

DISEÑO DE INTERFACES WEB
TÉCNICO EN DESARROLLO DE APLICACIONES WEB

Bibliotecas para la creación de interfaces (I)

ÍNDICE

| | |
|---|----|
| / 1. Introducción y contextualización práctica | 3 |
| / 2. Patrones de diseño. MVC | 4 |
| / 3. Introducción a React JS | 4 |
| / 4. Herramientas recomendadas para React.js | 5 |
| / 5. Componentes | 6 |
| / 6. <i>Props</i> | 7 |
| / 7. Estados en componentes | 8 |
| / 8. Caso práctico 1: “Diseño de una app con React (I). Implementar código de componentes reutilizable” | 9 |
| / 9. Renderizando componentes | 10 |
| / 10. Composición de componentes | 11 |
| / 11. Caso práctico 2: “Creación de componentes con acceso y modificación del estado” | 12 |
| / 12. Resumen y resolución del caso práctico de la unidad | 13 |
| / 13. Bibliografía | 13 |
| / 14. Webgrafía | 13 |

OBJETIVOS

Utilizar y valorar distintas aplicaciones para el diseño de documentos web.

Utilizar marcos, tablas y capas para presentar la información de manera ordenada.

Crear y utilizar plantillas de diseño.

Interpretar guías de estilo.

/ 1. Introducción y contextualización práctica

React.js es utilizado ampliamente para el desarrollo en *front-end* y será donde nos centremos en este capítulo. Ahora bien, también puede ser utilizado para el desarrollo de aplicaciones Android e iOS, en aplicaciones de escritorio, modelado de realidad virtual, e incluso en *back-end*.

Los **componentes** son el elemento esencial de React.js, estos están formados por tres pilares: en primer lugar, la estructura del componente (implementado en lenguaje JSX), los estilos del componente que permiten configurar la apariencia del mismo, y finalmente, podemos hablar del comportamiento.

En este capítulo, analizaremos el funcionalmente básico de la librería React JS y de las herramientas necesarias para su puesta en marcha. En concreto, hablamos de Node.js, una librería que resultará clave en esta unidad, así como en posteriores temas, para el uso de las librerías de desarrollo de interfaces web que se van a exponer.

Escucha el siguiente audio donde planteamos la contextualización práctica de este tema, encontrarás su resolución en el apartado Resumen y resolución del caso práctico.



Fig. 1. React. CSS + HTML5 +JS.



Audio intro. "Características y ventajas de React.js"
<https://bit.ly/3o7NXQH>



/ 2. Patrones de diseño. MVC

React JS es una librería de código abierto creada por **Facebook** e implementada sobre **JavaScript**, la cual permite el diseño e implementación de interfaces web. Se trata de una de las librerías más importantes en la actualidad, puesto que permite mejorar el diseño de interfaces web de una forma muy sencilla.

En estos momentos, el desarrollo de JavaScript está muy presente en los patrones de desarrollo MVC (Modelo-Vista-Controlador). Los patrones de diseño de *software* consisten en unos “esqueletos” ya probados y validados que permiten el diseño y desarrollo de nuevo *software* esquematizado de una forma más ágil y eficaz.

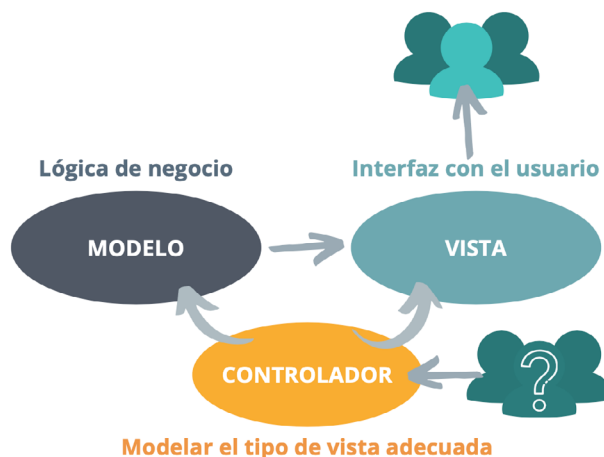


Fig 2. Modelo-Vista-Controlador.

Uno de los patrones de diseño más extendidos es el llamado Modelo-Vista-Controlador (MVC), el cual divide el código en tres partes claramente diferenciadas que tendrán una función específica.

- **Modelo.** Es la parte que se encarga del manejo de todos los datos, por lo tanto, proporcionará los mecanismos oportunos para la recuperación y modificación de los mismos. El modelo representa la lógica de negocio.
- **Vista.** Estos componentes son los que permiten al usuario una interacción directa con la aplicación o sitio web, por lo tanto, las vistas son las interfaces en las que tiene lugar la interacción entre el usuario y el programa o aplicación. **Podemos encontrar dos opciones, la primera desarrollada por Google (angular.js) y la segunda, por Facebook e Instagram (React).**
- **Controlador.** Finalmente, los controladores son los elementos que permiten seleccionar el tipo de vista adecuada para cada usuario. Se encargan de analizar la petición del usuario y redirigirla para poder generar una respuesta adecuada que será enviada de nuevo al usuario. Podemos decir que este elemento está entre el modelo y la vista, y se encarga de modelar que la interacción y extracción de la información sea la correcta para cada ocasión.

/ 3. Introducción a React JS

React.js se encarga del *front-end* y busca conseguir una experiencia de usuario similar a la de las aplicaciones de escritorio, de esta forma, quedan limitadas todo lo posible las interacciones entre el navegador y el servidor.



Fig. 3. Logotipo React.



Será **React** el que se encargue de actualizar los datos en la capa visual cuando estos cambien. Esta librería se basa en la implementación de componentes **reutilizables**, es decir, React.js no separa la implementación de un proyecto en capas por su lenguaje de programación, sino que lo hace dividiendo todo el desarrollo en componentes que puedan ser reutilizables.

Para ponerlo en funcionamiento, en primer lugar, se realiza la descarga de la librería desde el sitio oficial en el siguiente [enlace](#). Se recomienda realizar la descarga del Kit de inicio, el cual incluye: React.js y JSX *Transformer*, el compilador de JSX a JavaScript y, además, algunos proyectos de ejemplo.



Fig. 4. Interfaz sitio oficial React.

Tras realizar la descarga del paquete de herramientas, debemos descomprimirlo y colocarlo en una ruta de tu equipo fácilmente recordable, puesto que luego será necesario acceder a ella a través de línea de comandos para la ejecución de algunas instrucciones.

Ahora bien, para indicar en un fichero HTML que se deben incluir las dos librerías descargadas, se ha de colocar en la sección <head> el siguiente fragmento de código:

```
<script src="build/react.js"></script>
<script src="build/JSXTransformer.js"></script>
```

Código 1. Inspección de bibliotecas React.js y JSXTransformer.js.

/ 4. Herramientas recomendadas para React.js

Desde la sección “**Herramientas recomendadas**” de la página oficial de React.js, podemos encontrar una selección de aplicaciones útiles. En este caso, nos vamos a centrar en **Create React App**. Es una de las herramientas más utilizadas para crear proyectos “de una sola página” en React.js.

Las sentencias necesarias para poder instalarlo a través de línea de comandos y crear un proyecto al mismo tiempo son las que se muestran a continuación:

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

Código 2. Creación de una nueva aplicación con React.js. Video?

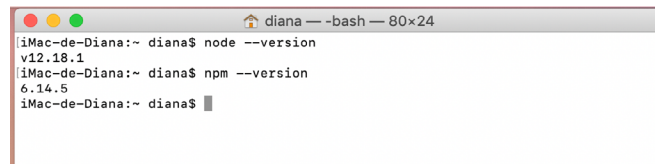
El comando **npx** es un comando de **Node.js** que, como vimos en el tema 5 de este módulo, es una tecnología en JavaScript que permite la instalación de programas implementados en lenguaje de programación JavaScript. Vamos a recordar cómo realizar la instalación de Node.js:

La descarga del paquete está disponible desde su sitio web: <https://nodejs.org/en/download/>.

Para realizar la instalación de Node.js en cualquier sistema operativo, tras la descarga del ejecutable correspondiente, basta con hacer clic sobre el ejecutable y se abrirá un instalador. El proceso de instalación es muy sencillo. Simplemente, se pulsará *Next* hasta completar la instalación. No necesita ninguna configuración específica.

Para comprobar si la instalación se ha realizado de forma correcta, se utilizan los comandos:

- **node --version.** Tras realizar el proceso de instalación anterior, para confirmar el resultado satisfactorio, se ejecuta esta línea por comandos y el resultado mostrará la versión de Node.js instalada.
- **npm --version.** NPM es un gestor de paquetes de JavaScript, si al ejecutarlo se muestra una versión, está instalado. Con la instalación de Node.js se instala también este gestor. Si se desea actualizar la versión, basta con ejecutar por línea de comandos **npm install -g npm**



```
iMac-de-Diana:~ diana$ node --version
v12.18.1
iMac-de-Diana:~ diana$ npm --version
6.14.5
iMac-de-Diana:~ diana$
```

Fig. 5. Comandos de instalación y resultado de la ejecución para Node.js.

/ 5. Componentes

Los componentes son las piezas que forman la interfaz de usuario en React.js. Cada bloque que aparece en la pantalla será un componente. Por lo tanto, podemos decir que serán fragmentos de código que quedan encapsulados bajo un determinado nombre y que pueden reutilizarse desde cualquier parte del código. Los componentes son objetos JavaScript.

Ahora bien, ¿cómo quedan reflejados estos componentes en la interfaz visual de un sitio web? Imaginemos una página en la que tenemos una cabecera compuesta por un icono y un elemento de búsqueda, y un cuerpo con dos párrafos de texto y dos botones, pues bien, cada uno de los elementos descritos sería un **componente**.

En este punto, ya toca mencionar a **JSX (JavaScript Extended)**, que permitirá escribir HTML dentro de JavaScript. Utilizando el compilador que se ha descargado en el kit inicial, será posible realizar la traducción entre JSX y JavaScript.

Por ejemplo, veamos el siguiente código y analicemos su funcionamiento:

```
import React from 'react'
const Button = () => (
  <a href="https://xxx" class="button1">
    Inicio
  </a>
)
export default Button
```

Código 3. Creación de un componente 'modo estático'.



- **import React from 'react'**: permite importar la librería React en el proyecto de desarrollo.
- **export default Button**: permite exportar el componente que se está implementado para poder ser utilizado en otro desarrollo distinto.
- **const Button = () => (...)**: se implementa el código del componente, una función en JavaScript que devuelve código HTML.

Este ejemplo está creado de forma estática, por lo que no recibe ningún parámetro que permita definir su contenido en función de las especificaciones del sistema, por ejemplo, para modificar la cadena de texto que está vinculada al enlace `<a href>`.

Por lo tanto, será deseable que el contenido pueda ser modificado de forma dinámica, y para esto, aparece un nuevo objeto llamado **props** (que estudiaremos a continuación). Este permite recibir los valores que van a ser utilizados en cada momento.



Audio 1. "Tipos de componentes en React"

<https://bit.ly/2ViuOiv>



/ 6. Props

Los denominados **props** son un elemento clave para el desarrollo de cualquier componente. Es un objeto que determina todas las propiedades de un componente y resulta especialmente útil para renderizar los mismos. Además, gracias a su implementación, es posible dotar de dinamismo al código, de lo contrario, los componentes resultantes serían estáticos y podrían reutilizarse en menor medida.

Su sintaxis se muestra a continuación. El uso de la palabra reservada *this* hace referencia al componente sobre el que está referenciado el indicador *props*. Al final, se añade el nombre del atributo que va a ser tomado en cada momento:

```
this.props.nombreAtributo
```

Código 4. Sintaxis básica props.

Un objeto de tipo **props** permite recibir por parámetro diferentes tipos de datos, desde cadenas de texto hasta otros componentes. Pero ¿cómo funciona?

En primer lugar, cuando desde el código JSX se utiliza un componente, se definen ciertos "atributos" de la siguiente forma:

```
const nuevoComponente=<nombreAtributo="...">
```

Código 5. Definición de atributos.

Este valor será recibido por el componente y desde el código JSX, se accede al atributo *nombreAtributo*, para lo cual, se debe indicar su nombre siempre entre llaves `{}`. (El tipo de elemento definido en las etiquetas HTML puede ser el que el usuario elija, se ha escogido `<p>` en este caso como ejemplo).

```
<p>{this.props.nombreAtributo}</p>
```

Código 6. Ejemplo uso props.

En la siguiente figura, podemos ver un diagrama de uso props.

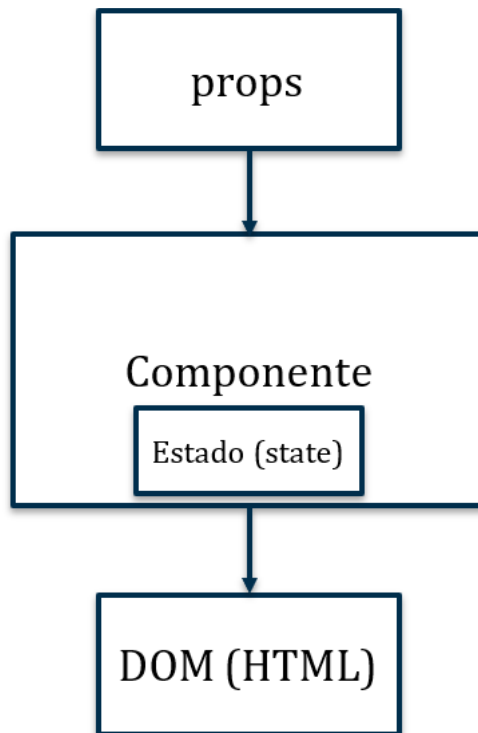


Fig. 6. Diagrama de uso props.

/ 7. Estados en componentes

Uno de los conceptos esenciales cuando hablamos de componentes en React.js es el **estado**. Cada componente va a tener su propio estado que indica cómo está el componente en cada momento (es decir, si el usuario ha interactuado con él, ha introducido algún tipo de texto, pulsado un botón...) y qué datos tiene en cada instante.



Fig. 7. Imagen ilustrativa interfaz.

Este estado “interno” de cada uno de los componentes solo es conocido por el propio elemento y sus elementos hijos. Para poder establecer una comunicación con otros componentes, utilizarán el estado denominado “estado global”.



Vídeo 1. „Análisis de flujo en React.js“
<https://bit.ly/3lnCM4v>





Por ejemplo, si para cambiar la imagen mostrada en una página web a través de un carrusel de fotografías es necesario pulsar un botón, el componente que modela este botón comunicará al estado global que ha sido accionado, y será entonces cuando se comunica al elemento contenedor de las imágenes que debe actualizar su contenido y mostrar la siguiente fotografía que hubiera almacenada en su repositorio de imágenes.

La propiedad que permite acceder al estado de un componente para modificar o consultar su valor es **state**. En concreto, para modificar su valor, la sintaxis que se implementa es la siguiente:

```
this.state={  
  nombreVariable: valor  
}
```

Código 7. Sintaxis cargar el valor estado en React.

De la misma forma, para modificar el valor del estado:

```
this.setState={  
  nombreVariable: nuevoValor  
}
```

Código 8. Sintaxis modificar el valor estado en React

Y para acceder a su valor:

```
this.state.nombreVariable
```

Código 9. Sintaxis acceder al valor estado en React.

/ 8. Caso práctico 1: “Diseño de una app con React (I). Implementar código de componentes reutilizable”

Planteamiento: Se pide adaptar el código estático para la creación de un componente de tipo botón, como el visto en el código 2, a una implementación más dinámica que permita reutilizar este componente en cualquier ocasión, puesto que el definido solo podría utilizarse en los términos descritos.

Además, se ha de implementar el código de creación de al menos dos botones nuevos utilizando el componente reutilizable desarrollado previamente.

Nudo: Para la adaptación de este componente a una versión más dinámica, será necesario utilizar el elemento **props**. Este objeto permite acceder a uno de los atributos que van a ser utilizados para desarrollar cada nuevo botón (código 11) y que utilizarán el componente de referencia creado (código 10).

Desenlace: El código utilizando **props** quedará de la forma:

```
Import React from 'react'  
const Button = props => (  
  <a href={props.url} class={props.class}>  
    {props.content}  
  </a>  
)  
export default Button
```

Código 10. Creación de un componente 'modo dinámico'.



Ahora bien, ya se habría creado el componente reutilizable, pero ¿cómo podemos crear botones usando este componente?

```
<Button  
url="urlA"  
class="buttonA"  
content="BOTÓN A"  
>  
<Button  
url="urlB"  
class="buttonB"  
content="BOTÓN B"  
>
```

Código 11. Creación de botones reutilizando un componente Button.

/ 9. Renderizando componentes

Los componentes React incorporan una función que permite renderizar en el navegador un componente. Esta función recibe el nombre de **render()**.

Como vamos a ver en el código 16 del caso práctico 2, cuando se crea un componente o se actualiza su valor, se llama a la función `render()`, puesto que será aquí donde se implemente el código JSX necesario para modelar el componente que será mostrado en el navegador.

De manera general, el esqueleto de creación de componente y su llamada a *render* quedaría de la forma:

```
React.Component {  
  constructor(...args) {  
    ...  
  }  
  render(){  
    ... JSX  
  }  
}
```

Código 12. Esqueleto de creación de componentes. Constructor + render.

Cuando realizamos el renderizado de componentes, es importante tener en cuenta el tratamiento de **propiedades**, es decir, los parámetros que un componente puede enviar a JSX para el modelado del código HTML que será mostrado en el navegador del usuario.

```
React.Component {  
  constructor(...args) {  
    ...  
  }  
  render(){  
    let opciones = ['A','B','C']  
    return <{opciones}>  
  }  
}
```

Código 13. Uso de render. Ejemplo.



Como se puede ver en el código anterior, a través de los corchetes `[]` es posible definir una lista de valores. Posteriormente, para acceder a su valor, se coloca el nombre de esta lista entre llaves `{}`, y dado que estamos modelando en HTML, será necesario insertarlo entre los símbolos `<>` propios de este lenguaje de desarrollo web.



Vídeo 2. "Análisis de código completo.
Constructor + render en React.js"
<https://bit.ly/37mCPsh>



/ 10. Composición de componentes

La **composición** de componentes ocurre cuando la **implementación de un componente implica el uso de otros componentes en su salida**. Por lo tanto, al desarrollar el código JSX, será posible incluir una llamada a otros componentes.

En el siguiente código, podemos ver cómo, en primer lugar, se desarrolla un primer componente que crea una cadena de texto para emitir un saludo. Este componente es "dinámico", es decir, adapta su salida en función de los parámetros de entrada, recibidos a través de **props**.

```
function Saludar(props) {  
  return <h2>Hola {props.name}</h2>;  
}  
function ComponentePrincipal() {  
  return (  
    <div>  
      <Saludar name="Diana" />  
      <Saludar name="Pepe" />  
    </div>  
  );  
}  
ReactDOM.render(  
  <ComponentePrincipal />,  
  document.getElementById('root')  
);
```

Código 14. Ejemplo de creación de componentes.

A continuación, se modela el componente principal, este invoca el componente *Saludar* y le pasa por parámetro diferentes valores, por lo que la salida de este código sería:

Hola Diana

Hola Pepe

La composición de componentes permite fragmentar el diseño de la página web que se realiza utilizando JSX. Hasta ahora, se ha hecho creando un solo bloque, un solo elemento *div* dentro del cual se ha colocado todo el código. No obstante, será posible fragmentar este código en trozos más pequeños, por ejemplo, si en el código anterior se quisiera identificar cada saludo de forma independiente para, posteriormente, aplicarle estilos distintos.

Se podría implementar de la siguiente forma:

```
function Saludar(props) {  
  return <h2>Hola {props.name}</h2>;  
}  
function ComponentePrincipal() {  
  return (  
    <div className="user1">  
      <Saludar name="Diana" />  
    </div>  
    <div className="user2">  
      <Saludar name="Pepe" />  
    </div>  
  );  
}
```

Código 15. Adaptación del código 14, función saludar fragmentada.

/ 11. Caso práctico 2: “Creación de componentes con acceso y modificación del estado”

Planteamiento: Se pide implementar el código necesario para crear un componente que almacene el estado de una aplicación. En concreto, la aplicación que se pide diseñar consiste en incrementar un determinado valor. Recuerda que el estado “interno” de cada uno de los componentes solo es conocido por el propio elemento y sus elementos hijos. Para poder establecer una comunicación con otros componentes, utilizarán el estado denominado “estado global”.

Nudo: Se tienen en cuenta los siguientes parámetros de diseño:

- La clase se llamará Incrementador.
- La variable inicial recibirá el nombre `variableIncrementar` y se inicializa a valor 0.
- Se pide implementar el contenido de la función `render`, la cual devuelve por pantalla (con las etiquetas HTML oportunas) el resultado de la acción.
- Implementa el constructor.

Desenlace: En el siguiente fragmento de código, se muestra cómo se crearía un componente que sí almacena estado, en este caso, se está desarrollando un elemento que incrementa su valor:

```
import React from 'react'  
export default class Incrementador extends React.  
  Component {  
  constructor() {  
    super()  
    this.state={  
      variableIncrementar:0  
    }  
  }  
  render(){  
    return (  
      <div> El valor es: {this.state.variableIncrementar}</div>  
    )  
  }  
}
```

Código 16. Caso práctico 1, código de solución.



/ 12. Resumen y resolución del caso práctico de la unidad

En esta unidad, hemos conocido que **React JS** es una librería de código abierto creada por **Facebook** e implementada sobre **JavaScript**, la cual permite el diseño e implementación de interfaces web. Se trata de una de las librerías más importantes en la actualidad, puesto que permiten mejorar el diseño de interfaces web de una forma muy sencilla. **React.js** se encarga del *front-end* y busca conseguir una experiencia de usuario similar a la de las aplicaciones de escritorio, de esta forma, quedan limitadas todo lo posible las interacciones entre el navegador y el servidor. Será **React** el que se encargue de actualizar los datos en la capa visual cuando estos cambien. Esta librería se basa en la implementación de componentes **reutilizables**.

Los **componentes** son las piezas que forman la interfaz de usuario en React.js. Cada bloque que aparece en la pantalla será un componente. Por lo tanto, podemos decir que serán fragmentos de código que quedan encapsulados bajo un determinado nombre y que pueden ser reutilizados desde cualquier parte del código. Los componentes son objetos JavaScript.

Finalmente, hemos comprobado que los denominados **props** son un elemento clave para el desarrollo de cualquier componente. Se trata de un objeto que determina todas las propiedades de un componente y resultan especialmente útiles para renderizar estos componentes.

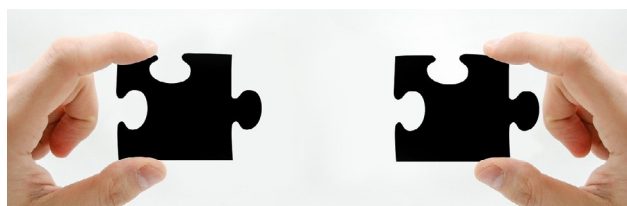


Fig. 8. Imagen ilustrativa 'Componentes interfaz'.

Resolución del caso práctico inicial

Como se ha visto a lo largo del tema, el uso de React resulta muy interesante para el desarrollo de aplicaciones y sus correspondientes interfaces web.

Tal y como se planteaba en el audio inicial, tras haber trabajado ampliamente sobre este tema, es posible extraer como conclusión algunas de las ventajas principales de **React.jx**:

- Permite trabajar a través de componentes, lo que permite la reutilización del código.
- Resulta relativamente sencillo su aprendizaje, puesto que la librería no es demasiado extensa.
- Es muy rápido.
- Utiliza un DOM virtual que permite agilizar el acceso a los elementos del DOM, puesto que, normalmente, suele ser uno de los hitos que ralentiza el funcionamiento de una página web.

/ 13. Bibliografía

García-Miguel, D. (2019). *Diseño de Interfaces Web* (1.a ed.). Madrid, España: Síntesis.

/ 14. Webgrafía

React.js. Recuperado de: <https://es.reactjs.org>