

DWEC

TEMA 9



REACT JS



ÍNDICE

- 9.1 Introducción
- 9.2 Atomic Design Methodology
- 9.3 Instalación
- 9.4 React básico: lenguaje JSX
- 9.5 Funcionamiento de React
- 9.6 Acceso al api.

9.1 INTRODUCCIÓN

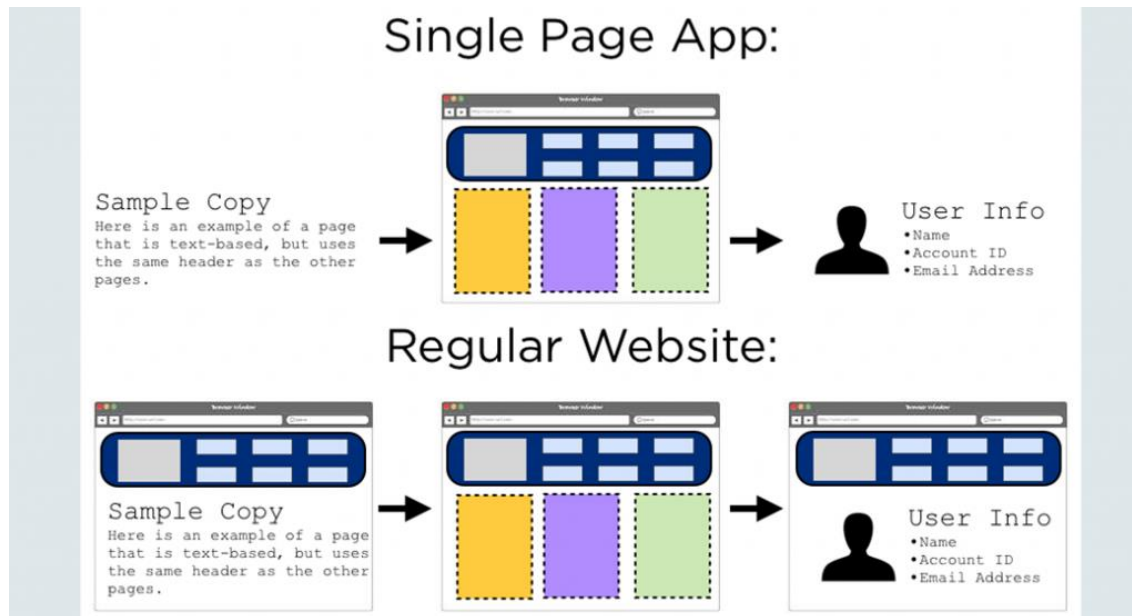
React JS es una librería Opensource creada por Facebook para el desarrollo de aplicaciones de una sola página (SPA)



9.1 INTRODUCCIÓN

SPA (aplicación de una sola página → Simple Page application)

Es una implementación de aplicación web que carga solo un documento web y luego actualiza el contenido del cuerpo de ese documento único a través de las API de JavaScript, como XMLHttpRequest y Fetch, cuando se muestra contenido diferente.

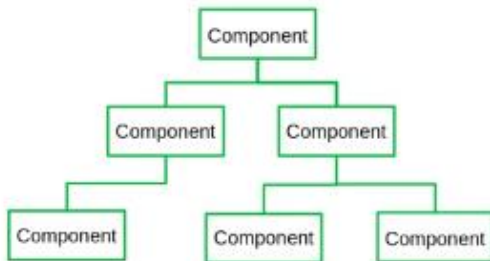


9.1 INTRODUCCIÓN

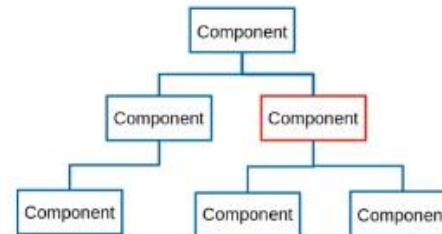
Virtual DOM

React aplica un renderizado para optimizar el uso del DOM, haciendo solo los cambios necesarios y optimizando su uso. Todo esto lo hace gracias a su DOM virtual .

DOM



 React
USA SU PROPIO
Virtual DOM

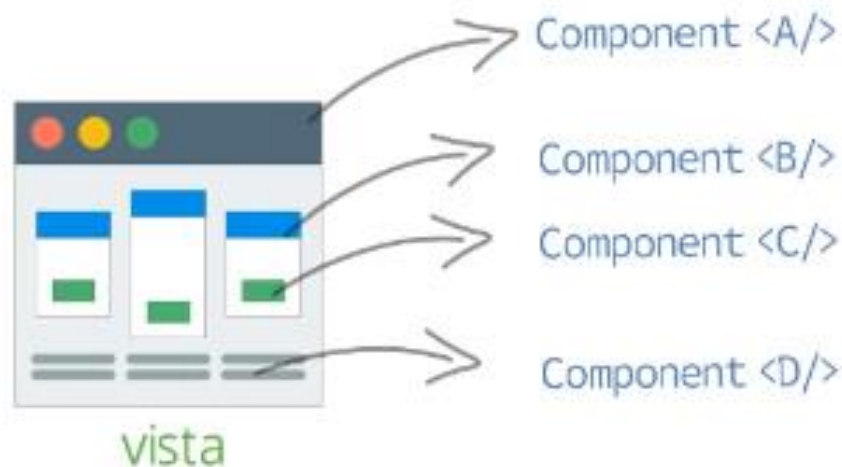


9.1 INTRODUCCIÓN

ATOM DESIGN METHODOLOGY

Su diseño esta basado componentes, estos componentes son reutilizables i se pueden agrupar. Se verá en el siguiente apartado .

VISTAS A BASE DE COMPONENTES



9.1 INTRODUCCIÓN

React tiene una gran comunidad y esta respaldada por Facebook cosa que hace que siempre se esté actualizando y tenga una continuidad del proyecto.

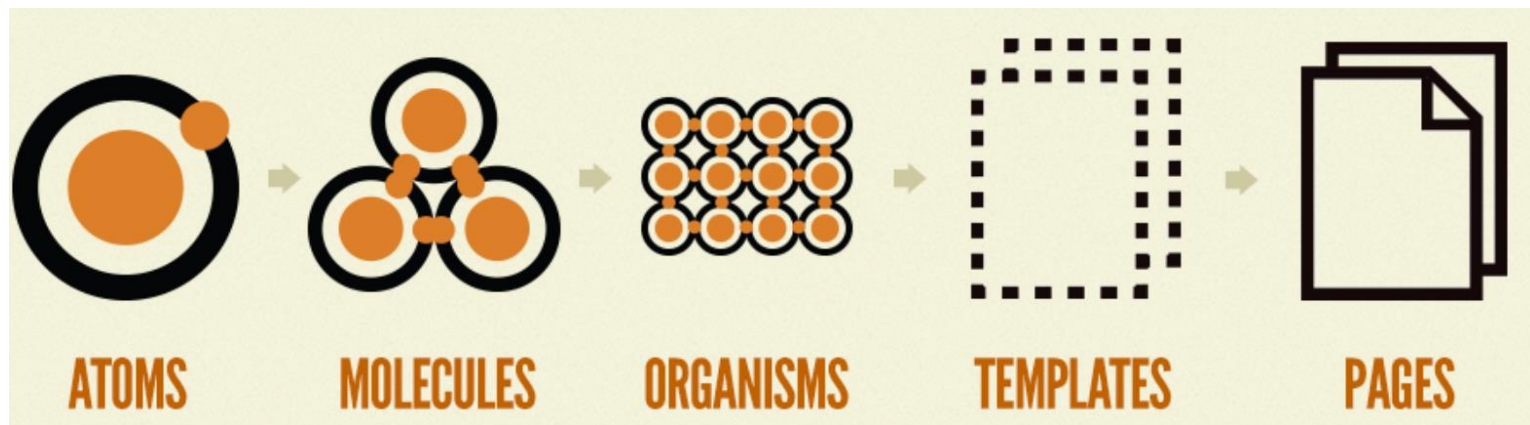


9.2 ATOMIC DESIGN METHODOLOGY

Es una metodología que consta de cinco etapas que trabajan juntas para construir un sistema de diseño de interfaz de una manera cuidada y jerárquica. Inspirada en la tabla periódica.

<https://www.youtube.com/watch?v=Yi-A20x2dcA>

<https://www.youtube.com/watch?v=4m3Dkk42uHk>



9.3 INSTALACIÓN

Para la instalación de un proyecto en React debemos de tener preinstalado Node js, los gestores de paquetes npm, yarn y npx.

- <https://nodejs.org/en/download/>
- Npm va incluido en la instalación de node.
- <https://classic.yarnpkg.com/en/docs/install#windows-stable>
- <https://www.npmjs.com/package/npx>

El proceso de instalación de React es muy complejo, teniendo que realizar muchos pasos y configuraciones que nos llevarían a complicar este apartado, existe un atajo para realizar una instalación estándar que nos servirá para la gran mayoría de proyectos que vamos a realizar.

- <https://create-react-app.dev/docs/getting-started>

9.3 INSTALACIÓN

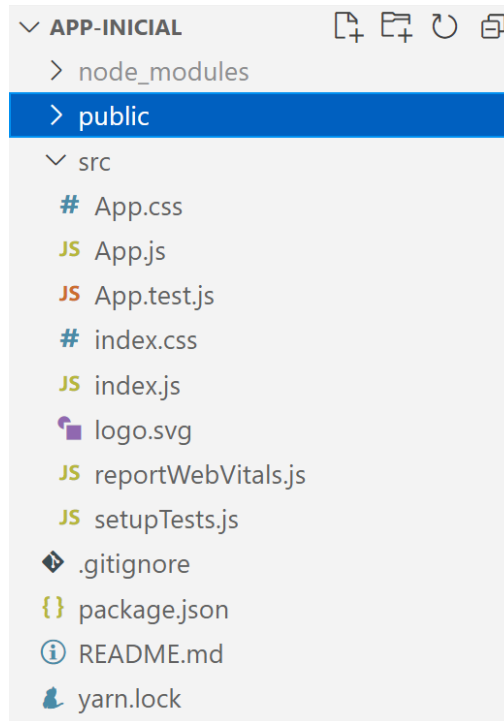
Una vez instalada la app para hacer las instalaciones de proyectos vamos a realizar la primera instalación:

- `yarn create react-app app-inicial`
 - `cd app-inicial`
 - `yarn start`
- `yarn start`



9.3 INSTALACIÓN

Ya instalada nuestra aplicación y arrancada podemos empezar a ver como se ha configurado la estructura de directorios del proyecto:



9.3 INSTALACIÓN

Cada directorio tiene una función, que pasamos a de tallar :

- **Node_module**: Contiene todas las librerías de las que depende React.
- **Public**: Contiene el fichero index.html que sirve como plantilla donde React montará la aplicación y donde incluirá automáticamente las referencias al JavaScript y al CSS optimizado que se genera.
- **Package.json**: El fichero que contine todas las propiedades del proyecto montado.
- **Yarn.lock**: Fichero utilizado por yarm que contiene las referencias de los paquetes y sus versiones.
- **Src**: Carpeta del código fuente que debemos editar para desarrollar el proyecto.
- **.gitignore** : fichero que informa los paquetes o ficheros que no se deben de subir al git.

9.3 INSTALACIÓN

Ahora vamos a revisar los ficheros mas importantes del proyecto explicando un poco su función:

Fichero package.json :

```
"dependencies": {  
  "@testing-library/jest-dom": "^5.14.1",  
  "@testing-library/react": "^12.0.0",  
  "@testing-library/user-event": "^13.2.1",  
  "react": "^17.0.2",  
  "react-dom": "^17.0.2",  
  "react-scripts": "5.0.0",  
  "web-vitals": "^2.1.0"  
},  
   Debug  
"scripts": {  
  "start": "react-scripts start",  
  "build": "react-scripts build",  
  "test": "react-scripts test",  
  "eject": "react-scripts eject"  
},
```

9.3 INSTALACIÓN

Ahora vamos a revisar los ficheros mas importantes del proyecto explicando un poco su función:

Fichero **package.json** :

```
"dependencies": {  
  "@testing-library/jest-dom": "^5.14.1",  
  "@testing-library/react": "^12.0.0",  
  "@testing-library/user-event": "^13.2.1",  
  "react": "^17.0.2",  
  "react-dom": "^17.0.2",  
  "react-scripts": "5.0.0",  
  "web-vitals": "^2.1.0"  
},  
   Debug  
  "scripts": {  
    "start": "react-scripts start",  
    "build": "react-scripts build",  
    "test": "react-scripts test",  
    "eject": "react-scripts eject"  
  },
```

9.3 INSTALACIÓN

Fichero **index.js**:

Es el fichero que inicia la aplicación React y que incluye un único componente que tiene en este caso: **App**. Básicamente, el inicio de React consiste en una llamada a Renderizar el elemento raíz `<App />` y se le indica en que elemento de la página se quiere cargar: **root**

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);
reportWebVitals();
```

9.3 INSTALACIÓN

Fichero **index.html**:

Dentro de la carpeta public se encuentra el contenido HTML donde se carga los componentes de Recat. Según lo que define el index.js, el componente raíz se debe de cargar dentro del elemento cuya id debería de ser root.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />

    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />

    <title>React App</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
  </body>
</html>
```


9.3 INSTALACIÓN

Fichero **app.js**:

Es el fichero de código que define el único componente que se utiliza. Es un componente bastante sencillo que directamente devuelve un contenido estático mediante un return. Lo que se devuelve parece que es HTML, pero se trata de **JSX** .

```
import logo from './logo.svg';
import './App.css';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  );
}

export default App;
```

9.3 INSTALACIÓN

Fichero **app.js**:

Es el fichero de código que define el único componente que se utiliza. Es un componente bastante sencillo que directamente devuelve un contenido estático mediante un return. Lo que se devuelve parece que es HTML, pero se trata de **JSX** .

```
import logo from './logo.svg';
import './App.css';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  );
}

export default App;
```

9.3 INSTALACIÓN

Fichero **app.css**:

Los componentes pueden tener estilos asociados. Esta sería la forma más sencilla de aplicarlos, asociando un CSS a cada componente.

```
.App {
  text-align: center;
}

.App-logo {
  height: 40vmin;
  pointer-events: none;
}

@media (prefers-reduced-motion: no-preference) {
  .App-logo {
    animation: App-logo-spin infinite 20s linear;
  }
}

.App-header {
  background-color: #282c34;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  font-size: calc(10px + 2vmin);
  color: white;
}
```

```
.App-link {
  color: #61dafb;
}

@keyframes App-logo-spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}
```

9.3 INSTALACIÓN

Fichero **registerServiceWorker.js**:

Es un fichero que registra un servicio que se ejecuta en segundo plano y funciona como una especie de proxy que permite una carga más rápida de la aplicación, mediante el uso de caché y la gestión de las peticiones a la red.

9.3 INSTALACIÓN

Plugins de React

El navegador y su consola de desarrolladores es uno de los mejores aliados para la programación web. Se encuentra disponible en todos los navegadores.

En el caso de React, puede descargarse un plugin específico, que se muestra en la imagen que aporta ayuda extra y más específica para este framework.



React Developer Tools

by [React](#)

⚠ This add-on is not actively monitored for security by Mozilla. Make sure you trust it before installing.

[Learn more](#)

React Developer Tools is a tool that allows you to inspect a React tree, including the component hierarchy, props, state, and more. To get started, just open the Firefox devtools and switch to the "Components" or "Profiler" tab.

[Add to Firefox](#)

<https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi?hl=es>

9.4 REACT BÁSICO: LENGUAJE JSX

JSX es en realidad JavaScript XML, no es más que un lenguaje tipo XML en que se pueden intercambiar sentencias de JavaScript. Esto permite crear componentes donde se puede añadir cierta lógica como variables, propiedades, condiciones, bucles, ect.

JSX también permite mezclar de forma natural etiquetas HTML con componentes de React, además facilita la vida a los diseñadores que no tienen por qué saber nada de programación.

9.4 REACT BÁSICO: LENGUAJE JSX

Normas básicas

JSX debe de cumplir una serie de normas básicas para ser válido, que se expresan a continuación.

- **Sin errores**

JSX, ante todo, es código y no puede tener errores de JavaScript o de JSX. Al mínimo error, el componente no se transpila y la aplicación no funciona correctamente. No es HTML y no se debe de esperar que se comporte como tal.

- **Cierre de etiquetas**

Todas las etiquetas deben tener un cierre, tal y como exige XML, y que las etiquetas puedan cerrarse a si mismas, como se ha visto en los ejemplos anteriores. La convención para distinguir el HTML puro con los componentes de React es porque su primera letra está en mayúsculas.

9.4 REACT BÁSICO: LENGUAJE JSX

- **Elemento raíz**

Además hay otra norma que también se hereda de XML: todo componente debe de tener una etiqueta raíz. Es decir, algo así sería incorrecto:

```
class Incorrect extends Component {  
  render () {  
    return (  
      <h2> Title </h2>  
      <div> This is the text </div>  
    );  
  }  
}
```

El transpilador mostrará un error indicando que se necesita un elemento raíz, o bien se utiliza *React.Fragment* o simplemente se puede arreglar así.

```
class Correct extends Component {  
  render() {  
    return(  
      <div>  
        <h2>Title</h2>  
        <div>This is the text</div>  
      </div>  
    )  
  }  
}
```


9.4 REACT BÁSICO: LENGUAJE JSX

Normas básicas

- **Atributo key en conjuntos**

Si se generan varios elementos de los componentes como listas, filas de tablas, etc. Utilizando bucles, se debe de asignar una clave a cada uno de esos elementos incluyendo un atributo *Key*. Si no se hace, en el propio generador de la aplicación React advertirá con un mensaje de aviso como el siguiente:

```
warning: Each child in a list should have a  
unique "key" prop.
```

9.4 REACT BÁSICO: LENGUAJE JSX

Comentarios

Dentro de JSX se pueden introducir comentarios. Pese a que los comentarios actualmente son una mala práctica, en el caso de que fuera preciso, estas serían las formas de introducirlos:

- Los comentarios propios de JavaScript, los clásicos multilínea o los de una única línea.
- Los comentarios en los componentes React.

```
/**
 * Multiline JavaScript comment
 */
class Comments extends React.Component {
  // Single line comment
  render () {
    let title = "this is the title";
    let content="This is some content";
    return (
      <div /*
       *Multiline comment within any component
       */
        <h1>{title}</h1>
        <p>{content}</p>
      </div>
    )
  }
}
```

9.4 REACT BÁSICO: LENGUAJE JSX

Variables

Se pueden incluir variables que alteran la apariencia y el comportamiento de un componente. Además de las propiedades y el estado de un componente que se verán más adelante, dentro del componente se permite definir variables que pueden ser utilizadas según convenga.

```
class Comments extends React.Component {  
  render () {  
    let title = "this is the title";  
    let content="This is some content";  
    return (  
      <div>  
        <h1>{title}</h1>  
        <p>{content}</p>  
      </div>  
    )  
  }  
}
```

9.4 REACT BÁSICO: LENGUAJE JSX

Variables

Se podría, incluso, definir dentro de variables distintas del componente y luego utilizar esas variables al retornar el componente en el método render.

```
const name = 'Josh Perez';  
const element = <h1>Hello, {name}</h1>;  
ReactDOM.render(  
  element,  
  document.getElementById('root')  
);
```

9.4 REACT BÁSICO: LENGUAJE JSX

Condicionales

Dentro del propio render, se pueden mostrar elementos de forma condicional de varias maneras:

- **Return condicional** : dentro del código JavaScript, se hará un return u otro. Se puede aplicar con *switch-case* o *else if*.

```
class Comments extends React.Component {  
  render () {  
    if (condition){  
      return <Component1/>  
    } else {  
      return <Component2 />  
    }  
  }  
}
```

9.4 REACT BÁSICO: LENGUAJE JSX

Condicionales

- **If simple en el render:** dentro del *return* que hace el *render* se pueden aplicar condicionales en el caso de que queramos que se muestre algo así

```
class Conditionals2 extends React.Component {  
  render () {  
    return (  
      <div>  
        {this.state.row.length > 0 &&  
          <table>  
            <tr>...  
            </tr>  
          </table>  
        }  
      </div>  
    );  
  }  
}
```

9.4 REACT BÁSICO: LENGUAJE JSX

Condicionales

- **If-else en render:** también se puede aplicar una estructura tipo if-else. Se trata de aplicar el operador ternario dentro del render .

```
class Conditionals2 extends React.Component {  
  render () {  
    return (  
      <div>  
        <h1> Display Data</h1>  
        {this.state.row.length > 0 ?  
          (  
            <table>  
              <tr>...  
            </tr>  
            </table>  
          ) : (  
            <div> No data available </div>  
          )  
        </div>  
      );  
    }  
  }  
}
```

9.4 REACT BÁSICO: LENGUAJE JSX

Bucles

Crear múltiples elementos

Se puede aprovechar el hecho de los componentes se crean con JavaScript para poder automatizar la generación de los elementos del componente. Por ejemplo si tenemos un array de objetos podemos iterarlos mediante el método ***map*** de los arrays. Ejemplo:

```
class Lopps extends React.Component {  
  render () {  
    const friends = [  
      {name: "John Doe", age: 34},  
      {name: "Gandalf", age: 1432},  
      {name: "Aragorn", age: 442}  
    ];  
    return(  
      <div>  
        <h1> My Friends </h1>  
        <ul>  
          {  
            friends.map ((ele, ind) => {  
              <li key={ind} ><b>{ele.name}</b>: {ele.age} years </li>  
            })  
          }  
        </ul>  
      </div>  
    )  
  }  
}
```


9.4 REACT BÁSICO: LENGUAJE JSX

JavaScript

También el JSX permite que se pueda alternar código JavaScript. Así pues dentro de JSX, cualquier cosa que vaya entre {} puede permitir meter código JS:

```
class codi extends React.Component {  
  render(){  
    return (  
      <div>  
        <h1> JavaScript will execute</h1>  
        {console.log('This will appear in the console')}  
      </div>  
    )  
  }  
}
```

9.4 REACT BÁSICO: LENGUAJE JSX

Eventos

A cada uno de los componentes se le puede incluir un atributo de evento. Los nombres son los mismos que en HTML, con la sutil diferencia de que en JSX debemos utilizar siempre el camel case para referirnos a ellos.

- onClick: Este es el evento más típico:

```
<div onClick={alert('Its Works!')}>  
  
</div>
```

Para que el método solo sea llamado cuando se hace click, se debe de definir una función arrow.

```
<div onClick={() => alert('Its Works!')}>  
  Click on this paragraph  
</div>
```

9.4 REACT BÁSICO: LENGUAJE JSX

Eventos

Dos estilos de llamadas

Como se ve en el siguiente componente, se puede llamar a métodos definidos dentro del propio componente, pero tenemos diferentes opciones:

- Introducir código JS dentro del onClick con un *arrow function*.
- Hacer referencia a un método que luego se debe de añadir como *arrow function* . (primer enlace)
- Usar un *arrow function* que invoque un método que definiremos de forma tradicional (segundo enlace)

9.4 REACT BÁSICO: LENGUAJE JSX

Eventos

Dos estilos de llamadas

Esto es lo que se ve a continuación:

```
class App extends React.Component {
  arrowHandleClick = (e) => {
    e.preventDefault ();
    alert ('The first link was clicked. '); }
  handleClick (e) {
    e.preventDefault ();
    alert ('The second link was clicked. '); }
  render(){
    return (
      <div>
        <button onClick={() => alert("Its works")}>
          Click on this button
        </button>
        <a href="#" onClick={this.arrowHandleClick}>
          Click first link </a>
        <a href="#" onClick={(e) => this.handleClick(e) }>
          Click second link </a>
        </div>
      );
  }
}
```

9.4 REACT BÁSICO: LENGUAJE JSX

Hojas de estilo

Como se ha visto desde la primera aplicación en React, se puede asociar una hoja de estilos generales editando el fichero *public/index.html*, donde se podría cargar por ejemplo frameworks como Bootstrap o cualquier elemento.

- Un CSS por cada componente

En general, se puede definir una hoja de estilo e importarla en nuestro componente.

Ver ejemplo del Fichero App.css

Hay que recordar que para aplicar una clase de CSS en JSX el atributo es **className**.

9.4 REACT BÁSICO: LENGUAJE JSX

Hojas de estilo

- Estilos JavaScript

React también puede usar su propio estilo, para aplicar CSS.

Con JS se pueden manipular los estilos del DOM. Para cada propiedad CSS, existe una propiedad JavaScript correspondiente, en la que únicamente varia la forma de escribir esa propiedad, por ejemplo :

- background-color en JS sería backgroundColor.
- font-size sería fontSize
- Etc.

9.4 REACT BÁSICO: LENGUAJE JSX

Hojas de estilo

- Estilos JavaScript

Lo que se hace básicamente es definir en variables un conjunto de propiedades en JSON dentro de la propia clase del componente. Ejemplo:

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

class App extends Component {
  render () {
    let h1class = {
      textAlign : 'center',
      color: 'red',
      fontWeight : 'bold'
    }
    let h2Class = { color: 'darkred' }

    return (
      <div>
        <h1 style={h1class}> Styled component</h1>
        <h2 style={h2Class}> Other Style</h2>
      </div>
    )
  }
}

export default App;
```

9.4 REACT BÁSICO: LENGUAJE JSX

Propiedades y estados

Todo componente de React tiene propiedades y estados. Se trata de dos atributos que puede utilizar o no para alterar su aspecto y su comportamiento.

- **Propiedades**

Las *properties* o propiedades son valores con los que se inicia un componente. Son como los parámetros que recibe al ser utilizado y la forma de pasarlos es como establecer un atributo en una etiqueta Web. Son **valores de solo lectura**, y por tanto el componente no las puede modificar. En el momento en el que se cambian las propiedades que se pasan al componente, su **método render es llamado automáticamente**.

9.5 FUNCIONAMIENTO DE REACT

Propiedades y estados

- Propiedades

Ejemplo:

Pasar propiedades :

```
let technologyData = ['React', 'Vue', 'Angular'];  
<App title="My title" version={1} tech={technologyData} />
```

Mostrar propiedades:

```
render(){  
  const title = this.props.title;  
  const tech = this.props.tech;  
  return(  
    <div>  
      <h1>{title}</h1>  
      <h2>{tech[0]}</h2>  
    </div>  
  )  
}
```

9.5 FUNCIONAMIENTO DE REACT

Propiedades y estados

- **Propiedades**

Valores por defecto: Mediante esta definición se puede establecer los valores por defecto de las propiedades, en el caso de que no se le pasen.

```
static defaultProps = {  
  title: 'Default title',  
  version: 0,  
}
```

Restricciones: También se puede aplicar restricciones para que a las propiedades no se asignen valores de manera incontrolada o para asegurar de que se pasan todas las propiedades requeridas. Se insertan a parte de la clase o función.

```
import PropTypes from 'prop-types';  
/**-----**/  
App.propTypes = {  
  title: PropTypes.string.isRequired,  
  version: PropTypes.number.isRequired  
}
```

9.5 FUNCIONAMIENTO DE REACT

Propiedades y estados

- **Propiedades:** Ejemplo Completo

```
/* Fichero index.jsx */
import React from "react";
import ReactDOM from "react-dom";
import './index.css';
import App from './App';
import registerServiceWorker from './registerServiceWorker';

const myText = 'This app rocks!';
const doIt = () => {
  alert ('It works')
};

ReactDOM.render (
  <App title='SuperApp' text={myText} fun={doIt} />,
  document.getElementById('root'))
registerServiceWorker ();
```

9.5 FUNCIONAMIENTO DE REACT

Propiedades y estados

- Propiedades: Ejemplo Completo

```
// Fichero App.jsx
import React, {Component} from 'react';
import PropTypes from 'prop-types';
import logo from './logo.svg';
import './App.css';

class App extends Component {

  static defaultProps = {
    title : 'Default title',
    text : 'Default text',
    version : 0,
    technologies : [],
    fun : this.dummy
  }

  dummy () {
    console.log('I do nothing');
  }

  render(){
    const text = this.props.text;
    const version = this.props.version;
```

9.5 FUNCIONAMIENTO DE REACT

Propiedades y estados

- Propiedades: Ejemplo Completo

```
    return (  
      <div className="App">  
        <header className="App-header">  
          <img src={logo} className="App-logo" alt="logo" />  
          <h1 className='App-title'>  
            {this.props.title} v {version}  
          </h1>  
        </header>  
        <p className='App-intro'>  
          {text}  
        </p>  
        <button onClick={this.props.fun}> Click Me</button>  
      </div>  
    )  
  }  
  App.propTypes = {  
    title: propTypes.string.isRequired,  
    text: propTypes.string.isRequired,  
    version: propTypes.number.isRequired,  
    technologies: propTypes.array.isRequired,  
    fun : propTypes.func.isRequired,  
  }  
  export default App;
```

9.5 FUNCIONAMIENTO DE REACT

Propiedades y estados

- **Estados:**

El *state* o estado nos permite controlar y modificar el componente mientras dura la ejecución. A diferencia de las propiedades, el estado SÍ se puede y debe modificar para poder controlar la apariencia y comportamiento del componente.

En el momento en que se modifica el estado, **el método *render* es llamado automáticamente**. Por tanto, el estado es la palanca con la que podemos hacer un componente cambie internamente.

9.5 FUNCIONAMIENTO DE REACT

Propiedades y estados

- **Estados:**

Valores por defecto

Todo componente puede tener definidos unos valores de estado iniciales. Eso se puede hacer mediante el constructor de nuestro componente.

```
constructor (props){  
  super (props);  
  this.state = {  
    name: '',  
    data: []  
  }  
}
```

9.5 FUNCIONAMIENTO DE REACT

Propiedades y estados

- **Estados:**

Modificación del estado

Para modificar los valores de estado, se debe usar un método propio del componente llamado *setState*, al que se le pasa un objeto con el nombre de la propiedad del estado que queremos cambiar y su nuevo valor.

```
this.setState ({name: 'Eugene Krabs', data: ['Krusty Krab']})
```


9.5 FUNCIONAMIENTO DE REACT

Propiedades y estados

- Estados: Ejemplo

```
// Fichero App.jsx
import React, {Component} from 'react';
import PropTypes from 'prop-types';
import logo from './logo.svg';
import './App.css';

class App extends Component {
  constructor (props){
    super (props);
    this.state = {
      title: 'Default',
      time: new Date ().toLocaleDateString(),
      number: 0,
      numbers: []
    }
  }
  changeState (){
    let number = Math.round ( Math.random() * 4);
    let numbers = this.state.numbers;
    numbers.push (number);
    this.setState({
      time: new Date().toLocaleTimeString(),
      numbers: numbers,
      number: number,
      title: ((number % 2 === 0) ? 'It is even' : 'It is odd')
    })
    console.log('changelstate --> ', this.state)
  }
}
```

```
render() {
  console.log ('Render was called -->', this.state);
  const colors = ['red','yellow','green','blue','orange'];
  const color =colors[this.state.number];
  return (
    <div className="App" style={{backgroundColor: color}}>
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <h1 className='App-title'>
          {this.state.title} - {this.state.number}
        </h1>
      </header>
      <div className='App-intro'>
        <div>{this.state.time}</div>
        Press this button to change state
      </div>
      <div>
        <button onClick={() => this.changeState()}> Change state</button>
      </div>
      <div>
        Generated numbers
        <ul>
          {this.state.numbers.map((x)=>(
            <li key={x}> {x} </li>
          ))}
        </ul>
      </div>
    </div>
  )}
}

export default App;
```

9.5 FUNCIONAMIENTO DE REACT

Propiedades y estados

- **Estados:** Ejemplo de React Hook como gestiona los estados:

```
import React, { useState } from "react";
import ReactDOM from "react-dom";

function FavoriteColor() {
  const [color, setColor] = useState("red");

  return (
    <>
      <h1>My favorite color is {color}!</h1>
      <button
        type="button"
        onClick={() => setColor("blue")}
      >Blue</button>
      <button
        type="button"
        onClick={() => setColor("red")}
      >Red</button>
      <button
        type="button"
        onClick={() => setColor("pink")}
      >Pink</button>
      <button
        type="button"
        onClick={() => setColor("green")}
      >Green</button>
    </>
  );
}

ReactDOM.render(<FavoriteColor />, document.getElementById('root'));
```

9.5 FUNCIONAMIENTO DE REACT

Ciclo de vida

Todo componente React tiene un ciclo de vida desde que se inicializa, se muestra, etc. A través de unos métodos concretos, se pueden introducir código para que se ejecute en el momento más conveniente.

El único que es obligatorio para todo componente React **es el método *render***. El resto son opcionales.

Prefijos

La lista de métodos tienen unos prefijos que nos dan pistas sobre el momento en el que se ejecuta:

- Empiezan por **will**: Se ejecutan **antes** de que pase algo.
- Empiezan por **did**: Se ejecutan **después** de que pase algo.

9.5 FUNCIONAMIENTO DE REACT

Ciclo de vida

Momentos clave

Hay tres momentos clave en el ciclo de vida de un componente en React:

- **Montaje del componente:** Es cuando el componente se crea y se inserta en el DOM del documento. Aquí se llama a los métodos:
 - constructor
 - componentWillMount
 - render
 - componentDidMount
- **Actualización del componente:** Ocurre cuando se producen cambios en el estado o en las propiedades, que también pueden pasarse entre componentes padres a hijos.
 - componentWillReceiveProps
 - shouldComponentUpdate
 - componentWillUpdate
 - render
 - componentDidUpdate

9.5 FUNCIONAMIENTO DE REACT

Ciclo de vida

Momentos clave

- **Desmontaje del componente:** Ocurre cuando este se elimina del DOM. En este caso se llama al siguiente método:
 - `componentWillUnmount`

Por lo tanto, si se declara cualquiera de esos métodos en el componente, se puede hacer que el código se ejecute en el momento que sea preciso.

Además, de las clases también se dispone de otros métodos que permiten llevar a cabo cambios en el componente:

- `setState`.
- `forceUpdate`.

Para más información sobre el ciclo de vida de los componentes podemos visitar la [URL oficial de React](#). (Ver ejemplo de reloj).

9.5 FUNCIONAMIENTO DE REACT

Multicomponentes

Hasta el momento hemos desarrollado pequeños ejemplos siempre con un único componente. Aunque se podría hacer así, resulta obvio que las aplicaciones deben dividirse en pequeños componentes; de hecho, esa es una de las mejores características de React. El poder organizar las interfaces de usuarios en un conjunto de componentes.

Programación hacia abajo

Este es un concepto clave que debe tenerse en cuenta. Al utilizar múltiples componentes se acabará construyendo una jerarquía de componentes, es muy probable que sea necesario pasar ciertas propiedades de un componente a otro: desde los padres a los hijos. Ejemplo :

```
<App name="My Title">
  <Section titleText={this.props.name}>
    <Title>{this.props.titleText}</Title>
  </Section>
</App>
```

9.5 FUNCIONAMIENTO DE REACT

Multicomponentes

Llama a funciones hacia arriba

La programación de propiedades va hacia abajo ; en cambio, desde un componente se puede llamar a un método que se encuentra en el elemento padre.

Las funciones pueden pasarse como una propiedad más.

Como ejemplo, se parte del componente base que ofrece el generador de aplicaciones de React, al que se le agrega un método que muestra un mensaje:

9.5 FUNCIONAMIENTO DE REACT

Multicomponentes

// Aplicación con un solo componente

```
class App extends Component {  
  showMesg(msg){  
    alert('This button works ' + msg)  
  }  
  render() {  
    return (  
      <div className='App'>  
        <header className='App-header'>  
          <img src={logo} className='App-logo' alt='logo' />  
          <h1 className='App-title'> Welcome to React</h1>  
        </header>  
        <p className="App-intro">  
          <button onClick={ () => this.showMesg('Hello')}>Click here</button>  
        </p>  
      </div>  
    )  
  }  
}  
  
export default App;
```


9.5 FUNCIONAMIENTO DE REACT

Multicomponentes

```
// Multicomponentes Fichero App.jsx
import React, {Component} from 'react';
import logo from './logo.svg';
import './App.css';
import Header from './header';
import Paragraph from './paragraph'
// Versión multicomponente
class App extends Component {
  showMesg(msg){
    alert('This button works ' + msg)
  }
  render() {
    return (
      <div className='App'>
        <Header logo={logo} />
        <Paragraph shMsg={this.showMesg} />
      </div>
    )
  }
}
export default App;
```

9.5 FUNCIONAMIENTO DE REACT

Multicomponentes

```
// Multicomponentes Fichero header.jsx
import { Component } from "react";

class Header extends Component {
  render () {
    return (
      <header className="App-header">
        <img src={this.props.logo} className="App-logo" alt="logo" />
        <h1 className="App-title">Welcome to React </h1>
      </header>
    )
  }
}

export default Header;
```

9.5 FUNCIONAMIENTO DE REACT

Multicomponentes

// Multicomponentes Fichero paragraph.jsx

```
import { Component } from "react";
```

```
class Paragraph extends Component {  
  render () {  
    return (  
      <div>  
        <p className="App-intro">  
          To get started edit <code>src/App.js</code> and save to reload.  
        </p>  
        <button onClick={() => this.props.shMsg('Hello')}> Click here</button>  
      </div>  
    )  
  }  
}  
export default Paragraph;
```

9.5 FUNCIONAMIENTO DE REACT

Versión mínima de componente

En React se puede crear componentes sin necesidad de crear una clase que herede de *React.Component*, ni siquiera metiendio un método *render*.

Son componentes que se declaran como funciones y no como clases, pueden tener parámetros o no y directamente hacen un *return* de su contenido.

```
// Declaración de Componente como función
export default function Header ({logo, title ="Welcome to React"}) {
  return (
    <header className="App-header">
      <img src={logo} className="App-logo" alt="logo" />
      <h1 className="App-title">{title} </h1>
    </header>
  )
}
```

9.5 FUNCIONAMIENTO DE REACT

React Fragments

Una de las condiciones básicas de React es que un componente debe de tener una etiqueta raíz, ya sea un componente o una etiqueta HTML. Eso significa que no se permite crear un método render que devuelva un par de bloques div sueltos: deben de ir dentro de otro elemento. Si no interesa que exista ese elemento padre que agrupe los bloques, esto se puede solucionar con `React.Fragment` Que es una etiqueta que permite agrupar elementos en un componente sin que se genere un elemento extra en el DOM o en el HTML final, pero cumpliendo las exigencias de JSX.

```
class Paragraph extends React.Component{
  render (){
    return (
      <React.Fragment>
        {this.props.phrases.map ( phrase =>
          <p>{phrase} </p>)}
      </React.Fragment>
    )
  }
}
export default Paragraph;
```

9.5 FUNCIONAMIENTO DE REACT

Formularios

Dentro de JSX también se puede introducir formularios que permiten poder introducir datos en las aplicaciones.

Existen dos tipos de maneras de utilizar elementos de formularios en React:

- Controlado o controlled : Se trata de un formulario en formato HTML, pero cuyos valores se gestionan al estilo React: cada campo del formulario se vincula a un valor de estado, así por cada cambio de campo se cambia el estado.
- Descontrolado o Uncontrolled : Sería la más similar a la tradicional, solo que, en React, se necesita utilizar el atributo *ref* para poder acceder a los campos.

9.5 FUNCIONAMIENTO DE REACT

Formularios

- **Campos controlados**

Esta es la manera de manejar los formularios asociando sus valores al estado del componente.

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

const Formuari = () => {

  const [dades, setDades] = useState({
    nom: '',
    cognom: ''
  })

  const handleImputChange = (event) => {
    console.log(event.target.value)
    setDades ({
      ...dades,
      [event.target.name]: event.target.value
    })
  }

  const enviarDades = (event) => {
    event.preventDefault();
    console.log(dades.nom + dades.cognom)
  }
}
```

9.5 FUNCIONAMIENTO DE REACT

Formularios

```
return (  
  <Fragment>  
    <h1>Formulari</h1>  
    <form className="row" onSubmit={enviarDades}>  
      <div className="col md-3">  
        <input  
          placeholder="introdueix el nom"  
          className="from-control"  
          type="input"  
          name="nom"  
          onChange={handleImputChange}  
        />  
      </div>  
      <div className="col md-3">  
        <input  
          placeholder="introdueix el cognom"  
          className="from-control"  
          type="input"  
          name="cognom"  
          onChange={handleImputChange}  
        />  
      </div>  
      <div className="col md-3">  
        <button  
          className="btn btn-primary"  
          type="submit"  
          onClick={enviarDades}  
        >Enviar</button>  
      </div>  
    </form>  
  </Fragment>  
)  
}  
  
export default Formulari;
```


9.5 FUNCIONAMIENTO DE REACT

Formularios

```
return (  
  <Fragment>  
    <h1>Formulari</h1>  
    <form className="row" onSubmit={enviarDades}>  
      <div className="col md-3">  
        <input  
          placeholder="introdueix el nom"  
          className="from-control"  
          type="input"  
          name="nom"  
          onChange={handleImputChange}  
        />  
      </div>  
      <div className="col md-3">  
        <input  
          placeholder="introdueix el cognom"  
          className="from-control"  
          type="input"  
          name="cognom"  
          onChange={handleImputChange}  
        />  
      </div>  
      <div className="col md-3">  
        <button  
          className="btn btn-primary"  
          type="submit"  
          onClick={enviarDades}  
        >Enviar</button>  
      </div>  
    </form>  
  </Fragment>  
)  
}  
  
export default Formulari;
```

9.5 FUNCIONAMIENTO DE REACT

Formularios

- **Campos descontrolados**

Como hemos comentado anteriormente la forma de acceder a los valores de los campos de un formulario es mediante el atributo **ref**. Todo aquello que tenga el atributo **ref** permite ser accedido desde los métodos del componente a través de la propiedad *this.refs.nombre_atributo.value*.

NOTA: *Este método está deprecated i no se recomienda su uso.*

9.5 FUNCIONAMIENTO DE REACT

Formularios

- Campos descontrolados

Ejemplo :

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

class Descontrolat extends Component {
  handleSubmit = (e) => {
    e.preventDefault();
    this.setState({
      login: this.refs.login.value,
      password: this.refs.password.value
    });
  }
  constructor(props) {
    super(props);
    this.state = {
      login: '',
      password: ''
    }
  }
  render() {
    return (
      <div>
        <form onSubmit={this.handleSubmit}>
          <div>
            <label> Login</label>
          </div>
          <div>
            <input type="text" ref="login" />
          </div>
          <div>
            <label> Password </label>
          </div>
          <div>
            <input type="password" ref="password" />
          </div>
          <div>
            <input type="submit" value="login" />
          </div>
        </form>
        <div> Login {this.state.login} , password { this.state.password}</div>
      </div>
    );
  }
};
export default Descontrolat;
```

9.5 FUNCIONAMIENTO DE REACT

Librerías de validación de formularios

La gestión de formularios, en especial su validación, es una de las tareas mas pesada pero necesarias en cualquier aplicación. Existen pequeñas librerías para facilitar en cierta medida esta tarea: una de las más populares en React es **Formik**. Gracias a ella se puede crear un componente con el mismo nombre cuyas propiedades básicas establece:

- Valores iniciales.
- Funciones de validación.
- Acción a ejecutar en el envío.

<https://formik.org/>

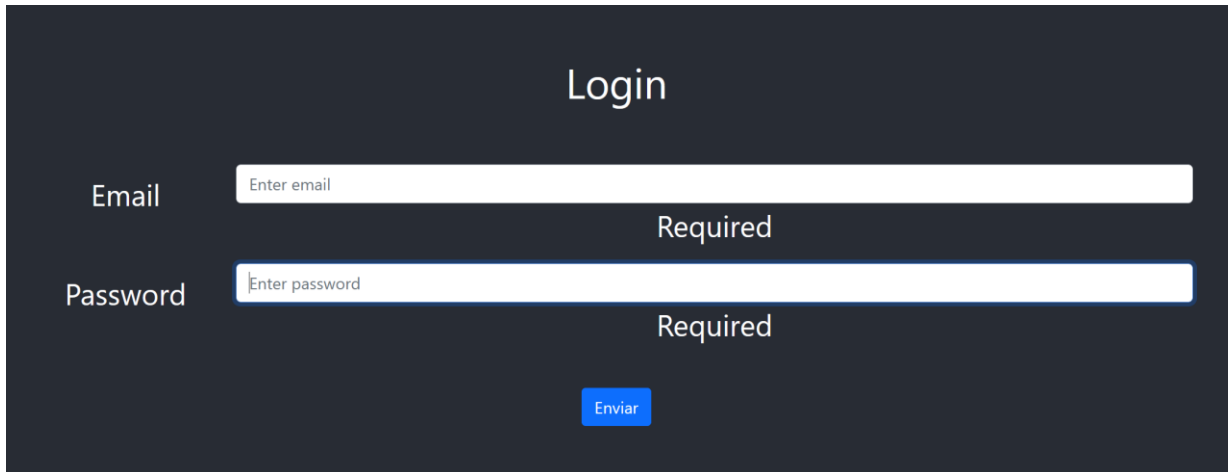
Otra componente que ayuda a validar los formularios es el componente YUP

<https://www.npmjs.com/package/react-yup>

9.5 FUNCIONAMIENTO DE REACT

Librerías de validación de formularios

Ejemplo simple :



A login form titled "Login" on a dark background. It contains two input fields: "Email" with a placeholder "Enter email" and "Password" with a placeholder "Enter password". Both fields are followed by the text "Required". Below the fields is a blue button labeled "Enviar".

```


Login



Email



Enter email



Required



Password



Enter password



Required



Enviar


```

9.5 FUNCIONAMIENTO DE REACT

Librerías de validación de formularios

Ejemplo simple :

```
import React from "react";
import { Formik, Form, Field } from "formik";
import "bootstrap/dist/css/bootstrap.css";
import * as Yup from 'yup';

const SignupSchema = Yup.object().shape({
  email: Yup.string().email('Invalid email').required('Required'),
  password: Yup.string().min(6, 'Too Short!').max(20, 'Too Long!').required('Required'),
});

class FormFormik extends React.Component {
  render() {
    return (
      <div className="container">
        <div className="row">
          <div className="col-lg-12">
            <Formik initialValues={{ email: "", password: "" }}
              validationSchema={SignupSchema}
              onSubmit={values => {
                console.log(values);
              }}
            >
          </div>
        </div>
      </div>
    );
  }
}
```

9.5 FUNCIONAMIENTO DE REACT

Librerías de validación de formularios

Ejemplo simple :

```
{{ errors, touched }} => (  
  <div>  
  
    <div className="row mb-5">  
      <div className="col-lg-12 text-center">  
        <h1 className="mt-5">Login</h1>  
      </div>  
    </div>  
  
    <Form>  
      <div className="mb-3 row">  
        <label htmlFor="email" className="col-sm-2 col-form-label">Email</label>  
        <div class="col-sm-10">  
          <Field  
            type="email"  
            name="email"  
            placeholder="Enter email"  
            autoComplete="off"  
            className="form-control"  
          />  
          {errors.email && touched.email ?  
            <div>{errors.email}</div> : null}  
        </div>  
      </div>  
    </Form>  
  </div>  
)
```

9.5 FUNCIONAMIENTO DE REACT

Librerías de validación de formularios

Ejemplo simple :

```
<div className="mb-3 row">
  <label htmlFor="password" className="col-sm-2 col-form-label"> Password </label>
  <div class="col-sm-10">
    <Field
      type="password"
      name="password"
      placeholder="Enter password"
      className="form-control"
    />
    {errors.password && touched.password ? (
      <div>{errors.password}</div>) : null}
  </div>

  <button
    type="submit"
    className="btn btn-primary btn-block mt-4"
  >
    Enviar
  </button>
</Form>
</div>
```


9.5 FUNCIONAMIENTO DE REACT

Librerías de validación de formularios

Ejemplo simple :

```
    })  
    </Formik>  
  </div>  
</div>  
</div>  
);  
}  
}  
  
export default FormFormik;
```

9.5 FUNCIONAMIENTO DE REACT

React Router

Por mucho que se tenga que organizar un componente en componentes más pequeños para facilitar su gestión, llega un momento en el que nuestra aplicación necesita ser capaz de poder pasar de un componente a otro, como si se hiciera un cambio de página. Para conseguirlo se utiliza la librería **React Router**.

Lo que se hace es definir una serie de **rutas o URL** que pueden ser requeridas y asociarles el componente correspondiente. Una vez definidas esas rutas, se pueden incorporar enlaces en la aplicación para poder pasar de un componente a otro. Además, se pueden pasar parámetros a través de los enlaces.

9.5 FUNCIONAMIENTO DE REACT

React Router

Para configurar las rutas debemos de importar el objeto *react-router-dom* e indicar en *index.jsx* que se van a utilizar el *BrowserRouter*.

```
import { BrowserRouter } from 'react-router-dom';  
ReactDOM.render(  
  <BrowserRouter>  
    <App />  
  </BrowserRouter>,  
  document.getElementById('root')  
);
```

Para mas información sobre React Route pincha [aquí](#).

9.5 FUNCIONAMIENTO DE REACT

React Router

Luego en el fichero app.jsx aplicaremos las rutas de nuestra aplicación :

```
import {
  Routes,
  Route
} from "react-router-dom";
export default function App() {

  return (
    <React.Fragment>
      <Header />
      <Container>
        <Routes>
          <Route path="/about" element={<About/>} />

          <Route path="/contacto" element={<Contacto/>} />

          <Route path="/users" element={<Users/>} />

          <Route path="/categoria/:id" element={<Categoria/>} />

          <Route path="/404" element={<Error404/>} />

          <Route path="/" element={<Home/>} />

          <Route path="*" element={<Error404/>} />
        </Routes>
      <Footer />
    </Container>
  </React.Fragment>
);
}
```

9.5 FUNCIONAMIENTO DE REACT

React Router

En el header.jsx tendremos los enlaces de los Endpoints.
Como se aprecia a continuación:

```
class Header extends Component{
  render (){
    return (
      <div>
        <Navbar bg="light" expand="lg">
          <Navbar.Brand href="/">React-Bootstrap</Navbar.Brand>
          <Navbar.Toggle aria-controls="basic-navbar-nav" />
          <Navbar.Collapse id="basic-navbar-nav">
            <Nav className="mr-auto">
              <Nav.Link href="/">Home</Nav.Link>
              <Nav.Link href="/about">About</Nav.Link>
              <Nav.Link href="/users">Users</Nav.Link>
              <Nav.Link href="/contacto">Contacto</Nav.Link>
              <NavDropdown title="News" id="basic-nav-dropdown">
                <NavDropdown.Item href="/categoria/1">New Products</NavDropdown.Item>
                <NavDropdown.Item href="/categoria/2">Awesome</NavDropdown.Item>
                <NavDropdown.Item href="/categoria/3">Something</NavDropdown.Item>
                <NavDropdown.Divider />
                <NavDropdown.Item href="/categoria/4">Last News</NavDropdown.Item>
              </NavDropdown>
            </Nav>
            <Form inline>
              <FormControl type="text" placeholder="Search" className="mr-sm-2" />
              <Button variant="outline-success">Search</Button>
            </Form>
          </Navbar.Collapse>
        </Navbar>
      </div>
    )
  }
}
```

9.5 FUNCIONAMIENTO DE REACT

React Router

Si queremos pasar parámetros al componente tendremos que controlar los parámetros que se pasan. Ver ejemplo:

```
import { Navigate, useParams } from 'react-router-dom';
function Categoria() {

  const {id} = useParams();

  switch(id) {
    case '1':
      return (<React.Fragment>
        <h1>New Products</h1>
        <a href="/">Home</a>
      </React.Fragment>
    );
    case '2':
      return (<React.Fragment>
        <h1>Awesome</h1>
        <a href="/">Home</a>
      </React.Fragment>
    );
    case '3':
      return (<React.Fragment>
        <h1>Something</h1>
        <a href="/">Home</a>
      </React.Fragment>
    );
    case '4':
      return (<React.Fragment>
        <h1>Last News</h1>
        <a href="/">Home</a>
      </React.Fragment>
    );
    default:
      return (<React.Fragment>
        <Navigate to="/404"/>
      </React.Fragment>
    );
  }
}

export default Categoria;
```

9.5 FUNCIONAMIENTO DE REACT

React-Bootstrap

Es la librería que reemplaza el JavaScript de Bootstrap. Cada componente se ha creado desde cero como un verdadero componente de React, sin dependencias innecesarias como jQuery.

Se ha creado teniendo en cuenta la compatibilidad, adoptando su núcleo de arranque y haciendo compatible con el ecosistema Bootstrap. Haciendo que funcione con los miles de temas de Bootstrap.

<https://react-bootstrap.github.io/>

9.5 FUNCIONAMIENTO DE REACT

React-Bootstrap

Instalación:

```
npm install react-bootstrap bootstrap @5.1.3
```

Importar los componentes necesarios:

```
import { Button } from 'react-bootstrap';
```

Importar CSS

- En el componente o aplicación.

```
import 'bootstrap/dist/css/bootstrap.min.css';
```

- Como CDN

[ver link](#)

Ver toda esta información actualizada en el siguiente link:

<https://react-bootstrap.github.io/getting-started/introduction>

9.5 FUNCIONAMIENTO DE REACT

React-Bootstrap

Ejemplo de uso:

Button #1

Button #2

Button #3

```
<Row className="mx-0">  
  <Button as={Col} variant="primary">Button #1</Button>  
  <Button as={Col} variant="secondary" className="mx-2">Button #2</Button>  
  <Button as={Col} variant="success">Button #3</Button>  
</Row>
```

Copy

9.5 FUNCIONAMIENTO DE REACT

React Hooks

Hasta ahora, la mayoría de componentes que se han mostrado son una clase. Sin embargo, es posible crear un componente en forma de función. Este es un patrón habitual y se suele aplicar cuando un componente simplemente muestra contenido estático alimentado o no por *properties*.

React Hooks es un mecanismo que permite crear componentes de este tipo, pero con estado. Mediante una configuración muy simple, se pueden inicializar, además de definir una función para su actualización. Gracias a esto, se pueden crear componentes muy sencillos, que además pueden tener estado.

9.5 FUNCIONAMIENTO DE REACT

React Hooks

El uso de propiedades en funciones se realiza de la siguiente forma:

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

<https://reactjs.org/docs/components-and-props.html>

9.5 FUNCIONAMIENTO DE REACT

React Hooks

En la página oficial de React tenemos un ejemplo que lo muestra de forma muy simple el uso de los estados.

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

export default function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);
  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

9.5 FUNCIONAMIENTO DE REACT

React Hooks

En la página oficial de React tenemos un ejemplo que lo muestra de forma muy simple el uso de los estados.

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

export default function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);
  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

9.5 FUNCIONAMIENTO DE REACT

React Hooks

Otra gran ventaja de React-hooks es que se simplifica el ciclo de vida del componente utilizando el `useEffect`. Con el siguiente ejemplo se explica su uso.

```
import React, { useState, useEffect } from 'react';
export default function Example() {
  const [count, setCount] = useState(0);

  // Similar to componentDidMount and componentDidUpdate:
  useEffect(() => {
    // Update the document title using the browser API
    document.title = `You clicked ${count} times`;
  });

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

9.5 FUNCIONAMIENTO DE REACT

React Context

Conforme se avanza el desarrollo de las aplicaciones de React, es probable que los programadores tenga la impresión de que muchas veces tienen que pasar propiedades desde componentes padres a los hijos pasando por varios componentes. Así como suele haber la necesidad de compartir determinado dato o método y en ocasiones se precisa gestionar la misma información entre componentes no directamente relacionados.

***React Context** es un mecanismo que permite definir valores para que sean compartidos dentro de un contexto o, lo que es lo mismo, en un conjunto de componentes.*

9.5 FUNCIONAMIENTO DE REACT

React Context

El valor a compartir puede ser cualquier cosa, desde un valor simple a un objeto que puede tener cualquier otro dato o incluso un método.

Primero se crea un contexto y , luego, se aplica en un componentes padr para que agrupe diferentes compoentes. A partir de ahí, todos los componentes que se definen en ese contexto tendran acceso a los datos de ese contexto.

Un uso típico del contexto podria ser para datos de usuario, con los que se distingue si esta en una sesion abierta o no. Eso podria modificar los menús, cabezeras y distintos componentes de una aplicación.

Más información : <https://reactjs.org/docs/context.html>

9.5 FUNCIONAMIENTO DE REACT

React Context

Ejemplo sencillo:

Definimos el contexto

```
//Fichero contexData.jsx
import React from 'react';

export const valores = { title: ' Super Program v1.0', color:'red'};
export const appContext = React.createContext (valores);
```

9.5 FUNCIONAMIENTO DE REACT

React Context

Ejemplo sencillo:

Apicamos el contexto a los componentes que agrupamos.

```
//Fichero App.jsx
import './App.css';
import {appContext, valores} from './Components/contextData'
import { Header } from './Components/header';
import {Footer} from './Components/footer';

function App() {
  return (
    <div className="App">
      <appContext.Provider value={valores}>
        <Header />
        <div> Cuerpo de del documento</div>
        <Footer />
      </appContext.Provider>
    </div>
  );
}

export default App;
```

9.5 FUNCIONAMIENTO DE REACT

React Context

Ejemplo sencillo:

Obtenemos el contexto para mostrarlo

```
//Fichero header.jsx
import React,{useContext} from 'react';
import logo from '../logo.svg';
import { appContext } from './contextData';
export const Header = () => {
  const valuesContext = useContext(appContext);
  return (
    <div className="App-header" style={{color: valuesContext.color}}>
      <img src={logo} className="App-logo" alt="logo" />
      <h1>This is the header</h1>
      <h2> {valuesContext.title}</h2>
    </div>
  )
}
```

9.5 FUNCIONAMIENTO DE REACT

React Context

Ejemplo sencillo:

Obtenemos el contexto para mostrarlo pero en el footer.

```
//Fichero footer.jsx
import React,{useContext} from 'react';
import { appContext } from './contextData';

export const Footer = () => {
  const valuesContext = useContext(appContext);
  return (
    <div>
      <small>This is the footer of {valuesContext.title}</small>
    </div>
  )
}
```

9.6 ACCESO AL API.

Una vez ya tenemos los conocimientos para poder crear una aplicación en react así como poder darle forma con bootstrap y react router, nos quedaria obtener y registrar los datos en nuestra API.

Este simple ejemplo nos muestra como mostrar datos de un API público:

```
import React,{useEffect,useState} from "react";
import { Button } from 'react-bootstrap';
import "bootstrap/dist/css/bootstrap.min.css";
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';
import { faEdit, faTrashAlt } from '@fortawesome/free-solid-svg-icons';

const Usuarios = () =>{

  const [users, setUsers] = useState([]);

  useEffect( ()=>{
    // obtener productos
    obtenerUsuarios();

  },[]);
```

9.6 ACCESO AL API.

Ejemplo GET.

```
const obtenerUsuarios = () =>{  
  fetch('https://jsonplaceholder.typicode.com/users')  
  .then(response => response.json())  
  .then(data => {  
    console.log(data);  
    setUsers(data);  
  });  
}
```

9.6 ACCESO AL API.

Ejemplo GET.

```
return(  
  <React.Fragment>  
    <h1>Usuarios</h1>  
    <Button onClick={()=>{console.log('agregar Usuario'); }}>Agregar Usuario</Button>  
  
    <table className="table ">  
      <thead>  
        <tr>  
          <th>Nombre</th>  
          <th>usuario</th>  
          <th>email</th>  
          <th>telefono</th>  
          <th>Acciones</th>  
        </tr>  
      </thead>  
    </table>  
  </React.Fragment>  
)
```

9.6 ACCESO AL API.

Ejemplo GET.

```
    <tbody>
      {users.map(user=>{
        return(
          <tr key={user.id}>
            <td>{user.name}</td>
            <td>{user.username}</td>
            <td>{user.email}</td>
            <td>{user.phone}</td>
            <td>
              <button className="btn btn-primary" onClick={()=>{console.log("modificar");}}>
                <FontAwesomeIcon icon={faEdit}/></button>
              {"  "}
              <button className="btn btn-danger" onClick={()=>{console.log("borrar");}}>
                <FontAwesomeIcon icon={faTrashAlt}/></button>
            </td>
          </tr>
        )
      })}
    </tbody>
  </table>
</React.Fragment>
);
}
```

```
export default Usuarios;
```