- menos personalizable

Extensiones

- React Developer Tools (Browser)
- Eslint (Proyecto)
- Prettier (Proyecto)

The React logo, a stylized atom with three intersecting elliptical orbits, is rendered in a dark gray color on a black background.

REACT

React: Browser extension

- Visualizar componentes
- Buscar mediante click o texto
- Estudiar props
- Monitorizar cambios

#>/<>

HACK A BOSS

<CODE YOUR TALENT>

npm

- gestor de paquetes
- package.json
- versionado "semántico"

git

- control de versiones

DEVTOOLS



IDE

- VSCode
- WebStorm

#>/<>

HACK A BOSS

<CODE YOUR TALENT>

Práctica: añadir elementos al DOM mediante funciones que devuelven JSX

- Modificar el código anterior para conseguir el siguiente HTML (mediante JSX y funciones)

```
<body>
  <div id="root">
    <div class="container">
      <div class="msg">Hola Mundo!</div>
      <div class="msg">Adios Mundo cruel 🤪</div>
    </div>
  </div>
</body>
```

Práctica: añadir elementos al DOM mediante funciones que devuelven JSX

1. Crear una función para generar algo como

```
<div class="msg">Mensaje</div>
```

dados:

- Un nombre de clase css
- Un texto como mensaje

2. Usar esta función en el código anterior

Solución: JSX mediante funciones

```
<script type="text/babel">
  function message(className, children) {
    return <div className={className}>{children}</div>
  }

  const element = (
    <div className="container">
      {message('msg', 'Hola Mundo!')}
      {message('msg', 'Adios Mundo cruel 🤪')}
    </div>
  )

  ReactDOM.render(element, document.getElementById('root'))
</script>
```


Práctica: añadir elementos al DOM usando funciones que devuelven JSX (II)

- Modificar el código anterior para que la función que genera `<div class="msg">Mensaje</div>` reciba un solo parámetro props:

```
const props = {
  className: 'msg',
  children: '¿Carreteras? A donde vamos no necesitamos carreteras',
}
```

Solución: añadir elementos al DOM usando funciones que devuelven JSX (II)

Es un solo parámetro, desestructurado



```
function message({className, children}) {  
  return <div className={className}>{children}</div>  
}
```

Práctica: elementos React a partir de funciones

- `React.createElement(type, props, children)`
 - **type** puede ser una función que devuelva algo pintable
 - Si es así, `createElement` va a pasar **props** como argumento a **type**
- Es decir, React va a “ejecutar” **type(props)**

Práctica: elementos React a partir de funciones

- Tarea: crear elementos React cuyo type sea la función creada anteriormente para pintar los mismos mensajes:

```
<div class="msg">Hola Mundo!</div>  
<div class="msg">Adios Mundo cruel 🤪</div>
```

Solución: elementos React a partir de funciones

```
<script type="text/babel">
  function message({children}) {
    return <div className="message">{children}</div>
  }
  const element = (
    <div className="container">
      {React.createElement(message, {children: 'Hola Mundo!'})}
      {React.createElement(message, {children: 'Adios Mundo cruel 🤪'})}
    </div>
  )
  ReactDOM.render(element, document.getElementById('root'))
</script>
```


Práctica: Usar elementos como JSX

- Babel se encarga de compilar JSX a Javascript
- JSX tiene unas reglas específicas
- ¿cómo cambiar lo último que hicimos para en

vez de `{React.createElement(message, {children: 'Mensaje'})}`

poder usar: `<Message>Mensaje</Message>` ?



COMPONENTES

04

React: Componentes

- ~~Elemento~~ “Trozo” visual reusable
- Conjunto de elementos React
- Puede tener estado
- Por convenio, PascalCase

function Component

- en forma de funciones
- return es render()
- ~~sin estado~~ (desde hooks también tienen estado)

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

Class component

- usando clases ES6
- extend React.Component

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

- pueden tener estado

React: Componentes

- *props* son "solo lectura"
- regla: All React components must act like pure functions with respect to their props.
- Componer componentes
- Extraer componentes

Componer componentes

```
function Welcome(props) {  
  return <h1>Hello, {props.name}</h1>;  
}  
  
function App() {  
  return (  
    <div>  
      <Welcome name="Sara" />  
      <Welcome name="Cahal" />  
      <Welcome name="Edite" />  
    </div>  
  );  
}
```

Extraer 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>  
  );  
}
```


Extraer componentes

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

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



ESTILOS

05

React: Estilos

- Dos formas “naturales” de establecer estilos
 - Estilos en línea con la prop `style`
 - CSS normal mediante la prop `className`
- Librerías avanzadas CSS-in-JS (emotion, styled-components...)
 - [Css In Your JS](#) ([transparencias](#))

React: prop style

- En HTML se pasa un string de CSS

```
<div style="margin-top: 20px; background-color: blue;"></div>
```

- En React se pasa un objeto de CSS

```
<div style={{marginTop: 20, backgroundColor: 'blue'}} />
```

- `{{...attr}}` → expresión que envuelve un objeto
- Las propiedades se pasan a `camelCase`



REACT

Práctica: Instalar CRA

- `npx create-react-app my-app`
- Comentar interioridades

#>/<>

HACK A BOSS

<CODE YOUR TALENT>

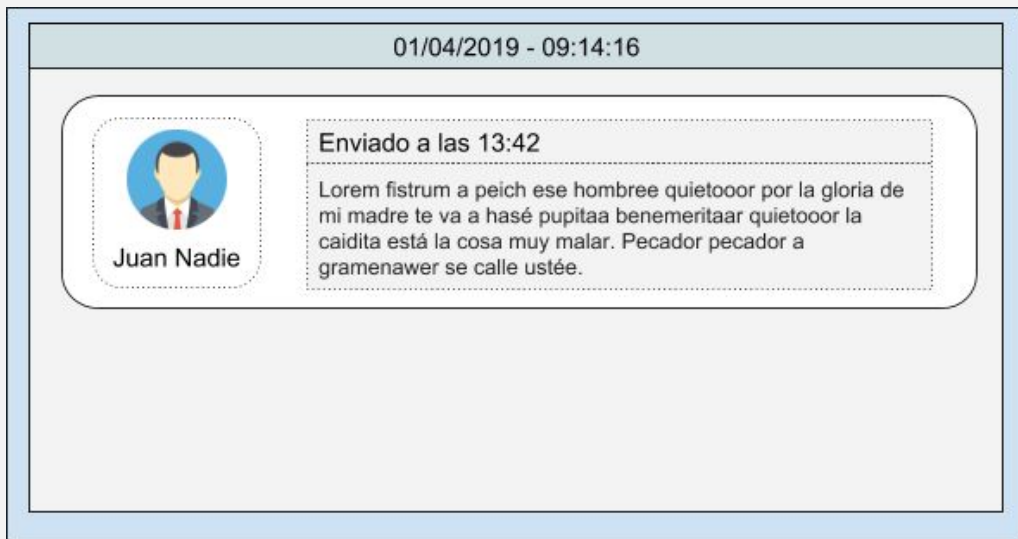
Práctica: Estilos

- Descargar los ficheros de la carpeta `ej-styles` del repositorio compartido y guardarlos en CRA
- Modificar `index.js` para que cargue `Styles.js` en lugar de `App.js`
- Dar estilo como se pide en el fichero `Styles.js`

Práctica: Estilos (apartado 2)

- Extraer componente Box
- Props `style` y `className`
- prop `size` (opcional)

P5: UI Inicial



P5: Sala de chat

- Descomponer UI
- Escribir componentes
- Todo hardcoded

P5: Componentes

- ChatRoom (Class)
- ChatRoomHeader (Class)
- Message
- MessageBody
- Avatar (function)

P5: Componentes

ChatRoom (Class)

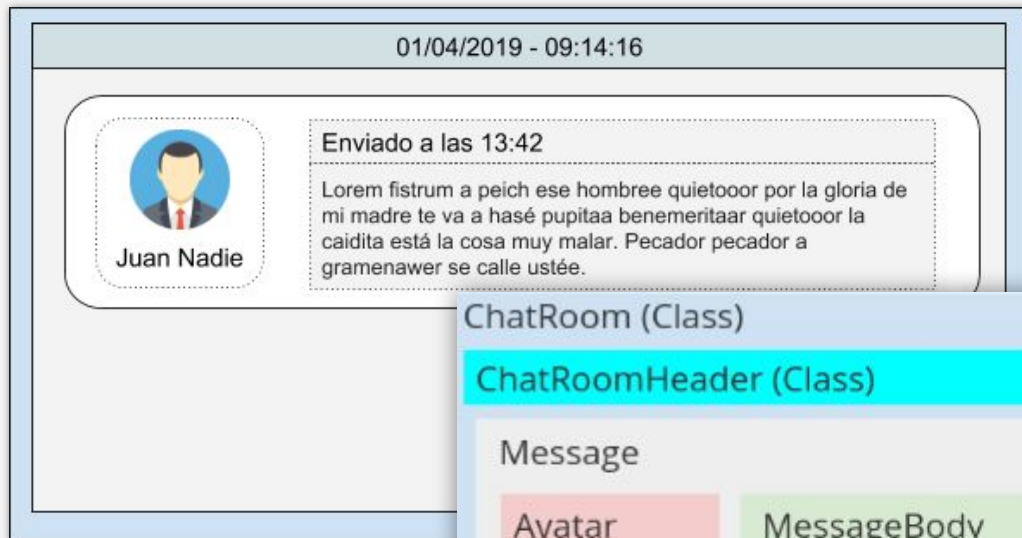
ChatRoomHeader (Class)

Message

Avatar
(function)

MessageBody

P5: Componentes



ChatRoom (Class)

ChatRoomHeader (Class)

Message

Avatar
(function)

MessageBody



ESTADO

06

React: Estado

- Local a cada componente
- **NUNCA** se cambia a mano
- Sólo class component (hasta que llegó hooks)

React: Estado

- constructor invoca super(props)
- Inicializar estado

```
class Clock extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = {date: new Date()};  
  }  
  render() {  
    return (  
      <div>  
        <h2>  
          {this.state.date.toLocaleTimeString()}  
        </h2>  
      </div>  
    );  
  }  
}
```

React: Estado

- acceso: `this.state[...]`
- modificación: `this.setState()`

MAL

```
this.state.carColor = 'red';
```

BIEN

```
this.setState({carColor: 'red'});
```


React: Estado

- modificación puede ser asíncrona
- cuando depende de props o state =>
- `this.setState(fn)`

MAL

```
this.setState({  
  counter: this.state.counter + this.props.increment,  
});
```

BIEN `fn: (state, props) => ({nextState})`

```
this.setState((state, props) => ({  
  counter: state.counter + props.increment  
}));
```

React: Estado

- `this.setState({})` mezcla
 - superficial (shallow)

```
//ANTES
this.state = {
  posts: [],
  comments: ["comment1", "comment2"]
};
this.setState({
  comments: ["comment3"]
})
//DESPUÉS
this.state = {
  posts: [],
  comments: ["comment3"]
};
```

React: useState

- Para... usar el state
- Desestructuración al asignar
- Parámetro: valor inicial

```
const [count, setCount] = useState(0);
```

Atención: esto es un **hook**.

Se explicarán en detalle más adelante

React: useState

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

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

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

P6: Hora actual

- ChatRoom con estado
 - db.json
- Pintar la hora real



**CICLO DE VIDA
(CLASES)**

07

Lifecycle: Mount

- **constructor()** asignar this.state directamente
- **static getDerivedStateFromProps()**
- **render()**
- **componentDidMount()**
- ~~**UNSAFE_componentWillMount()**~~

Lifecycle: Update

- static getDerivedStateFromProps()
- shouldComponentUpdate()
- **render()**
- getSnapshotBeforeUpdate()
- **componentDidUpdate()**
- ~~UNSAFE_componentWillUpdate()~~
- ~~UNSAFE_componentWillReceiveProps()~~

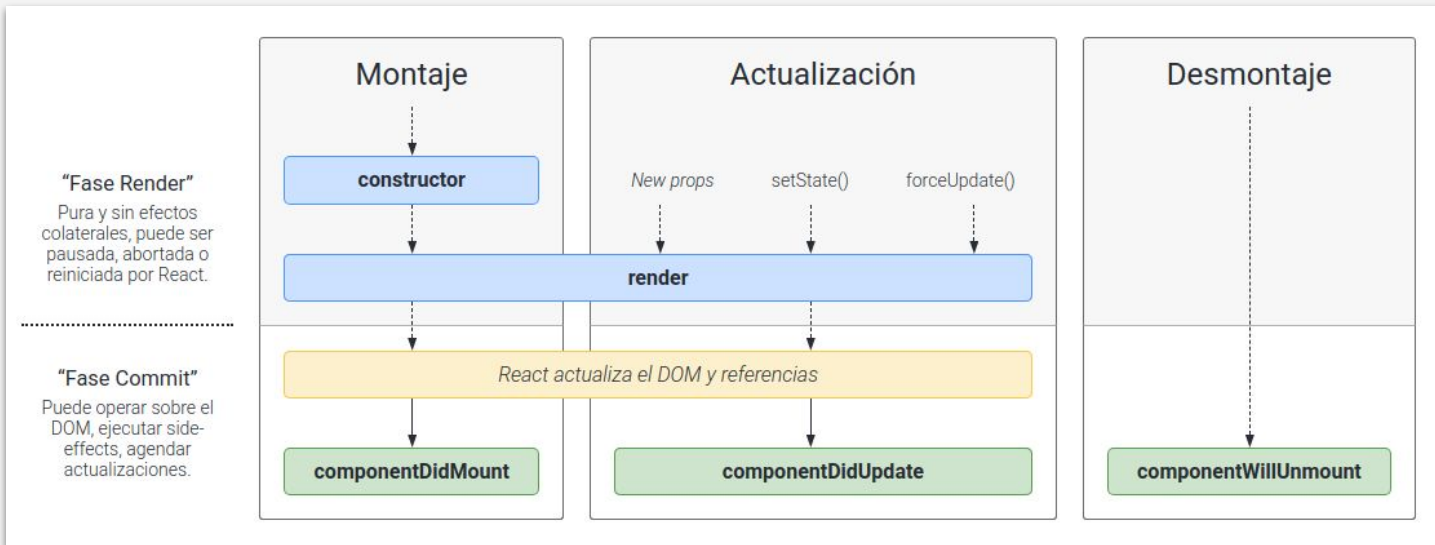
Lifecycle: Unmount

- [componentWillUnmount\(\)](#)

Lifecycle: Error handling

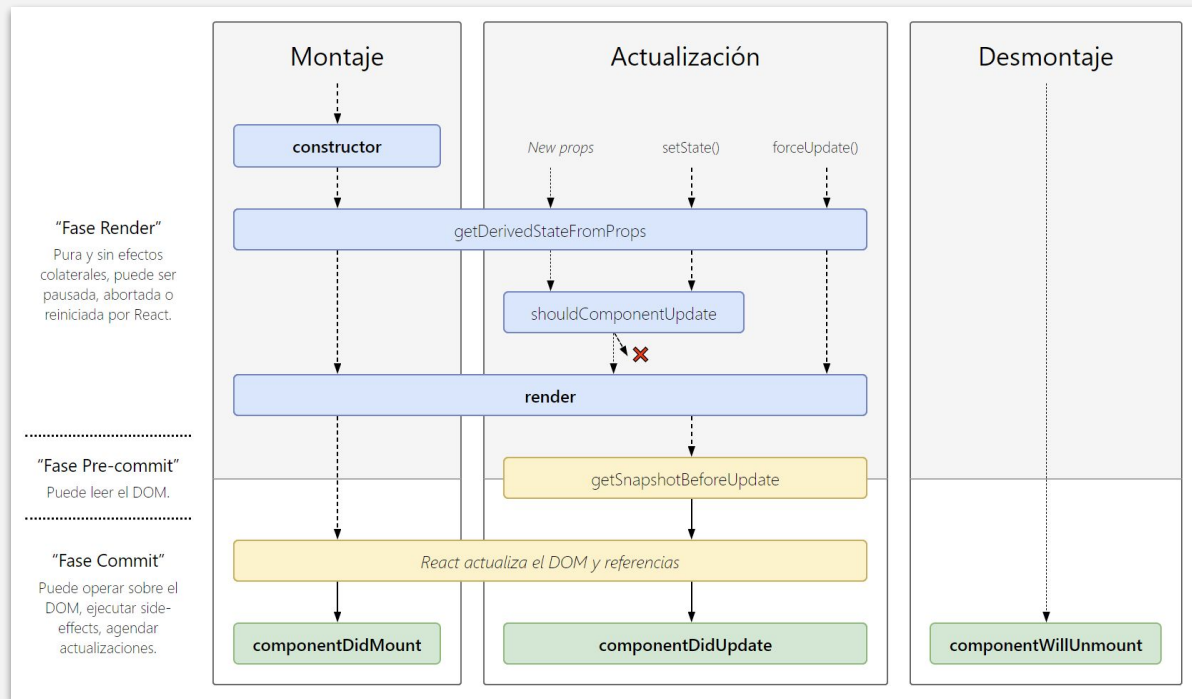
- [static getDerivedStateFromError\(\)](#)
- [componentDidCatch\(\)](#)

Lifecycle



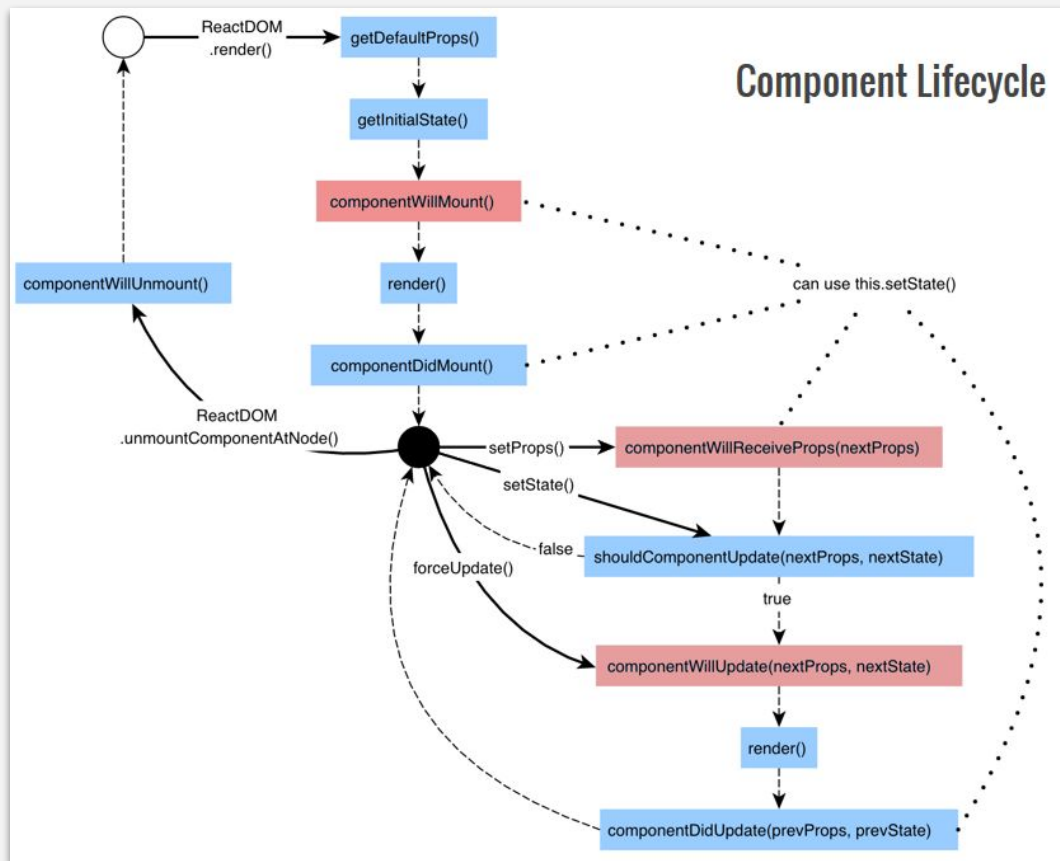
<http://projects.wojtekmaaj.pl/react-lifecycle-methods-diagram/>

Lifecycle



REACT

Lifecycle



#>/<>

HACK A BOSS

<CODE YOUR TALENT>

P7: Refrescar hora

- Usar `setInterval()`
- `componentDidMount()`
- `componentDidUnmount()`

P8: Componente Mensaje a partir de datos

- Se os proporciona un objeto que representa un mensaje
- El componente Message debe crear su aspecto visual (JSX) *en función de* ese objeto, que recibirá como prop



LISTAS

08

React: Lists

- elemento JSX = componente React
- Normalmente relación 1:1
- ¿Cómo se hacen listas?

React: Lists

- dentro de un Wrapper
- mucho uso de map()

```
function NumberList() {  
  const numbers = [1, 2, 3, 4, 5];  
  const listItems = numbers.map(  
    number => (<li>{number}</li>)  
  );  
  return (  
    <ul>{listItems}</ul>  
  );  
}
```

React: Lists

- Problema: detectar cambios
- Solución: keys

P9: Lista de mensajes

- Se os proporciona un array de objetos que representa una lista de mensajes
- Se debe pintar una lista de componentes Message
- Atención a key



EVENTOS

09

React: Events

- SyntheticEvent
- En camelCase (onClick)
- Se pasan funciones, no texto

```
<!-- HTML -->  
<button onclick="activateLasers()">  
  Activate Lasers  
</button>
```

```
//JSX  
<button onClick={activateLasers}>  
  Activate Lasers  
</button>
```

React: Events

- Se evalúa lo que esté entre llaves
- Se pasa el evento como parámetro

```
<button onClick={activateLasers}>  
  Activate Lasers  
</button>  
  
//se va a ejecutar activateLasers(e)  
//e va a ser el evento sintético click
```

Es decir: los "manejadores"
de eventos son funciones
que tienen una firma así:

```
function eventHandler(event){  
  //something  
}  
//equivalente  
const eventHandler = (event) =>  
{  
  //something  
}
```

React: Events

- minimizar llamadas a `addEventListener()`
- invocar `preventDefault()` explícitamente

```
function ActionLink() {  
  function handleClick(e) {  
    e.preventDefault();  
    console.log('The link was clicked.');  }  
  return (  
    <a href="#" onClick={handleClick}>  
      Click me  
    </a>  
  );  
}
```

React: Events

- Cuidado con el binding de this

```
class Toggle extends React.Component {
  constructor(props) {
    super(props);
    this.state = {isToggleOn: true};
    // This binding is necessary to make `this` work in the callback
    this.handleClick = this.handleClick.bind(this);
  }
  handleClick() {
    this.setState(state => ({
      isToggleOn: !state.isToggleOn
    }));
  }
  render() {
    return (
      <button onClick={this.handleClick}>
        {this.state.isToggleOn ? 'ON' : 'OFF'}
      </button>
    );
  }
}
```


React: Events

- Cuidado con el binding de this

```
class Toggle extends React.Component {
  constructor(props) {
    super(props);
    this.state = {isToggleOn: true};
  }
  handleClick: () => { //experimental!!!
    this.setState(state => ({
      isToggleOn: !state.isToggleOn
    }));
  }
  render() {
    return (
      <button onClick={this.handleClick}>
        {this.state.isToggleOn ? 'ON' : 'OFF'}
      </button>
    );
  }
}
```

React: Events

- Cuidado con el binding de this

```
class Toggle extends React.Component {
  constructor(props) {
    super(props);
    this.state = {isToggleOn: true};
  }
  handleClick(){
    this.setState(state => ({
      isToggleOn: !state.isToggleOn
    }));
  }
  render() {
    return (
      <button onClick={(e) => this.handleClick(e)}>
        {this.state.isToggleOn ? 'ON' : 'OFF'}
      </button>
    );
  }
}
```

Eventos: parámetros

- Nueva función


```
<button onClick={(e) => this.deleteRow(id, e)}>Delete Row</button>
```

- bind this

```
<button onClick={this.deleteRow.bind(this, id)}>Delete Row</button>
```

P09: Enviar mensaje

- Pintar formulario (input + boton)
 - mismo componente que la lista de mensajes
- onSubmit (formularios se ven más adelante)
 - uncontrolled
- guardar en la lista de mensajes



Darth Maul

P09: Enviar mensaje

```
const ChatRoomInput = () => {  
  const handleSubmit = (e) => {  
    e.preventDefault();  
    console.dir(e.target);  
    //Hacer algo aqui  
  };  
  
  return (  
    <form onSubmit={handleSubmit}>  
      <input type="text" id="message" />  
      <button type="submit">Enviar</button>  
    </form>  
  );  
};
```

React: refs

- Guardan referencias a elementos del DOM
- Guardan referencias a variables
 - que persisten durante la vida del componente (como useState)
 - que **NO** notifican a React de sus cambios (a diferencia de useState)

A large, dark gray, stylized React logo is positioned on the left side of the slide, partially overlapping the black background.

REACT

React: Ejemplo

Obtener los valores de un input mediante refs

#>/<>

HACK A BOSS

<CODE YOUR TALENT>



API REQUESTS

10

React: API requests

- Se pueden usar librerías
 - axios, jQuery ajax, standard fetch
- Su efecto es cambiar state
- React refleja esos cambios

React: API requests

- Inicialización:
 - `componentDidMount()`
 - `useEffect`
- Como efecto de un evento

React: API requests

- Casi siempre Promesa

```
const onFulfill = (value) => {  
  this.setState({name: value});  
}  
  
const onReject = (errorMsg) => {  
  console.error('Error al obtener el nombre:', errorMsg);  
  this.setState({problem: errorMsg});  
}  
  
const x = getName //imaginad una llamada ajax  
  .then(onFulfill, onReject)  
  .catch(onReject)
```

REACT

#>/<>

HACK A BOSS

<CODE YOUR TALENT>

```
const Fetcher = () => {
  const [posts, setPosts] = useState([]);
  const loadPosts = (e) => {
    e.preventDefault();
    fetch("https://jsonplaceholder.typicode.com/posts")
      .then(res => res.json())
      .then(res => setPosts(res), msg => console.error("Err:", msg))
  };
  return (
    <div className="App">
      <form onSubmit={loadPosts}>
        <button type="submit">Load</button>
      </form>
      <ul>{posts.map(post => <li key={post.id}>{post.title}</li>)}</ul>
    </div>
  );
};
```

REACT



#>/<>

HACK A BOSS

<CODE YOUR TALENT>

```
const Fetcher = () => {
  const [posts, setPosts] = useState([]);
  const loadPosts = async (e) => {
    try {
      const res = await fetch("https://jsonplaceholder.typicode.com/posts");
      const body = await res.json();
      setPosts(body);
    } catch (e) {
      console.error("Err:", e);
    }
  };
  return (
    <div className="App">
      <button onClick={loadPosts}>Load</button>
      <ul>{posts.map(post => <li key={post.id}>{post.title}</li>)}</ul>
    </div>
  );
};
```

React: API requests

- ATENCIÓN: gestionar cuando no hay datos
- Conditional Render
- <https://reactjs.org/docs/faq-ajax.html>

<https://medium.freecodecamp.org/how-to-easily-cancel-useeffect-http-calls-with-rxjs-d1be418014e8>

P10: Cargar los mensajes del servidor

- Tutorial: Arrancar servidor mock-json-server
- Nuevo botón: Cargar
- Lista de mensajes inicial = []
- Al pulsar el botón pedir al servidor la lista de mensajes guardados
 - Guardar esa lista en el estado



FORMULARIOS

11

React: Forms

- Source of truth (browser vs React)
- Form elements vs React
- Controlled vs Uncontrolled

A large, dark gray, stylized React logo is positioned on the left side of the slide, partially overlapping the black background.

REACT

React: Controlled

- value
- onChange
- this.setState
- useState

#>/<>

HACK A BOSS

<CODE YOUR TALENT>

REACT

#>/<>

HACK A BOSS

<CODE YOUR TALENT>

React: Controlled

```
class NameForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = {value: ''};
  }
  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>
    );
  }
}
```

REACT

#>/<>

HACK A BOSS

<CODE YOUR TALENT>

React: Controlled

```
const NameForm = () => {  
  const [value, setValue] = useState("");  
  
  const handleSubmit = (event) => {  
    alert("A name was submitted: " + value);  
    event.preventDefault();  
  };  
  
  const handleChange = (event) => {  
    setValue(event.target.value);  
  };  
  
  return (  
    <form onSubmit={handleSubmit}>  
      <label>  
        Name:  
        <input type="text" value={value} onChange={handleChange} />  
      </label>  
      <input type="submit" value="Submit" />  
    </form>  
  );  
}
```

React: Uncontrolled

- `input type="file"`
- `useRef`
- `modify imperatively`

React: Forms

- Librerías de ayuda:
 - Formik
 - React Hook Forms
- Librerías de validación
 - yup
 - joi

P11: Enviar mensaje

- Convertir a componente controlado



A chat input field with a blue border. The top bar contains a circular profile picture of Darth Maul and the text 'Darth Maul'. Below this is a large white text input area. To the right of the input area is a grey button with a black right-pointing triangle.

P11b: Enviar mensajes al servidor

- Modificar el botón de enviar mensaje para que envíe al servidor, en lugar de al estado local

Ejemplo: Petición POST

- Preparamos un objeto para enviar en el body
 - JSON.stringify()
- fetch con segundo parámetro “options”
- Importantes las cabeceras con Content-type

```
const data = { /* El objeto que sea... */ };
const serializedData = JSON.stringify(data);
const res = await fetch("http://localhost:3050/messages", {
  method: 'POST',
  body: serializedData,
  headers: {
    'Content-type': 'application/json'
  }
});
```



ESTADO (II)

12

React: Managing State

- Estado local al componente
- Lifting State Up
- Unidirectional data flow

React: Estado local

- No hay estado global "per se"
- Lo más parecido es el estado del "padre" de la jerarquía
- Cada componente se encarga de "lo suyo"

Ya no estrictamente cierto (Context API)

React: Estado global

"Application State"

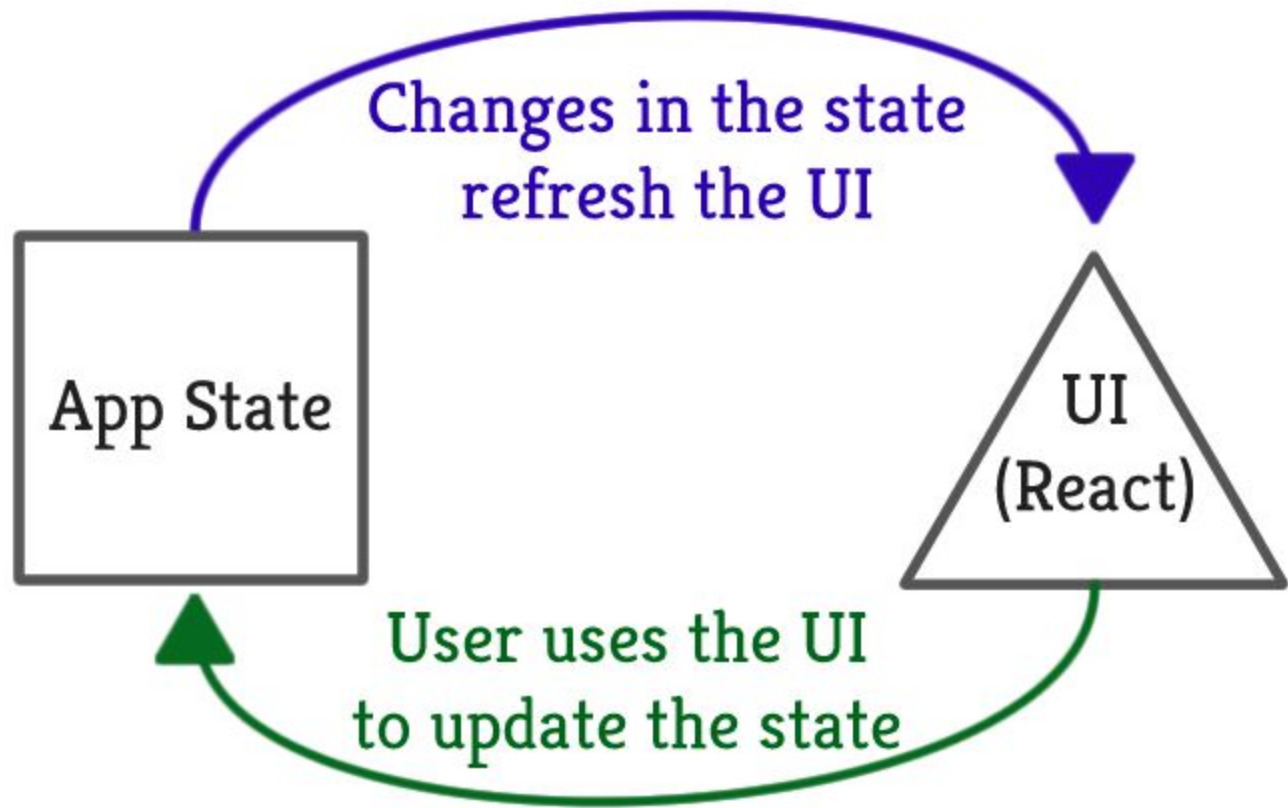
- Difícil "intencionadamente"
- Librerías para gestionar esto
 - REDUX (Muy usado, hay middlewares, conceptualmente interesante)
 - MOBX (Basada en observables)
 - UNSTATED (Usa el api Context, elegante)
- Context API

React: Lifting State Up

- El estado "sube" hasta el Componente ancestro más "bajo" en la jerarquía que lo necesite.
- Se pasa a los hijos mediante props.
- Se pasan a los hijos métodos para cambiar estado

REACT

Unidirectional Data Flow



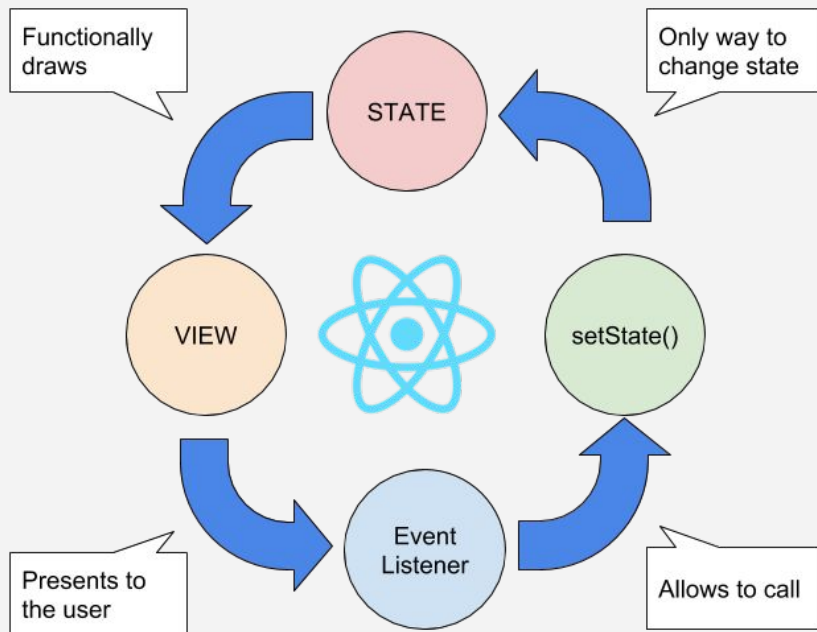
#>/<>

HACK A BOSS

<CODE YOUR TALENT>

REACT

Unidirectional Data Flow



#>/<>

HACK A BOSS

<CODE YOUR TALENT>

Unidirectional Data Flow

"Top-Down data flow"

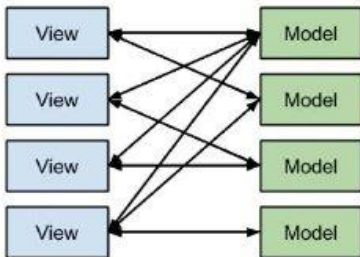
- Los datos van en una dirección: de padre a hijo
- Ningún ancestro trabaja con el estado de sus hijos.
- Ningún componente sabe si otro es stateful o stateless

Unidirectional Data Flow

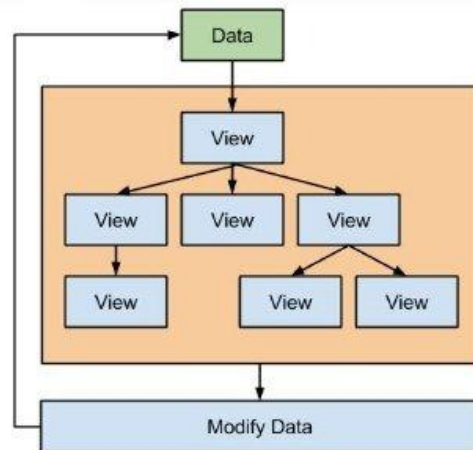
"Top-Down data flow"

The Tree of Components + 1 way data flow

- **1 way data flow** is the **secret sauce** of React
- Top – Down rendering



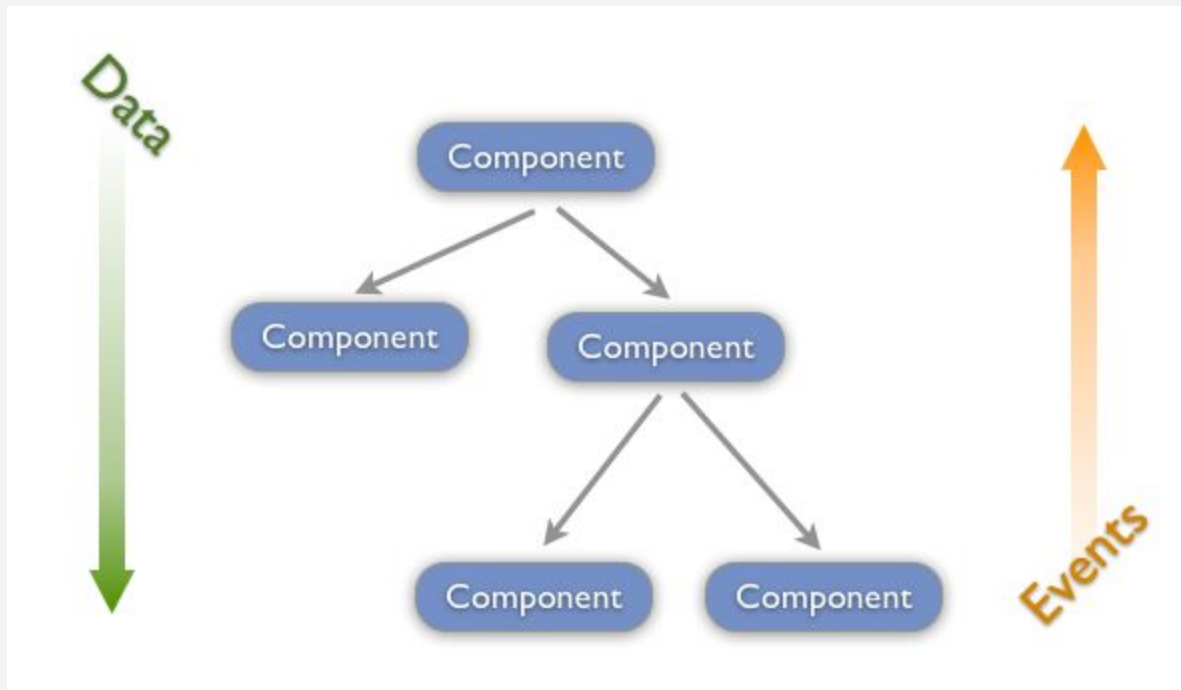
MV* Data Binding



React

Unidirectional Data Flow

"Top-Down data flow"



P12: Estructura de archivos

- Carpeta “componentes”
- Cada componente en un archivo
 - Esto **no** es ley! Es un ejercicio



HOOKS

13

React: Hooks

- Para usar state y otras cosas sin escribir clases
- Opcional
- "wrapper hell"
- Habilita optimizaciones

React: Reglas de hooks

- Sólo dentro de function Components o custom hooks (no en vuestras funciones)
- Sólo "top-level" del componente. Nunca dentro de if, bucles, o funciones anidadas
- Hay *linter* para ayudar con esto.

React: Hooks

- **useState**
- **useEffect**
- `useContext`
- `useRef, useReducer, ...`

React: useState

- Para... usar el state
- Destructuración al asignar
- Parámetro: valor inicial

```
const [count, setCount] = useState(0);
```

React: useState

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

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

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

React: useEffect

- Efectos secundarios (side effects):
 - Llamadas al API
 - Configurar una suscripción
 - Manipular a mano el DOM
- Limpiar al salir

React: useEffect

- "Reemplaza":
 - componentDidMount
 - componentDidUpdate
 - componentWillUnmount

React: useEffect

- Parámetro:
función de "efecto"
- Return del "efecto":
función de limpieza (opcional)

```
useEffect(() => {  
  let timer = setTimeout(  
    () => setDate(new Date())  
    , 1000  
  );  
  
  return(() => clearTimeout(timer));  
}, /*segundo parámetro*/);
```

React: useEffect

- Labor: hacer algo tras el render
- Se ejecuta tras TODOS los render
- Hay que optimizar

React: useEffect

Optimizar

- 2º parámetro:
 - lista de variables a monitorizar
 - si cambian, se ejecuta el efecto otra vez
- Si no se pasa nada, se ejecuta siempre
- Si se pasa [] se ejecuta una sola vez
(componentDidMount)

React: useEffect

Optimizar

```
useEffect(() => {  
  function handleStatusChange(status) {  
    setIsOnline(status.isOnline);  
  }  
  
  ChatAPI.subscribeToFriendStatus(props.friend.id, handleStatusChange);  
  return () => {  
    ChatAPI.unsubscribeFromFriendStatus(props.friend.id,  
    handleStatusChange);  
  };  
}, [props.friend.id]); // Only re-subscribe if props.friend.id changes
```


P13: ChatRoom con hooks

- Lograr que ChatRoom pase de ser un Class Component a un function Component
- ChatRoom debe seguir teniendo estado y efectos

React: custom hook

- Función que empieza con "use" y puede llamar a otros hooks

```
import React, { useState, useEffect } from 'react';
function useFriendStatus(friendID) {
  const [isOnline, setIsOnline] = useState(null);
  useEffect(() => {
    function handleStatusChange(status) {
      setIsOnline(status.isOnline);
    }
    ChatAPI.subscribeToFriendStatus(friendID, handleStatusChange);
    return () => {
      ChatAPI.unsubscribeFromFriendStatus(friendID, handleStatusChange);
    };
  });
  return isOnline;
}
```

React: custom hook

- Cómo se usa? Pues...

```
const FriendListItem = (props) => {  
  const isOnline = useFriendStatus(props.friend.id);  
  
  return (  
    <li style={{ color: isOnline ? 'green' : 'black' }}>  
      {props.friend.name}  
    </li>  
  );  
}
```

React: custom hook

- Se debe llamar empezando por "use"? SÍ
- Dos componentes pueden usar el mismo custom hook? SÍ
- Comparten estado si comparten hook? NO
- Cómo paso información entre hooks? Son funciones: mediante parámetros

REACT

Hook Flow

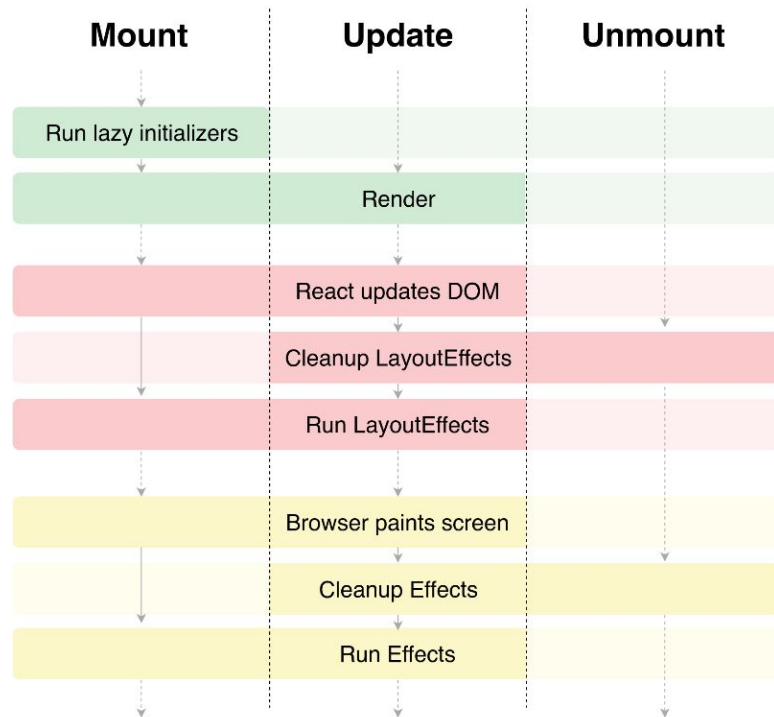
#>/<>

HACK A BOSS

<CODE YOUR TALENT>

React Hook Flow Diagram

v1.3.1 github.com/donavon/hook-flow



Notes:

1. Updates are caused by a parent re-render, state change, or context change.
2. Lazy initializers are functions passed to useState and useReducer.

P13: Cargar los mensajes periódicamente desde el servidor

- usar `useEffect` para iniciar un temporizador que cargue los mensajes del servidor cada segundo
 - limpiar el temporizador al acabar!
- A continuación, crear un custom hook “`useRemoteMessages`” que combine ese `useEffect` y el `useState` con la lista de mensajes

P13: Cargar la lista de usuarios del servidor

- Mostrar un `<select>` con la lista de usuarios del servidor
 - Componente independiente
 - 1º `useState` + `useEffect` (llamada remota)
 - 2º `customHook`
- Mostrar debajo una ficha con el usuario seleccionado: Avatar + nombre



CONTEXTO

14

Context

- Compartir información entre componentes distantes de una jerarquía
- Datos casi “globales”
- No tiene por qué ser toda la jerarquía
- **Evita pasar props por elementos intermedios**
- Ejemplos: usuario autenticado, idioma del sitio, tema seleccionado

Context

- *Prop drilling*: pasar props por elementos intermedios
- Evitarlo es Tentador!! pero no siempre ideal
- Solución mejor en muchos casos: **Componer Componentes**

Context

```
// Se declara el contexto (fuera de componentes)
const MyContext = React.createContext(defaultValue);

// Se envuelve un trozo de JSX en el Provider de ese
// contexto y se le asigna un valor
<MyContext.Provider value={/* some value */} >
  <Algo />
</MyContext.Provider>

// los descendientes (ej. Algo) pueden usar useContext
const value = useContext(MyContext);
```

- Y cómo se modifica el valor del contexto
 - Creando un custom Provider con estado ⇒

Context

```
const CounterContext = React.createContext(defaultValue);

const CounterProvider = props => {
  const [count, setCount] = React.useState(0);
  return <CounterContext.Provider value={ [count, setCount] }>
    {props.children}
  </CounterContext.Provider>
}

const CounterDisplay = () => {
  const [count] = React.useContext(CounterContext);
  return <div>La cantidad actual es {count}</div>
}

const App = () => (
  <CounterProvider>
    <CounterDisplay />
  </CounterProvider>
)
```

Context: Michael Jackson

- <https://www.youtube.com/watch?v=3XaXKiXtNjw>



MICHAEL JACKSON @mjackson · 15 nov. 2019

...

Unpopular opinion: All that junk you're putting in React context should just be props. Legit uses for context are rare, and mostly for library code, not your app.

45

126

619



MICHAEL JACKSON @mjackson · 15 nov. 2019

...

The canonical use case for context is when you're building compound components; or multiple components that need to work together as if they were one. For example, a `<Select>` + `<Option>` combo, where it doesn't make sense to use one w/out the other. That's where it really shines.

2

4

59



MICHAEL JACKSON @mjackson · 15 nov. 2019

...

React is fundamentally about composition. Context is bit of a cheat. It's useful in some situations, but the more you use it the further you get from encapsulation and composition.

Practice the fundamentals: state and props. Get good at those and you won't need to cheat so much.

7

2

80



P13: Contexto

- usar contexto para almacenar al usuario seleccionado
- Mostrar al usuario seleccionado en el input de escribir el mensaje
- Queremos que el usuario seleccionado sea quién envía el mensaje



PATRONES

15

A large, faint, dark gray React logo is positioned on the left side of the slide, behind the 'REACT' text.

REACT

React: Patrones

- Presentational vs Container
- Higher Order Components
- Render Props

#>/<>

HACK A BOSS

<CODE YOUR TALENT>

React: Presentational

- Sólo aspecto visual
- Reciben qué pintar por props
- Stateless
- Function components
- “Hijo” de un Container

React: Container

- Sin aspecto visual
- Reciben datos por props
- Normalmente stateful
- Devuelven un Presentational

React: HOC

- Un HOC es una función que recibe un Componente y devuelve otro Componente
- Redux "connect"

React: HOC

- Se usan para abstraer funcionalidad común
- Firma:

```
const EnhancedComponent = higherOrderComponent(WrappedComponent);
```

REACT

React: HOC

```
const withInterval = effectFunction => WrappedComponent => {
  return class extends React.Component {
    constructor(props) {
      super(props);
      this.state = { timer: null };
    }
    componentDidMount() {
      let timer = setInterval(
        () => this.setState({data: effectFunction()})
        , 1000);
      this.setState({ timer: timer });
    }
    componentWillUnmount() {
      clearInterval(this.state.timer);
    }
    render() {
      return <WrappedComponent {...this.props} data={this.state.data}/>;
    }
  };
};
```

#>/<>

HACK A BOSS

<CODE YOUR TALENT>

React: HOC

```
function getDate() {  
  let d = new Date();  
  return d.toLocaleString("es-ES");  
}  
  
const dateUpdater = () => getDate();  
  
class ChatRoomHeader extends Component {  
  render() {  
    return <header className="ChatRoomHeader">{this.props.data}</header>;  
  }  
}  
  
export default withInterval(dateUpdater)(ChatRoomHeader);
```

React: Render Props

- Componente con una *prop* cuyo valor es una función que devuelve un componente
- React Router
- react-motion

React: Render Props

- También se usan para abstraer funcionalidad común
- `render()` usa la "render prop"
- Firma:

```
<DataProvider render={data => (  
  <h1>Hello {data.target}</h1>  
)}>
```


React: Render Props

- Es una *prop* con una función que el componente usa para saber qué pintar
- Se pueden reescribir casi todos los HOC usando render props
- Más potente que HOC

P14: Render props

Sea `MessageList` el componente responsable de pintar la lista de mensajes. Queremos:

- Que el aspecto visual de los mensajes individuales se pueda definir desde el padre
- Usar una render prop para esto



AUTENTICACIÓN

16

JWT

- Registro: POST /register

```
{  
  "email": "olivier@mail.com",  
  "password": "bestPassw0rd"  
}
```

- Login: POST /login

```
{  
  "email": "olivier@mail.com",  
  "password": "bestPassw0rd"  
}
```

<https://www.npmjs.com/package/json-server-auth>

JWT

- Para hacer consultas identificado hay que añadir la siguiente cabecera en las peticiones:

```
GET /600/users/1
```

```
Authorization: Bearer xxx.xxx.xxx
```

Usando el access token que nos devuelve la llamada a login o a register:

```
{  
  "accessToken": "xxx.xxx.xxx"  
}
```

JWT: Seguridad

- JWT no se recomienda para mantener sesiones
- Vulnerable a XSS. Mejor Cookies de sesión
- Alternativa:

JWT in memory + Refresh token in cookie

- <http://crypto.net/~joepie91/blog/2016/06/13/stop-using-jwt-for-sessions/>
- <https://stormpath.com/blog/where-to-store-your-jwts-cookies-vs-html5-web-storage>
- https://hasura.io/blog/best-practices-of-using-jwt-with-graphql/#refresh_token
- <https://supertokens.io/blog/cookies-vs-localstorage-for-sessions-everything-you-need-to-know>

P15.1: Custom useLocalStorage hook

Queremos hacer algo similar a useState pero que guarde la información también en localStorage

- useState + useEffect
- window.localStorage.setItem
- window.localStorage.getItem (pensad dónde)
- misma interfaz que useState ⇒ debéis poder reemplazar useState por useLocalStorage
- Aplicar al mensaje escrito y refrescad aplicación

P15.2: Hacer formulario de registro

- input email + password
- POST a /register
 - el body debe tener el formato:

```
{
  "email": "alguien@sitio.dominio",
  "password": "passwordEnClaro",
}
```

- guardar accessToken en localStorage 🚒

P15.3: Hacer formulario de login

- input email + password
- POST a /login
- Sois capaces de hacerlo sin repetir el JSX?

P15.4: Obtener mensajes “protegidos”

- la consulta a mensajes pasa a ser /660/messages
- hay que mandar la cabecera

`Authorization: Bearer ...`



ENRUTAMIENTO

17

React Router

- Para simular enrutamiento en el lado del cliente
- SPA
- Utiliza history API para manipular la URL y el historial de URL.
- Necesita ayuda del servidor web para la primera visita a la página
 - <https://create-react-app.dev/docs/deployment/#serving-apps-with-client-side-routing>

React Router: Componentes

- Router: envuelve la aplicación
 - `<BrowserRouter />`
- Route Matcher: comprueban la url para decidir qué componente pintar
 - `<Switch />` y `<Route />`
- Route Changer: Para navegar
 - `<Link />`, `<NavLink />`, `<Redirect />`

<https://reactrouter.com/web/guides/quick-start>

P16: URLs en la aplicación

- Hacer que la aplicación soporte 3 rutas
 - `/register`: Pinta formulario de registro
 - Si autenticado \Rightarrow /
 - `/login`: Pinta formulario de login
 - Si autenticado \Rightarrow /
 - Enlace a `/register`
 - `/`: Pinta ChatRoom
 - Si no autenticado \Rightarrow `/login`



UPLOAD

18

Subir ficheros al servidor

- Llamada al servidor => Fetch API
- Método => **POST**
- data => **FormData**
- Content-type => No especificar (depende del servidor)

Subir ficheros al servidor

- Formulario en JSX:

```
<div className="App">
  <form onSubmit={uploadFile}>
    <div>
      <label>Select file to upload</label>
      <input type="file" onChange={onFileChange}/>
    </div>
    <button type="submit">Upload</button>
  </form>
</div>
```

Subir ficheros al servidor

- Usamos useState

```
function uploadFile() {  
  let data = new FormData();  
  data.append('image', file);  
  fetch('http://localhost:3050/files', {  
    method: 'POST',  
    body: data,  
  })  
    .then(response => response.json())  
    .then(success => {  
      // Do something with the successful response  
    })  
    .catch(error => console.log(error))  
  );  
}  
  
const onFileChange = event => {  
  const f = event.target.files[0];  
  setFile(f);  
}
```

P17: Subir imagen para vuestro avatar

- Utilizar un input tipo file para que el usuario seleccione una imagen
 - No puede ser “controlado”
- Utilizar Fetch para subir la imagen
 - Method: POST
 - body: FormData



**LIBRERÍAS DE
COMPONENTES**

19

Librerías de componentes

- Permiten “esquivar” las tareas de diseño y UX
 - Los creadores de las librerías ya se encargaron de eso
- Sus componentes están pensados para funcionar en conjunto
- **Accesibilidad!** Casi ninguna la tiene al 100%

P18: Usar Material-UI

- Instalar Material-UI (MUI) en nuestra aplicación
- Cambiar nuestro componente Avatar para que use un “Avatar” de MUI
 - Atención al conflicto de nombres en los imports
- Cambiar el componente Message (individual) para que use un Card de MUI



RECURSOS 20

Recursos: Conexión remota

- [react-query](#) (hook library)
- [swr](#) (hook library)
- fetch api (direct request)
- [axios](#) (direct request)
- GraphQL API ⇐ [Apollo GraphQL](#)

Recursos: Presentación

- [Material UI](#)
- [Ant Design](#)
- [Bit.dev](#) (suelos)
- [Tailwind](#)
 - [For CRA](#)
 - [TailwindUI](#) (pago)
 - [Example](#)
- [Bumbag.style](#)
- [React Bootstrap](#)
- [React Foundation](#)
- [Semantic UI](#)
- [React Toolbox](#)
- [Atlassian](#)

Recursos: Formularios

- [react-hook-form](#)
- [formik](#)
- [react final form](#)

Recursos: Gestión del estado

- [XState](#)
 - interesante para componentes, no para toda la App
- [Redux](#)
- [Mobx](#)

Recursos: Gráficos

- [Rechart](#)
- [Victory](#)
- [Nivo](#)
- [Vis](#)
- [VX](#)
- [Ant Design Charts](#)
- [React financial charts](#)

Recursos: Frameworks

- [Next.js](#)
- [Gatsby.js](#)
- [Remix](#) (pago)

Recursos: Animación

Animación

- [React Spring](#)
- [Framer Motion](#)
 - <https://dev.to/joserfelix/getting-started-with-react-animations-308a>

Recursos: Extra

- [Awesome React](#)
- [Awesome React Components](#)
- [javascript.info](#)
- [https://kentcdodds.com/blog/](#)
- [https://www.joshwcomeau.com/tutorials/react/](#)
- [http://reactivex.io/learnrx/](#)

Recursos: Extra

- [React Three Fiber](#)
- <https://webxr.autovrse.in/3d-automotive-configurator>
 - <https://github.com/Epiczzor/r3f-template>
- <https://todomvc.com/>