1-Fundamentos de las herramientas de desarrollo

Veremos herramientas utiles para el desarrollo con **REACT** como **NPM**, **BABEL**, **WEBPACK** aunque tambien se utilizaran

Entorno de desarrollo de REACT JS

Se puede implementar **REACT** utilizando la libreria sin compilar pero lo mas recomendado es utilizar **JSX** el cual es un lenjuage compilado. Para la implementacion de **REACT** se utilizara el modulo *create-react-app*.

Explicacion de NPM, Webpack y babel

NPM: Es un gestor de modulos de JS.

BABEL: Es un traductor de codigo JS con estandares nuevos, lo que permite que el projecto sea utilizable en cualquier navegador, aunque este no soporte los ultimos estandares del **ecmascript**.

WEBPACK: Permite ulizar modulos para reutilizar codigos, es muy util utilizarlo junto con **BABEL**.

Chrome DevTools

Para abrir las **DevTools** se utiliza *ctrl+shift+i*.

Las **DevTools** posee una *consola* la cual permite una interaccion entre el programa y el navegador.

Creando Stack con REACT

Para crear una app de **REACT** utilizamos el comando **CREATE-REACT-APP *nombre app***, este comando puede ser instalado con **NPM**.

Comandos utiles

- **NPM START**: Ejecuta la aplicacion en un servidor local.
- **NPM RUN**:

Estructura del proyecto

Dentro de la carpeta **public** se encuentra el **INDEX.html** en el cual se puede encontrar un *div* de clase *root* que en donde se generara nuestra aplicacion. El archivo **manifest.json** se encuentra las cosas necesarias para realizar una **PWA**.

Dentro de la carpeta **src** se encuentran todos los archivos de la aplicacion , en el **Index.js** se encuentra importa el modulo *serviceWorker* junto con el archivo **serviceWorker.js** se utiliza para generar una **PWA**.

2-Introduccion REACT CORE

Esta seccion permite comprender el entorno de **REACT**, y todo su entorno como **JSX** y las **expresiones** las cuales permiten generar codigo *html* de forma de reutilizar codigo. Junto con la aplicacion de codigo de **CSS**

para mejor las paginas web.

Los temas vistos en esta seccion son:

1. Introduccion a JSX
2. Componentes
3. JSX en profundidad
4. Ecmascript 6

Introduccion a JSX

JSX es una combinacion entre **JS** y **HTML** la cual permite indexar directamente **HTML** en codigo **JS**.

Creacion de elementos e insercion en el DOM

Utilizar **REACT** debo importar el modulo ***React*** y ***ReactDOM***.

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import * as serviceWorker from './serviceWorker';

const app = <h1>Hola React</h1>;

ReactDOM.render(app, document.getElementById('root') );
// If you want your app to work offline and load faster, you can change
// unregister() to register() below. Note this comes with some pitfalls.
// Learn more about service workers: https://bit.ly/CRA-PWA
serviceWorker.unregister();
```

Expresiones

Para insertar una expresion en **JSX**, utilizando llaves, dentro de ellos puedo colocar cualquier expresion valida de **JS**.

```
function crearFrase( name ){
    return 'Bienvenido ' + name
}

let name = 'Lautaro';

const app = <h1> { crearFrase( name ) } </h1>
```

Componentes

Los componentes permiten la reutilizacion de codigo . Existen 2 grandes grupos de componentes los **FUNCIONALES** y **BASADOS EN CLASES**. Los mas recomendables para utilizar son los componentes **FUNCIONALES** ya que necesitan menor cantidad de lineas de codigo gracias a los **HOOKS**.

Funcionales

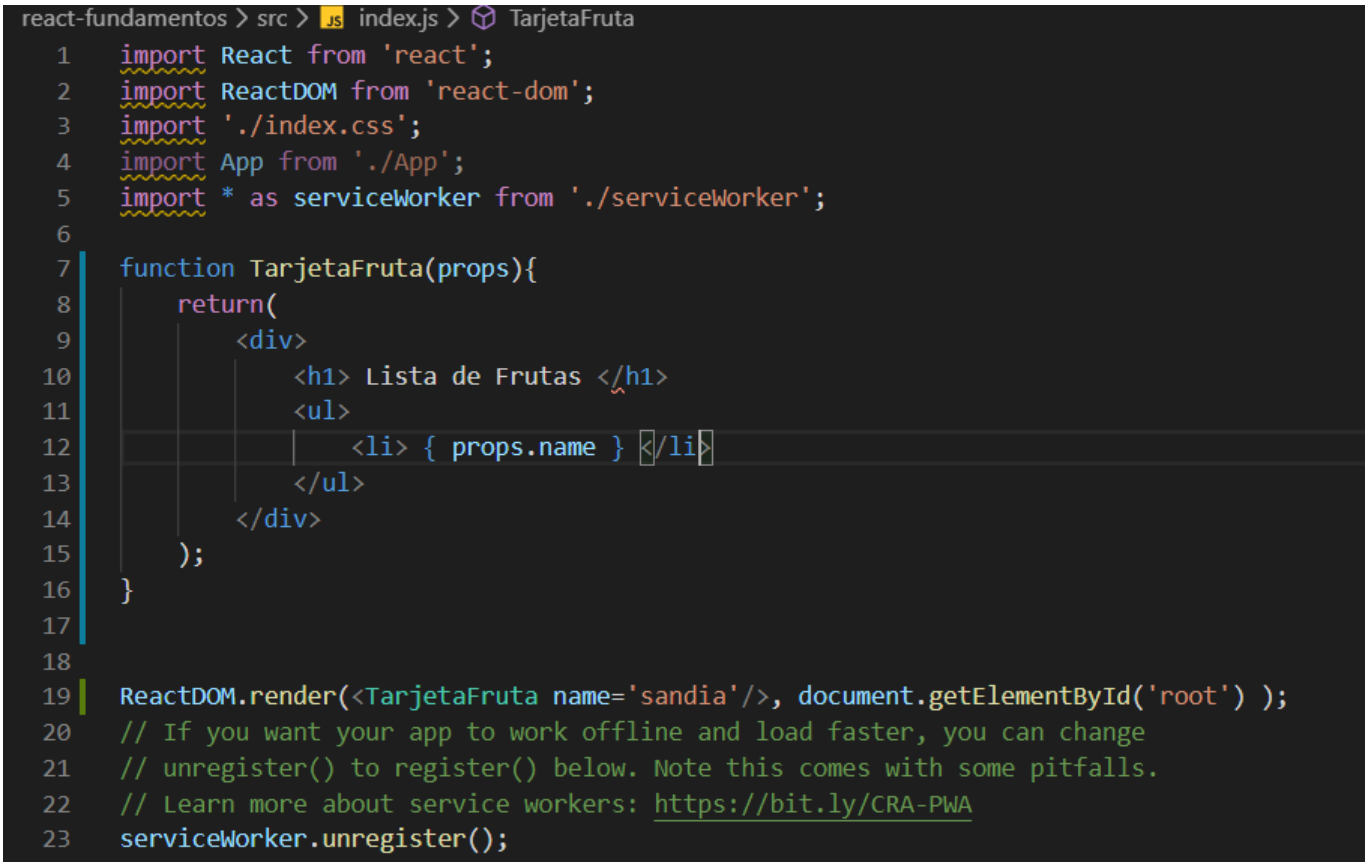
Son funciones que retornan una unica etiqueta de **HTML**. Para llamar un componente un funcional se debe utilizar la nomenclatura clasica de **HTML**.

react-fundamentos > src > index.js > ...

```
1   import React from 'react';
2   import ReactDOM from 'react-dom';
3   import './index.css';
4   import App from './App';
5   import * as serviceWorker from './serviceWorker';
6
7   function TarjetaFruta(){
8       return(
9           <div>
10              <h1> Lista de Frutas </h1>
11          </div>
12      );
13  }
14
15
16  ReactDOM.render(<TarjetaFruta/>, document.getElementById('root') );
17  // If you want your app to work offline and load faster, you can change
18  // unregister() to register() below. Note this comes with some pitfalls.
19  // Learn more about service workers: https://bit.ly/CRA-PWA
20  serviceWorker.unregister();
```

Props

Las **props** son variables de entrada de los componentes, los componentes que se le pasan a un **props** se deben utilizar llaves aunque para los datos de tipo **String** son opcionales.



Clases

Son clases que extienden de **React.Component** y debe implementar un metodo **render** el cual renderiza el componente.

Los componentes basados en clases soportan una actualizacion dinamica, utilizando una variable de clase **state** y para actulizar el componente se utiliza el metodo *setState*.

ACLARACION: Para los metodos que utilicen **this** es recomendable utilizar el metodo **bind(this)** para todos los metodos que lo requieran. Es recomendable utilizar inicializadores de propiedades.

Inicializadores de Propiedades

Se crea un atributo fuera del constructor y permite presindir del mismo ya que la unica llamada necesaria es la del constructor del padre.

```
class TarjetaFruta extends React.Component{
    state = {
        cantidad: 0,
    }


    agregar = () => {
        this.setState( {
            cantidad: this.state.cantidad + 1
        } );
    }

    quitar = () => {
        const cantidadNueva = (this.state.cantidad>0) ? this.state.cantidad-1 : 0;
        this.setState({
            cantidad: cantidadNueva
        });
    }

    render(){
        return(
            <div>
                <h3>Fruta: { this.props.name }</h3>
                <h4>Cantidad: {this.state.cantidad }</h4>
                <p>Value: ${ this.props.price }</p>
                <button onClick={ this.agregar }>+1</button>
                <button onClick={ this.quitar }>-1</button>
            </div>
        );
    }


}
```

Factorizacion de componentes

Es muy recomendable tener los componentes separados en diferentes carpetas, para lograr tener el codigo mucho mas ordenado.

JSX en profundidad

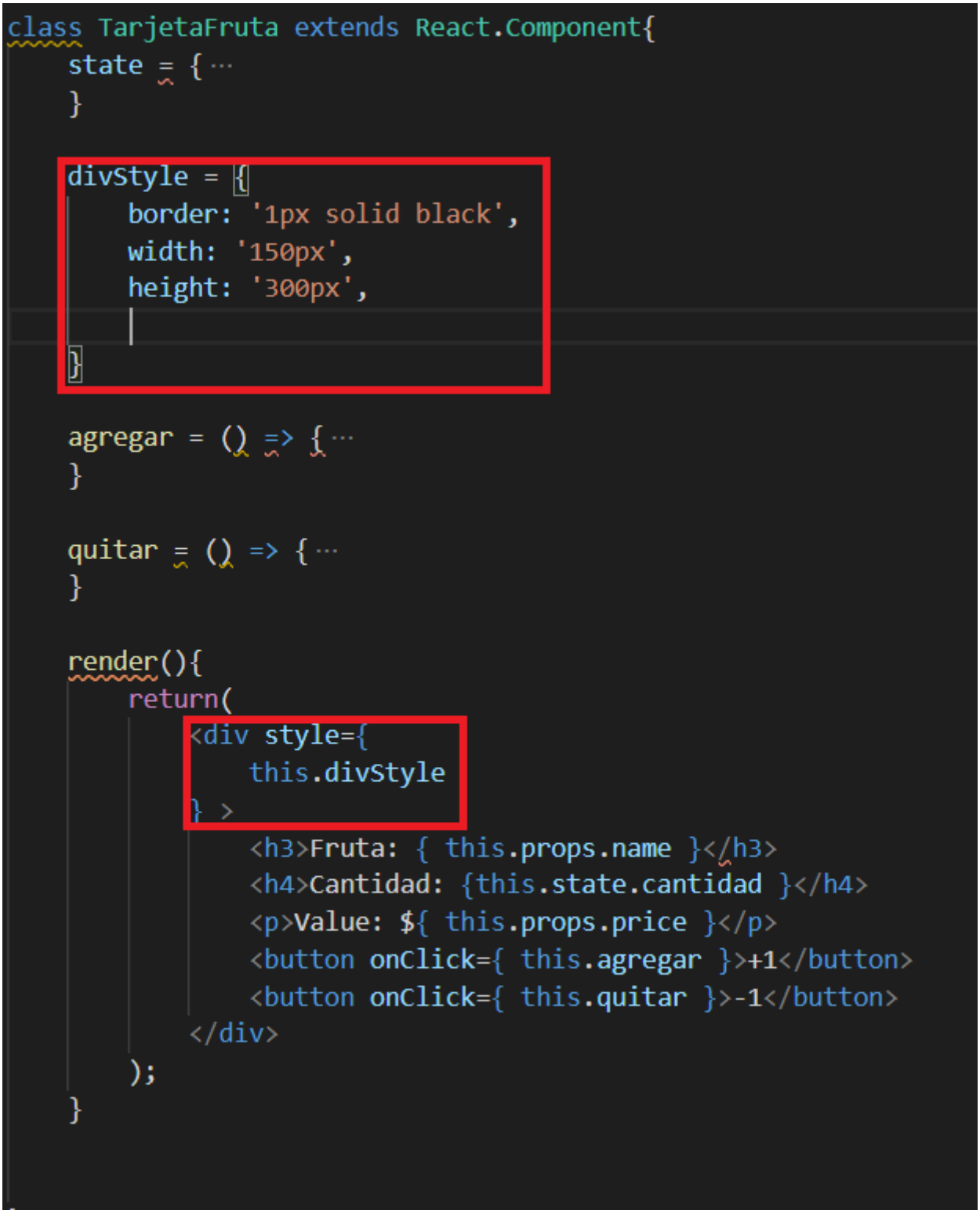
Los componentes deben comenzar con mayuscula, ya que sino **Babel** no tomara como valido.

Para agregar una clase a una etiqueta se utiliza **className**.

JSX permite declarar modulos de componentes, es decir, es posible crear objetos **JSON** donde cada atributo sea un componente.

CSS en React

Para indexar codigo **CSS** debo colocar el atributo **style** y pasar como argumento un objeto que contenga las propiedades.



```
class TarjetaFruta extends React.Component{
    state = {···
    }

    divStyle = {
        border: '1px solid black',
        width: '150px',
        height: '300px',
    }

    agregar = () => {···
    }

    quitar = () => {···
    }

    render(){
        return(
            <div style={
                this.divStyle
            } >
                <h3>Fruta: { this.props.name }</h3>
                <h4>Cantidad: {this.state.cantidad }</h4>
                <p>Value: ${ this.props.price }</p>
                <button onClick={ this.agregar }>+1</button>
                <button onClick={ this.quitar }>-1</button>
            </div>
        );
    }
```

Si el estilo posee un gion medio (-) se utiliza la notacion **camelCase**. **JSX** permite la implementacion de operadores ternarios dentro de los objetos, lo cual hace mucho mas sencilla la implementacion de estilos segun los estados.

```jsx
render(){
    const hasItems = this.state.cantidad > 0;

    const divStyle = {
        border: '1px solid black',
        width: '150px',
        height: '300px',
        borderRadius: '0.5em',
        padding: '1em',
        marginBottom: '1em',
        marginLeft: '2em',
        background: hasItems ? 'linear-gradient(45deg, #adebad, #33cc33)' : 'linear-gradient(45deg, black, red)',
        color: hasItems ? 'white' : 'black',
        transition: 'all 450ms ease-out',
    }

    return(
        <div style={
            divStyle
        } >
            <h3>Fruta: { this.props.name }</h3>
            <h4>Cantidad: {this.state.cantidad }</h4>
            <p>Value: ${ this.props.price }</p>
            <p>
                Total: ${ this.props.price * this.state.cantidad }
            </p>
            <button onClick={ this.agregar }>+1</button>
            <button onClick={ this.quitar }>-1</button>
        </div>
    );
}
```

CSS utilizando clases

Es recomendable utilizar dentro de cada carpeta del componente sus estilos, lo que permite tener componentes aun mas modularizados. **JSX** permite importar archivos **CSS** directamente, esto sucede gracias al paquete de **WebPack**.

```
import React from 'react';
import ReactDOM from 'react-dom';

import './style.css';

class TarjetaFruta extends React.Component{
    state = {···
    }

    agregar = () => {···
    }

    quitar = () => {···
    }

    render(){
        const hasItems = this.state.cantidad > 0;
        const clases = `box ${ hasItems ? 'box-activa' : '' }`;
        return(
            <div className = { clases }>
                <h3>Fruta: { this.props.name }</h3>
                <h4>Cantidad: {this.state.cantidad }</h4>
                <p>Value: ${ this.props.price }</p>
                <p>
                    Total: ${ this.props.price * this.state.cantidad }
                </p>
                <button onClick={ this.agregar }>+1</button>
                <button onClick={ this.quitar }>-1</button>
            </div>
        );
    }

}
```

Sistemas de Modulos

Cuando se desea utilizar un sistema de modulos de **CSS** el arvhivo debe ser nombrado de la siguiente forma:

***nombre*.modules.css**

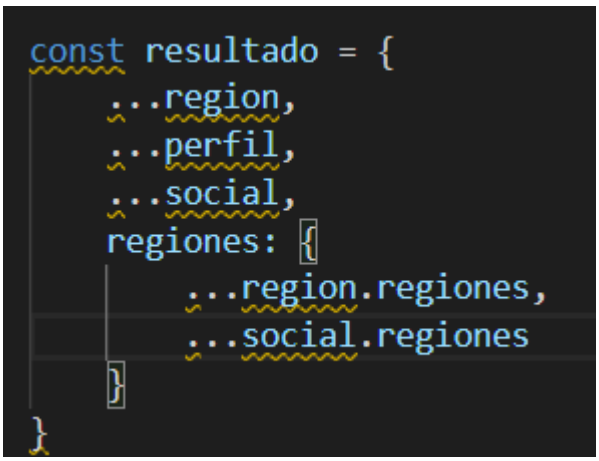
Ecmascript 6

Object.assing() 👍🏽

Este metodo sirve para combinar objetos, pero si existe una coincidencia entre los atributos de los objetos estas seran sobreescritas por la ultima encontrada. La solucion de este problema puede es utilizar el metodo **Assing** sobre los atributos que se pisan creando un objetonuevo dentro de ese atributo.

Operador Spread [...]

Cuando poseo objetos dentro de objetos se utiliza el operador **Spread**. El cual remplaza al metodo **Assing** con una sintaxis mucho mas amigable.



EL operador **spread** sirve para concatenar **arrays** de una forma muy util

Mutacion de componenetes utilizando funciones

Debido a que el metodo **setState** es asincrono es muy util utilizar una funcion de callback dentro del metodo.