MI PRIMERA APLICACIÓN CON REACT

Primer paso: Instalación de SW

- **Node.js:** Node.js es un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor (pero no limitándose a ello) basado en el lenguaje de programación ECMAScript, asíncrono, con I/O de datos en una arquitectura orientada a eventos y basado en el motor V8 de Google. Nos permite compilar y ejecutar código JavaScript en el entorno del servidor.
- Descargar en *https://nodejs.org/es/download/*
- Una vez instalado podemos comprobar que está bien instalado en el terminal. *$ node -v*
- **Visual Studio Code**

Segundo paso: Creación de la App

- En el terminal ejecutamos *npx create-react-app mi-primera-app* donde **"mi-primera-app"** será el nombre de la aplicación (debe ser en minúsculas).
- Una vez acabado el proceso veremos como se nos ha creado una carpeta con el nombre de nuestra app, que contiene la estructura básica de la misma.
- La ejecutamos: *npm start*
- Vamos al navegador y en la barra de direcciones introducimos http://localhost:3000/. Veremos la página por defecto.

Tercer Paso: Modificación de la App

- Lanzamos el Visual Studio Code y editamos el archivo *App.js* (*mi-primera-app/src/App.js*)

c: > Users > jacobo > MiPrimeraAppReact > miprimerappreact > src > JS App.js > App

```
1   import React from 'react';
2   import logo from './logo.svg';
3   import './App.css';
4
5   function App() {
6     return (
7       <div className="App">
8         <header className="App-header">
9           <img src={logo} className="App-logo" alt="logo" />
10          <p>
11            Edit <code>src/App.js</code> and save to reload.
12          </p>
13          <a
14            className="App-link"
15            href="https://reactjs.org"
16            target="_blank"
17            rel="noopener noreferrer"
18          >
19            Learn React
20          </a>
21        </header>
22      </div>
23    );
24  }
25
26  export default App;
```

- Vamos a modificar la aplicación para que muestre un mensaje de bienvenida

c: > Users > jacobo > MiPrimeraAppReact > miprimerappreact > src > JS App.js > App

```
1   import React from 'react';
2   import logo from './logo.svg';
3   import './App.css';
4
5   function App() {
6     return (
7       <div className="App">
8         <header className="App-header">
9           <img src={logo} className="App-logo" alt="logo" />
10          <h1>Hola</h1>
11        </header>
12      </div>
13    );
14  }
15
16  export default App;
17
```

- Guardamos el archivo y volvemos al navegador para ver cómo ha cambiado

Cuarto paso: Creación de componentes.

- React se basa en componentes. De hecho todo son componentes: botones, tablas, formularios, cajas de texto... La creación de componentes nos permite aislar funcionalidades y reutilizar código.
- Vamos a crear un componente (*stateless*) para la cabecera H1 que hemos puesto.
- Creamos una carpeta llamada "*components*" dentro de "*src*" y en el editor creamos un nuevo archivo que llamaremos *HelloComponent.js*
- Vemos que todos los componentes tienen un método *render()* que es el que dibuja o, más bien, genera dinámicamente el código HTML que se visualizará en el navegador para ese componente.

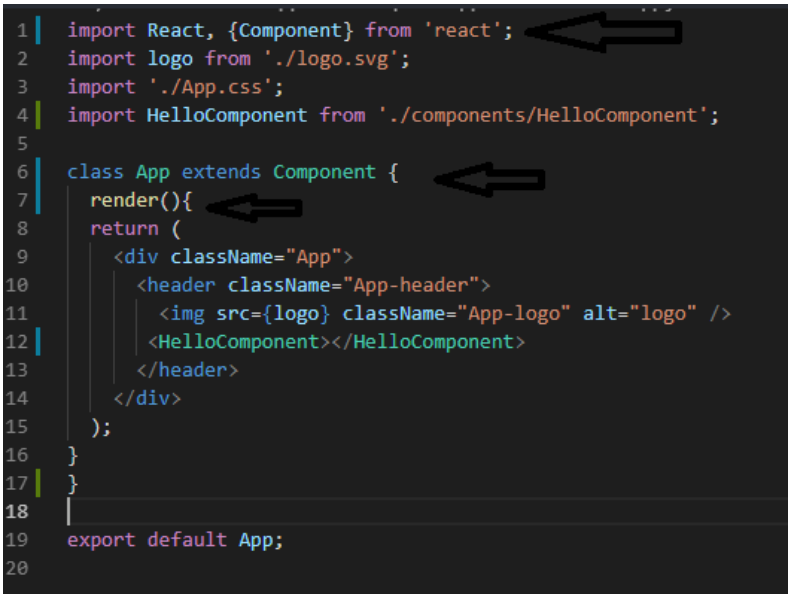
```
import React, {Component} from 'react'

export default class HelloComponent extends Component{
    render(){
        return (
            <div>
             <h1>Hola</h1>
            </div>
        )
    }
}

```

- Ahora debemos modificar el código del archivo *App.js* para que use este nuevo componente, en vez de la etiqueta H1

```jsx
import React from 'react';
import logo from './logo.svg';
import './App.css';
import HelloComponent from './components/HelloComponent';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <HelloComponent></HelloComponent>
      </header>
    </div>
  );
}

export default App;
```

- No debemos olvidarnos de hacer el *import* del componente. Guardamos el archivo y volvemos al navegador para comprobar que seguimos viendo el mensaje de bienvenida.

- Vamos a hacer unas pequeñas modificaciones en el archivo *App.js* para convertirla en un componente más.



```
import React, {Component} from 'react';
import logo from './logo.svg';
import './App.css';
import HelloComponent from './components/HelloComponent';

class App extends Component {
  render(){
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
       <HelloComponent></HelloComponent>
      </header>
    </div>
  );
}
}

export default App;
```

Quinto Paso: Asociar estado a nuestros Componentes.

- En React tenemos componentes "*stateful*" con estado y "*stateless*". En los primeros, sus atributos (*state*) varían en el tiempo, tienen un estado que definen y actualizan, y cada cambio tanto en *props* como en su estado invoca al método *render()*. En los componentes sin estado, los únicos datos con los que trabajan este tipo de componentes es con las *props* recibidas.
- Vamos a hacer que el componente principal, la app, tenga estado. Este estado tendrá un atributo "*name*", al que le damos el valor inicial de nuestro nombre.

```
import React, {Component} from 'react';
import logo from './logo.svg';
import './App.css';
import HelloComponent from './components/HelloComponent';

class App extends Component {
  constructor(){
    super()
    this.state={
     name: 'Jacobo'

    }

  }
  render(){
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
       <HelloComponent></HelloComponent>
      </header>
    </div>
  );
}
}

export default App;
```

Sexto Paso: Comunicación entre un componente padre con su hijo (props).

- En React los componentes están relacionados. El componente principal, la clase *App*, es padre del componente "*HelloComponent*".
- La comunicación del padre (*App*) con el hijo (*HelloComponent*) se hace a través de *props*, o propiedades.

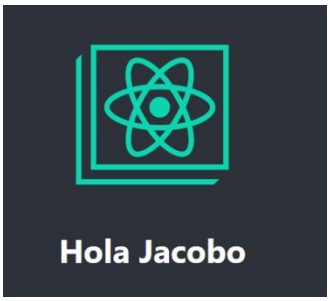
- Ahora vamos a hacer que el atributo "*name*" se le pase al componente hijo "*HelloComponent*" como parámetro para que lo muestre junto a "Hola".
- Introducimos la siguiente modificación en el archivo *App.js*

```
render(){
return (
  <div className="App">
    <header className="App-header">
      <img src={logo} className="App-logo" alt="logo" />
     <HelloComponent nombre={this.state.name}></HelloComponent>
    </header>
  </div>
);
```

- Estamos diciendo que la *prop* "*nombre*" del "*HelloComponent*" tome el valor del atributo "*name*" del estado del componente principal. También debemos modificar el componente "*HelloComponent*" . No tenemos que definir la variable, simplemente indicar que la vamos a usar.

```
1  import React, {Component} from 'react'
2
3
4  export default class HelloComponent extends Component{
5
6      render(){
7          return (
8              <div>
9              <h1>Hola {this.props.nombre}</h1>
10             </div>
11         )
12     }
13 }
14
```

- Guardamos los archivos y vemos el resultado en el navegador.



Séptimo Paso: Eventos.

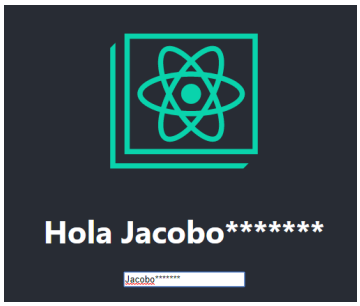
- Vamos ahora a hacer que cuando se introduzca un texto en una caja de texto, se actualice el atributo "*name*" del estado del componente principal "*App*". La actualización del atributo "*name*" provocará la actualización en todos los componentes que usen ese atributo. Es decir, cada vez que escribamos en la caja de texto, también se modificará el atributo "*nombre*" de las *props* del componente "*HelloComponent*".
- En primer lugar definimos en el archivo *App.js* un método "*changeName*" que ante un evento (por ejemplo, teclear en una caja de texto), actualiza su estado con el texto introducido.

```
class App extends Component {
  constructor(){
    super()
    this.state={
     name: 'Jacobo'

    }

  }
  changeName=(event)=>{
    this.setState({
      name: event.target.value
    })

  }
  render(){
```

- Ahora asociamos ese método con el evento "*Onchange*" de la caja de texto.

```
render(){
return (
  <div className="App">
    <header className="App-header">
      <img src={logo} className="App-logo" alt="logo" />
     <HelloComponent nombre={this.state.name} ></HelloComponent>
     <input value={this.state.name} onChange={this.changeName}></input>
     </header>
  </div>
);
```

- Haremos que la *prop* "*value*" de la caja de texto tome el valor del atributo "*name*" del estado del componente principal (App).
- Guardamos el archivo, vamos al navegador y vemos como todo lo que escribimos en la caja de texto aparece automáticamente escrito junto al "hola".



Octavo Paso: Comunicación entre un componente hijo con su padre (callbacks).

- Vamos a crear otro componente. Una caja de texto (*stateless*) que sustituirá a la original <*input*> de HTML. Creamos un nuevo archivo llamado "*InputComponent*"**.**

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

export default class InputComponent extends Component{
    render(){
        return (
            <div>
                <input value={this.props.nombre}></input>
                {/*Hacemos que el valor que aparezca inicialmente se el del atributo "name" del props del input */}
            </div>
        );
    }
}

```

-

- Igual que hicimos antes, haremos que la *prop* "*nombre*" del "*InputComponent*" tome el valor del atributo "*name*" del estado del componente principal (*App.js*).

```
render(){
return (
  <div className="App">
    <header className="App-header">
     <img src={logo} className="App-logo" alt="logo" />
    <HelloComponent nombre={this.state.name} ></HelloComponent>
    <InputComponent nombre={this.state.name}  ></InputComponent>
    </header>
  </div>
);
```

- De nuevo recordar que no debemos olvidarnos de hacer el *import* del nuevo componente.
- Vamos a hacer ahora, igual que hicimos con el <*input*>, que cuando se escriba en la caja de texto, se actualice el texto que aparece junto a "hola". En este caso debemos a añadir el evento "*OnChange*" en el componente (*InputComponent*).

```
1  import React, {Component} from 'react'
2
3  export default class InputComponent extends Component{
4      render(){
5          return (
6              <div>
7                  <input value={this.props.name} onChange={this.props.cambiarNombre}></input>
8
9              </div>
10         );
11     }
12 }
13
14
```

- Con esto, estamos diciendo que cuando alguien escriba en la caja de texto (evento *Onchange*) se ejecute el valor que tenga el atributo

"*cambiarNombre*" del *props* del "*InputComponent*". Así es como se comunica un componente "hijo" con su "padre", invocando métodos del padre. Esto es lo que se llama "*callback*".

- Modificamos ahora el archivo *App.js*.

```
render(){
return (
  <div className="App">
    <header className="App-header">
      <img src={logo} className="App-logo" alt="logo" />
     <HelloComponent nombre={this.state.name} ></HelloComponent>
      <InputComponent name={this.state.name}  cambiarNombre={this.changeName}></InputComponent>
    </header>
  </div>
);
```

- Podemos comprobar que haciendo este cambio se establece el valor del atributo "*cambiarNombre*" del "*InputComponent*" como el método "*changeName*" del archivo App.js

- Guardamos y lo probamos en el navegador.