Primeros pasos en React

https://reactjs.org/blog/2022/03/08/react-18-upgrade-guide.html

¿Qué aprenderemos en esta sección?

- Vamos a ver como ponerle tipado a nuestros diferentes componentes.
- Nuestra primera aplicación - Hola Mundo
- Exposiciones sobre los componentes
- Creación de componentes (Functional Components)
- Propiedades - Props
- Impresiones en el HTML
- PropTypes
- DefaultProps
- Introducción general a los Hooks
- useState

Es una sección importante, especialmente para todos los que están empezando de cero en React, ya que dará las bases de cómo segmentar la lógica de nuestra aplicación en pequeñas piezas más fáciles de mantener.

1. ¿Qué son los componentes?

La idea de los componentes es fragmentar nuestra aplicación macro en pequeñas piezas que nos ayude a nostoros a que nuestra aplicación sea mas mantenible y que cada una de las piezas realice una tarea en especifico.

Es una buena practica que los componentes usen Upper-camel-case.

Un componente es una pequeña pieza de código encapsulada re-utilizable que puede tener estado o no y realice un trabajo en especifico.

El estado es como se encuentra la informacion del componente en un punto determinado del tiempo.

Un router nos ayuda a hacer navegación entre páginas sin hacer refresh del navegador web.

2. Primera aplicación React

Para iniciar con nuestra app de React debemos tener instalado nodeJS y npm (Node Package Manager).

Seguir las instrucciones de la siguiente pagina: https://create-react-app.dev/

npm start levanta un web-pack-dev-server para que podamos tener un servidor local.

3. Estructuras de directorios

Documentaciones: https://developers.google.com/web/ilt/pwa - https://create-react-app.dev/docs/available-scripts/ - https://developers.google.com/search/docs/advanced/robots/intro?hl=es&visit_id=637858351351919784-3314713950&rd=1

Los archivos pueden cambiar dependiendo de lo que crea conveniente la gente que mantiene comando create-react-app.

node-modules: Contiene las librerias y paquetes que hacen que mi aplicación de React funcione en desarrollo. No quiere decir que cuando generemos una aplicación vayamos a subir/utilizar todos estos modulos pre-instalados. No se recomienda manipular las carpetas manualmente, si desde la terminal de comandos.

Pantallazo general de los archivos.

public: Parece una pagina web común y corriente.

Tiene un favicon para el index.html.

index.html que es la página que se despliega con el npm start y nuestra página principal.

Los logos png son recursos estaticos que estamos utilizando. Seria como los iconos si nosotros instalamos nuestra aplicación web como una PWA (Progresive Web Application).

La imagen que se despliega en el servidor en realidad es un vector.

manifest.json Es una configuración que se hace de las PWA.

Esto de la PWA no esta encargado de realizarlo React.

Robots.txt Son unas configuraciones que vienen por defecto. No son importantes y no tienen que ver con React tampoco.

Lo unico que es propio de react es el index.html.

El index.html tiene cosas como %PUBLIC_URL% que son variables incrustadas. La razon de utilizar una PUBLIC_URL es que cuando hacemos un build de producción el PATH puede cambiar.

El PUBLIC_URL es una variable de ambiente que hace referencia al PATH relativo desde la carpeta contenedora de nuestra APP hasta donde se encuentra ese PUBLIC_URL. Asi nos ahorramos tener que escribir distintos paths cuando levantemos a producción.

El meta theme-color sirve para que cuando utilicemos el task manager de android tenga el color que querramos.

El .gitignore es un archivo que se utiliza para indicarle que carpetas y archivos ignorar para no darle seguimiento.

La carpeta coverage es algo que se genera cuando hacemos cierto tipos de pruebas.

La carpeta build es la carpeta de producción, cuando hablamos de producción hablamos de producto final.

El package.json son archivos de configuracion, dependencias, scripts, start app, test app etc.

El readme no es mas que un archivo que describe la aplicación.

4. El contenido de la carpeta src

App.css: es un archivo de estilos para importar en app.js.

App.js: Es un componente.

App.test.js: la extension .test le dice al npm run test que es un archivo a ejectuar para realizar pruebas. Y que estas pruebas estan relacionadas al nombre del archivo.

index.css: Es el archivo de estilos a importar en index.js

index.js: Es el punto inicial de nuestra aplicación. Cuando se lanza el webpack lee el archivo por defecto. React importa el JSX, el DOM para manipular el HTML, importacion de estilos, componentes y el serviceWorker (es propio de la PWA).

logo.svg: Es la imagen que se muestra renderizada en el index.js.

serviceWorker.js: Informacion del serviceWorker para hacer nuestra aplicación web progresiva.

setupTests.js: Es algo que se ejecuta la primera vez que ejecutamos pruebas. Va a configurar todo lo que vamos a necesitar para nuestras pruebas.

5. Hola Mundo en React

En el momento que utilicemos cualquier etiqueta dentro de JS se transforma en JSX vamos a tener que importar React para que podamos utilizalo.

React se va a encargar de crear la etiqueta por nosotros.

Para renderizar el JSX en HTML vamos a tener que importar el {createRoot} de react-dom/client, este va a ser el encargado para realizar renderizaciones en HTML.

En el HTML vamos a renderizar practicamente todo dentro del div root. El nombre del div lo podemos cambiar.

¿Por qué renderizamos dentro de un div?
https://stackoverflow.com/questions/56705154/why-do-we-use-divroot-instead-of-body

ReactDOM contiene metodos para manipualar de manera mas sencilla el DOM.

Por ejemplo con root.render(nuevo componente); Vamos a poder renderizar nuestro JSX en el HTML.

Ejemplo:

const saludo = <h1>Hola mundo</h1>; -> JSX

const divRoot = document.querySelector("#root"); -> Selecionamos nuestro div root

const root = createRoot(divRoot); -> Nuestra constante root va a ser donde se va a renderizar el JSX. Al createRoot vamos a pasarle el divRoot para generar el vinculo

root.render(saludo); -> Renderizamos el JSX

Utilizamos ReactDOM porque nos ofrece otras caracteristicas, entre las cuales nos va permitir empezar a crear nuestro arbol de componentes y eso a su vez nos permite comunicarnos entre componentes de una manera sencilla.

6. Nuestro primer componente

En react tenemos dos tipos de componentes, los basados en clases y los basados en funciones. React hoy en dia esta empujando a trabajar orientado a funciones, por ello vamos a trabajar mucho con funciones.

Estas funciones se llaman Functional Components, antes se llamaban Stateless Functional Components, porque no podian manejar estados o no podian manejar estados de la manera que hacian las clases. Pero desde que React introdujo los

Hooks ahora el codigo de los Functional Components es mucho mas fácil de leer y de mantener.

Los componentes se esperan que devuelvan algo que le sirva a la renderizacion de HTML.

root.render(<COMPONENTE/>); Asi vamos a renderizar un Functional Component en el root.

import './index.css'; Asi vamos a importar y vincular un archivo CSS a un js.

7. Retornar elementos en el Componente - Fragment

Con React.Fragment o importando {Fragment} de react vamos a poder retornar mas de una etiqueta HTML envuelta en un fragmento. Esto nos resuelve el problema de tal vez tener que tener un div con muchas etiquetas contenidas y que esos divs no los vayamos a utilizar.

Ejemplo:

```
<Fragment>
   <h1>Hola Mundo</h1>
   <p>Mi primera aplicación</p>
</Fragment>
```

Tambien poniendo <> y </> React lo interpreta como si fuese un fragmento.

```
<>
   <h1>Hola Mundo</h1>
   <p>Mi primera aplicación</p>
</>
```

8. Impresión de variables en el HTML

Para renderizar constantes dentro de JSX vamos a utilizar las llaves {VARIABLE}. Puede contener valores primitivos como string, char, int, float, etc. Si le pasamos un array va a imprimir de manera separada todos los valores que tiene dentro. No renderiza booleanos. No renderiza objetos salvo que lo manipulemos con un JSON.stringify y manipularlo como JSON.

9. Comunicación entre componentes

Hay un concepto fundamental en React que son las propiedades que son enviadas a los componentes. Se las conocen cono properties pero le ponen la abreviatura de props.

props son las propiedades enviadas del padre hacia los componentes hijos.

Las props son objetos vacios si no se envia nada.

Para enviar props, al momento de renderizar un componente podemos enviar propiedades. Ejemplo: root.render(<PrimeraApp saludo="Hola, Soy Goku"/>);

Lo que se envia como props es un JSON.

Puedo acceder a una property recibiendo un argumento en mi Functional Component y con el nombre que le asigne el operador '.'NOMBRE_PROPERTY. Normalmente se utiliza la desestructuracion y no el operador . para ingresar al valor.

Si mi property esperaba un valor puedo obligar a que reciba algo (lo veremos mas adelante) o puedo poner un valor por defecto en caso de que no se envie.

10. PropTypes

Podemos forzar que la persona que utilice el componente enviandole una prop para su correcto. Una manera es preguntar si el prop existe o no, en caso que no enviar un error. Otra manera, mucho mas eficiente, es importar PropTypes from prop-types. Prop-types lo vamos a definir debajo de la funcion y le vamos a indicar como si fuese un objeto clave:valor que nombre de variable espera y que tipo de valor. Si ese valor se espara si o si y no puede ser un undefined vamos a utilizar .isRequired

Ejemplo:

```
FUNCION_COMPONENTE.propTypes = {
   NOMBRE_VARIABLE : PropTypes.TIPO_ESPERADO.isRequired
}
```

11. DefaultProps

DefaultProps nos da la posiblidad de no tener que inicalizar valores por defecto cuando recibo props. Tambien nos sirve para que el prop quede registrado y exista, ya que si pasamos un valor por defecto ese prop NO existe fuera del ambito de la funcion.

Esta contenido en la importacion de React.

```
PrimeraApp.defaultProps = {

   subtitulo : "Soy un subtitulo"

}
```

El snippet rafcp crea un Functional Component con PropTypes

12. Evento click (Eventos en general)

Documentacion: https://es.reactjs.org/docs/events.html

Ejemplo: <button onClick={Manejador_useState}>+1</button>

13. useState - Hook

Documentacion: https://es.reactjs.org/docs/hooks-intro.html - https://es.reactjs.org/docs/hooks-overview.html

Video explicativo: https://www.youtube.com/watch?v=4pO-HcG2igk

Un hook es una función especial de React. Se utilizan para hacer reactivo los componentes, es decir, que se renderize una parte del componente cuando utilicemos el hook.

useState Es normalmente utilizada para establecer el valor a una variable. Contiene una variable y una funcion en un array de 2 posiciones. La primera posicion sera un valor X y la segunda un setter para cambiar el valor que tiene en su primera posicion el array de useState.

const [VALOR_ACTUAL, FUNCION_SETTER] = useState(value); Desestructurando el array.

Definicion Documentacion:

Los Hooks son funciones que te permiten “enganchar” el estado de React y el ciclo de vida desde componentes de función. Los hooks no funcionan dentro de las clases — te permiten usar React sin clases. (No recomendamos reescribir tus componentes existentes de la noche a la mañana, pero puedes comenzar a usar Hooks en los nuevos si quieres).

React proporciona algunos Hooks incorporados como useState. También puedes crear tus propios Hooks para reutilizar el comportamiento con estado entre diferentes componentes. Primero veremos los Hooks incorporados.