**ReactJS**

**REACT JS**

Es el paquete que nos trae toda la API de React js.

**REACT-DOM**

Nos permite interactuar con el DOM.

**BABEL-CORE**

Es el núcleo compilador de babel.

**BABEL-LOADER**

Este paquete nos permitirá trabajar con webpack.

**BABEL-PRESET-ES2015**

Este paquete nos permite codear con ecmascript 6.

**BABEL-PRESET-REACT**

Este paquete nos permite codear jsx .

**PACKAGE.JSON**

contiene información del proyecto, así como enumera las dependencias de npm, tanto para desarrollo como para producción.

**JSX**

Es una extensión de la sintaxis XML. Todo codigo javascript que se introduzca, debe ir encerrado por llaves

Sintaxis

return (

<h2>{Javascript Code}</h2>);

**Componentes**

Un componente es cada uno de los modulos de la aplicacion, es una trozo del sitio web que cumple un objetivo, son como etiquetas nuevas de HTML que encapsulan su contenido, presentación y funcionalidad. A base de tener unos componentes que engloban a otros es como se consigue la arquitectura de componentes

Sintaxis

class HelloMessage extends React.Component {

render() {

return (<h2>Hello </h2>);

}

}

**Dom Virual React**

El render del reactDOM solo admite una etiqueta, que en este caso seria <Component/>, para llamar varios componentes se debe usar la herencia de componentes, es decir dentro del componente “Component” debe estar dentro de etiquetas <div> y dentro llamar el resto de componentes

Sintaxis

class Component extends React.Component {

render() {

return (

<div>

<Component2/>

<Component3/>

</div>

);

}

}

ReactDOM.render(

<Component/>,

document.getElementById('root')

);

**State**

El **state** es el estado o caracteristicas propias de un componente. Los estados sólo se definen en el interior del componente y son de tipo ( Objeto {llave:valor}), y son declarados en el constructor del componente.

Sintaxis

class IntroducirTexto extends React.Component{

constructor(){

super();

this.state={ ///Creacion del state

nombre:"Juan" ///

}

}

cambiarNombre(event){

console.log(event.target.value) //Obteniendo el dato del onchange

this.setState({nombre: event.target.value}) //Seteando el nuevo valor del state

}

render(){

return (

<div>

<h2>{ this.state.nombre }</h2> //Llamada al state

<input onChange={this.cambiarNombre.bind(this)}/>

</div>

);

}

}

**Props**

**OnChange**

El evento onChange se lanza cuando un elemento select, text o textarea de un formulario pierde el focus (o cursor) y además su valor ha sido cambiado, esto es, si se está introduciendo datos en una casilla de texto, el evento no se activará hasta que no se pase a otra casilla. La sintaxis de su uso es:

Sintaxis

<input type="tipo de elemento" onChange="función a ejecutar">

cambiarNombre(event){

console.log(event.target.value) //Obteniendo el dato del onchange

this.setState({nombre: event.target.value})

}

render(){

return (

<div>

<h2>{ this.state.nombre }</h2>

<input onChange={this.cambiarNombre.bind(this)}/> //Aplicando el onchange al input

</div>

);

}

}