React Básico

Class	Finalizado
Reviewed	☐
Certificado	☐

1-Introduccion a React

¿Qué es un SPA?

Una web SPA (Single page application) es una forma de desarrollo web en la que la página web está contenida en un único archivo HTML. Mientras navegamos por la web, se van solicitando los contenidos al servidor. Se mejoran los tiempos de respuesta y, por tanto la experiencia de usuario.

React es una libreria de JavaScript para crear interfaces de usuario.

Caracteristicas:

- Velocidad ⇒ Uno de los aspectos que mas se destacan, esto lo consigue trabajando sobre un DOM Virtual sobre el que aplica los cambios que sufra la aplicacion y luego actualiza unicamente los elementos que se hayan modificado.
- Componentes ⇒ Permiten separar la interfaz de usuario en piezas independientes, reutilizables y pensar cada pieza de forma aislada. Nos proporciona varios tipos de componentes: puros, de contenedor, de clase, de funcion, etc.

- Desarrollo Declarativo ⇒ Las aplicaciones que creemos estaran formadas por camponentes. Tanto la aplicacion global como cada componente tiene un estado propio, y es por este motivo que React es declarativo.
- Anidacion de Componentes ⇒ Pueden ser anidados, los componentes de orden superior propagan datos a los de orden inferior. Su comunicacion es unidireccional y se usan los eventos para que los componentes inferiores sean reconocidos por los de orden superior.
- Isomorfismo ⇒ Conocido como JavaScript Universal, capacidad con la que podemos renderizar tanto en el servidor como en el cliente. Esto hace que solucione problemas y mejore el posicionamiento.
- Agilidad de Desarrollo ⇒ Disponemos de todas las funcionalidades que nos ofrece JQuery, ambas tecnologias pueden convivir.

2-Componentes y jerarquía

Un componente es un elemento que es reutilizable, que son reactivos, que reaccionen a eventos, que sean dinamicos, etc. En src agregar una carpeta llamada components y adentro creamos una carpeta llamada pure, donde vamos a crear componentes puros, componentes sencillos que no tengan demasiada lógica, ni tampoco complejidad, se utilizan las props, que es para pasar info de padre a hijo. Creamos un componente llamado Greeting.jsx. Elemento html que podamos renderizar agregandolo al arbol de componentes.

Extensiones ⇒

- Babel JavaScript
- Debugger for Chrome
- Better Comments (documentar código)
- Bootstrap
- Bracket Pair Colorizer 2
- CodeStream
- Color Highlight
- ES7/Redux/GraphQL...
- EsLint
- Image Preview
- Jest / Jest Test Explorer

- Js JSX Snippets
- npm / npm intellisense
- Version Lens

Crear nuestro primer componente de tipo clase

```
import React, { Component } from 'react';
import PropTypes from 'prop-types';
// extendemos una clase de componente, y tenemos un
//render que nos devuelve algo dentro del return
class Greeting extends Component {
  constructor(props){
    super(props)
    this.state = {
      age: 29
    }
  }

  render(){
    return (
      <div>
        <h1>
          Hola! { this.props }
        </h1>
        <h2>
          Tu edad es: { this.state.age }
        </h2>
        <div>
          <button onClick={ this.birthday }>
            cumplir años
          </button>
        </div>
      </div>
    );
  }

  birthday = () => {
    this.setState((prevState) => {
      {
        age: prevState.age + 1
      }
    })
  }
}

Greeting.propTypes = {
  name: PropTypes.string,
};

export default Greeting;
```

Los componentes de tipo clase, tienen constructores, tienen clases, tienen métodos propios, etc.

Tienen un propio constructor y podemos pasarle informacion, para darle informacion debemos hacerlo desde un constructor.

Tenemos `props` que es informacion que va a recibir mediante atributos, las props que va a recibir va a venir en este caso desde el app.js. Pero tambien tenemos `state` para manejar el estado del componente de clase.

El `this.state` es un dato o valor totalmente privado, es información que tiene el componente que no puede modificarse, y que cuando lo haga actualizaria la vista. Se pueden generar nuevos estados que significan una nueva actualizacion, se pueden realizar con una función llamada `setState,` es el único metodo que nos va a generar un nuevo estado y actualizaria la vista, si no lo hacemos la vista quedará inmutada, con el `prevState` le pasamos el estado previo en una `arrow function` y le sumamos +1, con el `onClick` en el button habremos generado un estado que actualice por ejemplo en este caso de 29 a 30, etc.

El `render()` debe devolver con un `return()` el html, podemos renderizar determinadas variables.

Los `propTypes` nos dice que tipo de dato es que le estamos pasando, son el contenido que yo le puedo pasar a un contenido de orden superior, requieren de tipos, en este caso le decimos que el proptypes es de tipo `string`, pero pueden ser arrays, objects, number, booleans, etc. En react solo podemos devolver un elemento padre, en este caso es el `<div></div>.`

Para importar un archivo lo que debemos hacer es poner en App, en la parte superior:

```
import Greeting from './components/pure/greeting';
```

Los nombres de componentes van con mayúscula.

Para saber si tenemos anidado Greeting en el App, debemos ir a las developer web tools.

Crear un componente de tipo función

Los componentes que se usan en react hoy son los componentes funcionales. Para hacer un componente funcional:

```
import React, { useState} from 'react';
import PropTypes from 'prop-types';

const Greetingf = (props) => {
//funcion que nos permite crear un estado privado
//const [variable, metodo para actualizar] = useState(estado inicial)
```

```
  const [age, setAge] = useState(29);

  //Actualizamos el State
  const birthday = () => {
    setAge(age + 1);
  }

  return (
    <div>
        <h1>
          Hola! { props.name } desde componente
          funcional!
        </h1>
        <h2>
          Tu edad es: { this.state.age }
        </h2>
        <div>
          <button onClick={ this.birthday }>
            cummplir años
          </button>
        </div>
      </div>
  )
};

Greetingf.propTypes = {
  name: propTypes.string;
};

export default Greeting;
```

Como es un función no tenemos `props` como en el componente de clase, tampoco tenemos `state`, debemos utilizar un `hooks` y las props la tenemos que definir manualmente. Adentro de la arrow function le ponemos `props,` y trae las propiedades que se le van a pasar, incluso se le pueden especificar que tipo de props le vamos a pasar ⇒ `({name})`. En `PropTypes` tambien le ponemos que tipo de dato le vamos a pasar, en este caso es `string`. Para renderizarlo, lo llevamos a App.js asi ⇒ `<Greetingf name=”German”></Greetingf>`. En este caso German es nuestro props llamado `name`. Al ser una funcion no tenemos un constructor como tal, entonces tenemos que ver como manejamos el estado del componente. Existe `useState`, que es una funcion que nos permite crear un estado privado para este componente de tipo función, para traernos el hooks, debemos importarlo:

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

Entonces lo debemos definir:

Su estructura es la siguiente ⇒

```
const [variable(va a tener el valor inicial), método para
actualizarlo(actualiza el valor inicial)] = useState(valor inicial);
```

```
const [age, setAge] = useState(29);
```

Luego actualizamos el State ⇒

```
const birthday = () => {
  setAge(age + 1);
}
```

Genera un nuevo estado, y luego actualizara el componente.

⇒ El componente de tipo clase es como se trabajaba tradicionalmente en React,es una clase, tiene constructores, tiene un estado privado, tiene props, etc, por otro lado tiene funciones y métodos privados. El componente de clase necesita el setState para poder actualizar el componente y poder renderizar los cambios en la vista.

⇒ El componente de tipo función es mucho mas sencillo, lo podemos embeber en cualquier sitio, la podemos utilizar como un callback, podemos utilizarlos para en cualquier momento generar una funcion que devuelva un elemento html sin necesidad de crear clases. Esto hace que sea muy fácil declarar componentes en React y muy fácil utilizarlos, es recomendable trabajar con componentes de tipo función en su gran mayoria. Además existen hooks, no solo para trabajar con el state, sino para utilizar referencias, trabajar con el contexto de los datos.

Formas de trabajar:

Existen varias tecnicas:

1. Empezar por componentes mayores, cuando se detectan que se repiten algun componente podemos sacar esos elementos repetidos.
2. Diseño modular, bloques mas pequeños y vamos subiendo a bloques mayores.

Hicimos las tareas, componentes que vamos a renderizar con una tarea por default(), clases que nos va a traer las tareas, y componente que nos dice el estado de la tarea.

3-Los hooks de React

Es una buena práctica al trabajar con VsCode es tener .editorconfig ⇒ Es un archivo, tiene su extension, nos permite configurar como queremos tener el indentado, que las tabulaciones ocupen un determinado espacio.

```
indent_style = tab
end_of_line = lf
insert_final_newline = true
tab_width = 2
charset = utf-8
trim_trailing_whitespace = true
```

Los Hooks son funciones que se incorporaron en el año 2018, simplifican o especifican mecanismos para trabajar con componentes funcionales.

Ejemplo 1 ⇒

src/hooks/Ejemplo1.jsx

```
//Ejemplo de uso de useState()
// Crear componente de tipo funcion, acceder a su estado
//privado a traves de un hooks, y ademas, poder modificarlo.
import React, { useState } from "react"

const Ejemplo1 = () => {
  // Valor inicial para un contador
  const valorInicial = 0;

 // Valor inicial para una persona
  const personaInicial = {
    nombre : "German",
    email: "german_94_tw@hotmail.com"
  }

  // Queremos que el ValorInicial y PersonaInicial
  // sea parte del estado del componente, para gestionar su
  // cambio para reflejarlo en la vista del componente.

 // const {nombreVariable,fucnionparacambiar} = useState(valorinicial)

  const {contador, setContador} = useState(valorInicial)
  const {persona, setPersona} = useState(personaInicial)

/*
  Funcion para actualizar el estado privado que contiene el contador
*/
  function incrementalContador(){
    //? funcionParaCambiar(nuevoValor)
      setContador(contador + 1)
  }
//Funcion para actualizar el estado de persona en el comp
  function actualizarPersona(){
      setPersona(
```

```
          {
            nombre: "Pepe",
            email: "pepe@imaginagroup.com"
          }
      )
  }

  return (
    <div>
      <h1>Ejemplo de useState</h1>
      <h2>CONTADOR : {contador}</h2>
      <h2>DATOS DE LA PERSONA:</h2>
      <h3>NOMBRE: {persona.nombre}</h3>
      <h3>EMAIL: {persona.email}</h3>
      <button onClick={incrementarContador}>Incrementar contador</button>
      <button onClick={actualizarPersona}>Actualizar persona</button>
    </div>
  )
}
export default Ejemplo1;
```

App.js

```
function App(){
  <Ejemplo1 />
  <Ejemplo2 />
}
```

Ejemplo 2 ⇒

useEffect() ⇒ Nos va a ayudar a controlar los cambios en la vista.

useRef() ⇒ Referenciar elementos dentro de la vista.

src/hooks/Ejemplo2.jsx

```
/*
  Ejemplo de uso de:
  -useState()
  -useRef()
  -useEffect()
*/
import React, {useState, useRef, useEffect} from "react"

const Ejemplo2 = (){
  const {contador1,setContador1} = useState(0)
  const {contador2,setContador2} = useState(0)
//Creamos una referencia para asociar una variable con un
//elemento del DOM del componente (vista HTML)
  const miRef = useRef()

  function incrementar1(){
    setContador1(contador1 + 1)
  }
  function incrementar2(){
    setContador2(contador2 + 1)
```

```
  }

  //Trabajando con useEffect
  //Caso 1 => Ejecutar siempre un snippet de codigo
  // cada vez que cambia el componente se ejecuta un
  //aquello que esta dentro del useEffect()
  useEffect(() => {
      console.log("Cambio en el estado del comp")
      console.log("Mostrando ref al elemento DOM")
      console.log(miRef)
  })


  return (
      <div>
        <h1>Ejemplo de useState(),useRef(),useEffect()</h1>
        <h2>CONTADOR 1: {contador1}</h2>
        <h2>CONTADOR 2: {contador2}</h2>
        {/*Elemento referenciado*/}
        <h4 ref=(miRef)>
          Ejemplo de elemento referenciado
        </h4>
        <div>
            <button onClick={incrementar1}>Incrementar contador 1</button>
            <button onClick={incrementar2}>Incrementar contador 2</button>
        </div>
      </div>
  )
}

export default Ejemplo2;
```

caso 2 ⇒

```
/*
  Ejemplo de uso de:
  -useState()
  -useRef()
  -useEffect()
*/
import React, {useState, useRef, useEffect} from "react"

const Ejemplo2 = (){
  const {contador1,setContador1} = useState(0)
  const {contador2,setContador2} = useState(0)
//Creamos una referencia para asociar una variable con un
//elemento del DOM del componente (vista HTML)
  const miRef = useRef()

  function incrementar1(){
    setContador1(contador1 + 1)
  }
  function incrementar2(){
    setContador2(contador2 + 1)
  }

  //Trabajando con useEffect
  //Caso 2 => Ejecutar solo cuando cambie contador 1
  // cada vez que cambia contador 1, se ejecuta lo
```

```
  //que diga el useEffect(). Si cambia contador 2 no
  //habrá cambio
  useEffect(() => {
      console.log("Cambio en el estado del contador 1")
      console.log("Mostrando ref al elemento DOM")
      console.log(miRef)
  },[contador1])


  return (
      <div>
        <h1>Ejemplo de useState(),useRef(),useEffect()</h1>
        <h2>CONTADOR 1: {contador1}</h2>
        <h2>CONTADOR 2: {contador2}</h2>
        {/*Elemento referenciado*/}
        <h4 ref=(miRef)>
          Ejemplo de elemento referenciado
        </h4>
        <div>
            <button onClick={incrementar1}>Incrementar contador 1</button>
            <button onClick={incrementar2}>Incrementar contador 2</button>
        </div>
      </div>
  )
}

export default Ejemplo2;
```

Solo podemos tener un `useEffect()`, o si declaramos varios, el ultimo va a ser el que se ejecutará.

caso 3 ⇒

```
/*
  Ejemplo de uso de:
  -useState()
  -useRef()
  -useEffect()
*/
import React, {useState, useRef, useEffect} from "react"

const Ejemplo3 = (){
  const {contador1,setContador1} = useState(0)
  const {contador2,setContador2} = useState(0)
//Creamos una referencia para asociar una variable con un
//elemento del DOM del componente (vista HTML)
  const miRef = useRef()

  function incrementar1(){
    setContador1(contador1 + 1)
  }
  function incrementar2(){
    setContador2(contador2 + 1)
  }

  //Trabajando con useEffect
  //Caso 3 => Ejecutar solo cuando cambie contador 1 o contador 2
  // cada vez que cambia contador 1, se ejecuta lo
```

```
  //que diga el useEffect(). Si cambia contador 2 ejecuto lo que
  //diga useEffect()
  useEffect(() => {
      console.log("Cambio en el estado del contador 1/ contador 2")
      console.log("Mostrando ref al elemento DOM")
      console.log(miRef)
  },[contador1, contador2])


  return (
      <div>
        <h1>Ejemplo de useState(),useRef(),useEffect()</h1>
        <h2>CONTADOR 1: {contador1}</h2>
        <h2>CONTADOR 2: {contador2}</h2>
        {/*Elemento referenciado*/}
        <h4 ref=(miRef)>
          Ejemplo de elemento referenciado
        </h4>
        <div>
            <button onClick={incrementar1}>Incrementar contador 1</button>
            <button onClick={incrementar2}>Incrementar contador 2</button>
        </div>
      </div>
  )
}

export default Ejemplo2;
```

caso 4 ⇒

useContext() ⇒ Especialmente interesante para trabajar con datos, poder acceder o utilizar el contexto y pasarselo a componentes que sean inferiores:

```
/*
  Ejemplo de hooks:
    -useState()
    -useConntext()
*/
import React, {useState, useContext} from "react"


const miContexto = React.createContext(null)
//Componente1 => Dispone de un contexto que va a tener
//un valor que recibe desde el padre
const Componente1 = () => {
  const state= useContext(miContexto)
  //Inicializamos un estado vacio que va a rellenarse
  //con los datos del contexto del padre
  return (
    <div>
      <h1>El Token es: {state.token}</h1>
      {/*Pintamos el componente 2*/}
      <Componente2></Componente2>
    </div>
  )
}
```

```
const Componente2 = () => {

  const state = useContext(miContexto)
  return (
    <div>
      <h2>La sesion es : {state.sesion}</h2>
    </div>
  )
}


const default function MiComponenteConContexto(){
  const estadoInicial = {
    token: '234324893',
    sesion: 1
  }


  //creamos el estado de este componente
  const [sessionData, setSessionData] = useState(estadoInicial)

  function actualizarSesion(){
    sessionData(
      {
        token: '12738127312',
        session: sessionData.sesion + 1
      }
    )
  }

  return(
    <miContexto.Provider value={sessionData}>
      {/*Todo lo que este aqui dentro puede leer los datos
      de sessionData. Ademas si se actualizan los componente de
      aqui tambien se actualizan*/}
      <Componente1></Componente1>
      <button onClick={actualizarSesion}>Actualizar Sesion</button>
    </miContexto.Provider>
  )
}

export default Ejemplo4;
```

caso 5 ⇒

props.children() => Podemos pasar elementos html

```
import React from 'react'

const Ejemplo5 = (props) => {
  return (
    <div>
      <h1>Ejemplo de CHILDREN PROPS</h1>
      <h2>
        Nombre:{props.children}
      </h2>
      {props.children}
    </div>
```

```
  )
}
```

4-Dando estilos a un proyecto de React

Podemos trabajar con css y sass, usar sass nos permite hacer uso de variables y mixins. Bootstrap utiliza scss y sass.

Hay que instalar ⇒ `npm i --save node-sass` nos va a permitir utilizar sass, y luego hacer el build. En app.js en la primer linea tenemos el `import './App.css'`

En css podemos utilizar la especificidad. Cada componente tiene su propia hoja de estilo. Se van a heredar los estilos siempre que tenga un padre que los haya definido.

`src/styles/task.scss`

```
h1 {
  color: cyan;
}
.task-name {
  font-weight: bold;
  color: tomato;
}
```

Luego importar la hoja de estilo de task.scss como ⇒ `import '.../../styles/task.scss'`, luego

```
return(
  <h2 className="task-name">
    Nombre: {task.name}
  </h2>
)
```

`src/components/pure/greetingStyle.jsx:`

```
import React,{ useState } from 'react'
//Definiendo estilos en constantes:
// Estilos para usuario logueado
const loggedStyle = {
  color: 'white',
};
// Estilo para usuario no logueado
const unloggeadedStyle = {
  color: 'tomato',
  fontWeight: 'bold',
}
```

```
const Greetingstyled = () => {

  //Generamos un estado para el componente
  // y si el usuario esta o no logueado
  const [logged, setLogged] = useState(false)

  return (
    <div style={logged ? loggedStyle : unloggedStyle}>
        {logged ?
            {<p>Hola, {props.name}</p>}
            :
            {<p>Please Login</p>
        }


        <button onClick={() => {
          console.log("Boton pulsado")
          setLogged(!logged);
        }}>
          {logged ? 'Loggout' : 'Login'}
        </button>
    </div>
  )
}
```

Instalar bootstrap ⇒ `npm i bootstrap --save`, luego ir a nuestro elemento principal, e importarlo ⇒ `import 'bootstrap/dist/css/bootstrap.css'`

Importante ⇒ Los estilos propios deben ir debajo de bootstrap para que no nos pise.

5-Manejo de Eventos

Un evento puede ocurrir tanto en el padre como en el hijo. En la carpeta `container` creamos un componente llamado `father.jsx` ⇒

```
import React from "react";
import Child from "../pure/child";

const Father = () => {
  return (
    <div>
      <Child />
    </div>
  );
};
export default Father;
```

y en la carpeta pure creamos el `child.jsx` ⇒

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

const Child = ({ name, send, update }) => {
  const messageRef = useRef("");
  const nameRef = useRef();

  function pressButton() {
    const text = messageRef.current.value;
    alert(`Text in Input: ${text}`);
  }
  function pressButtonParams(text) {
    alert(`Text: ${text}`);
  }
  function submitName(e) {
    e.preventDefault();
    update(nameRef.current.value);
  }

  return (
    <div style={{ backgroundColor: "blue", padding: "30px" }}>
      <p
        onMouseOver={() => console.log("On mouse over 1")}
        style={{ color: "white" }}
      >
        Hello, {name}
      </p>
      <button onClick={() => console.log("Press Button 1")}>Botón 1</button>
      <button onClick={pressButton}>Botón 2</button>
      <button onClick={() => pressButtonParams("Hello")}>Botón 3</button>
      <input
        placeholder="Send a text to your father"
        onFocus={() => console.log("Input focused")}
        onChange={(e) => console.log("Input changed:", e.target.value)}
        onCopy={() => console.log("Copied text from Input")}
        ref={messageRef}
      />
      <button onClick={() => send(messageRef.current.value)}>
        Send Message
      </button>
      <div style={{ marginTop: "20px" }}>
        <form onSubmit={submitName}>
          <input ref={nameRef} placeholder="New Name" />
          <button type="submit">Update Name</button>
        </form>
      </div>
    </div>
  );
};
export default Child;
```

onMouseOver ⇒ React ocurre cuando el puntero del mouse se mueve sobre un elemento (un **div** , un **botón** , una **entrada** , un área de **texto** , etc.). La función del controlador de eventos se activará y podremos ejecutar nuestra lógica allí.

onFocus ⇒ Hacemos foco en el elemento.

onChange ⇒ Evento que se dispara cuando hay un tipo de cambio.

onCopy ⇒ Si hemos copiado el contenido.

Todos aquellos que se empiezan con on son eventos, se controlan de esta manera.

Podemos llamar a un metodo que no esta dentro del componente a través de un evento, con el metodo onClick={}, en el componente padre debemos pasarle por props la funcion que queremos que se ejecute en el otro componente. Esto se ejecuta en father pero lo esta ejecutando en child. Hacemos una refencia que es el hook que nos sirve para referenciar elementos en el DOM.

6-Depuración del código

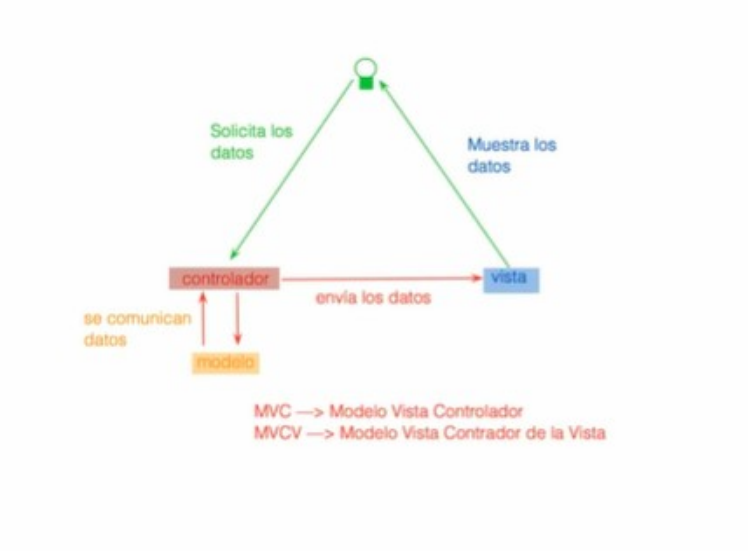
Depuracion busca resolver errores. Es cuando algo falla poder controlar ese evento y poner un punto de ruptura. En los developer tools, en los componentes. El watch sirve para observar expresiones o valores que pueden ser de interes.

En visual studio code podemos utilizar el debugger, debemos instalar la extension de chrome y nos va a lanzar una carpeta llamada vscode con un launch.json, eso podemos agregarlo al .gitignore para que no se suba al desplegar nuestro proyecto.

7-Buenas prácticas de estructuración de proyectos

Esquema de Modelo-Vista-Controlador

Cada componente esta creado por un modelo de datos. Tenemos una vista asociada a cada componente, a cada controlador.



Vistas en el proyecto

Estamos hablando de un componente que va a engloblar otros componentes. Esta pensado para englobar otros componentes de orden inferior. Estas pages o views contienen componentes que nos permitan interactuar con comp de orden inferior.

Vistas y paginas, contenedores y puros. La diferencia entre una vista y un componente contenedor es que una vista pueden tener varios componentes contenedor. mientras un componente contenedor engloba un componente puro internamente para pasarle determinada informacion y lógica. Las operaciones mas complejas tambien van a estar en un componente contenedor.

El modelo esta asociado a los datos que se deben representar en la vista a través del controlador, el componente puede ser privado o puede ser global. Con redux va a actualizar el estado global de la aplicacion y que los componentes puedan estar actualizados constantemente.

Herramientas para peticiones HTTP: Axios

Las peticiones https las hace el controlador, debemos asociar determinadas logicas para poder que actue como un middleware entre la vista y el controlador, para poder estar persistiendo entre el local y sesion storage. Lo hace a traves de una rest full.

Axios nos traera los datos, el modelo que nosotros tenemos asociados a unos datos que vengan desde afuera.

8: Aplicando renderizado condicional a la aplicación

src/components/pure/optionalRender.jsx ⇒

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

const OptionalRender = () => {
  const [access, setAccess] = useState(true);

  const updateAccess = () => {
    setAccess(!access);
  };

  let optionalButton;
  if (access) {
    optionalButton = <button onClick={updateAccess}>Logout</button>;
  } else {
    optionalButton = <button onClick={updateAccess}>Login</button>;
  }

  return <div>{optionalButton}</div>;
};
export default OptionalRender;
```

Otra forma de renderizar distintos elementos en nuestra vista:

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

// Login / Logout buttons
const LoginButton = ({ loginAction }) => {
  return <button onClick={loginAction}>Login</button>;
};
const LogoutButton = ({ logoutAction }) => {
  return <button onClick={logoutAction}>Logout</button>;
};

// ? (Expresion true) && expresion => Se renderiza la expresion
// ? (Expresion false) && expresion => No se renderiza la expresion

const OptionalRender = () => {
  const [access, setAccess] = useState(true);
  const [nMessages, setNMessages] = useState(0);

  const loginAction = () => {
    setAccess(true);
  };
  const logoutAction = () => {
    setAccess(false);
  };

  let optionalButton;
  if (access) {
    optionalButton = <LogoutButton logoutAction={logoutAction}></LogoutButton>;
  } else {
```

```
    optionalButton = <LoginButton loginAction={loginAction}></LoginButton>;
  }

  // Unread messages
  let addMessages = () => {
    setNMessages(nMessages + 1);
  };

  return (
    <div>
      {/* Optional Buttons */}
      {optionalButton}
      {/* N messages unread */}
      {nMessages > 0 && nMessages === 1 && (
        <p>You have {nMessages} new message...</p>
      )}
      {nMessages > 1 && <p>You have {nMessages} new messages</p>}
      {nMessages === 0 && <p>There are no new messages</p>}
      <button onClick={addMessages}>Add new message</button>
    </div>
  );
};
export default OptionalRender;
```

Otra manera de renderizar dos elementos segun su caso, es hacerlo con un operador ternario, pero solo se puede hacer con dos elementos, con mas no se puede realizar.

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

// Login / Logout buttons
const LoginButton = ({ loginAction }) => {
  return <button onClick={loginAction}>Login</button>;
};
const LogoutButton = ({ logoutAction }) => {
  return <button onClick={logoutAction}>Logout</button>;
};

// ? (Expresion true) && expresion => Se renderiza la expresion
// ? (Expresion false) && expresion => No se renderiza la expresion

const OptionalRender = () => {
  const [access, setAccess] = useState(true);
  const [nMessages, setNMessages] = useState(0);

  const loginAction = () => {
    setAccess(true);
  };
  const logoutAction = () => {
    setAccess(false);
  };

  let optionalButton;
  if (access) {
    optionalButton = <LogoutButton logoutAction={logoutAction}></LogoutButton>;
  } else {
    optionalButton = <LoginButton loginAction={loginAction}></LoginButton>;
  }
```

```
  // Unread messages
  let addMessages = () => {
    setNMessages(nMessages + 1);
  };

  return (
    <div>
      {/* Optional Buttons */}
      {optionalButton}
      {/* Ternay Operator */}
      {nMessages > 0 ? (
        <p>
          You have {nMessages} new messages {nMessages > 1 ? "s" : null}{" "}
        </p>
      ) : (
        <p>There are no new messages</p>
      )}
      <button onClick={addMessages}>
        {nMessages === 0 ? "Add your first message" : "Add new message"}
      </button>
    </div>
  );
};
export default OptionalRender;
```

Es fundamental en React utilizar el operador ternario. El renderizado condicional, consiste en pintar varias veces el elemento, puede ser tambien para los estilos, como en greetingStyle.

Renderizado condicional para poner estilos según el caso ⇒

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

let red = 0;
let green = 200;
let blue = 150;

//Estilo para usuario logueado
const loggedStyle = {
  backgroundColor: "red",
  color: "black",
  fontWeight: "bold"
};
//Estilo para usuario no logueado
const unloggedStyle = {
  backgroundColor: `rgb(${red},${green},${blue})`,
  color: "black",
  fontWeight: "bold"
};

// Login / Logout buttons
const LoginButton = ({ loginAction, propStyle }) => {
  return (
    <button style={propStyle} onClick={loginAction}>
      Login
    </button>
  );
```

```
};
const LogoutButton = ({ logoutAction, propStyle }) => {
  return (
    <button style={propStyle} onClick={logoutAction}>
      Logout
    </button>
  );
};

// ? (Expresion true) && expresion => Se renderiza la expresion
// ? (Expresion false) && expresion => No se renderiza la expresion

const OptionalRender = () => {
  const [access, setAccess] = useState(false);
  const [nMessages, setNMessages] = useState(0);

  const loginAction = () => {
    setAccess(true);
  };
  const logoutAction = () => {
    setAccess(false);
  };

  let optionalButton;
  if (access) {
    optionalButton = (
      <LogoutButton
        propStyle={loggedStyle}
        logoutAction={logoutAction}
      ></LogoutButton>
    );
  } else {
    optionalButton = (
      <LoginButton
        propStyle={unloggedStyle}
        loginAction={loginAction}
      ></LoginButton>
    );
  }

  // Unread messages
  let addMessages = () => {
    setNMessages(nMessages + 1);
  };

  return (
    <div>
      {/* Optional Buttons */}
      {optionalButton}
      {/* Ternay Operator */}
      {access ? (
        <div>
          {nMessages > 0 ? (
            <p>
              You have {nMessages} new message{nMessages > 1 ? "s" : null}{" "}
            </p>
          ) : (
            <p>There are no new messages</p>
          )}
          <button onClick={addMessages}>
            {nMessages === 0 ? "Add your first message" : "Add new message"}
```

```
          </button>
        </div>
      ) : null}
    </div>
  );
};
export default OptionalRender;
```

9-Formularios

Utilizamos Formik y Yup. Siempre al form debemos poner el e.preventDefault(e), a través del onSubmit. El botón debe tener submit.

Formik ⇒ OpenSource. Crear formularios a traves de un com llamado formik, en combinacion con yup que es un esquema para validacion, que lo hace mas facil. Para instalarlo ⇒ npm i –save formik yup

Con Yup vamos a crear un esquema de valores.

```
import React from "react";
import { Formik, Field, Form, ErrorMessage } from "formik";
import * as Yup from "yup";
//Creamos un objeto, el form esta compuesto email y pasw
//shape permite especificar la estructura de ese objeto
const loginSchema = Yup.object().shape({
  //Tipos de datos, como strings, arrays
  //Email espera el mensaje de error en caso de que no cumpla
  email: Yup.string()
    .email("Invalid email format")
    .required("Email is required"),
  password: Yup.string().required("Password is required")
});

const LoginFormik = () => {
  const initialCredentials = {
    email: "",
    password: ""
  };

  return (
    <div>
      <h4>Login Form</h4>
      <Formik
        /* Initial values that the form will take */
        initialValues={initialCredentials}
        /* Validation schema */
        validationSchema={loginSchema}
        /* onSubmit Event */
        onSubmit={async (values) => {
          await new Promise((r) => setTimeout(r, 1000));
          alert(JSON.stringify(values, null, 2));
          /* We save the data in the localStorage */
          localStorage.setItem("credentiasl", values);
        }}
      >
```

```
        {/* We obtein props from Formik */}
        {({
          errors,
          touched,
          values,
          isSubmitting,
          handleBlur,
          handleChange
        }) => {
          return (
            <Form>
              <label htmlFor="email">Email</label>
              <Field
                type="email"
                id="email"
                name="email"
                placeholder="example@example.com"
              />
              {/* Emails error */}
              {errors.email && touched.email && (
                <ErrorMessage name="email" component="div" />
              )}
              <label htmlFor="password">Password</label>
              <Field
                id="password"
                name="password"
                placeholder="************"
                type="password"
              />
              {/* Password errors */}
              {errors.password && touched.password && (
                <ErrorMessage name="password" component="div" />
              )}
              <button type="submit">Enviar</button>
              {isSubmitting ? <p>Login your credentials...</p> : null}
            </Form>
          );
        }}
      </Formik>
    </div>
  );
};
export default LoginFormik;
```

10-Sistema de enrutado de React JS

Esencial de las SPA, vamos a utilizar react-router-dom, en este caso es la version 5, ahora esta en la version 6. Cambian las cosas.

Para instalarlo ⇒ `npm i - -save react-router-dom`

Creamos todo la enrutacion principal, la raiz del programa es `HomePage.jsx` por eso le ponemos exact ⇒

```
import { BrowserRouter as Router, Route, Link, Switch } from "react-router-dom";
import HomePage from "./pages/home/HomePage";
```

```
import NotFoundPage from "./pages/404/NotFoundPage";
import AboutPage from "./pages/about-faqs/AboutPage";
import ProfilePage from "./pages/profile/ProfilePage";
import TaskPage from "./pages/tasks/TaskPage";
import TasksDetailspage from "./pages/tasks/TasksDetailspage";

export default function AppRoutingOne() {
  return (
    <Router>
      <div>
        <aside>
          <Link to="/">|| Home | </Link>
          <Link to="/about">| ABOUT | </Link>
          <Link to="/faqs">| FAQs || </Link>
          <Link to="/any404">| Not Existing Route || </Link>
        </aside>
        <main>
          <Switch>
            <Route exac path="/" component={HomePage} />
            <Route path="/(about|faqs)" component={AboutPage} />
            <Route path="/profile" component={ProfilePage} />
            <Route path="/tasks" component={TaskPage} />
            <Route path="/tasks/:id" component={TasksDetailspage} />
            {/* 404 Page Not Found */}
            <Route component={NotFoundPage} />
          </Switch>
        </main>
      </div>
    </Router>
  );
}
```

Creamos los componentes de la ruta.

HomePage.jsx ⇒

```
import React from "react";
import { useHistory, useLocation } from "react-router-dom";

const HomePage = () => {
  const location = useLocation();
  const history = useHistory();

  console.log("We are in Route:", location.pathname); //'about | faqs '

  const navigate = (path) => {
    history.push(path);
  };

  return (
    <div>
      <h1>Home Page</h1>
      <button onClick={() => navigate("/profile")}>Go To Profile</button>
    </div>
  );
};
export default HomePage;
```

NotFoundPage.jsx ⇒

```
import React from "react";
import { useHistory } from "react-router-dom";

const NotFoundPage = () => {
  const history = useHistory;
  const navigateTo = (path) => {
    history.push(path);
  };

  return (
    <div>
      <h1>404-Page not found</h1>
      <button onClick={() => navigateTo("/")}>Go Back To Home</button>
    </div>
  );
};
export default NotFoundPage;
```

AboutPage.jsx ⇒

```
import React from "react";
import { useLocation, useHistory } from "react-router-dom";

const AboutPage = () => {
  const location = useLocation();
  const history = useHistory();

  console.log("We are in Route:", location.pathname);

  const navigate = (path) => {
    history.push(path);
  };

  const goBack = () => {
    history.goBack();
  };

  const goForward = () => {
    history.goForward();
  };

  return (
    <div>
      <h1>About | FAQs Page</h1>
      <div>
        <button onClick={() => navigate("/")}>Go To Home</button>
        <button onClick={goBack}>Go Back</button>
        <button onClick={goForward}>Go Forward</button>
      </div>
    </div>
  );
};
export default AboutPage;
```

ProfilePage.jsx ⇒

```
import React from "react";
import { useHistory } from "react-router-dom";

const ProfilePage = ({ user }) => {
  const history = useHistory();

  const goBack = () => {
    history.goBack();
  };

  return (
    <div>
      <h1>Your Profile</h1>
      <button onClick={goBack}>Go Back</button>
    </div>
  );
};
export default ProfilePage;
```

TaskDetailspage.jsx ⇒

```
import React from "react";
import { useParams } from "react-router-dom";

const TasksDetailspage = () => {
  const { id } = useParams();

  return (
    <div>
      <h1>Task Details - {id}</h1>
    </div>
  );
};
export default TasksDetailspage;
```

11-Gestión avanzada de rutas en React JS

AppRoutingOne.jsx ⇒

```
import {
  BrowserRouter as Router,
  Route,
  Link,
  Switch,
  Redirect
} from "react-router-dom";
import HomePage from "./pages/home/HomePage";
import NotFoundPage from "./pages/404/NotFoundPage";
import AboutPage from "./pages/about-faqs/AboutPage";
import ProfilePage from "./pages/profile/ProfilePage";
import TaskPage from "./pages/tasks/TaskPage";
```

```
import TasksDetailspage from "./pages/tasks/TasksDetailspage";
import LoginPage from "./pages/auth/LoginPage";
import { useEffect } from "react";
import Statepage from "./pages/home/Statepage";

export default function AppRoutingOne() {
  const logged = false;

  let taskList = [
    { id: 1, name: "task 1", description: "My first Task" },
    { id: 2, name: "task 2", description: "My second Task" }
  ];

  useEffect(() => {
    const logged = localStorage.getItem("credentials");
    console.log("User logged?", logged);
  }, []);

  return (
    <Router>
      <div>
        <aside>
          <Link to="/">|| Home | </Link>
          <Link to="/about"> ABOUT | </Link>
          <Link to="/faqs"> FAQs | </Link>
          <Link to="/task/1"> Task 1 | </Link>
          <Link to="/task/2"> Task 2 | </Link>
          <Link to="/any404"> Not Existing Route | </Link>
          <Link to="/login"> LOGIN || </Link>
        </aside>
        <main>
          <Switch>
            <Route exact path="/" component={HomePage} />
            <Route exact path="/online-state" component={Statepage} />
            <Route path="/login" component={LoginPage}>
              {logged
                ? () => {
                    alert("You must be logged in, Redirecting go home...");
                    return <Redirect to="/login" />;
                  }
                : () => {
                    return <LoginPage />;
                  }}
            </Route>
            <Route path="/(about|faqs)" component={AboutPage} />
            <Route path="/profile" component={ProfilePage}>
              {logged ? (
                <ProfilePage />
              ) : (
                () => {
                  alert("You must be logged in, Redirecting go login...");
                  return <Redirect to="/" />;
                }
              )}
            </Route>
            <Route path="/tasks" component={TaskPage} />
            <Route
              exact
              path="/tasks/:id"
              render={({ match }) => (
                <TasksDetailspage task={taskList[match.params.id - 1]} />
```

```
              )}
            ></Route>
            {/* 404 Page Not Found */}
            <Route component={NotFoundPage} />
          </Switch>
        </main>
      </div>
    </Router>
  );
}
```

HomePage.jsx ⇒

```
import React from "react";
import { useHistory, useLocation } from "react-router-dom";

const HomePage = () => {
  const location = useLocation();
  const history = useHistory();

  console.log("We are in Route:", location.pathname); //'about | faqs '

  const navigate = (path) => {
    history.push(path);
  };

  const navigateProps = (path) => {
    history.push({
      pathname: path,
      search: "?online=true", //Query Params
      state: {
        online: true
      }
    });
  };

  return (
    <div>
      <h1>Home Page</h1>
      <button onClick={() => navigateProps("/online-state")}>
        Go To Page with State / Query Params
      </button>
      <button onClick={() => navigate("/profile")}>Go To Profile</button>
    </div>
  );
};
export default HomePage;
```

Statepage.jsx ⇒

```
import React from "react";
import { useLocation } from "react-router-dom";

const Statepage = () => {
  const location = useLocation();
  console.log("Location State:", location.state.online); //State sent
```

```
  console.log("Query Params:", location.search); //Query params

  return (
    <div>
      <h1>
        State:
        {location.state.online ? "The user is Online" : "The user is Offline"}
      </h1>
    </div>
  );
};
export default Statepage;
```

TaskDetailpage.jsx ⇒

```
import React from "react";
import { useParams } from "react-router-dom";

const TasksDetailspage = ({ task }) => {
  const { id } = useParams();

  return (
    <div>
      <h1>Task Details - {id}</h1>
      <h2>{task.name}</h2>
      <h3>{task.description}</h3>
    </div>
  );
};
export default TasksDetailspage;
```

LoginPage.jsx ⇒

```
import React from "react";
import LoginFormik from "../../components/pure/forms/loginFormik";

const LoginPage = () => {
  return (
    <div>
      <h1>Login Page</h1>
      <LoginFormik />
    </div>
  );
};
export default LoginPage;
```

LoginFormik.jsx ⇒

```
import React from "react";
import { useHistory } from "react-router-dom";
import { Formik, Field, Form, ErrorMessage } from "formik";
import * as Yup from "yup";
//Creamos un objeto, el form esta compuesto email y pasw
```

```
//shape permite especificar la estructura de ese objeto
const loginSchema = Yup.object().shape({
  //Tipos de datos, como strings, arrays
  //Email espera el mensaje de error en caso de que no cumpla
  email: Yup.string()
    .email("Invalid email format")
    .required("Email is required"),
  password: Yup.string().required("Password is required")
});

const LoginFormik = () => {
  const initialCredentials = {
    email: "",
    password: ""
  };

  const history = useHistory();

  return (
    <div>
      <h4>Login Form</h4>
      <Formik
        /* Initial values that the form will take */
        initialValues={initialCredentials}
        /* Validation schema */
        validationSchema={loginSchema}
        /* onSubmit Event */
        onSubmit={async (values) => {
          await new Promise((r) => setTimeout(r, 1000));
          alert(JSON.stringify(values, null, 2));
          /* We save the data in the localStorage */
          await localStorage.setItem("credentiasl", values);
          history.push("/profile");
        }}
      >
        {/* We obtain props from Formik */}
        {({
          errors,
          touched,
          values,
          isSubmitting,
          handleBlur,
          handleChange
        }) => {
          return (
            <Form>
              <label htmlFor="email">Email</label>
              <Field
                type="email"
                id="email"
                name="email"
                placeholder="example@example.com"
              />
              {/* Emails error */}
              {errors.email && touched.email && (
                <ErrorMessage name="email" component="div" />
              )}
              <label htmlFor="password">Password</label>
              <Field
                id="password"
                name="password"
```

```
                placeholder="************"
                type="password"
              />
              {/* Password errors */}
              {errors.password && touched.password && (
                <ErrorMessage name="password" component="div" />
              )}
              <button type="submit">Enviar</button>
              {isSubmitting ? <p>Login your credentials...</p> : null}
            </Form>
          );
        }}
      </Formik>
    </div>
  );
};
export default LoginFormik;
```

12-Repaso con procesos asíncronos

Procesos asincronos en js, entender las promesas, pueden tardar X tiempos. Async nos va a permitir trabajar con la asincronia dentro de cuaquier funcion, await nos sirve para hacer paradas dentro de nuestro codigo, nos ayuda a que no lancemos codigo antes de que llegue la peticion.

AsyncExample.jsx ⇒

En una promesa tenemos:

resolve ⇒ Devolver el dato final.

all ⇒ Devolver el resultado final con todas las promesas que se tienen que resolver.

reject ⇒ generar una promesa rechazada por algun tipo de error.

race ⇒ Generar una promesa que se resuelve o se rechaza cuando otras promseas son resueltas o rechazadas.

allSettled ⇒ Nos sirve para comprobar cuando tosa las promesas estan listas.

Hasta que una promesa no se resuelve no tenemos el valor final.

Con el método then() es la manera en la cual nosotros esperamos la resolucion de esa promesa. Con el método catch() nos devuelve un error, es un reject en una promesa. Con el finally() es para mostrar que a promesa finalizó.

```
import React from "react";

const Asyncexample = () => {
  async function generateNumber() {
    return 1;
  }

  async function generatePromiseNumber() {
    return Promise.resolve(2);
  }

  function obtainNumber() {
    generateNumber()
      .then((response) => alert(`Response: ${response}`))
      .catch((error) => alert(`Something went wrong: ${error}`));
  }
  function obtainPromiseNumber() {
    generatePromiseNumber()
      .then((response) => alert(`Response: ${response}`))
      .catch((error) => alert(`Something went wrong: ${error}`));
  }

  async function saveSessionStorage(key, value) {
    sessionStorage.setitem(key, value);
    return Promise.resolve(sessionStorage.getItem(key));
  }

  function showStorage() {
    saveSessionStorage("name", "German")
      .then((response) => {
        let value = response;
        alert(`Saved Name: ${value}`);
      })
      .catch((error) => alert(`Something went wrong: ${error}`))
      .finally(() => console.log("SUCCESS: Name saved and retreived"));
  }

  async function obtainMessage() {
    let promise = new Promise((resolve, reject) => {
      setTimeout(() => resolve("Hello World"), 2000);
    });

    let message = await promise;
    await alert(`Message received: ${message}`);
  }

  const returnError = async () => {
    await Promise.reject(new Error("Oooooops!"));
  };

  const consumeError = () => {
    returnError()
      .then((response) => alert(`Everything is OK: ${response}`))
      .catch((error) => alert(`Something went wrong: ${error}`))
      .finally(() => alert("Finally executed"));
  };

  const urlNotFound = async () => {
    try {
      let response = await fetch("https://invalidURL.com");
```

```
      alert(`Response: ${JSON.stringify(response)}`);
    } catch (error) {
      alert(`Something went wrong with your URL: ${error}{check your console}`);
    }
  };

  const multiplePromise = async () => {
    let results = await Promise.all([
      fetch("https://reqres.in/api/users"),
      fetch("https://reqres.in/api/users?page=2")
    ]).catch((error) =>
      alert(`Something went wrong with your URL: ${error} {check your console}`)
    );
  };

  return (
    <div>
      <h1>Async, Promise example</h1>
      <button onClick={obtainNumber}>Obtain Number</button>
      <button onClick={obtainPromiseNumber}>Obtain Promise Number</button>
      <button onClick={showStorage}>Save Name and Show</button>
      <button onClick={obtainMessage}>Received message in 2 seconds</button>
      <button onClick={consumeError}>Obtain Errors</button>
      <button onClick={urlNotFound}>Request to Unknown URL</button>
      <button onClick={multiplePromise}>Multiple Promises</button>
    </div>
  );
};
export default Asyncexample;
```

Los observables se puede devolver varios o un valor, por jemplo una función devuelve algo y ya esta, un iterador devuelve varios valores de forma sincrona mientras que de forma asincrona tenemos las promesas para devolver un valor mientras que los observables devuelven multiples valores.

Armamos una carpeta llamada services, dentro creamos un archivo observableService.js ⇒

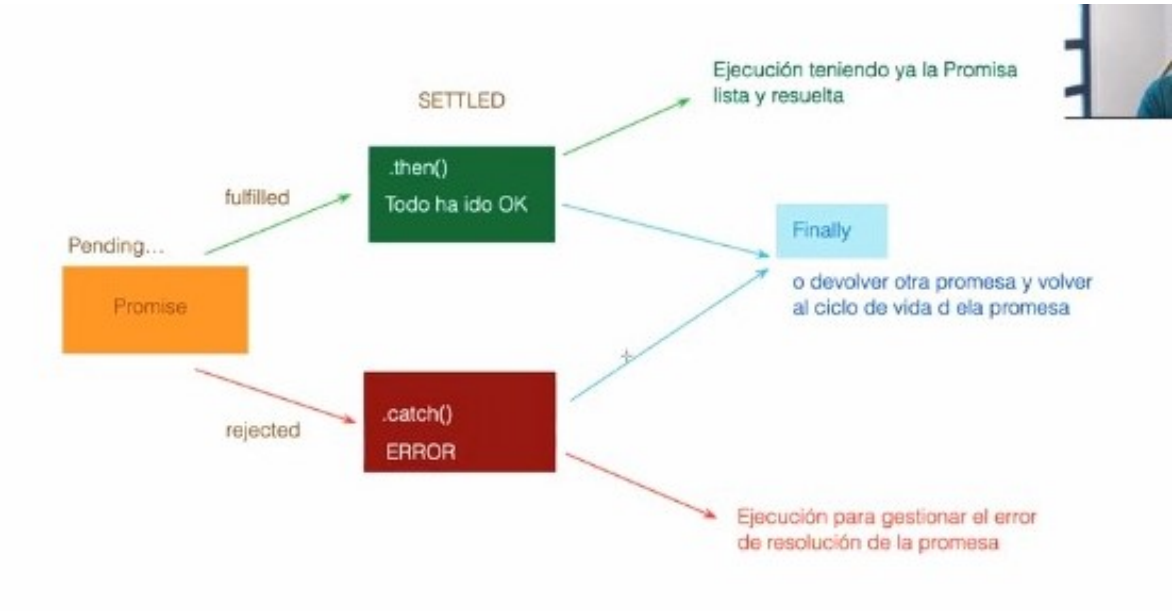
```
 import { Observable } from "rxjs";

export const getNumbers = new Observable((subscriber) => {
  // We Emit values
  subscriber.next(1); // Emits 1
  subscriber.next(2); // Emits 2
  subscriber.next(3); // Emits 3
  setTimeout(() => {
    subscriber.next(4); // Emits 4
    subscriber.complete(); // Finally, the Observable completes & finishes
  }, 1000); // Waits is
});
```

En la carpeta pure:

ObservableExample.jsx ⇒

```
import React, { useState } from "react";
import { getNumbers } from "../../services/observablesService";

const Observableexample = () => {
  const [number, setNumber] = useState(0);

  const obtainNewNumber = () => {
    console.log("Subscription to Observable");
    getNumbers.subscribe({
      next(newNumber) {
        console.log("New Number:", newNumber);
        setNumber(newNumber);
      },
      error(error) {
        alert(`Something went wrong: ${error}`);
        console.log("Error in observable");
      },
      complete() {
        alert(`Finished with: ${number}`);
        console.log("Done with the observale");
      }
    });
  };

  return (
    <div>
      <h2>Number: {number}</h2>
      <button onClick={obtainNewNumber}>Obtain new Numbers</button>
    </div>
  );
};
export default Observableexample;
```

Asi funcionan los procesos asincronicos ⇒



13-Realizando peticiones HTTP con Fetch

src/services/fetchService.js ⇒

```
// LLamar a todos los users
export const getAllUsers = async () => {
  let response = await fetch(`https://reqres.in/api/users`);

  console.log("Response:", response);
  console.log("Status:", response.status);
  console.log("Ok?:", response.ok);
  // We return the json
  return response.json();
};

export const getAllPagedUsers = async (page) => {
  let response = await fetch(`https://reqres.in/api/users?page=${page}`);

  console.log("Response:", response);
  console.log("Status:", response.status);
  console.log("Ok?:", response.ok);
  // We return the json
  return response.json();
};

export const getUserDetails = async (id) => {
  let response = await fetch(`https://reqres.in/api/users/${id}`);

  console.log("Response:", response);
  console.log("Status:", response.status);
  console.log("Ok?:", response.ok);
  // We return the json
  return response.json();
};
```

src/components/pure/FetchExample.jsx ⇒

```
import React, { useState, useEffect } from "react";
import {
  getAllPagedUsers,
  getAllUsers,
  getUserDetails
} from "../../services/fetchService";

const Fetcheexample = () => {
  const [users, setUsers] = useState([]);
  const [selectedUser, setSelectedUsers] = useState(null);
  const [totalUsers, setTotalUsers] = useState(12);
  const [usersPerPages, setUsersPerPages] = useState(6);
  const [pages, setPages] = useState(2);

  useEffect(() => {
    obtainUsers();
  }, []);

  const obtainUsers = () => {
```

```
  getAllUsers()
    .then((response) => {
      console.log("All users", response.data);
      setUsers(response.data);
      setTotalUsers(response.total);
      setUsersPerPages(response.per_pages);
      setPages(response.total_pages);
    })
    .catch((error) => alert(`Error while retreiving your users: ${error}`))
    .finally(() => {
      console.log(`Ended obtaining users:`);
      console.table(users);
    });
};

const obtainPagedUsers = (page) => {
  getAllPagedUsers(page)
    .then((response) => {
      console.log("All Paged users", response.data);
      setUsers(response.data);
      setTotalUsers(response.total);
      setUsersPerPages(response.per_pages);
      setPages(response.total_pages);
    })
    .catch((error) => alert(`Error while retreiving your users: ${error}`))
    .finally(() => {
      console.log(`Ended obtaining users:`);
      console.table(users);
    });
};

const obtainUserDetails = (id) => {
  getUserDetails(id)
    .then((response) => {
      console.log("All Paged users", response.data);
      setSelectedUsers(response.data);
    })
    .catch((error) => alert(`Error while retreiving your users: ${error}`))
    .finally(() => {
      console.log(`Ended obtaining users:`);
      console.table(selectedUser);
    });
};

return (
  <div>
    <h2>Users:</h2>
    {users.map((user, index) => (
      <p key={index} onClick={() => obtainUserDetails(user.id)}>
        {user.first_name} {user.last_name}
      </p>
    ))}
    <p>
      Showing {usersPerPages} users of {totalUsers} in total
    </p>
    <button onClick={() => obtainPagedUsers(1)}>1</button>
    <button onClick={() => obtainPagedUsers(2)}>2</button>
    <div>
      <h3>User Details</h3>
    </div>
    {selectedUser && (
```

```
        <div>
          <p>Name: {selectedUser.first_name}</p>
          <p>Last Name: {selectedUser.last_name}</p>
          <p>Email: {selectedUser.email}</p>
          <img
            alt="avatar"
            src={selectedUser.avatar}
            style={{ height: "50px", width: "50px" }}
          />
        </div>
      )}
    </div>
  );
};
export default Fetcheexample;
```

14-Realizando peticiones HTTP con Axios

`fetchService.js` ⇒

```
// LLamar a todos los users
export const getAllUsers = async () => {
  let response = await fetch(`https://reqres.in/api/users`);

  console.log("Response:", response);
  console.log("Status:", response.status);
  console.log("Ok?:", response.ok);
  // We return the json
  return response.json();
};

export const getAllPagedUsers = async (page) => {
  let response = await fetch(`https://reqres.in/api/users?page=${page}`);

  console.log("Response:", response);
  console.log("Status:", response.status);
  console.log("Ok?:", response.ok);
  // We return the json
  return response.json();
};

export const getUserDetails = async (id) => {
  let response = await fetch(`https://reqres.in/api/users/${id}`);

  console.log("Response:", response);
  console.log("Status:", response.status);
  console.log("Ok?:", response.ok);
  // We return the json
  return response.json();
};

export const login = async (email, password) => {
  let body = {
    email: email,
    password: password
  };
  let response = await fetch(`https://reqres.in/api/login`, {
```

```
    method: "POST",
    //mode: "no-cors",
    //credentials: "omit",
    //cache: "no-cache",
    //headers: {
    //  "Content-type": "application/json"
    //},
    body: JSON.stringify(body)
  });
  console.log("Response:", response);
  console.log("Status:", response.status);
  console.log("Ok?:", response.ok);
  return response.json();
};
```

FetchExample.jsx ⇒

```
import React, { useState, useEffect } from "react";
import {
  getAllPagedUsers,
  getAllUsers,
  getUserDetails,
  login
} from "../../services/fetchService";

const Fetcheexample = () => {
  const [users, setUsers] = useState([]);
  const [selectedUser, setSelectedUsers] = useState(null);
  const [totalUsers, setTotalUsers] = useState(12);
  const [usersPerPages, setUsersPerPages] = useState(6);
  const [pages, setPages] = useState(2);

  useEffect(() => {
    obtainUsers();
  }, []);

  const obtainUsers = () => {
    getAllUsers()
      .then((response) => {
        console.log("All users", response.data);
        setUsers(response.data);
        setTotalUsers(response.total);
        setUsersPerPages(response.per_pages);
        setPages(response.total_pages);
      })
      .catch((error) => alert(`Error while retreiving your users: ${error}`))
      .finally(() => {
        console.log(`Ended obtaining users:`);
        console.table(users);
      });
  };

  const obtainPagedUsers = (page) => {
    getAllPagedUsers(page)
      .then((response) => {
        console.log("All Paged users", response.data);
        setUsers(response.data);
        setTotalUsers(response.total);
        setUsersPerPages(response.per_pages);
```

```
      setPages(response.total_pages);
    })
    .catch((error) => alert(`Error while retreiving your users: ${error}`))
    .finally(() => {
      console.log(`Ended obtaining users:`);
      console.table(users);
    });
};

const obtainUserDetails = (id) => {
  getUserDetails(id)
    .then((response) => {
      console.log("All Paged users", response.data);
      setSelectedUsers(response.data);
    })
    .catch((error) => alert(`Error while retreiving your users: ${error}`))
    .finally(() => {
      console.log(`Ended obtaining users:`);
      console.table(selectedUser);
    });
};

const authUser = () => {
  login("eve.holt@reqres.in", "cityslicka")
    .then((response) => {
      console.log("TOKEN", response.token);
      sessionStorage.setItem("token", response.token);
    })
    .catch((error) => alert(`Error while login users: ${error}`))
    .finally(() => {
      console.log(`Ended login users. Navigate to Home...`);
    });
};

return (
  <div>
    {/* Button to simulate login */}
    <button onClick={authUser}>Auth User</button>
    <h2>Users:</h2>
    {users.map((user, index) => (
      <p key={index} onClick={() => obtainUserDetails(user.id)}>
        {user.first_name} {user.last_name}
      </p>
    ))}
    <p>
      Showing {usersPerPages} users of {totalUsers} in total
    </p>
    <button onClick={() => obtainPagedUsers(1)}>1</button>
    <button onClick={() => obtainPagedUsers(2)}>2</button>
    <div></div>
    {selectedUser != null ? (
      <div>
        <h3>User Details</h3>
        <p>Name: {selectedUser.first_name}</p>
        <p>Last Name: {selectedUser.last_name}</p>
        <p>Email: {selectedUser.email}</p>
        <img
          alt="avatar"
          src={selectedUser.avatar}
          style={{ height: "50px", width: "50px" }}
        />
```

```
        </div>
      ) : (
        <h6> Please click on a User to see its details</h6>
      )}
    </div>
  );
};
export default Fetcheexample;
```

Instalar ⇒ `npm i --save axios`

Creamos una carpeta de config de axios, dentro de `src`, creamos `utils`, dentro `config` y adentro el archivo `axios.config.js` que sirve para unificar todas las gestiones para tener un punto de config de axios

```
import axios from "axios";

//Default config for AXIOS
export default axios.create({
  baseURL: "https://randomuser.me/api",
  responseType: "json",
  timeout: 6000
});
```

src/service/axiosService.js ⇒

```
import APIRequest from "../utils//config/axios.config";

export function getRandomUser() {
  return APIRequest.get("/", {
    //Resuelve solo si el codigo es menos de 500
    validateStatus: function (status) {
      return status < 500; //Resolve only if the status code is less than 500
    }
  }); // https://randomuser.me/api
}
```

axiosExample.jsx ⇒

```
import React, { useState, useEffect } from "react";
import { getRandomUser } from "../../services/axiosService";

const AxiosExample = () => {
  const [user, setUser] = useState(null);

  useEffect(() => {
    obtainUser();
  }, []);

  const obtainUser = () => {
    getRandomUser()
      .then((response) => {
```

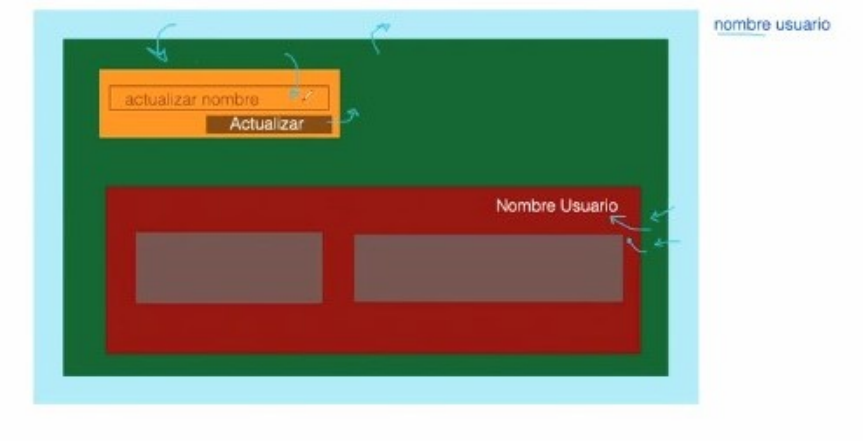
```
        if (response.status === 200) {
          setUser(response.data.results[0]);
        }
        console.log(response);
      })
      .catch((error) => {
        alert(`Something went wrong: ${error}`);
      });
  };

  return (
    <div>
      <h1>Axios Example</h1>
      {user != null ? (
        <div>
          <h2>
            {user.name.title} {user.name.first} {user.name.last}
          </h2>
          <h3>{user.email}</h3>
        </div>
      ) : (
        <div>
          <p>Generate a New User</p>
          <button onClick={obtainUser}>Random user</button>
        </div>
      )}
    </div>
  );
};
export default AxiosExample;
```

15-Problema de gestión del estado de la aplicación-Parte I

Nuevo Proyecto en Codesandbox, instalar ⇒ npm i redux react-redux --save, luego en las dependencias de desarrollo instalar npm i redux-devtools-extension --save. Solo instalar en modo de desarrollo, no en produccion ⇒

```
"devDependecies": {
  "redux-devtools-extension": "^2.13.9"
}
```

Estado de app, pasar estados mediante props ⇒



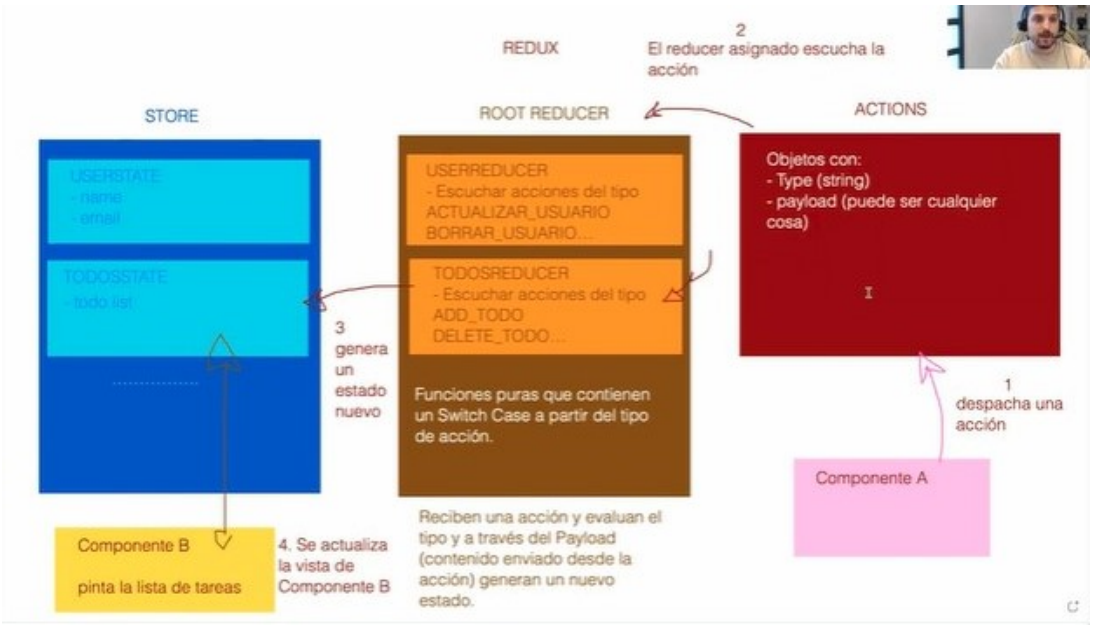
REDUX

Conceptos ⇒ Crear los archivos y carpetas, diseñar el estado de la aplicacion.

Stores ⇒ Es donde almacenamos los diferentes estados de la aplicacion, que son listas, objetos, informacion, etc. Se recomienda crear estados pequeños o medianos que estén relacionados.

Reducers ⇒ Conjunto de funciones que esta compuesta por el estado inicial y que a partir de acciones devuelven un estado nuevo, funciones con un switch case, escuchando las acciones que se pueden realizar sobre el estado de un usuario.

Actions ⇒ Objetos con tipos(strings) y payload(cualquier objeto).



16-Problema de gestión del estado de la aplicación-Parte II

openBootcamp-react-redux

openBootcamp-react-redux by gergati using react, react-dom, react-redux, react-router-dom, react-scripts, redux

☐ https://codesandbox.io/s/openbootcamp-react-redux-sjiidg?file=/src/components/pure/Filter.jsx

17-Estados globales asíncronos

Con Redux, existe redux-tank, redux-promise o redux-saga, son middlewares que podemos incorporar para la gestion del estado de la aplicacion para que se interpongan entre el propio reducer que se encarga de gestionar el estado de la aplicacion, a traves de mecanismos que permiten contorlar la peticion https, esperar la respuesta que el reducer capturará y actualizará. Sagas nos permite a partir de funciones generadoras, vamos a tener un watcher que es una funcion generadora, que va a estar escuchando en este caso peticiones https.

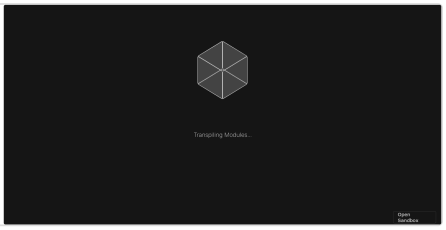
18-Gestionando el estado con Hooks

useReducer ⇒ Se utiliza para almacenar y actualizar estados de un componente, utiliza una funcion asociada que seria nuestro reducer, va a ser un switch case. Hook creado para simular el comportamiento de Redux. Un useReducer va a tener un switch case, que va a devolver un nuevo estado, y podemos trasladarla de modo global con el useContext.

useReducerAnduseContextExamples

useReducerAnduseContextExamples by gergati using react, react-dom, react-scripts

☐ https://codesandbox.io/s/usereducerandusecontextexamples-s7v1sm



19-Introducción a Webpack y configuración

Solucion para empaquetar nuestras builds. Crear una carpeta, abrir la terminal, con npm init creamos el package.json, si a npm init le agregamos el -u le decimos a todo que si.

```
package name: [proyecto-final]
version: (1.0.0)
description: Proyecto final curso React Js
entry point: (index.js)
text comand:
git repository:
keywords: react
author: gergati
license: (ISC)
is this OK? yes
```

instalar ⇒ npm i --save @babel/cli

instalar ⇒ npm i --save @babel/core

instalar ⇒ npm i --save @babel/preset-env @babel/preset-react

Si por ejemplo nos hemos equivocado, podemos borrar los node_modules y packege.lock.json y luego borrar dentro del package.json las dependencies, y luego con el tag --save-dev:

instalar ⇒ npm i --save-dev @babel/cli @babel/core @babel/preset-env @babel/preset-react

Instalar ⇒ npm i --save-dev babel-loader

Instalar ⇒ npm i --save-dev css-loader

Instalar ⇒ npm i --save-dev documentation

Instalar ⇒ npm i --save-dev eslint

Instalar ⇒ npm i --save-dev webpack webpack-cli webpack-dev-server

Instalar ⇒ npm i --save-dev node-sass sass-loader

Instalar ⇒ npm i --save-dev html-webpack-plugin

Instalar ⇒ npm i --save-dev mini-css-extract-plugin

Instalar ⇒ npm i --save-dev file-loader source-map-loader

Instalación a nivel global, herramientas de produccion:

Instalar ⇒ npm i --save react react-dom react-router-dom

Instalar ⇒ npm i --save redux react-redux redux-saga

Instalar ⇒ npm i babel/plugin-transform-modules-commonjs

Instalar ⇒ npm i babel/plugin-transform-runtime

Instalar ⇒ `npm i --save axios bootstrap`

Configuraciones:

.babelrc ⇒

```
{
  "presets": [
    "@babel/preset-env",
    "@babel/preset-react"
  ],
  "plugins": [
    [
      "@babel/plugin-transform-runtime",
      {
        "absoluteRuntime": false,
        "corejs": false,
        "regenerator": true,
        "useESModule": true,
        "version": "^7.16.0"
      }
    ]
  ]
}
```

.editorconfig ⇒

```
indent_style = tab
end_of_line = lf
insert_final_newline = true
tab_width = 2
charset = utf-8
trim_trailing_whitespace = true
```

Crear el `index.html`, archivo de inicializacion del proyecto, en el `body` poner ⇒

```
<div class="root">
</div>
```

webpack.config.js ⇒

```
const path = require('path');
//PLUGINS Y MINIFICADORES DE CSS Y SCSS/SASS
//Para reducir el tamaño de las hojas de estilo de proy
const HtmlWebpackPlugin = require("html-webpack-plugin")// Para el template del HTML que va a usar webpack
const MiniCssExtractPlugin = require("mini-css-extract-plugin")// Para reducer los CSS
const { SourceMapDevToolPlugin } = require('webpack') //Para conocer el Source Map de nuestro proyecto

//Configuraciones de puerto
const port = process.env.PORT || 3000;
```

```
// Exportar configuracion de webpack
module.exports = {
  entry: './src/index.jsx',
  output: {
    path: path.join(__dirname, '/dist/'),
    filename: 'bundle.[hash].js',
    publicPath: '/'
  },
  context: path.resolve(__dirname),
  devServer: {
    port,
    inline: true,
    historyApiFallback: true
  },
  devtool: 'eval-source-map',
  module: {
    rules: [
      //Reglas para archivos de JS y JSX
      // Tienen que pasar el LINTING para poder pasar
      {
        enforce: 'pre',
        test: /(\.js|\.jsx)$/,
        exclude: /node_modules/,
        use: [
          'eslint-loader',
          'source-map-loader'
        ],
      },
      //Reglas para archivos JS y JSX
      // Reglas de Babel ES y JSX
      {
        test: /(\.js|jsx)$/,
        exclude: /node_modules/,
        use: {
          loader: 'babel-loader'
        },
        query: {
          presets: [
            '@babel/env',
            '@babel/react',
          ],
        },
      },
        //Reglas para archivos CSS, SASS y SCSS para minificarlos y cargarlos en el bundle
        {
          test: /(\.css|\.scss)$/,
          loader: [
            MiniCssExtractPlugin.loader,
            'css-loader',
            'sass-loader'
          ]
        },
        //Reglas para los archivos de imágenes
      {
        test: /\.(png|jpe?g|gif)$/,
        use : [
          {
            loader: 'file-loader'
          }
        ]
```

```
      }
    ]
  },
  plugins: [
    //Template HTML
    new HtmlWebpackPlugin(
      {
        template: './index.html'
      }
    ),
    new MiniCssExtractPlugin(
      {
        filename: './css/styles.css'
      }
    ),
    new SourceMapDevToolPlugin(
      {
        filename: '[file].map'
      },
    )
  ],
  resolve: {
    extensions: ['.js','.jsx','.css','.scss','.sass'],
    modules: [
      'node_modules'
    ]
  }
}
```

Instalar ⇒ `npm i -g eslint`

Luego `eslint --init`, poner: `To check syntax, find problems, and enforce code style`. Luego `Javascript modules (import/export)`,poner React, luego si usamos Typescript, `Browser` o `node` podemos marcarlas poniendo espacio. Poner `Use a popular style guide`, utilizamos `Standard` o `Airbnb`. Nos pregunta como queremos generar el archivo de configuracion, ponemos `JSON`. Ponemos install? `Yes`. Nos crea el `.eslintrc.json`.

Dentro ponemos

```
"settings":{
  "react":{
     "version": "detect"
    }
},
"overrides": [
  {
    "files": [
      "*.jsx",
      "*.js"
    ]
  }
],
"globals":{
  "document": true
}
```

Para sobreescribirlo utilizaremos el comando eslint fix.

Dentro del package.json tenemos que generar unos scripts ⇒

```
"scripts": {
  "test": "echo/"Error: no test specified\" && exit 1",
  "lint": "eslint ./src",
  "lint:fix": "eslint --fix ./src",
  "dev": "webpack-dev-server --mode development --hot --open --content-base",
  "docs:build": "documentation build src/** -f html -o docs",
  "build": "",
  "build:prod": "webpack"
}
```

Dentro de src le agregamos el index.js, dentro components, env, routes, assets, styles, pages.

20-Build del proyecto

```
"scripts": {
  "test": "echo/"Comando no specified\" && exit 1",
  "lint": "eslint ./src",
  "lint:fix": "eslint --fix ./src",
  "dev": "webpack-dev-server --mode development --hot --open --content-base",
  "docs:build": "documentation build src/** -f html -o docs",
  "build": "webpack --mode development",
  "buil:prod": "webpack --mode production"
}
```