**React y sus cosas.**

*Importante de react, su JSX, es javascript pero como con etiquetas.*

*Buena práctica, hacer un .js que exporte todas las llamadas a la API, devolviendo lo que sea.*

*IMPORTANTE: Mejor hacer export de las funciones, para luego al importar hacemos {nombre} y nos aseguramos de mantener el orden*

Otra cosa importante de react son los estados, que controlan los campos.

En nuestro APP.jsx habrá una función app que devolverá el componente principal con lo que queramos. Antes de eso deberíamos trabajar con los estados que hemos mencionado:

El const de launches esta declarando una variable, que aquí se hace con estados, el setLaunches es un setter normal y el valor dentr4o de useState el valor inicial

Texto

Descripción generada automáticamente

Aquí con el useEffect, que necesita 2 parámetros (función y array), y esto se ejecuta al principio, lo que hacemos es llamar a la .js de la API que hemos hecho que nos devuelve el array de lanzamientos, entonces cada vez que cambie algo del array que pasamos se vuelve a ejecutar, y con el set se guarda en Launches, que luego lo metemos en el map en un return para que nos lo muestre todo. IMPORTANTE, si no pusiéramos en el use effect el array vacio de dependencias, estaríamos haciendo un bucle infito de actualiuzaciones, así solo se ejecuta una vez al principio

Texto, Escala de tiempo

Descripción generada automáticamente

Además para el ejemplo de nuestro .js con la api, tendremosun método que devolverá el array de cosas, luego para recorrerlo será necesario un .map, puesto que con jsx se necesitan funciones que devuelvan cosas, en estas funciones, por cada elemento se necesitara una **KEY.**

    //se suele usar object destructuring con las props se puede poner que nos llegan props y luego de props cogemos lo que sea

    // O se puede crear variables con las cosas. En nuestro ejemplo comentado tendriamos ahora la variable persona y prueba

    //Pero también podríamos hacer en la funcion Personaje({persona, prueba}){ y seria lo mismo

    /\*

    const {persona, prueba} = props

    \*/

Podemos usar props.children para acceder a cualquier cosa que metamos entre la tag de apertura y la de cierre del componente, como si dentro de un botón fuera.

IMPORTANTE CON LA EJECUCIÓN DE LOS EVENTOS,

Si tienen parámetros se van a ejecutar al cargar la página por lo que lo suyo es hacer un arrow function sin parámetros que llame a la función con parámetros.

<div className="card personaje" onClick={console.log("MOUSE OVER")}> {/\* El problema es que esto se ejecuta nada mas cargar la aplicación, entonces para evitar eso la hacemos arrow function como vemos debajo \*/}

             <Tag  variant='solid' colorScheme={comprobarPlaneta()} size='sm' w={20}>

                 <Link to={`/planeta/${planeta.name}`}>{planeta.name} {props.prueba}</Link>

            </Tag>

            <div className="card-body">

                <h2 className="card-title" onClick={() => console.log("MOUSE OVER 2")}> <strong>{props.persona.name}</strong> </h2>

BIEN= 

MAL:



**SI QUEREMOS UTILIZAR HOOKS, el nombre del archivo debe tener la primera letra en mayúsculas, el componente o lo que sea**

El **UseState** y el setLoquesea tienen una peculiaridad con los objetos.

Si llamamos a setLoquesea lo cambiamos entero, imaginemos que solo queremos cambiar un valor de una propiedad del objeto, entonces tendríamos que copiar el objeto entero antes y modificarlo, sería así: con el …person como propiedad dentro del objeto que creamos en el setPerson. Con un array sería parecido.

Texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

(SEGUNDA FOTO ES UNA MEZCLA CON LO SIGUIENTE TOO)

Se pueden hacer tantos useState como quieras.

A veces al hacer el setLoquesea no toma bien el valor anterior por timeouts o por retrasos, entonces habría que hacerlo con una función flecha dentro del set: (comentado seria lo normal, un incrementador con tiempo, que solo incrementa 1 vez aunque clickemos 5 porque coge el valor al clicar, no al ejecutarse), el parámetro prevState coge automáticamente el valor de value parece, de lo que sea el STATE. IMPORTANTISIMO EL RETURN, SINO NO VA

Una captura de pantalla de un celular

Descripción generada automáticamente con confianza media

EL USE EFFECT es como un observable que se ejecuta con cada renderizado. Tiene unos detalles. Si ponemos un segundo argumento un array vacio solo se ejecutará al renderizar la página por primera vez, en cambio si ponemos una variable o algo se ejecutará cuando esa variable cambie, es decir, como un observable

Texto

Descripción generada automáticamente

Podemos tener todos los que queramos.

Importante, tener cuidado con los listener si los metemos en el useEffect, que se van a crear muchos, entonces limpiarlos.

Captura de pantalla con letras

Descripción generada automáticamente

Cuando hacemos el fetch tenemos que hacer el json, pero podemos poner luego otro then y seguir tratando los datos:

Pantalla de computadora con letras

Descripción generada automáticamente con confianza media

En esta imagen, además, (poque sino con el catch no iría) comprobamos que la petición nos dé un código adecuado, y si no, ponemos que sea error

NUEVO JS:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

El first value quiere decir que si text es false devuelve hello world (false o vacio), pero si no lo es devuelve el texto, en este caso k.

Con second value es al revés, si text es falso lo devuelve (y solo lo muestra si es etiquetasdo, rollo div), en cambio si es true devuelve el hello world

BOTON CON TOGGLE INGENIOSO

Texto

Descripción generada automáticamente

Además, otra manera de mostrar el h1… solo cuando esError es true, si es false no se muestra nada

FORMULARIOS

Texto

Descripción generada automáticamente

El modal de VUE aquí es un value con onChange pasando el evento.

Cuando tenemos muchos campos y generamos un objeto se puede utilizar el NAME que le hemos dado al input:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Lo que hacemos es llamar en el onChange directamente a este método, que actualizará la persona con la propiedad adecuada (porque la coge de name) y con el value adecuado

Truco para generar ids de manera rápida y luego mapear bien con key:

