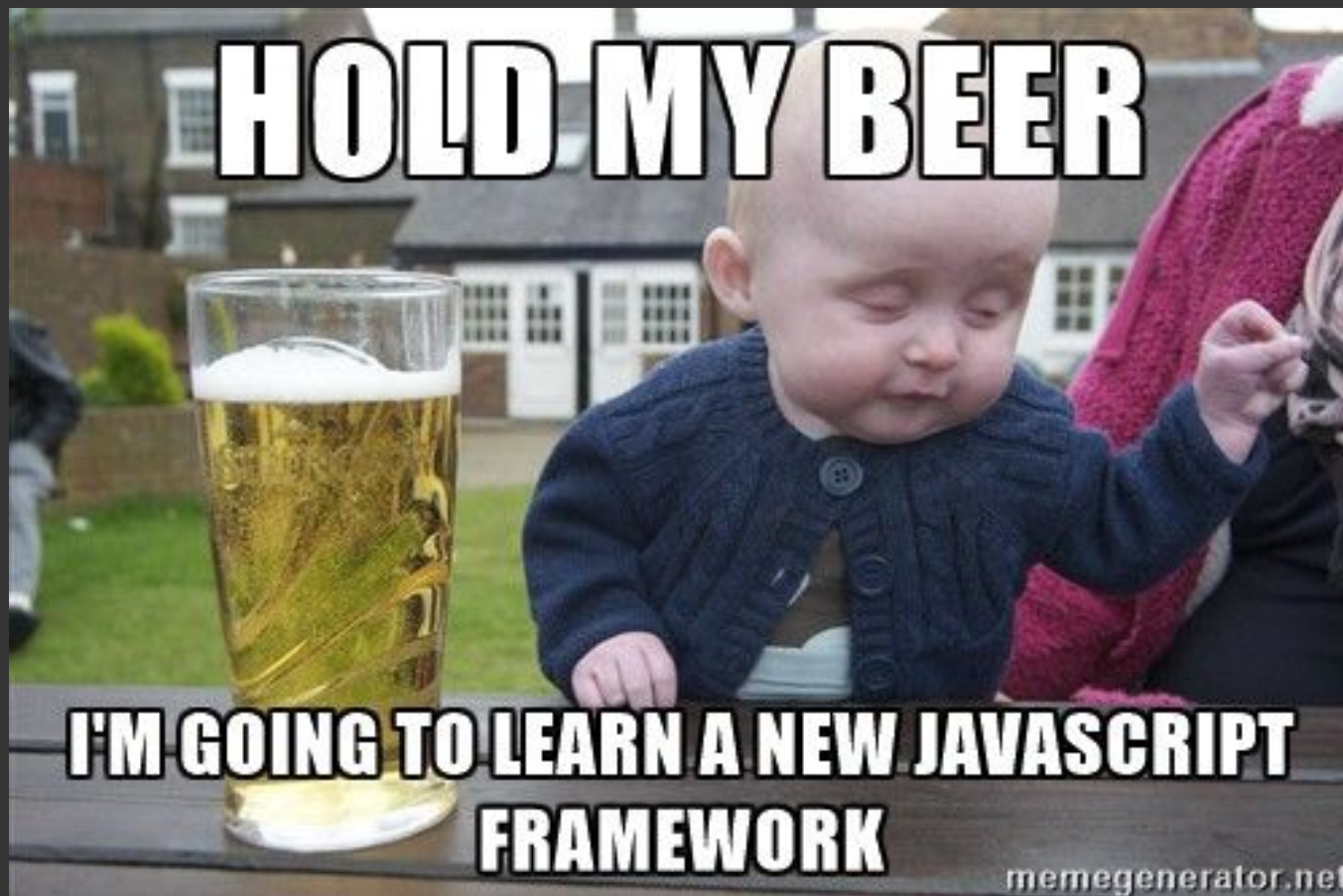
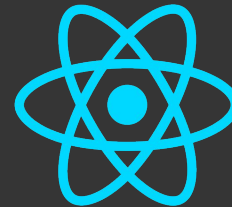


React

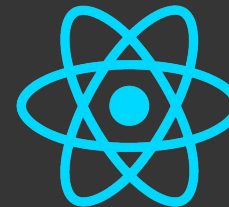
Bootcamp Frontend Developer

Upgrade  hub

REACT



REACT



Es una biblioteca Javascript de código abierto diseñada para crear interfaces de usuario con el objetivo de facilitar el desarrollo de aplicaciones en una sola página. Es mantenido por Facebook y la comunidad de software libre, han participado en el proyecto más de mil desarrolladores diferentes.

Inicio de desarrollo 2010 - Primera release Mayo 2013

React es usado por empresas como Netflix, Airbnb...

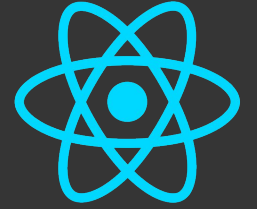
Ventajas de usar React:

- Flexibilidad
- Composición por componentes
- Desarrollo declarativo
- Flujo de datos unidireccional
- Rendimiento
- Isomorfismo

Desventajas:

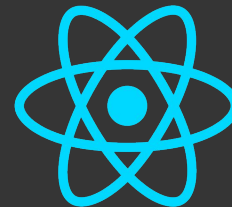
- Solo la vista del MVC
- Aprendizaje de sintaxis JSX

REACT



- Librería para crear interfaces de usuario
- No es un framework, es una librería de UI
- Se puede complementar fácilmente mediante plugins/extensiones
 - Router
 - Redux
 - Flow

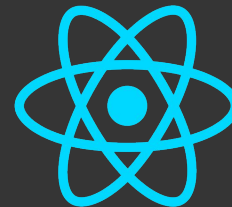
REACT



Modificar el DOM es una operación costosa y React ofrece una manera sencilla y optimizada para realizar esta operación con el Virtual DOM.

- Los componentes de React no generan HTML directamente
- Generan código, una descripción virtual del DOM
- Se estructura en árbol, con un único nodo principal
- Cada vez que se realiza un render, React guarda un snapshot en memoria

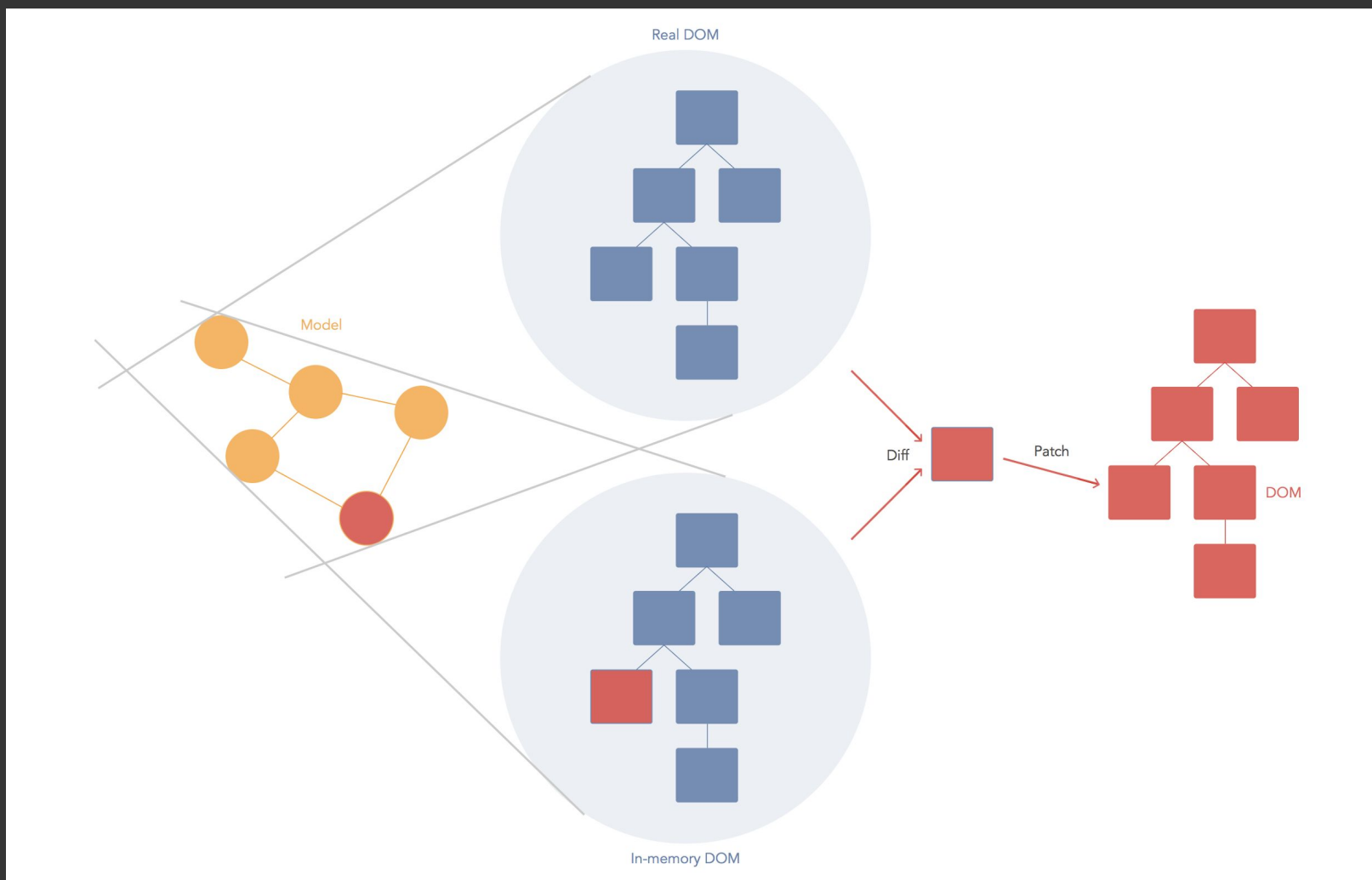
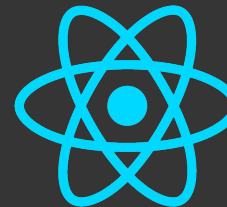
REACT



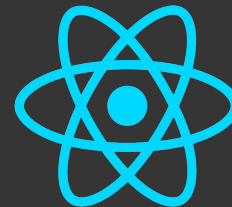
Cuando hay algún cambio en los datos de los componentes a mostrar:

- React compara la nueva versión de los componentes, con el snapshot guardado en memoria del render anterior
- De ese modo decide que debe ser modificado en DOM real
- React garantiza que se realizan el menor número posible de operaciones sobre el DOM para mostrar el nuevo estado
- A nivel de desarrollo, es igual que si se renderiza la aplicación por completo de nuevo
- De este modo, esta operación se realiza de la manera más eficiente

REACT



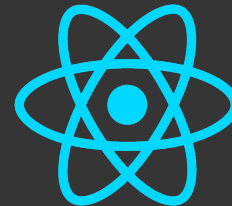
REACT



- La “API” que expone React parece que repinta la aplicación completa en cada render
- Los componentes de la app, definen su estado en momento dado del flujo de la aplicación
- Una aplicación React, es en realidad un componente formado por una composición de componentes

Para realizar esto, React usa una sintaxis especial JSX y ese código hay que transpilarlo.

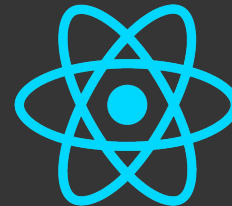
REACT



```
const Hello = React.createClass ({  
  render() {  
    return (  
      <h1>Hello world!</h1>  
    );  
  }  
})
```

```
var Hello = React.createClass({  
  displayName: "Hello",  
  render: function render() {  
    return React.createElement("h1", null, "Hello world!");  
  }  
});
```

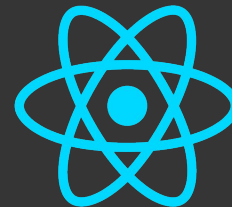
REACT



```
const Hello = React.createClass ({
  render() {
    return (
      <div id="wrapper">
        <h1>Hello world!</h1>
      </div>
    );
  }
})
```

```
var Hello = React.createClass({
  displayName: "Hello",
  render: function render() {
    return React.createElement("div", {
      id: "wrapper"
    }, React.createElement("h1", null, "Hello world!"));
  }
});
```

REACT



A la hora de realizar el transpilado, se utiliza babel:

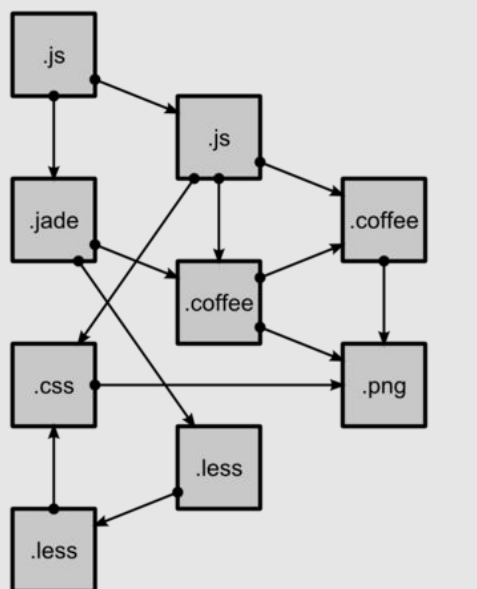
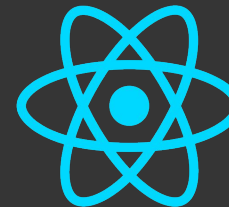


Además nos permite utilizar toda la sintaxis de ES6, ES7, ES8... y lo traduce a ES5 de modo que cualquier navegador pueda ejecutar nuestra aplicación

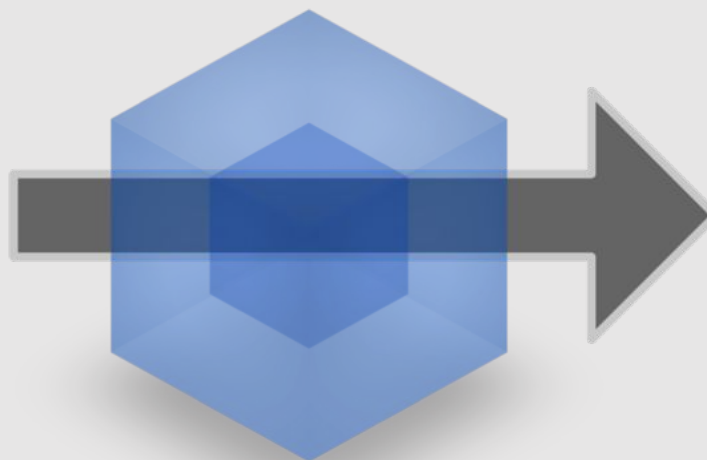
Para generar el bundle de la aplicación, se usa webpack:



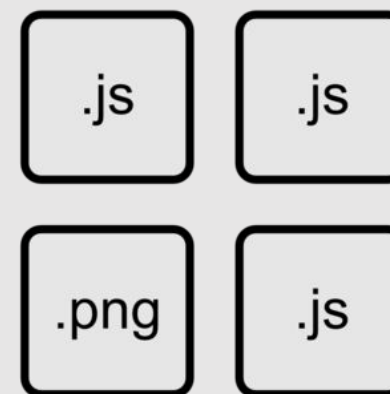
Podemos separar nuestro código en carpetas y ordenarlo por componentes utilizando npm como gestor de dependencias y tareas generando un único fichero JS con nuestra aplicación



modules
with dependencies

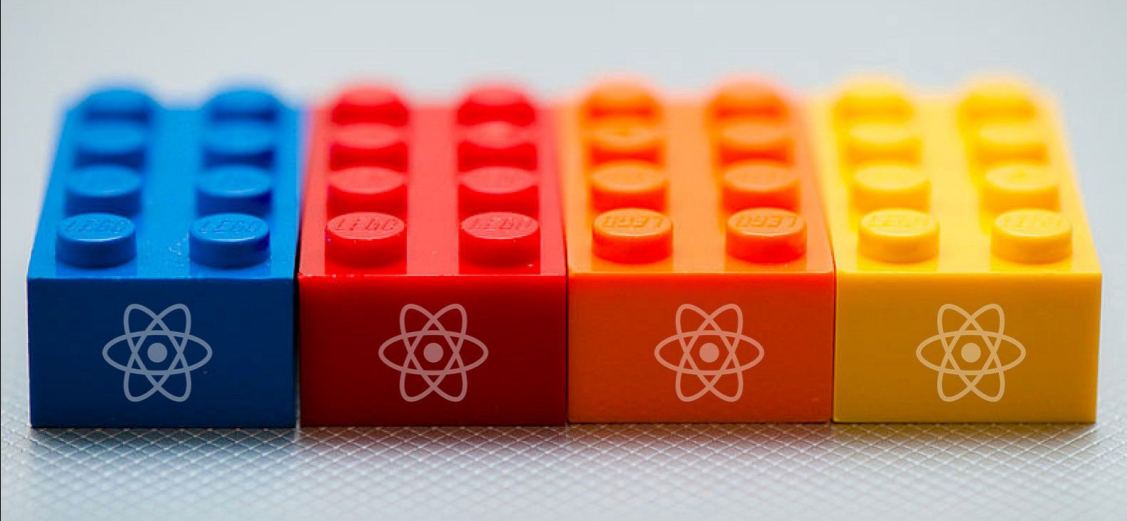
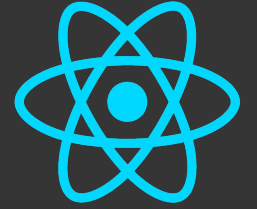


webpack
MODULE BUNDLER



static
assets

REACT: Componentes

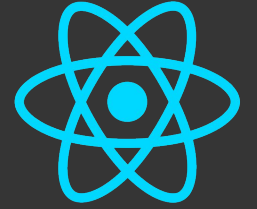


React desde un inicio se centró en ser una librería para construir componentes - UI

Cada “pieza” de una web es susceptible de ser un componente, desde un botón a una tabla de resultados con acciones sobre cada fila.

React permite la flexibilidad de granularizar tanto como queramos (¡o no!) el desarrollo de la aplicación.

REACT: Create React App



Es la herramienta de línea de comandos creada por el mismo equipo de React, requiere de NodeJs y se usa mediante npm:

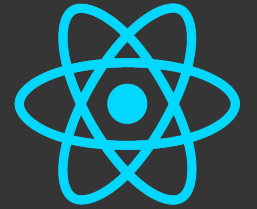
```
npx create-react-app my-app  
cd my-app  
npm start
```



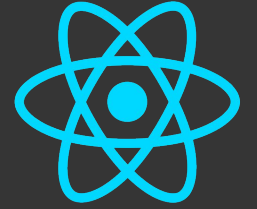
Con estos 3 comandos, hemos creado nuestra primera aplicación React y la hemos arrancado en <http://localhost:3000/>

El “CLI” de React nos ofrece varias funcionalidades
(ver más en: <https://github.com/facebook/create-react-app#whats-included>)

REACT: Create React App



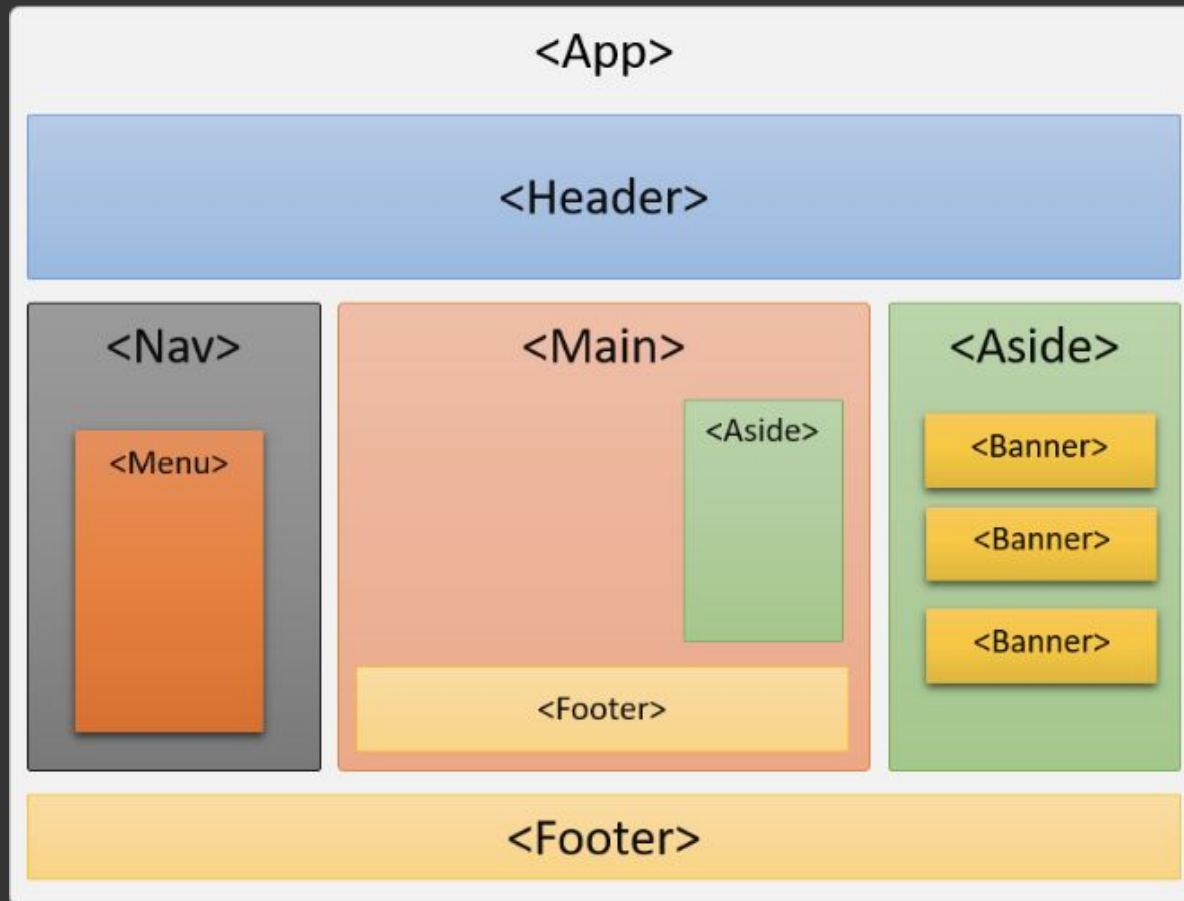
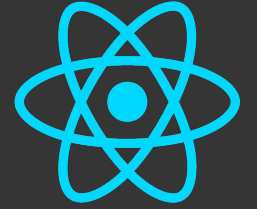
REACT: puntos clave



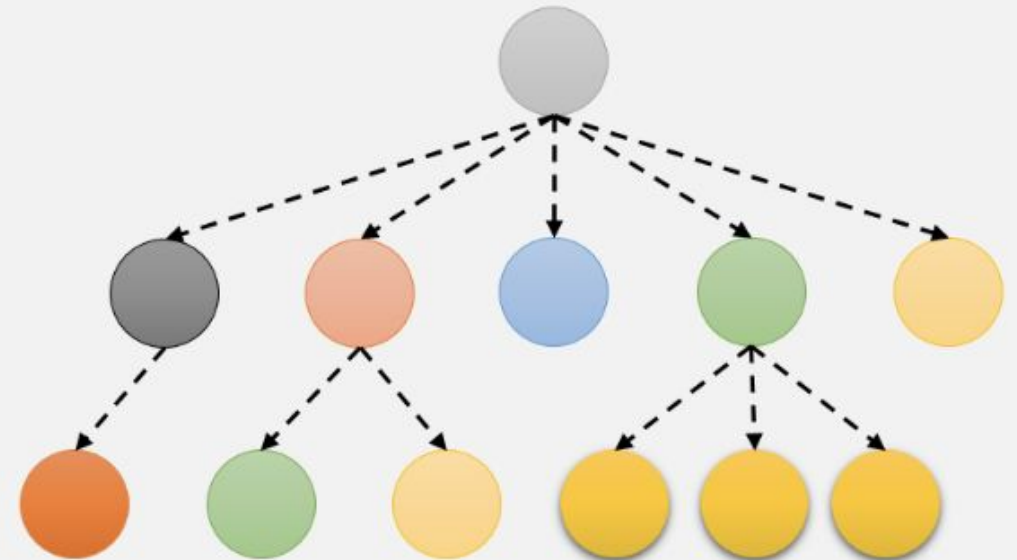
- Construir interfaces de usuario mediante una jerarquía de componentes
- Utiliza virtual DOM para mayor eficiencia
- Genera automáticamente cambios necesarios en DOM real (navegador)
- Simplicidad para el programador
- Cada componente define su salida como una función pura: render ()

Una función pura es aquella cuya ejecución no tiene efectos secundarios

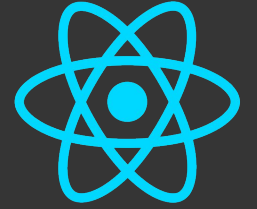
REACT: puntos clave



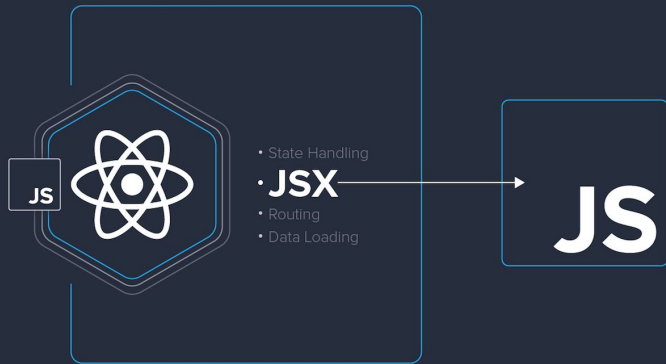
Jerarquía de componentes



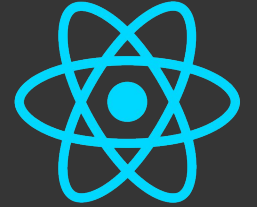
REACT: JSX



- Es una sintaxis basada en XML
- Al igual que XML es muy similar a HTML (ver más en [MDN](#))
- Su mayor diferencia es que se compila/transpila a código javascript
- La interfaz del componente y su lógica de negocio están englobados en el propio componente



REACT: JSX



```
import React from 'react';
```

Hay que importar React en el fichero

```
const HolaMundo = React.createClass({
```

Factoría de React

```
  render: function() {
```

OBLIGATORIO, se llama para pintar el componente

```
    return (
```

```
      <div className="wrapper">¡Hola mundo!</div>
```

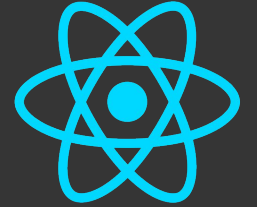
Lo que devuelve el componente

```
    );
```

```
  }
```

```
});
```

REACT: JSX

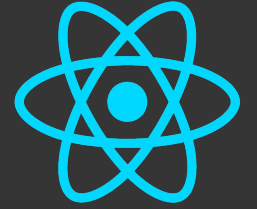


```
import React, { Component } from 'react';

class HolaMundo extends Component {
  render() {
    return (
      <div className="wrapper">¡Hola mundo!</div>
    );
  }
};

export default HolaMundo;
```

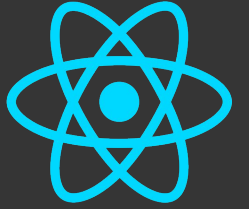
REACT: JSX



Aporta comodidad para no tener que estar escribiendo `React.createElement...`

- `class` -> `className`
- `for` -> `htmlFor`
- camelCase en eventos (`onClick`, `onChange`, ...)
- `style` tiene que recibir un objeto

REACT: JSX



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

const myStyle = {color: 'silver', border: '1px solid #000'};

class HolaMundo extends Component {

  render() {

    return (

      <h1 style={myStyle}>Hola mundo!</h1>

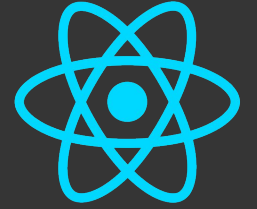
    )

  }

};

export default HolaMundo;
```

REACT: JSX



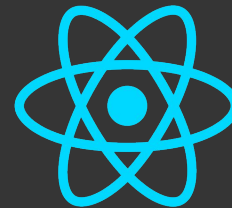
Un componente, puede generar:

- Otros componentes
- Elementos HTML

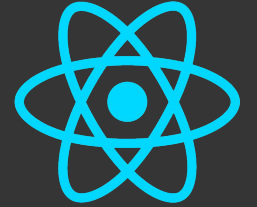
Para hacer fácilmente identificable cada uno, la convención en JSX es:

- `<etiqueta>` : elementos HTML
- `<Etiqueta>` : componentes de React

REACT: JSX



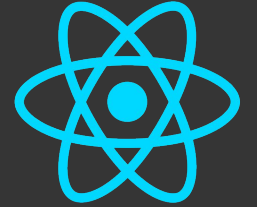
REACT: JSX



- En la función render podemos escribir código “cualquier” javascript
- Sin embargo, dentro del return solo puede haber expresiones:


```
class ComponentWithExpressions extends Component {  
  render() {  
    var user = {  
      name: "John",  
      surname: "Connor"  
    };  
    return (<div>  
      <p>Nombre: {user.name} Apellido: {user.surname}</ p>  
    </div>);  
  }  
}
```

REACT: JSX



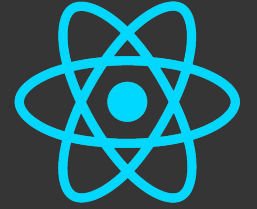
Un método render solo puede devolver un nodo (elemento HTML u otro componente), pero solo uno:

```
class ComponentWithExpressions extends Component {  
  render() {  
    var user = {  
      name: "John", surname: "Connor"  
    };  
    return (<p>Nombre: {user.name}</p><p>Apellido: {user.surname}</p>);  
  }  
}
```

A large red prohibition sign (a circle with a diagonal slash) is drawn over the code block, indicating that the provided code is invalid because it returns multiple JSX elements.

El render de un componente sólo puede devolver una llamada a una función: `React.createElement()`

REACT: JSX

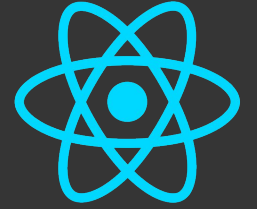


Sin embargo, esto nos ofrece una limitación... ¿cómo pintamos una lista?

Son necesarios 2 componentes:

- Padre que será el contenedor y en su render incluiremos tantos
- Hijo(s) como sean necesarios

REACT: JSX



```
import React from 'react';

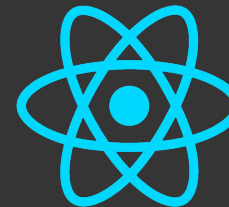
const Item = function () {
  return (<div>Item</div>);
}

const List = function () {
  let items = [];

  for (let i = 0; i < 100; i++) {
    items.push(<Item />);
  }

  return (<div>{items}</div>);
}
```

REACT: JSX



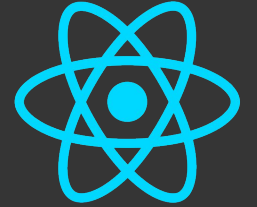
✖ ▶Warning: Each child in a list should have a unique "key" prop.

vendors~main.9e2083c...d9.bundle.js:108304

Check the top-level render call using `<Styled(div)>`. See <https://fb.me/react-warning-keys> for more information.

```
in useState(withState(withHandlers(lifecycle(branch(BaseComponent)))))  
in Unknown  
in Unknown  
in Unknown  
in Unknown (created by Context.Consumer)  
in _default (created by Layout)  
in div (created by Context.Consumer)  
in Styled(div) (created by Panel)  
in Panel (created by Layout)  
in div (created by Context.Consumer)  
in Styled(div) (created by Main)  
in div (created by Context.Consumer)  
in Styled(div) (created by Main)  
in Main (created by Layout)  
in Layout (created by Context.Consumer)  
in WithTheme(Layout) (created by ResizeDetector)  
in ResizeDetector  
in div (created by Context.Consumer)  
in Styled(div)  
in Unknown  
in Unknown (created by ResizeDetector)  
in ResizeDetector  
in Unknown  
in Unknown (created by Manager)  
in ThemeProvider (created by Manager)  
in Manager (created by Context.Consumer)  
in Location (created by QueryLocation)  
in QueryLocation (created by Root)  
in LocationProvider (created by Root)  
in HelmetProvider (created by Root)  
in Root
```

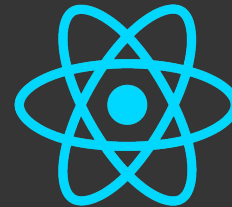
REACT: JSX



Para realizar una correcta gestión del DOM y mantener su eficiencia a la hora de realizar las actualizaciones inteligentes, necesita identificar los componentes idénticos y para ello necesita dar un identificador (key) que tiene que ser único dentro del array.

```
const Lista = function () {  
  var items = [];  
  for (var i = 0; i < 100; i++) {  
    items.push(<Item key={i}/>);  
  }  
  return (<div>  
    {items}  
  </div>  
);  
}
```

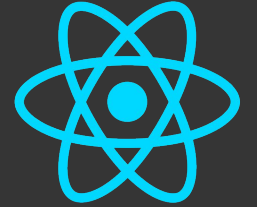
REACT: JSX



- Los componentes aceptan propiedades
- Se les pueden pasar como atributos en su etiqueta JSX
- Los reciben dentro de un objeto props que React crea uniendolos todos

```
class HelloWorld extends Component {  
  render() {  
    return <h1>Hola {this.props.name}</ h1>  
  }  
}
```

REACT: JSX



Podemos pasar como props cualquier cosa: texto, números, arrays, componentes, funciones... JSX === JS

Todas las props se pasan entre llaves:

```
<Component propNumber={0} />
```

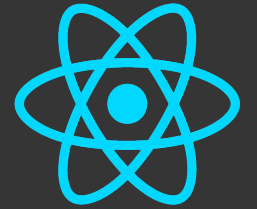
```
<Component propArray={[1, 2, 3]} />
```

Salvo el texto que va entre comillas:

```
<Component propText="Hello world" />
```

Esto nos permite crear la UI y reutilizar los componentes en diferentes módulos de nuestra aplicación

REACT: JSX



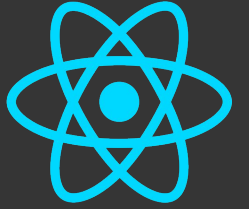
Los componentes:

- Pueden acceder a las props
- No pueden modificarlas

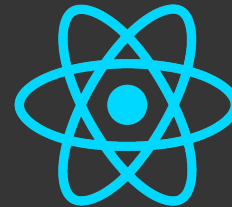
El componente padre es el único responsable de las props pasadas a sus hijos



REACT: JSX



REACT: Estado

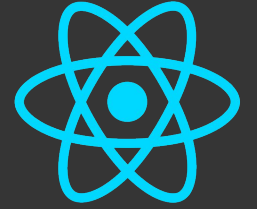


Los componentes de React tienen un estado interno al que pueden acceder:

- Es un acceso de solo lectura, no pueden modificarlo
- Para definir su estado inicial, lo hacen desde el constructor
- Los componentes funcionales no tienen estado

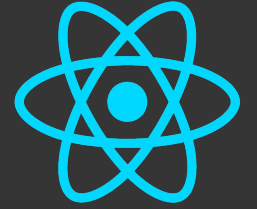
```
class MyComponent extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = { name: 'Jhon' };  
  }  
}
```

REACT: Estado



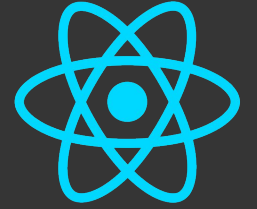
```
export class Count extends Component {  
  constructor(props) {  
    super(props);  
    this.state = {count: 0};  
  },  
  render() {  
    return ( <div>Count: {this.state.count} </div> );  
  }  
};
```

REACT: Estado



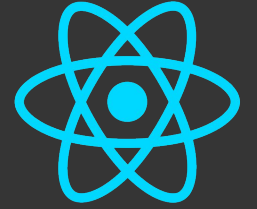
- El estado, es el modelo de datos del componente
- Para modificar el estado, los componentes de React tienen que hacer uso de un método `setState`
- Este método realiza un “merge” entre el estado anterior y el nuevo estado
- De este modo no es necesario pasar todos los datos del estado para modificar uno de ellos
- Una vez ha modificado el estado con la nueva propiedad, el componente fuerza un render
- En el render los datos del estado se pueden utilizar para:
 - Incluir su valor directamente en la salida del componente
 - Convertirlos en props que le pasamos otro componente hijo

REACT: Estado



```
export class Count extends Component {  
  constructor(props) {  
    super(props);  
    this.state = {count: 0};  
  }  
  plusOne() {  
    this.setState({count: this.state.count + 1})  
  }  
  render() {  
    return ( <div>Count: {this.state.count} </div> );  
  }  
};
```

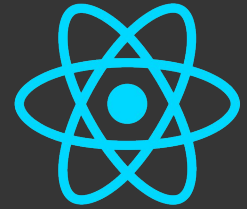
REACT: Estado



La filosofía de React nos dice que debemos tener el menor número de componentes con estado:

- Sin estado siempre mejor que con estado
- Es decir, muchos componentes “tontos” y pocos “listos”
- Los componentes con estado, normalmente serán los que tengan más carga lógica
- Los componentes sin estado solo tienen que pintar correctamente las props que reciben

REACT: Eventos

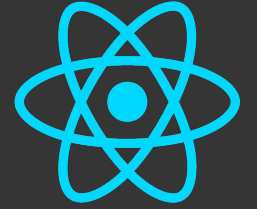


Para poder hacer interactiva nuestra aplicación, tenemos que poder capturar los eventos sobre los componentes.

- Se pueden controlar los eventos de click, cambio de valor...
- Mediante una prop `onXxxx`
- El valor de la prop es la función a ejecutar tras el evento



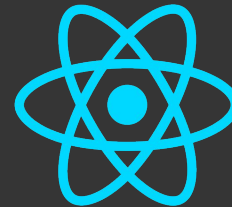
REACT: Eventos



```
render() { return (  
  <button onClick={ this.handleClick }>Click here</button> )  
}
```

```
render() { return (  
  <button onClick={ () => { alert('CLICKED!') } }>Click here</button> )  
}
```

REACT: Estado



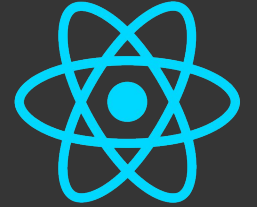
Cuando utilizamos una referencia a una función, el contexto de `this` no es el correcto, por lo que:

```
render() { return (  
  <button onClick={ this.handleClick }>Click here</button> )  
}
```

No funcionará salvo que corriamos el contexto para la ejecución de la función pasada.

¿Alguna idea?

REACT: Eventos



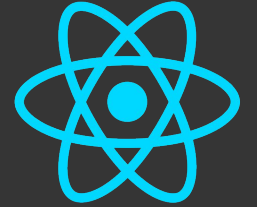
```
import React, { Component } from 'react';

class EventHandler extends Component {
  handleClick(e) {
    alert('Click!');
    this.setState({ clicked: true })
  }

  render() {
    return <button onClick={ this.handleClick }>Click me</button>;
  }
}

export default EventHandler;
```

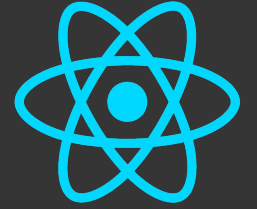
REACT: Estado



El método **bind()** crea una nueva función, que cuando es llamada, asigna a su operador `this` el valor entregado

```
class EventHandler extends Component {  
  constructor() {  
    super()  
    this.handleClick = this.handleClick.bind(this);  
  }  
  handleClick(e) {  
    alert('Click!');  
    this.setState({ clicked: true })  
  }  
  render() {  
    return <button onClick={ this.handleClick }>Click me</ button>;  
  }  
}
```

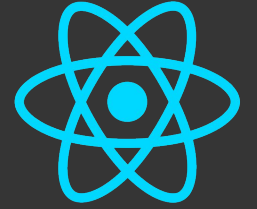
REACT: Eventos



Eventos de ratón:

- `onClick`
- `onDoubleClick` `onMouseDown`
- `onMouseUp`
- `onMouseEnter`
- `onMouseLeave`
- `onMouseEnter`
- `onMouseLeave` `onMouseMove`
- `onMouseOver`
- `onMouseOut`
- `onWheel`

REACT: Eventos



Eventos de teclado:

- onKeyDown
- onKeyPress
- onKeyUp

Eventos del portapapeles:

- onCopy
- onCut
- onPaste

Eventos de foco:

- onFocus
- onBlur

Eventos de formulario:

- onChange
- onInput
- onSubmit