Curso React 11330

Preguntas Frecuentes (FAQ)

Autor: Mariano L. Acosta [TUTOR]

Desafio 1 - Crear Proyecto

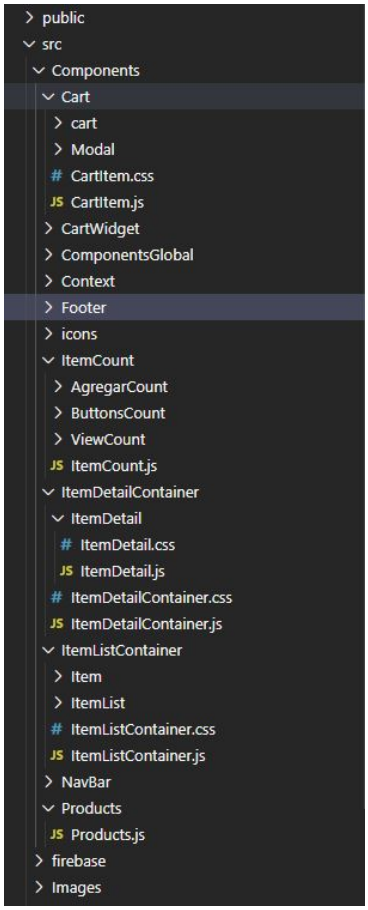
Recomendaciones:

- No hay mucho que decir, usen `npx create-react-app nombre` y sigan las instrucciones para crear un repo en Github.
- Colocar un nombre representativo a la tienda, ya que van a trabajar sobre este proyecto por el resto de la cursada

Desafio 2 - NavBar

Reglas a seguir:

- Pueden usar Bootstrap, SASS, etc... sí asi lo desean. Para eso deben instalarlo primero con el npm. Por ejemplo para usar bootstrap: `npm i bootstrap --save`
- El componente `<App/>` solo debe retornar un `<NavBar/>`. No debe haber ningún otro elemento en App.js
- Cada componente de React debe estar obligatoriamente declarado en un archivo aparte, dentro de una carpeta y en el directorio /Components. Ej: Navbar.js va dentro de /NavBar y esta última carpeta va dentro de /Components (/Componentes se crea dentro de /src). **Este criterio lo tienen que aplicar en todos los desafios**.
- Los archivos de estilos **deben** estar incluidos en la misma carpeta de su respectivo componente. Está **totalmente prohibido** declarar el estilo de un componente en App.css, index.css u otro archivo de jerarquía superior.
- Para aplicar estilo a un componente, crean por ejemplo un archivo navbar.css, lo editan y luego lo incluyen en NavBar.js asi: `import './navbar.css';` (dicho archivo css debe estar dentro de la carpeta /Navbar)
- Cada carpeta y cada archivo debe llevar **exactamente** el mismo nombre que su componente.
- Les dejo un ejemplo sobre cómo deberia verse su carpeta de componentes y su estructura:



Desafio 3 - Vista Home

Recomendaciones:

- Aprendan cómo realizar destructuring porque es muy importante de aquí en más. Les dejo un tutorial: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment
- En vez de hacer `function Home (props)` les recomiendo emplear

  `function Home ({ title })`

  De esta manera pueden acceder directamente a `title` sin hacer `props.title` (es un ejemplo de destructuring).
- Asegurense que pasan el título por props. Otras maneras de implementarlo no serán aceptadas.
- Por ahora el `<ItemListContainer/>` está vacio.
- Existen varias maneras para importar imágenes en React. Para el curso les recomiendo que creen una carpeta `/Img` dentro de `/Public` y coloquen todas sus

  fotos ahí. Luego utilizen este path en **TODOS** sus componentes : `./img/nombre_img.foo` . Esto funciona siempre porque luego del bundling todos los archivos quedan referenciados relativamente al index.html que se encuentra dentro de /Public.

- La otra manera es haciendo un import: `import nombre_img from ´/*path a la imagen*/´`. Después, la variable que importaron es su imagen. Si bien esta forma también es útil no se la recomiendo para la cursada, debido a que cuando utilicen Firebase obligatoriamente van a necesitar tener todas sus imágenes en `/Public`.

Desafio 4 - Contador

- Si tienen dificultades para realizarlo les dejo un tutorial que grabé en Twitch: https://www.twitch.tv/videos/780674436
- A estas alturas la estructura de carpetas dentro de /Componentes debe reflejar la jerarquia de componentes. Por ej: si el contador lo utiliza sólo el ItemDetail entonces la carpeta /Contador va dentro de la carpeta /ItemDetail, etc...

Desafio 5 - Promises

Recomendaciones:

- Creen en un archivo aparte un array con la información de sus productos, por ej, `products.js`. Luego, exporten dicho array:

```
let products = [/*..datos de sus productos..*/];

export default products;
```

Después, importen el array asi:

```
import products from '/*path al archivo products.js*/'
```

De esta manera su código va a ser mucho más legible.

- Declaren el promise afuera del componente:

```
let request = new Promise((res)=>{
    setTimeout(res(products),1500)
});
```

Luego lo usan asi:

```
request.then((res)=>{
  /*usen el setter del estado*/
})
.catch(/*opcional*/)
.finally(/*opcional*/)
```

- **Importante:** En el campo `photo` guarden el nombre del archivo, ej: 'remera.jpg'. Más adelante lo usan asi: `<img src = {"./img/" + photo}/>`
- Necesariamente tienen que utilizar un .map(). De esta manera se ahorran de repetir para cada producto de forma manual. La forma de utilizarlo es la siguiente:

```
products.map((item)=>{
    return <Item /*aqui van los params de cada item*/ />
})
```

Desafio 6 - Consumiendo APIs

Reglas a Seguir:

- **NO** vuelvan a definir el array de productos para el ItemDetail. Si hicieron lo que les dije anteriormente de crear un archivo aparte para *products*, sólo lo necesitan volver a importar.
- Por ahora el producto que buscan usando el promise debe estar hardcodeado, es decir, el índice queda fijo. Más adelante van a usar un .find().
- Pueden usar un spinner para cuando todavía esta cargando la aplicación. Para eso buscan "spinner" en https://www.npmjs.com/ y lo instalan en npm. Luego, crean un estado 'loading' con el *useState(true)* y lo setean 'false' cuando termina de cargar. Para que quede más claro:

```
const [loading, setLoading] = useState();

useEffect(() => {
    request.then((item)=>{
         setLoading(false);
       /*resto del código*/
    })
  }, [])

return (
    <>
        {loading && <ItemDetail item={item} />}
    </>
)
```

Desafio 7 - Routing

Reglas a Seguir:

- El id del producto se obtiene asi: `{id} = useParams()`
- Para buscar el item con el id usen .find() ya que devuelve un objeto. Si usan .filter() devuelve un array con un objeto, por lo tanto van a tener que deconstruirlo primero para seguir usándolo.
- Es posible que el .find() no les funcione porque utilizaron un *strict equal* (===) en vez de un *equal* (==). Esto se debe a que en el primer caso el datatype tiene que ser idéntico, en el segundo no. Es un problema habitual ya que el useParams() devuelve **String** y ustedes suelen guardar el id como **Number**.
- El routing es bastante sencillo, sigan la diapositiva de la clase para ver cómo realizarlo.
- *Opcional*: Si en el futuro desean crear una version de producción para su tienda, utilicen `<HashRouter/>` en vez de `<BrowserRouter/>` ya que este último

no funciona para SPA (Single Page Application).

Desafio 8 - Sincronizar Contador

Reglas a Seguir:

- Utilicen un `evt.stopPropagation()` para evitar que el onClick() se propague a los ancestros.
- Guardar en un estado del ItemDetail el siguiente objeto `{cantidad: /*cantidad*/, item: /*guardar todo el item aqui*/}` . Minimante debe quedar asi, otras formas no son válidas ya que el resto del trabajo se basa en ese estado.
- Asegurarse que desaparezca el contador cuando le dan a Agregar y en su lugar aparezca un botón que diga **Terminar Compra**. Para realizarlo es muy similar al ejemplo que dejé en el Desafio 6, sólo que ahora van a necesitar usar el negador lógico `!`. Por ejemplo:

```
{compra <ItemCount/>}
{!compra <Button/>}
```

El estado *compra* se vuelve verdadero cuando agregan los items al cart.

Desafio 9 - Context

Reglas a Seguir:

- El parametro *value* recibe un **único** objeto que contienen todas las variables, estados, setters, etc... que desean incluir en el context. ej: `<CartContext.provider value = {{ cart,setCart, /*etc...*/ }}>`

  Nótese las dobles llaves.

- Usen este patron de diseño (archivo aparte llamado *context.js*):

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

export const CartContext = React.createContext();

export const useCartContext = () => useContext(CartContext);

export default function CartProvider({ children, defaultCart }) {
    const [cart, setCart] = useState([]);
    /*definir demás estados*/

    function addToCart(item, qnt) {
      if (cart.length !== 0) {
            /*
              NO USEN .push() en el state!!!
              creen una variable auxiliar y despues utilicen el setter.
            */
        }
    function removeFromCart(itemID, qnt) {
    }
```

```
    function clearCart() {
    }

    return (
        <CartContext.Provider value={{ cart, /*etc...*/ }}>
            {children}
        </CartContext.Provider>)
}
```

Después toda su aplicación debe estar dentro de `<CartProvider></CartProvider>`

- Es clave que usen este patron para agregar un nuevo elemento: `setCart([...cart, /*nuevo elemento*/])`. Los `...` desarma un array en sus elementos constituyentes.
- Para utilizar las variables del context usen la funcion auxiliar *useAppContext()*

```
  import {useCartContext} from /*path*/ /*va entre llaves ya que el export no es default*/

  {cart, setCart, /*etc..*/} = useCartContext()
```

Desafio 10 - Cart View

Reglas a Seguir:

No hay mucho que agregar, asegurarse que Cart.js:

- Debe mostrar el desglose de tu carrito y el precio total.
- Debe estar agregada la ruta ‘cart’ al BrowserRouter.
- Debe mostrar todos los ítems agregados agrupados.
- Por cada tipo de ítem incluye un control para eliminar ítems.

Hagan uso de las funciones auxilares definidas en el CartContext.provider

- Tengan cuidado que los precios y demás variables numéricas sean del tipo Number y no String, porque después cuando quieran sumar no les va a funcionar (1 + 1 les va a dar 11 y no 2).

Desafio 11 - Firebase I

Reglas a Seguir:

- **NO GUARDEN LAS IMÁGENES EN FIRESTORE**. Sigan las indicaciones sobre imágenes que deje en el apartado sobre el Desafio 3. Guarden en el nombre de los archivos en un string. Entonces sólo va a ser necesario que realizen un `"./img/" + nombreFoto`
- Guarden los datos de accesso en .env.local usando variables de entorno. Para la corrección envian aparte el archivo .env.local. Ejemplo:

```
  REACT_APP_FS_API_KEY = AIzaSyWBxU0pskMtbEanNu_qkZgiqjmgqueLEs
  REACT_APP_FS_AUTH_DOMAIN = coder-tienda.firebaseapp.com
  REACT_APP_FS_DATABASE_URL = https://ccoder-tienda.firebaseio.com
```

```
REACT_APP_FS_PROJECT_ID = coder-tienda
REACT_APP_FS_STORAGE_BUCKET = coder-tienda.appspot.com
REACT_APP_FS_MESSAGING_SENDER_ID = 774223403805
REACT_APP_FS_APP_ID = 1:77451703805:web:e132ce27ca0c0504a49ff4
```

- Las instrucciones para usar firestore están detalladas en la diapositiva.
- *NO* deben crear un campo ID en firestore, el servicio debería generarles un ID automáticamente.

Desafio 12 - Firebase II

Reglas a Seguir:

- Cuando se haga click en **comprar** en el CartView, **SI o SI** se deben tomar los datos del cliente y mostrar el **ID DE COMPRA** en la pantalla.
- Les recomiendo que utilicen este código para crear la orden (úsenlo como punto de partida):

```
async function createOrder() {
    setLoading(true);
    const buyer = {
        name: document.getElementById("fname").value,
        lastName: document.getElementById("sname").value,
        number: document.getElementById("pnumber").value,
        email: document.getElementById("email").value
    };

    const newOrder = {
        ...buyer,
        items: cart,
        date: firebase.firestore.FieldValue.serverTimestamp(),
        total: total
    };

    const db = getFirestore();
    const orders = db.collection("orders");

    try {
        await orders.add(newOrder).then(id => {
            console.log('Order created with id: ', id.id);
            setOrderID(id.id);
            setChecked(true);
            setLoading(false);
        });

    }
    catch (err) {
      /*error handling*/
    }}
```