React

Nacho Viano

Índice

1. ¿Qué es un framework?

Un Framework, también traducido como Marco de Trabajo, en el caso del front-end se suelen tratar de estructuras vacía que se utilizan para crear una app. Existen distintos frameworks para diferentes lenguajes de programación entre los que podemos destacar para Javascript:

- React: creado por Facebook y mantenido por la comunidad. *(librería)
- Angular: creado por Google, que lanza una nueva versión cada 6 meses.
- Vue: Creado por Evan You, ex-trabajador de Google a partir de Angular JS.

Los frameworks mantienen un estándar a la hora de programar ya sea de forma independiente o en equipo. Sus principales características son:

- Buenas prácticas: buen uso de patrones de diseño y convenciones. Fácil de mantener y escalable.
- DRY: Don't Repeat Yourself, reutilizar código ya desarrollado evitando tener que volver a escribirlo.
- Al contrario que las librerrías, proporcionan herramientas para la gestión casi completa de un proyecto.

2. ¿Qué es React y para qué sirve?

Aunque muchos lo consideran como un Framework, React es una “librería” muy extensa de Javascript orientada al desarrollo de aplicaciones web. No confundir con React Native, framework para desarrollo multiplataforma y habitualmente utilizado para iOS y Android.

React fue desarrollada por Facebook para mejorar el rendimiento de su plataforma, pero fue liberada y ahora es mantenida por la comunidad de desarrolladores.

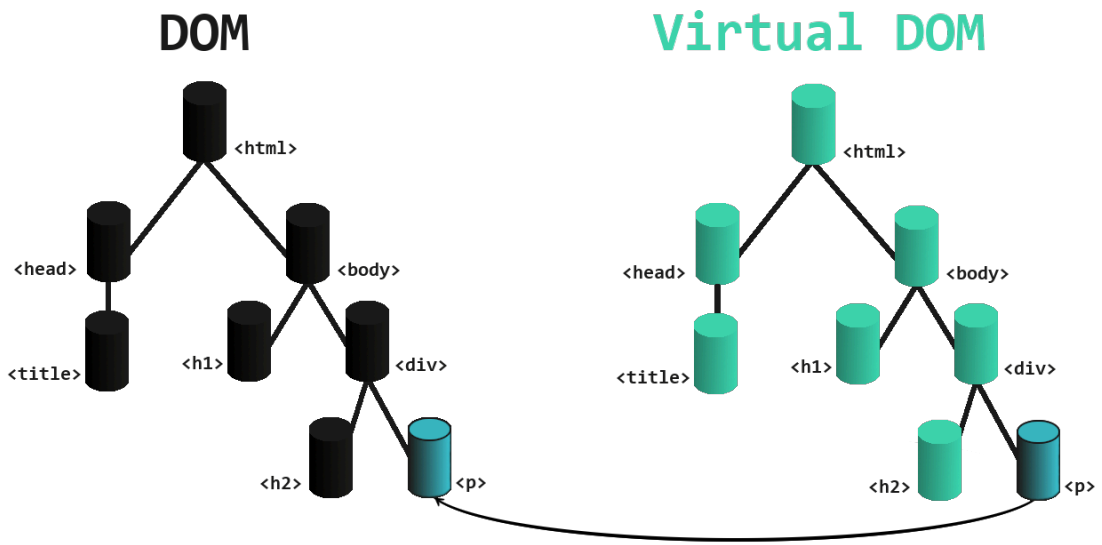
Estos son algunos de los sitios que utilizan React:

- Facebook
- Instagram
- Airbnb
- Netflix
- WhatsApp Web

A diferencia de jQuery, React presenta un comportamiento parecido al de un framework, por eso en ocasiones se le denomina como tal.

3. El Virtual DOM

El Virtual DOM es una copia simplificada del árbol DOM. Tiene como objetivo mejorar el rendimiento del navegador reduciendo los costes ocasionados cuando se modifica un nodo.



El Virtual DOM recopila los cambios realizados en el mismo y los va cargando directamente al DOM evitando tener que redibujarlo por completo.

4. Importando React en un proyecto

React se puede importar en un proyecto del mismo modo que jQuery:

- Creamos un archivo HTML y creamos una etiqueta con un id.
- Vamos al sitio web de React: es.reactjs.org/docs/add-react-to-a-website.html y copiamos los scripts.
- Importaremos también Babel si queremos utilizar JSX.
- Podremos crear elementos y renderizarlos dentro del div previamente incluido en el HTML.

```
<body>
     <div id="root"></div> <!-- Toda nuestra app estará dentro de este div -->

    <!-- Importamos react -->
    <script src="https://unpkg.com/react@17/umd/react.development.js" crossorigin></script>
    <script src="https://unpkg.com/react-dom@17/umd/react-dom.development.js" crossorigin></script>
    <!-- Importamos Babel para procesar JSX -->
    <script src="https://unpkg.com/babel-standalone@6/babel.min.js"></script>

    <script type="text/babel">
        const root = document.querySelector("#root");
        const title = <h1>Hola Futuro</h1>;
        ReactDOM.render(title, root);
    </script>
</body>
```

5. ¿Qué es Babel?

Babel es una utilidad que transforma el código JavaScript moderno a una versión retrocompatible con navegadores antiguos.

En el caso de React, Babel se utiliza para compilar el código JSX (JavaScript XML).

En este recurso podemos ver cómo se traduce en JavaScript.

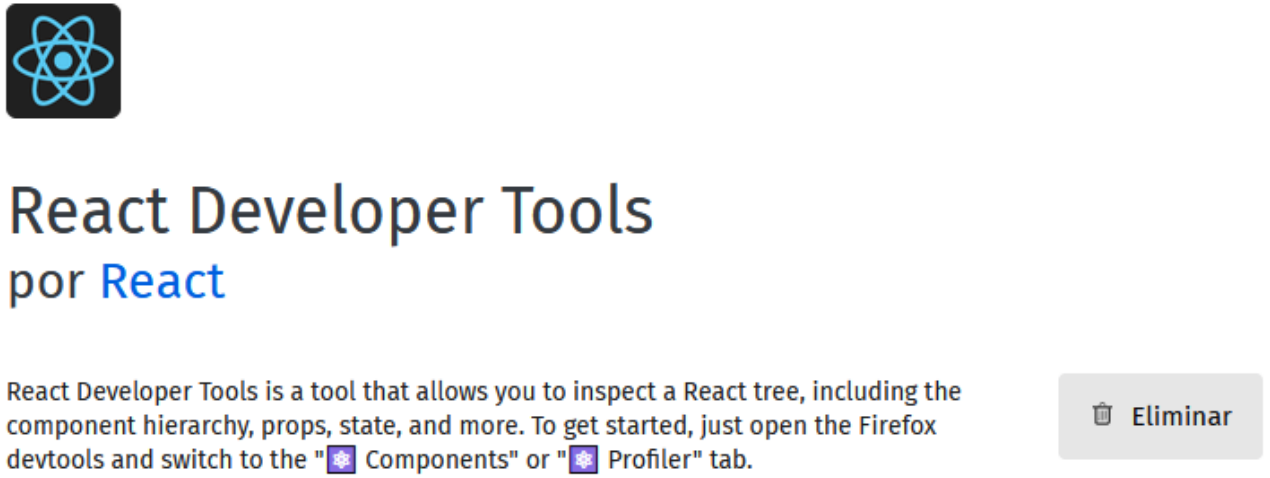

```jsx
function hello() {
  return <h1>Esto es syntactic sugar</h1>;
}
```

```jsx
function hello() {
  return /*#__PURE__*/React.createElement("h1", null, "Esto es syntactic sugar");
}
```

6. Herramientas útiles

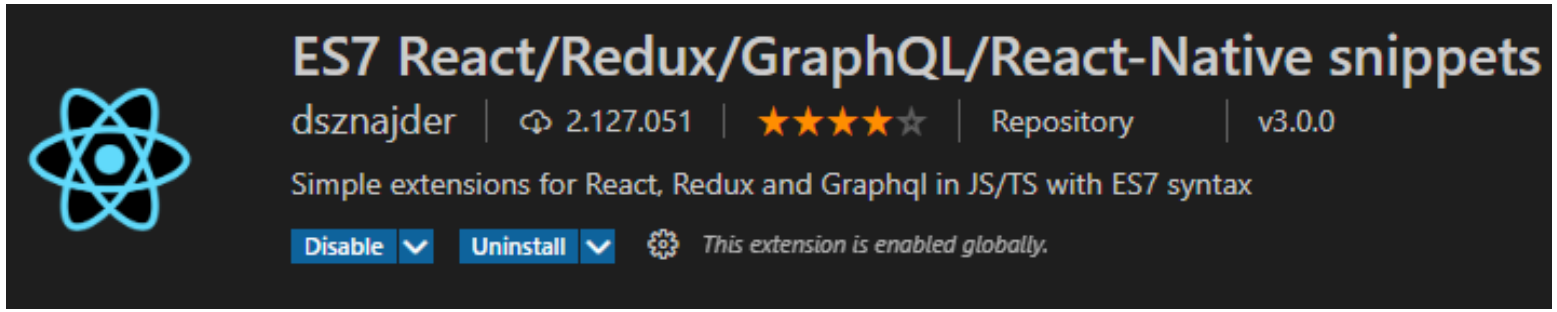
Existen dos herramientas bastante útiles disponibles para Mozilla Firefox y Google Chrome que nos ayudarán a trabajar con React:

- React Developer Tools



Snippets de React en VS Code:

- ES7 React/Redux/GraphQL/ React-Native snippets.



7. Proyecto React

Preparar proyecto

Siguiendo estos pasos y con Nodejs instalado, arrancaremos un nuevo proyecto React:

- Desde una terminal ejecutamos npx create-react-app nombre-proyecto, este comando creará un nueva carpeta con todo el contenido del proyecto.
- Accedemos a dicha carpeta con el comando cd nombre-proyecto.
- Y arrancamos el servidor virtual con el comando npm start

Al ejecutar el último comando nos abrirá la pantalla del navegador con una página de ejemplo del proyecto que hemos creado en la ruta http://localhost:3000

7. Proyecto React

Estructura de un proyecto React

- node_modules: Carpeta de Nodejs que incluye todos los módulos del proyecto.

- public: Carpeta que contiene la estructura mínima de la aplicación:
  - logos: imágenes que se usan cuando se genera la PWA.
  - manifest.json: Es el documento que configura la PWA.
  - index.html: Debe contener el nodo raíz de la aplicación.

- src: Contiene los archivos con los que trabajaremos en React
  - App.js, App.css: Componente y hoja de estilo generados automáticamente con el proyecto.
  - Index.js e index.css: Componente principal de la aplicación que carga el proyecto en el DOM.
  - setupTest.js: archivo que se ejecuta antes de comenzar las pruebas unitarias.

8. Componentes

Un componente en React es el bloque básico con el que se construyen las apps. Permiten reusabilidad y modularidad del código entre otras ventajas. Se pueden crear con Javascript o utilizando JSX (syntactic sugar). React without JSX.

Existen dos formas de definir los componentes en React: con clases o con funciones.

```
export default class Card extends Component {
  render() {
    return (<div>
              <h1>Soy un componente</h1>
            </div>)
  }
}
```

Class Component

```
export default function Card() {
  return (
    <div>
      <h1>Soy un componente de prueba</h1>
    </div>
  )
}
```

Functional Component

Hay una tendencia creciente en el uso de componentes funcionales por su simplicidad.

8. Componentes

Crear un componente

Podemos crear varios componentes por archivo aunque suele ser habitual tener un componente por archivo.
En el segundo caso, el nombre del archivo también será CamelCase igual que el nombre de la función.

```
function PrimerComponente() {
  return (
    <div>
      <h3>Entrada 16</h3>
      <small>16 de Agosto de 2025</small>
      <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit.
        Adipisci inventore qui incidunt temporibus dicta suscipit!</p>
    </div>
  )
}
```

8. Componentes

Utilizar componente

Una vez creamos un componente en React, podemos reutilizarlo en cualquier punto. Tendremos que importarlo en el componente padre para utilizarlo como si se tratara de una una etiqueta HTML.

```
import "./App.css";
import PrimerComponente from './PrimerComponente';

function App() {
  return (
    <div>
      {/* Cargamos el componente como una etiqueta más. */}
      <PrimerComponente />
    </div>
  )
}
```

8. Componentes

CSS en los componentes

Una práctica común es crear un archivo css para cada componente. Para que el componente se vea afectado por las reglas, debemos importarlo en dicho componente o en un componente superior.

```
import "./PrimerComponente.css";

function PrimerComponente() {
  return (
    <div>
      <h3>Entrada 16</h3>
      <small>16 de Agosto de 2025</small>
      <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit.
        Adipisci inventore qui incidunt temporibus dicta suscipit!</p>
    </div>
  )
}
```

8. Componentes

Contenido

Un componente puede retornar un único nodo que englobe al resto. Hasta ahora hemos utilizado una etiqueta para ello, pero hay otros métodos.

```jsx
import FirstComp from "./FirstComp";

export default function App() {
  return (
    <div>
      <FirstComp />
      <p>Lorem, ipsum dolor.</p>
    </div>
  )
}
```

div

```jsx
import {Fragment} from "react";
import FirstComp from "./FirstComp";

export default function App() {
  return (
    <Fragment>
      <FirstComp />
      <p>Lorem, ipsum dolor.</p>
    </Fragment>
)}
```

Fragment

```jsx
import FirstComp from "./FirstComp";

export default function App() {
  return (
    <>
      <FirstComp />
      <p>Lorem, ipsum dolor.</p>
    </>
  )
}
```

Fragment abreviado

8. Componentes

Props

Las props son argumentos que se les puede “pasar” a los componentes. De esta forma, enviamos valores desde el componente padre al componente hijo.

```jsx
import MyComp from "./MyComp";

export default function App() {
  return (
    <> {/* Enviando props al hijo como atributos */}
      <MyComp name="Christopher" lastname="Nolan"/>
      <p>Párrafo de App.js</p>
    </>
  )
}
```

Se envían las props desde el componente padre

```jsx
export default function MyComp(props) {
  return (
    <div>
      <h3>Entrada 16</h3>
      <small>Escrito por {props.name} {props.lastname}</small>
      <p>Lorem ipsum dolor sit amet.</p>
    </div>
  )
}
```

El hijo recibe un objeto por parámetro con ellas

9. JSX

¿Qué es JSX?

Se trata de una extensión de la sintaxis de JavaScript combinada con la estructuración de XML, un lenguaje de marcado similar a HTML.

```
const paragraph = <p>Esto es un párrafo JSX</p>
```

JSX genera etiquetas HTML y aunque no sea requerido, nos ahorra mucho trabajo para crear la estructura directamente desde JavaScript.

9. JSX

Imprimir datos utilizando JSX

Lo habitual es utilizar constantes para trabajar la información de una aplicación.

Para ejecutar código JavaScript dentro de JSX, tenemos que rodearlo con llaves {}.

```
export default function MyComp() {

  const post = {
    title: "Magna occaecat",
    date: "22 de Agosto de 2026",
    content: "Sint laborum sit do ut culpa aliquip."
  }

  return (
    <div> {/* Podemos imprimir variables entre llaves */}
      <h3>{post.title}</h3>
      <small>{post.date}</small>
      <p>{post.content}</p>
    </div>
  )
}
```

9. JSX

Condicionales en JSX

Podemos gestionar los condicionales fuera del return y posteriormente añadir los elementos resultantes, pero en ocasiones es muy práctico hacer la validación directamente en el JSX.

```jsx
export default function Component() {
    const isAdmin = true;

    // Si isAdmin es true,
    // se procesará la segunda parte.
    return  (
        <div>
            {isAdmin && <button>Abrir dashboard</button>}
        </div>
    )
}
```

Condición simple con &&

```jsx
export default function Component() {
    const isPremium = false;

    return  (
        <div>
            {isPremium ? (
                <a href="premium.html">Ver</a>
            ) : (
                <p>Tienes que aumentar tu plan.</p>
            )}
        </div>
)}
```

Condición ternaria con ? y :

9. JSX

Estilos en JSX

JSX hace un uso peculiar de la propiedad style, debe configurarse con la notación de llaves y utiliza el nombre de las propiedades de JS con camelCase.

```
export default function Component() {
    const sentence = "Lorem ipsum dolor sit amet " +
        "consectetur adipisicing elit. Temporibus, corporis.";

    return  (
        <p style={{color: "red", fontSize: 18}}>
            {sentence}
        </p>
    )
}
```

Creación de estilos en el atributo

```
export default function Component() {
    const sentence = "Lorem ipsum dolor sit amet " +
        "consectetur adipisicing elit. Temporibus, corporis.";
    const style = {color: "red", fontSize: 18};
    return  (
        <p style={style}>
            {sentence}
        </p>
    )
}
```

Crear objeto y asignarlo al atributo

9. JSX

Clases en JSX

React reemplaza el uso del atributo class por el de className, además permite el uso de llaves JSX para insertar variables o crear condiciones.

```
export default function Component() {
    const sentence = "Lorem ipsum dolor sit amet " +
        "consectetur adipisicing elit. Temporibus, corporis.";

    return  (
        <p className="mainParagraph">
            {sentence}
        </p>
    )
}
```

Asignar una clase con className

```
export default function Component() {
    const sentence = "Lorem ipsum dolor sit amet " +
        "consectetur adipisicing elit. Temporibus, corporis.";
    const isTitle = true;
    return  (
        <p className={isTitle ? "title" : "paragraph"}>
            {sentence}
        </p>
    )
}
```

Utilizar una condición ternaria en clases

9. JSX

Recorrer elementos en JSX

Para generar un nodo para cada elemento de un array directamente en JSX, podemos utilizar la función map.

```
export default function Component() {
    const shoppingList = ["Bread", "Milk",
                          "Soap", "Eggs", "Pasta"];

    return  (
        <ol>
            {shoppingList.map(item => <li>{item}</li>)}
        </ol>
    )
}
```

Recorriendo un Array de strings

```
export default function Component() {
    const users = [
        {name: "John", age: 46}, {name: "Jane", age: 52},
        {name: "Sarah", age: 24}
    ];

    return  (
        <>
            {users.map(item => {
                return  <ol>
                            <li>{item.name}</li>
                            <li>{item.age}</li>
                        </ol>
            })}
        </>
)}
```

Recorriendo un Array de objetos

Índice - Parte 2

10. Propiedades por defecto y validación de tipos

Cuando se reciben props por React, éstas pueden llegar vacías, para evitarlo podemos definir valores por defecto con defaultProps. Además, podemos validar el tipo de dato que recibimos en cada elemento. Si no se cumple, la consola nos avisará. Los PropTypes son string, number, array, bool, func, object y any. Podemos añadir isRequired junto al tipo para que sea requerido.

```
export default function Component(props) {
    return  (
        <>
            <p>{props.name}</p>
            <p>{props.lastname}</p>
            <p>{props.age}</p>
        </>
    );
}

Component.defaultProps = {
    name: "Sin nombre",
    lastname: "",
    age: 0
};
```

```
import PropTypes from "prop-types";

export default function Component(props) {
    return  (<>
                <p>{props.name}</p>
                <p>{props.lastname}</p>
                <p>{props.age}</p>
            </>);
}

Component.propTypes = {
    name: PropTypes.string.isRequired,
    lastname: PropTypes.string,
    age: PropTypes.number
};
```

10. Propiedades por defecto y validación de tipos

Los eventos en React son casi idénticos a los de JavaScript, la mayor diferencia reside en su notación camelCase y que estos reciben una función en formato JSX.

RATÓN	TECLADO	PORTAPAPELES	FORMULARIOS
onClick	onKeyDown	onCopy	onChange
onContextMenu	onKeyUp	onCut	onInput
onDoubleClick	onKeyPress	onPaste	onInvalid
onDrag			onReset
onDrop			onSubmit
onMouseUp			
onMouseDown			
onMouseEnter			
onMouseLeave			

Puedes ver todos los eventos disponibles para React aquí: https://es.reactjs.org/docs/events.html

12. Eventos en React

Uso de eventos

Existen dos métodos que se utilizan comúnmente para disparar funciones al utilizar un evento:

```
function App(){
    const verMensaje = (evento) => {
        alert("Bienvenid@ a Releevant!");
        //captura del evento desde parámetros:
        console.log("Botón pulsado: " + evento.target.innerText);
    }

    return (
        <>
            <button onClick={verMensaje}>Lanzar alerta</button>
        </>
    )
}
```

Procedimientos sin retorno

```
function App(){
    const verMensaje = (nombre) => {
        return (evento) => {
            //Recuperamos el parámetro recibido:
            alert("Bienvenid@ a Releevant!" + nombre);
            //captura del evento desde parámetros:
            console.log("Botón pulsado: " + evento.target.innerText);
        }
    }

    return (
        <>
            <button onClick={verMensaje("Elena")}>Lanzar alerta</button>
        </>
    )
}
```

Funciones que reciben parámetros

13. Hooks

¿Qué es un Hook?

Un Hook es una función que permite “conectarse” a características de React. Cambiar el estado de un componente, hacer determinadas acciones cuando el componente se carga. Compartir información entre componentes a través del contexto...

Aparecieron a partir de la versión 16.8 y el motivo de su existencia es permitir generar Functional Components que puedan manejar cambios de estado, algo que hasta entonces tenía que hacerse con Class Components.

Estas funciones deben colocarse dentro un componente funcional y no pueden estar anidadas dentro de ningún condicional o función.

Los Hooks mas comunes son:
- useState
- useEffect
- useContext

13. Hooks

useState

Devuelve dos valores: El estado actual y una función que lo actualiza.

El primer dato nos servirá para “leer” o acceder al estado, y el segundo para modificarlo.

A la función useState() se le asigna un valor inicial que puede ser cualquier tipo de dato.

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

function Counter() {
  // Declaramos un nueva variable de estado que será “count”
  const [count, setCount] = useState(0);

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

13. Hooks

Combinando Hooks y eventos

En la práctica, useState suele ir de la mano con los eventos.

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

function Messager() {
  const [message, setMessage] = useState("");

  const changeMessage = e => setMessage(e.target.value);

  return (
    <>
      <h3>Hooks + Eventos</h3>
      <input type="text" onChange={changeMessage} placeholder="Deje su mensaje"/>
      <p>{message}</p>
    </>
  );
}
```

14. Procesar formularios

A la hora de procesar un formulario es importante tener en cuenta que se van a utilizar tantos hooks de estado como inputs haya, de esta manera guardaremos la información.

En cada evento change de los inputs, actualizaremos cada estado individualmente.

En el evento submit evitaremos el comportamiento por defecto y procesaremos todos los estados según el objetivo que tenga el formulario.

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

function Login() {
    const [username, setUsername] = useState("");
    const [password, setPassword] = useState("");

    const handleUsername = e => setUsername(e.target.value);
    const handlepassword = e => setPassword(e.target.value);

    const handleSubmit = e => {
        e.preventDefault();
        if(username === "root" && password === "1234") {
            alert("Sesión iniciada");
        } else {
            alert("Error al iniciar sesión.");
        }
    };

    return (
        <form onSubmit={handleSubmit}>
            <input type="text" onChange={handleUsername}/>
            <input type="password" onChange={handlepassword}/>
            <input type="submit" value="Log in"/>
        </form>
    );
}
```

15. Comunicación de componente hijo a padre

Con el uso del Hook *useState* se puede modificar el estado del componente Padre a través del hijo.

```jsx
import {useState} from 'react';
import './App.css';

import NuevaComida from './NuevaComida';

function App() {
  const [compra, setCompra] = useState(['Galletas', 'Pan', 'Queso']);

  return (
    <>
      {/* Le pasamos al componente hijo la función que cambia el estado del
      componente compra: */}
      <NuevaComida setCompra={setCompra} />
      <ul>
        {
          compra.map((lista)=>{
            return <li key={lista}>{lista}</li>
          })
        }
      </ul>
    </>
  );
}

export default App;
```

El Componente padre carga al hijo y le pasa el set

```jsx
import {useState} from 'react';

function NuevaComida({setCompra}) {
    const [elemento, setElemento] = useState('');

    const handleElemento = (evento) => {
        setElemento(evento.target.value);
    }

    const handleCompra = (evento) => {
        evento.preventDefault();
        setCompra(compra => [...compra, elemento]);
        setElemento('');
    }

    return (
        <form onSubmit={handleCompra}>
            <input type="text" value={elemento} placeholder="Añade mas comida"
            onChange={handleElemento}/>
        </form>
    )
}

export default NuevaComida;
```

El hijo recibe el set y modifica al padre