**React**

**Package.json**

Es como el pom.xml (de aquí se abre la aplicación), contiene la configuración del proyecto, versiones, etc.

Dentro de este archivo podemos encontrar:

“**scripts**”, assign a key word to a react function

“**start**”: “react-scripts start”

if you type npm start, is the same to type react-script start.

“**build**”: “react-scripts build” (

if you type npm build, is the same as type npm react-scripts builds.

“**eslintConfig**”

“**react-app**”, active the react control of the code, and if you have some components or elements that you don’t use, react suggest to delete them.

“**browserslist**”,

specify for witch version of each browser, Babel are going to convert the react code to a javascript

**.gitignore**

Set what files don’t be update to github. The reason is that it is useful only for the notebook localhost.

**Public Files**

contiene cierta informacion sobre nuestra aplicacion

**Favicon.ico**, es el icono de nuestra aplicacion

**React192.png**, es la imagen del logo chica

**React512.png**, es la imagen del logo grande

**Index.html**

This file contain the main structure of our application.

<head></head>

**<body>** aqui hay un

**<noscript></noscript>**

If the user browser doesn’t have javascript enabled, show this message. The application doesn’t work.

**<div id=”root”></div>**

Here react build the application. Use the root id, to get the div element (with the javascript function getElementByID), and inside the div, put all Html and javascript. To do that, React need to convert the jsx code (javascript and xhtml).

**</body>**

**Directorio SRC,**

Contiene los archivos de la aplicacion que desarrollamos, entre ellos Podemos destacar:

Index.js

Es el archive mas importante, desde aca se arma el Proyecto, que sera insertado dentro del **<div id=”root”></div>,** del archive index.html que esta en la carpeta public. Para que suceda eso react debe importer React, ReactDom, index.css (el estilo de la aplicacion), y por ultimo crear import App from ‘./App’; crea esta aplicacionen la variable App y la va a renderizar luego.

Para hacer esto crea una const root (el mismo id que tiene el nombre del div en el archive index.html) y con ReactDom.CreatRoot(document.getElementByID(‘root’)). This last line assign the new element to root.

Luego al contenido de la constante la renderiza con el siguiente codigo

Root.render(<App />); y listo esta desplegada la aplicacion.

App.js

This is the main file of the application.

This files has a function to create the Application and called App(). This is the main component of my application.

First top of the files, we need to import the components:

Import the files of the logo

Import the flies with the Css.

**import React from 'react';** If you do not import, Babel do it for you.

Inside this function, we have somes div, that made the html. This code isn´t html, is a combination of javascript and xhtml, this code is name jsx.

This function is a React component, and return the application.

**Comments:**

All the tags are writing in lower case, and this is because they are a reacts components.

className, are similar to classname, the first is a React, the second is a html.

Components

The file name is set as TodoItem.js

**The first section,** we add the import commands.

**The second section,** we create a function with the same name of the file.

***function TodoItem(props) {***

The function is a component, start with upper case.

The elements that are inside the components, are render to html.

React, create the:

html with the components and their elements and

the JavaScript functions.

Props (Objects) are used for components to communicate with each other. They receive parameters but are different because they have another form to send the information.

The props are objects, that have different properties.

The props children, react transform all the components inside another component in their children. TodoItem is automatically a children of TodoList.

***<TodoList>***

***<TodoItem />***

***</TodoList>***

Warning = If you type the props inside the (), first you need to put {}. That say to react that this word are props objects.

The props don’t become in attribute of html, because they are components.

Warning: To create and html attribute you need to create an attribute in an element of react, then react transform it in an attribute of an html element.

All function has a return section.

***return (***

Inside the return section, the code is JavaScript and xhtml, but not html, is similar.

We need to set at first one html element inside the return. <div>, <p>, etc. The reason is because React need a html to render all the code.

Tips: If we don’t need a <div> element, we can set the first html element with

***<React.Fragment>***

***</React.Fragment>***

React do not render this element, so we don’t have <div> that they are not necessarily.

***<li>***

***<span>V</span>***

***<p>Llorar con la Llorona</p>***

***<span>X</span>***

***</li>***

***)***

***}***

**The last section,** we need to export the component, and after that, the component can be used for all react application.

We use the named export, this is more secure and avoid error.

***export { TodoItem };***

**Functions:**

**Map**, create an array from another array.

First array:

// This is an array, you create two variables, text and completed

***const defaultTodos = [***

***{ text: 'Cortar cebolla', completed: true },***

***{ text: 'Tomar el Curso de Intro a React.js', completed: false },***

***{ text: 'Llorar con la Llorona', completed: false },***

***{ text: 'LALALALALA', completed: false },***

***];***

// The map function creates a new array from the defaultTodos array.

// after that iterate inside and for each element create a TodoItem

// While create the TodoItem, send the value of the array in the key word.

***{*** // Set the element as javascript code

***defaultTodos.map(*** // create an array from defaultTodos array

***todo => (*** // create for each code

***<TodoItem*** // create TodoItem component

***key={todo.text}*** // this is the uniqueidentifier

***text={todo.text}*** // this the text variable

***completed={todo.completed}***

***/>***

***)*** // End the for each code

***)*** // End the created array

***}*** // End the javascript code

**toLowerCase**, this method converts the character in lower case using ASCII code.

The toLocaleLowerCase method converts the character in lower case using the rules of the specific location languages.

**noTildes**, this function take out the special letter with tildes

Function without tildes

**const noTildes = (text) => {**

**return text.normalize('NFD').replace(/[\u0300-\u036f]/g, '');**

**};**

Convert toLowerCase and take out the tildes

**const TodoTextLC = noTildes(todo.text.toLowerCase**());

// This method after convert the useState toLowerCase, call the noTildes function to replace this type of letter.

**const searchTextLC = noTildes(searchValue.toLowerCase());**

// This method after convert the searchValue toLowerCase, call the noTildes function to replace this type of letter.

**return TodoTextLC.includes(searchTextLC);**

// Apply the filter and return the selected value

// ------------------------------------------------------------------------------------------------------------------------------

const normalizeString = (string) => {

string = string || "";

string = string.toLowerCase();

// This line convert the string received toLowerCase.

string = string.normalize("NFD").replace(/[\u0300-\u036f]/g, "");

// This line removes the accent from the string received.

string = string.trim();

// This line removes the extra spaces.

return string;

// This line return the result of the method applied to the string received.

};

// This function apply the filter, but before call the normalizeString

const filteredTodos = todos.filter(

(todo) => { // This line is for each value

let { text: normalizedTodo } = todo;

// This line create a new array from todo useState.

normalizedTodo = normalizeString(normalizedTodo);

// This line apply the normalizeString method to the todo useState.

let normalizedSearch = normalizeString(searchValue);

// This line create a new array from the searchValue, but applying the normalizeString.

return normalizedTodo.includes(normalizedSearch);

// This line apply the search to the normalizedTodo array.

});

Events in React

React takes all the word that’s begin with **on** is an event. React takes this word and transform it to addEventListening. This eventListener do not show the event in the html element, but the button has the event onclick but no with the attribute onclick.

Example: onClick is the same, onChange

// This event is assigned to an input box, so each letter that the user type, this event is triggered.

onChange={ // React take onChange and with addEventListening add the onchange event.

// All event in react need a function. When the event occur, the function is triggered.

(event) =>

{

// console.log, send the event to the console.

console.log('Escribiste en el TodoSearch'); // Show the message

console.log(event); // Show the event in the log with all event information.

console.log(event.target); // Show the target event, is the html element.

console.log(event.target.value); // Show the value of the target, It is the value written by the user.

} // End function

} // End event

States in React

The states are a React.js tool, so to work with, first we need to import to the file.

**import React from 'react';**

React have many states to save and track the page history. You can look for one element and save the state in an array.

We use React states and not variable, because with states after each update React know that need to update the render, so the html is updated. If you use javaScript variables, you can change the value of the variables, but React never know when need to update the pages.

Every time we change the state of a component, only this component is re-rendered.

To do that you must:

// This code save the state of TodoSearch component.

// In the state you can get the value and save new data, so that, you need an array to store the value.

// The first parameter get the history, and the second save the new data

**const [searchValue, setSearchValue] = React.useState('');** // The value of the first input is empty

Tips: use let, instead of const, if you are going to work with numbers.

**console.log('Los usuarios buscan todos de ' + searchValue);**

The state in React can not be modificated, so you need to make a function to update the new state. We do that add the key word set to the first parameter.

**const [searchValue, setSearchValue]**

**const [todoItem, settodoItem]**

**const [searchValue, setSearchValue] = React.useState('');**

// The value of the first input is empty

// This code save the state of TodoSearch component.

// In the state you can get the value and save new data, so that, you need an array to store the value.

// The first parameter get the history, and the second save the new data

**console.log('searchValue is ' + searchValue);**

// This show the searchValue

**console.log('setSearchValue is ' + setSearchValue);**

// This show setSearchValue is function () { [native code] }, because is a function to assign the new value to the state.

**<input**

**placeholder="Agregar TODOs" // This is the suggestion message**

**className="TodoSearch" // This is the Css style**

**value={searchValue}**

// Define the value of the state as the value of the input box

// This event is assigned to an input box, so each letter that the user type, this event is triggered.

// so when the user type L, the event is triggered the one time

// after LA is triggered two times, one for L and one for A.

**onChange={**

// React take onChange and with addEventListening add the onchange event.

// All event in react need a function. When the event occur, the function is triggered.

**(event) =>**

**{**

**setSearchValue(event.target.value);**

// This line update the State of the searchValue. So when you type one letter, the event update the state setSearchValue and after that, the user can se what is written because you set value={searchValue} as the value of the inputbox.

**} // End function**

**} // End event**

**/>**

If we only let console.log('searchValue is ' + searchValue); enable, and type Cortar Cebolla, the result is:

**searchValue is C**

**TodoSearch.js:16 searchValue is Co**

**TodoSearch.js:16 searchValue is Cor**

**TodoSearch.js:16 searchValue is Cort**

**TodoSearch.js:16 searchValue is Corta**

**TodoSearch.js:16 searchValue is Cortar**

**TodoSearch.js:16 searchValue is Cortar**

**TodoSearch.js:16 searchValue is Cortar c**

**TodoSearch.js:16 searchValue is Cortar ce**

**TodoSearch.js:16 searchValue is Cortar ceb**

**TodoSearch.js:16 searchValue is Cortar cebo**

**TodoSearch.js:16 searchValue is Cortar cebol**

**TodoSearch.js:16 searchValue is Cortar ceboll**

**TodoSearch.js:16 searchValue is Cortar Cebolla**

**Important**

**Comparison Operators == and ===**

In JavaScript, == and === are comparison operators used to check equality between two values or variables.

The main difference between the two operators is how they compare values. The == operator compares the values of two variables after performing type conversion if necessary. On the other hand, the === operator compares the values of two variables without performing type conversion.

Here's an example to help illustrate the difference:

javascript

Copy code

const num = 10;

const str = "10";

console.log(num == str); // true - The values are the same after type conversion

console.log(num === str); // false - The values are different types and not equal

In the first console.log statement, the == operator compares the two variables after converting the string "10" to a number, resulting in true. In the second console.log statement, the === operator compares the two variables without converting them, resulting in false.

Here's another example:

javascript

Copy code

const bool = true;

const num = 1;

console.log(bool == num); // true - The boolean value true gets converted to 1

console.log(bool === num); // false - The values are different types and not equal

In the first console.log statement, the == operator converts the boolean value true to the number 1, resulting in true. In the second console.log statement, the === operator compares the two values without converting them, resulting in false.

**Icons:**

**Relation between js files**,

1. App.js

We create the html <TodoList> and inside it we call the component <TodoItem and pass throw the props the values.

1. TodoItems.js

In this file is created the html <li> and inside it we call the CompleteIcon and the DeleteIcon.

1. CompeteIcon.js or DeleteIcon.js

Inside this component we do not create any html element. We only call the <TodoIcon> component and pass throw the props the values.

1. TodoIcons.js

Inside this component we create the real icon. To do that:

First, import React, Css, and the Icons.svg

Second, create an object to manipulate the svg style.

Third, in the function assign the values that receive from the component that call the TodoIcon.

1. TodoIcon.Css

This file has the format used in the TodoIcon component.

**Deploy**

To deploy the project, you need to add this line to the bottom of the package.json

**,**

**"homepage": "/ApplicationsTutorial/Platzi/ReactIntro/build/"**

Then we need to copy the path of the build/index.html. And paste in the browser. The page open and found all the files.

<file:///C:/ApplicationsTutorial/Platzi/ReactIntro/build/index.html>

To deploy the project in github pages, you need to add this line to the bottom of the package.json

**,**

**"homepage":"https://JcaPetri.github.io/ReactIntro",**

Then we need to install with this command:

**npm i --save-dev gh-pages**

copy the path of the build/index.html. And paste in the browser. The page open and found all the files.

<file:///C:/ApplicationsTutorial/Platzi/ReactIntro/build/index.html>

Cuando cambia una version de React o da errores inexplicables, hay que borrar las siguientes carpetas y se reinstala todo

rm -rf node\_modules

// Borra el modulo node

rm package-lock.json

// Borra el archive de package

npm i

// Instala nuevamente react

Para crear un nuevo Proyecto desde cero hay que hacer

**npx create-react-app cratest**

// instala las dependencias, las ejecuta, crea el Proyecto y las borra.