# MINIPROYECTO UT5.

IMPLANTACION DE CONTENIDO MULTIMEDIA

Tabla de contenido

[MINIPROYECTO UT5. 1](#_Toc32527971)

[2. PREPARACIÓN DEL ENTORNO 2](#_Toc32527972)

[3. PROYECTO REACT 3](#_Toc32527973)

1. **INTRODUCCIÓN**

En esta unidad de trabajo vamos a tratar sobre **React**.

React es una biblioteca de JavaScript orientada a crear interfaces de usuario de una sola página (single page application).

Se utiliza un paradigma de programación orientada a componentes, los cuales son clases que heredan de la clase **Component**. Nuestro trabajo consistirá en crear esos componentes e importarlos a la aplicación principal. Podremos tener tantos componentes como requiera nuestra aplicación.

# **PREPARACIÓN DEL ENTORNO**

Para poder llevar a cabo proyectos en React es necesario tener el entorno configurado. Necesitamos para ello tener instalado Node.js y npm.

* 1. **Node.js**

Node.js es un entorno en tiempo de ejecución que utiliza código abierto de JavaScript y que ha sido creado para generar aplicaciones web de forma muy optimizada.

Node cambia el concepto que tenemos de JavaScript, ya que hasta ahora siempre lo hemos utilizado en el lado del cliente, pues bien, con Node se proporciona un entorno de ejecución del lado del servidor que compila y ejecuta aplicaciones a velocidades altas. Utiliza un modelo asíncrono y dirigido por eventos.

* 1. **npm**

Es el sistema de gestión de paquetes que utiliza por defecto Node.js. Esto nos va a permitir gestionar nuestros proyectos de forma sencilla distribuyendo paquetes, agregar dependencias, administrar módulos.

* 1. **Instalación de Node.js**

Como yo estoy utilizando MacOs como sistema operativo la instalación que he llevado a cabo ha sido distinta a la de clase. Aunque es multiplataforma, la instalación desarrollada en clase me daba problemas así que opte por utilizar el gestor de paquetes homebrew para la instalación.

Para ello hay que usar el comando: brew install node

Imagen que contiene reloj, interior, monitor, botella

Descripción generada automáticamente

Con este comando ya se llevará a cabo tanto la instalación de node, como la instalación de npm. Para comprobar que todo se ha instalado correctamente podemos ejecutar en la terminal los comandos: node –version y npm -v

Imagen que contiene reloj, objeto, interior, monitor

Descripción generada automáticamente

Una vez ejecutado esos comandos ya sabemos que tenemos instalado todo lo necesario para comenzar con nuestro proyecto React.

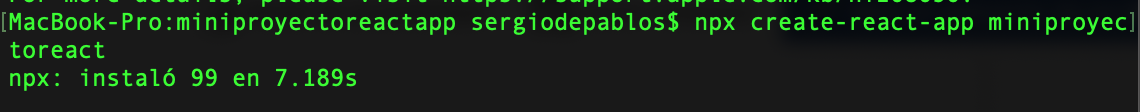
# **PROYECTO REACT**

Ahora que ya tenemos todo nuestro equipo configurado podemos empezar a crear nuestro proyecto React.

Tenemos varias formas de crear un proyecto React, pero nosotros vamos a utilizar el entorno preconfigurado llamado “Create React App”. Este entorno es configurado por Facebook, quien también es la creadora de la biblioteca React.

Vamos a ver los pasos necesarios para poder crear esta “react app”.

* Lo primero que debemos hacer es crear nuestra carpeta contenedora del proyecto. Importante no poner mayúsculas en la creación de este proyecto. Dentro de esta carpeta, hay que ejecutar el comando: npx create-rect-app nombreProyecto.



Una vez ejecutado el comando, se han creado todos los directorios y ficheros necesarios para nuestro proyecto react.

* Lo siguiente necesario para seguir con nuestro proyecto es ejecutar el comando: npm start.

Con esto nuestro proyecto quedará iniciado y el servidor se pondrá en modo escucha, con lo que ya no es ni necesario recargar nuestra web cuando se produzcan cambios.

Al ejecutar este comando también se abrirá en nuestro navegador la aplicación que esta escuchando en el puerto 3000.

Imagen que contiene captura de pantalla, computadora

Descripción generada automáticamente

* Por último, cuando queramos subir nuestro proyecto a un servidor remoto, hay que ejecutar el comando: npm run build.

Imagen que contiene verde, negro, computadora, blanco

Descripción generada automáticamente

Esto creará el proyecto optimizado para poder subirlo a nuestro servidor remoto.

* 1. **CREACIÓN DE NUESTRO PRIMER PROYECTO Y COMPONENTES.**

Al utilizar el proyecto preconfigurado por Facebook hay muchas ficheros que no vamos a utilizar, así que lo primero que vamos a hacer es quedarnos solo con aquellos ficheros que vamos a utilizar. Para ello en el directorio src vamos a eliminar todos los ficheros salvo: index.js e index.css.

Imagen que contiene captura de pantalla

Descripción generada automáticamente

Ahora ya nos hemos quedado con los ficheros que nos importaban.

El siguiente paso a seguir es crear el fichero App.js. Este fichero va a ser la aplicación principal de nuestro proyecto a donde se importaran los demás componentes que vayamos generando.

Lo primero que vamos a hacer es crear la clase App que va a heredar de la clase Component.

Lo primero que hay que hacer es importar la clase Component

Imagen que contiene medidor

Descripción generada automáticamente

Una vez hecho creamos la clase App que hereda de Component:

Imagen que contiene firmar, dibujo

Descripción generada automáticamente

Ahora hay que poner el render(){} y dentro el return. Indispensables para la creación de componentes.

Imagen que contiene pantalla, laptop, teléfono

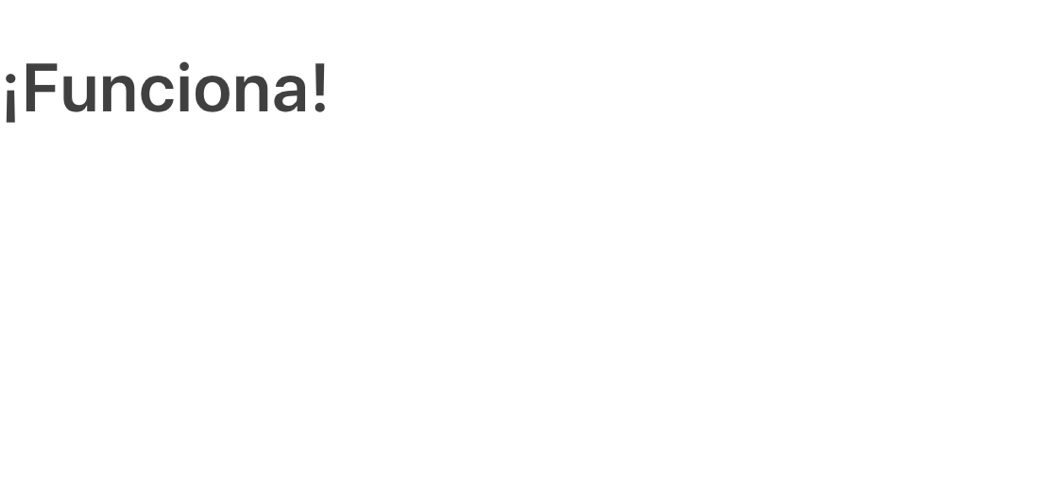
Descripción generada automáticamente

Por último es necesario exportar el componente para poder utilizarlo. Luego habrá que importarlo en el fichero index.js

Imagen que contiene firmar, calle, jugador, pelota

Descripción generada automáticamente

Con esto ya hemos creado el fichero App.js y nuestra aplicación funcionará, si ejecutamos el comando npm start podremos comprobar como se abre en nuestro navegador:



* 1. **SINTAXIS ESPECÍFICA DE REACT**

En React se utiliza la sintaxis JSX, que es una mezcla de JavaScript y XML. Esta sintaxis esta más cerca de JavaScript que de HTML.

Peculiaridades a tener en cuenta:

* Se utiliza **className** en vez de class para aplicar clases CSS (esto es porque class es una palabra reservada en JavaScript).
* Las propiedades y métodos se aplicarán con la segunda parte del nombre en mayúscula (**camelCase**). Por ejemplo: onclick se convertirá en onClick
* Las etiquetas que se cierran en sí mismas, por ejemplo img, lo harán de esta forma: <img />
  1. **VARIABLES EN REACT**

Para asignar variables en React usamos la siguiente sintaxis:

Imagen que contiene dibujo

Descripción generada automáticamente

Y para mostrar esa variable:



La salida en el navegador sería la siguiente:

Un conjunto de letras negras en un fondo blanco

Descripción generada automáticamente

* 1. **COMPONENTES EN REACT. COMPONENTES DE CLASE Y COMPONENTES SIMPLES.**

Las aplicaciones en React se componen de componentes, estos componentes se pueden desarrollar en el mismo documento o desarrollarlos separados para una mejor organización.

* + 1. **COMPONENTES DE CLASE**

Estos componentes son encapsulados dentro de una clase, la cual siempre será herencia de la clase Component. Por este motivo será necesario importar la clase Component utilizando la siguiente sentencia:

Import React, {Component} from ‘react’

Ahora ya podremos desarrollar nuestro componente de clase, vamos a poner un ejemplo desarrollando una tabla como componente. Lo primero es crear el fichero capitalizado.

Tras eso creamos la clase Tabla y ponemos tanto el render como el return Imagen que contiene dibujo

Descripción generada automáticamente

Ahora ya podemos meter nuestro código dentro del return. Por último es necesario exportarlo para poder utilizarlo en nuestra aplicación principal, en esta última será necesario importarlo:



Por último lo importamos y usamos el componente en la parque que queramos de nuestra aplicación principal:Captura de pantalla de un celular

Descripción generada automáticamente

Nuestra página se habrá recargado y ya aparecerá nuestra tabla:Captura de pantalla de un celular

Descripción generada automáticamente

* + 1. **COMPONENTES SIMPLES**

Este tipo de componentes son básicamente funciones y no utilizan la palabra reservada class.

Para ver un ejemplo de componente simple vamos a crear una lista ordenada:

Utilizamos la sintaxis ES6 para realizar las funciones de los componentes:

const ListaOrdenada = () =>{  
 return(  
 <ol>  
 <li>Ragnar Lothbrok</li>  
 <li>Rollo Lothbrok</li>  
 <li>Floki boatbuilder</li>  
 <li>Bjorn Ironside</li>  
 <li>king Ecthbeck</li>  
 </ol>  
 )  
}

Aquí creamos una lista ordenada como componente simple. Además voy a crear otra lista desordenada como componente simple:

const ListaDesordenada = () =>{  
 return(  
 <ul>  
 <li>Saul Goodman</li>  
 <li>Mike Hermantraut</li>  
 <li>Gustavo Fring</li>  
 </ul>  
 )  
}

Por último creo un componente de clase llamado listas donde voy a utilizar los dos componentes simples anteriormente elaborados:

class Listas extends Component{  
 render() {  
 return(  
 <div>  
 <h2>Lista ordenada Vikings componente simple</h2>  
 <ListaOrdenada />  
 <h2>Lista desordenada Better Call Saul componente simple</h2>  
 <ListaDesordenada/>  
 </div>  
 )  
 }  
}

Ahora lo exportamos y ya podemos utilizarlo en la aplicación principal:Captura de pantalla con letras y números

Descripción generada automáticamente

El resultado que obtendríamos en el navegador es el siguiente:Imagen que contiene pájaro, ave, árbol, flor

Descripción generada automáticamente

* 1. **PASO DE PARAMETROS**

El paso de parámetros en React se puede hacer (hasta donde conocemos) de dos formas, por props o state.

* + 1. **PROPS**

Lo característico de los props es que son parámetros (datos) que solo son de lectura, no podemos modificarlos.

Vamos a ver un ejemplo, para poder usarlo vamos a crear unos datos en formato JSON dentro del render de la aplicación principal.Captura de pantalla de un celular

Descripción generada automáticamente

Ahora vamos a crear el componente tabla compuesta de dos componentes simples:

Podemos ir poniendo el componente en la aplicación principal para así ya ir pasando los parámetros, es decir, hay que implementar la siguiente línea:

<h3>Tabla creada con componentes simples y paso de parámetros a través de props</h3>  
<TablaComponentesSimplesProps datosActoresActrices={actoresactrices}/>

Como vemos, al componente “TablaComponentesSimplesProps” le pasamos los datos que previamente creamos anteriormente, es decir le pasamos actoresactrices y lo llamamos datosActoresActrices.

Ahora vamos a crear el componente. Vamos a crear un componente simple para la cabecera de la tabla, otro componente simple para el cuerpo de la tabla y por último un componente de clase donde juntaremos todo. **Hay que tener en cuenta, que los datos los recogemos en el componente de clase y desde ahí lo pasamos al componente simple del cuerpo de la tabla que es quien va a usarlos.**

Vamos a ver el paso del prop al componente de clase:

class TablaComponentesSimplesProps extends Component{  
 render() {  
 const {datosActoresActrices} = this.props; //aquí llegan los datos de la aplicación principal  
 return(  
 <table>  
 <TablaCabecera/>  
 <TablaCuerpo datosActoresActrices = {datosActoresActrices}/>  
 </table>  
 )  
 }  
}

Ya tenemos los componentes en una variable, la cual luego se la pasamos al componente de clase. Utilizamos **this.props**, que es donde están todos los props que haya en la aplicación, en este caso solo existe uno, por lo que no es necesario que especifiquemos el nombre, pero si existieran más si sería necesario.

const TablaCuerpo = props =>{  
 const filasDatos = props.datosActoresActrices.map((fila, indice)=>{  
 return(  
 <tr key={indice}>  
 <td>{fila.nombre}</td>  
 <td>{fila.apellidos}</td>  
 </tr>  
 )  
 }  
 )  
 return (  
 <tbody>  
 {filasDatos}  
 </tbody>  
 )  
}

Seguimos en TablaComponentesSimplesProps.js. Como podemos observar a continuación, utilizaremos el props como parámetro para el componente simple TablaCuerpo. Utilizamos el método **map** para retornar una fila de tabla por cada objeto del array datosActoresActrices. Con índice llevamos la cuenta del número de filas. Hay que tener en cuenta que vamos a utilizar una **key** para identificar el elemento dentro del array. Las keys sólo se suelen utilizar cuando el orden de estos elementos no va a cambiar.

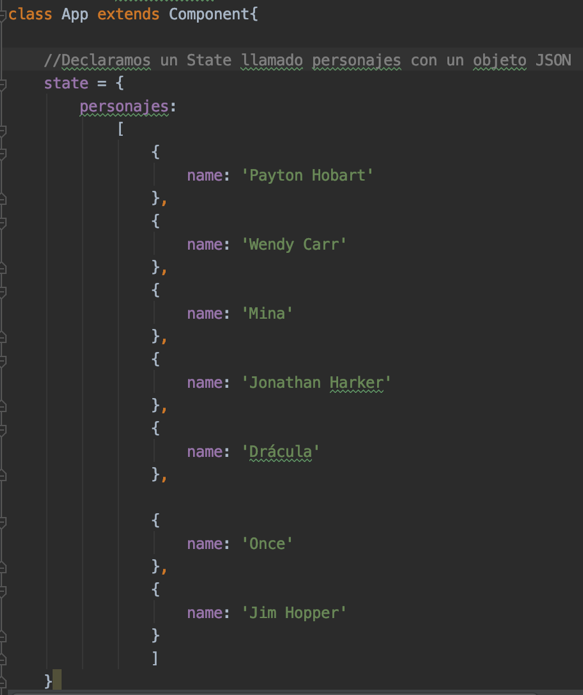
Si todo ha funcionado correctamente la salida en el navegador es la siguiente:



* + 1. **STATE**

State, al igual que props nos va a permitir pasar datos a un componente. Además estos datos podrán ser modificados por nosotros.

Vamos a crear un state constituido por un objeto JSON, para ello vamos a crearlo en la aplicación principal antes de render().



Una vez creado, vamos a crear un método para borrar elementos de este objeto JSON. Como podemos modificar los datos, podremos borrarlos con la función. Vamos a implementar la función en la aplicación principal justo a continuación de la declaración del state:

borrarPersonaje = indice =>{  
 const {personaje} = this.state;  
  
 this.setState(  
 {  
 personaje: personaje.filter((personaje, i) =>{  
 return i !==indice; /\*Devuelve todos los distintos de indice, es decir los que no tiene que borrar \*/  
  
 })  
 }  
 )  
}

El siguiente paso es inicializar el objeto personajes dentro del render () de la aplicación principal:

/\* Inicializamos el objeto personajes \*/  
const {personaje} = this.state;

Tras esta inicialización vamos ahora a poner el componente simple dentro de nuestra aplicación principal, y le pasamos lo parámetros. De primeras nos dará error, ya que no hemos creado el fichero ni lo hemos importado.

<h3>Tabla con componentes simples y paso de parámetros con State</h3>  
<TablaComponentesSimplesState datosPersonaje = {personajes} borrarPersonaje={this.borrarPersonaje}/>

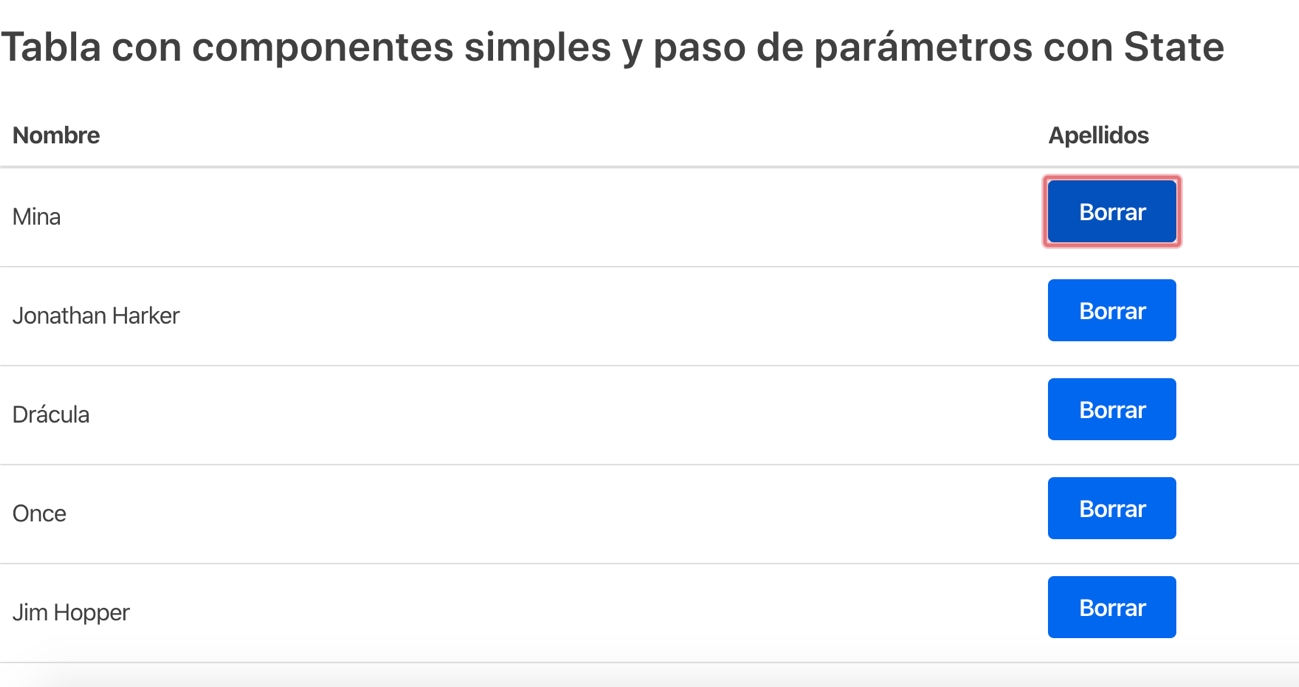
Ahora es el momento de crear el fichero y los componentes. Vamos a reutilizar el fichero anterior, añadiendo en el componente tablaCuerpo un botón al que le asociaremos un evento click que llamará a la función borrarPersonaje pasándole el índice.

const TablaCuerpo = props =>{  
 const filasDatos = props.datosPersonaje.map((fila, indice)=>{  
 return(  
 <tr key={indice}>  
 <td>{fila.name}</td>  
 <td><button onClick={()=>props.borrarPersonaje(indice)}>Borrar</button> </td>  
 </tr>  
 )  
 }  
 )  
 return (  
 <tbody>  
 {filasDatos}  
 </tbody>  
 )  
}

Ahora hay que modificar el componente donde importamos los otros dos, **lo vamos a poner como un componente simple para que state no de problemas, con un componente de clase nos daría problemas.**

const TablaComponentesSimplesState = (props) =>{  
 const {datosPersonaje, borrarPersonaje} = props;  
 return(  
 <table>  
 <TablaCabecera/>  
 <TablaCuerpo datosPersonaje ={datosPersonaje} borrarPersonaje={borrarPersonaje}/>  
 </table>  
 )  
}  
  
export default TablaComponentesSimplesState

Por último hay que exportarlo.



* 1. **TRATAMIENTO DE FICHEROS JSON**

Puede darse el caso, de hecho es muy probable, que los datos que tengamos que utilizar nuestra aplicación nos vengan de un fichero externo, el cual estará en formato JSON. Vamos a ver como habría de implementarse:

Lo primero es crear el fichero en formato JSON, es decir creamos un fichero .json y le metemos la siguiente información:

[  
 {  
 "name": "Payton Hobart"  
 },  
 {  
 "name": "Wendy Carr"  
 },  
 {  
 "name": "Mina"  
 },  
 {  
 "name": "Jonathan Harker"  
 },  
 {  
 "name": "Drácula"  
 },  
 {  
 "name": "Once"  
 },  
 {  
 "name": "Jim Hopper"  
 }  
]

Una vez tenemos creado el fichero es necesario importarlo a nuestra aplicación:

import personajes from "./personajes"

Con esto ya podremos usar este fichero declarándolo como state o props, ya queramos modificar o no estos datos.

* 1. AÑADIR DATOS DESDE UN FORMULARIO

Vamos ahora a ver como recoger datos y añadirlos a un state. Para ello vamos lo primero a declarar un state vacío que será donde se almacenen los datos que recojamos:

state = { actores: []}

Una vez hecho esto hay que crear una función que recupere los datos del formulario al hacer el envío. Vamos a implementarla:

manejarEnvio = actor => {   
 this.setState({actores: [...this.state.actores, actor]});   
}

Ahora vamos a poner el componente de clase al que le pasamos la función, el componente aun no esta creado, lo crearemos posteriormente:

<h2>Añadir nuevo</h2>  
<Formulario manejarEnvio={this.manejarEnvio} />

Por último creamos el componente de clase del formulario:

import React, {Component} from 'react';  
  
class Formulario extends Component {  
 constructor(props) {  
 super(props);  
 this.initialState = {name: ''};  
 this.state = this.initialState;  
 }  
  
 manejarCambio = evento => {  
 const {name, value} = evento.target;  
 this.setState({[name]: value});  
 }  
 enEnvioFormulario = (evento) => {  
 evento.preventDefault();  
 this.props.manejarEnvio(this.state);  
 this.setState(this.initialState);  
 }  
  
 render() {  
 const {name} = this.state;  
 return (  
 <form onSubmit={this.enEnvioFormulario}><label for="nombreid">Nombre</label> <input type="text" name="name"  
 id="nombreid"  
 value={name}  
 onChange={this.manejarCambio}/>  
 <button type="submit"> Enviar Datos</button>  
 </form>);  
 }  
}  
  
export default Formulario

Si cambiamos declaramos la variable state con otro nombre distinto de personajes habría que cambiar también los componentes de las tablas para que se visualice correctamente, la salida por pantalla es la siguiente:

Captura de pantalla de un celular

Descripción generada automáticamente

Si añadimos un nuevo valor se actualizará la tabla.

* 1. **CREAR BUILD DE PRODUCCIÓN**

Un proyecto completo en React es muy pesado, por lo que subirlo completamente a un servidor puede ser una tarea ardua, por ello es posible mediante el comando npm run build, construir una versión optimizada de nuestro proyecto pensada para nuestro servidor. Se generará así una carpeta build que será la que tenemos que subir a nuestro servidor:

Imagen que contiene dibujo

Descripción generada automáticamente