React Js

Guía de actividades para el Proyecto Final

CODERHOUSE

¿Qué es?

La **Guía de Actividades** es un espacio que nuclea todas las actividades prácticas que se relacionan directamente con los temas del proyecto final, abordados en el curso y que luego se evaluarán en las **preentregas correspondientes.**

La Guía fue creada para que puedas afianzar, potenciar y poner en práctica los saberes adquiridos en clase. Se desarrolla de manera **asincrónica. Sin embargo, su resolución es fundamental para la construcción del proyecto final.**

¡A practicar! 😉

Nota: te recomendamos que descargues el archivo para que lo puedas editar

CODERHOUSE

Instancias prácticas

Actividades en clase

Ayudan a poner en práctica los conceptos y la teoría vista en clase. No deben ser subidas a la plataforma y se desarrollan en la clase sincrónica.

¿Dónde las encontraré?
En las clases identificadas correspondientemente.

CODERHOUSE

Instancias prácticas

Actividades para el Proyecto final

Actividades relacionadas con el Proyecto Final. Su resolución es muy importante para llegar con mayor nivel de avance a las preentregas. Se desarrollan de **forma asincrónica.**

¿Dónde las encontraré?
Dentro de la plataforma en la Guía de Actividades y linkeadas en cada clase que se corresponda a la actividad.

CODERHOUSE

Instancias prácticas

Guía de ejercicios complementarios

Propuestas de ejercitación práctica complementaria, basadas en problemáticas comunes del desarrollo de aplicaciones. Se desarrollan en los AfterClass.

¿Dónde las encontraré?
En la carpeta de la comisión dentro de la [Guía de ejercicios complementarios](#).

CODERHOUSE

Instancias prácticas

Pre-entregas

Entregas obligatorias con el estado de avance de tu proyecto final que deberás subir a la plataforma a lo largo del curso y hasta 7 días luego de la clase, para ser corregidas por tu tutor/a.

¿Dónde la encontraré?
Dentro de la plataforma en la Hoja de Ruta del curso y linkeadas en cada clase que corresponda.

CODERHOUSE

GRILLA DE PRÁCTICAS Y PRE ENTREGAS

Clases	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Actividades de la Guía					◎	◎	◎		◎	◎	◎	◎	◎		
Proyecto Final				📊					📊						📊

Proyecto Final

CODERHOUSE

CLASE 05

Componentes II

CODERHOUSE

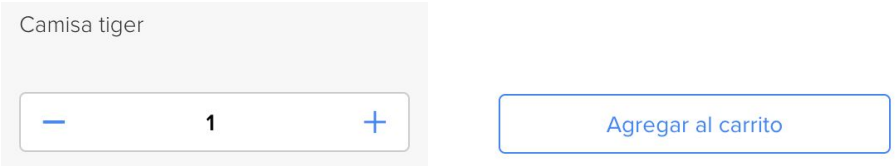
Actividad Nº 1

Segunda pre-entrega PF

Contador con botón

Descripción de la actividad.

- ✓ Crea un componente ItemCount.js, que debe estar compuesto de un botón y controles, para incrementar y decrementar la cantidad requerida de ítems



No es necesario usar este estilo, sirve a modo de orientación

CODERHOUSE

Actividad Nº 1

Contador con botón

Recomendaciones.

- ✓ El número contador nunca puede superar el stock disponible.
- ✓ De no haber stock el click no debe tener efecto y por ende no ejecutar el callback onAdd.
- ✓ Si hay stock al clickear el botón se debe ejecutar **onAdd** con un número que debe ser la cantidad seleccionada por el usuario.

Tener en cuenta.

- ✓ Como sabes, todavía no tenemos nuestro detalle de ítem y este desarrollo es parte de él, así que por el momento puedes probar e importar este componente dentro del **ItemListContainer**, solo a propósitos de prueba. Después lo sacaremos de aquí y lo incluiremos en el detalle del ítem.

Actividad Nº 1

Contador con botón

Ejemplo inicial.

function ItemCount({ stock, initial, onAdd }) {
 // Desarrollar lógica
}
- Adicionalmente, tendremos un número inicial (initial) de cantidad de ítems, de tal modo que si lo invoco del siguiente modo:
<ItemCount stock="5" initial="1" />
debería ver el contador inicializado en 1 por defecto

CLASE 06

Promises, asincronía y MAP

CODERHOUSE

Actividad Nº 2

Segunda pre-entrega PF

Catálogo con MAPS y promises

Descripción de la actividad:

- ✓ Crea los componentes Item.js e ItemList.js para mostrar algunos productos en tu ItemListContainer.js. Los ítems deben provenir de un llamado a una promise que los resuelva en tiempo diferido (setTimeout) de 2 segundos, para emular retrasos de red

Actividad Nº 2

Catálogo con MAPS y promises

Recomendaciones:

- ✓ Item.js: Es un componente destinado a mostrar información breve del producto que el user clickeará luego para acceder a los detalles (los desarrollaremos más adelante)
- ✓ ItemList.js Es un agrupador de un set de componentes Item.js (Deberías incluirlo dentro de ItemListContainer de la primera pre-entrega del Proyecto Final)
- ✓ Implementa un async mock (promise): Usa un efecto de montaje para poder emitir un llamado asincrónico a un mock (objeto) estático de datos que devuelva un conjunto de item { id, title, description, price, pictureUrl } en dos segundos (setTimeout), para emular retrasos de red.

Actividad Nº 2

Catálogo con MAPS y promises

Ejemplo inicial.

```
function Item({ item }) {
 // Desarrolla la vista de un ítem donde item es de tipo
 // { id, title, price, pictureUrl }
}

function ItemList({ items }) {
 // El componente va a recibir una prop `items` y va a mapear estos
`items` al componente `<Item ... />`
}
```

CODERHOUSE

CLASE 07

Consumiendo API's

Actividad Nº 3

Segunda pre-entrega PF

Detalle del producto - Parte I

Descripción de la actividad.

- ✓ Crea tu componente ItemDetailContainer con la misma premisa que ItemListContainer.

Recomendaciones

- ✓ Al iniciar utilizando un efecto de montaje, debe llamar a un async mock, utilizando lo visto en la clase anterior con Promise, que en 2 segundos le devuelva un 1 ítem, y lo guarde en un estado prop

Actividad Nº 3

Detalle del producto – Parte I

Ejemplo inicial.

```
const getItem = () => { /* Esta función debe retornar la promesa que
resuelva con delay */ }
function ItemDetailContainer() {
// Implementar mock invocando a getItem() y utilizando el resolver then
 return /* JSX que devuelva un ItemDetail (punto 2) */
```

Actividad Nº 3

Segunda pre-entrega PF

Detalle del producto - Parte II

Descripción de la actividad.

- ✓ Crea tu componente ItemDetail.js

Recomendaciones.

- ✓ ItemDetail.js, que debe mostrar la vista de detalle de un ítem incluyendo su descripción, una foto y el precio

CODERHOUSE

Actividad Nº 3

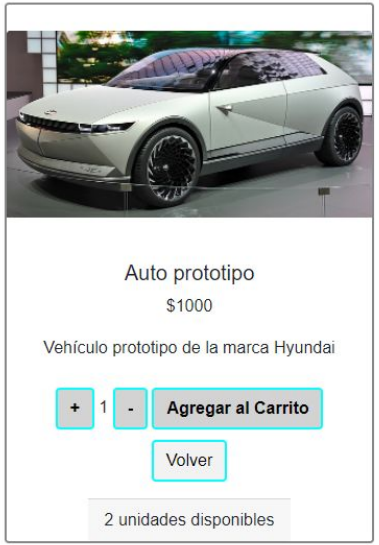
Detalle del producto – Parte II

Ejemplo inicial.

```
function ItemDetail({ item }) {

  return <>
            …
            // Desarrolla la vista de detalle
expandida del producto con su imagen, título,
descripción y precio
            ...
</>;

}
```



CODERHOUSE

CLASE 10

Eventos

Actividad Nº 4

Proyecto Final

Sincronizar Counter

Descripción de la actividad.

- ✓ Importa el ItemCount.js de la primera pre-entrega del PF en el counter ItemDetail.js, y configura el evento de compra, siguiendo los detalles de manual.

Recomendaciones

- ✓ Debes lograr separar la responsabilidad del count, del detalle del ítem, y esperar los eventos de agregado emitidos por el ItemCount
- ✓ Cuando ItemCount emita un evento onAdd almacenarás ese valor en un estado interno del ItemDetail para hacer desaparecer el ItemCount
- ✓ El botón de terminar mi compra debe poder navegar a un componente vacío por el momento en la ruta '/cart'.

Actividad Nº 4

Sincronizar Counter

Ejemplo inicial.

```
function ItemDetail({ item }) {
 onAdd(quantityToAdd) {
 // Hemos recibido un evento del ItemCount
 }
return <>
            …
     <ItemCount > // Configura las props de ItemCount. (
habría que mandarle? 👀

</>;
}
```

CODERHOUSE

CLASE 11

Context

Actividad Nº 5

Proyecto Final

Cart Context

Descripción de la actividad.

- ✓ Implementa React Context para mantener el estado de compra del user

Recomendaciones

- ✓ Al clickear comprar en ItemDetail se debe guardar en el CartContext el producto y su cantidad en forma de objeto { name, price, quantity, etc. } dentro del array de productos agregados
- ✓ Detalle importante: CartContext debe tener la lógica incorporada de no aceptar duplicados y mantener su consistencia.
- ✓ Métodos recomendados:
- ✓ addItem(item, quantity) // agregar cierta cantidad de un ítem al carrito
- ✓ removeItem(itemId) // Remover un item del cart por usando su id
- ✓ clear() // Remover todos los items
- ✓ isInCart: (id) => true|false

Actividad Nº 5

Cart Context

Ejemplo inicial.

```
import { createContext, useState } from "react";
export const CartContext = createContext();
const CartContextProvider = ({ children }) => {
    const [cartList, setCartList] = useState([]);
    const addToCart = (item, qty) => {  //implementa la funcionalidad para agregar un producto al carrito
    }
    const removeList = () => {//implementa la funcionalidad para dejar el carrito vacío
    }
    const deleteItem = (id) => {    //implementa la funcionalidad para borrar un producto del carrito
    }
    return (
        <CartContext.Provider value={{cartList, addToCart, removeList, deleteItem}}>
            { children }
        </CartContext.Provider>
    );
}
export default CartContextProvider;
```

CLASE 12

Técnicas de rendering

Actividad Nº 6

Proyecto Final

Cart View

Descripción de la actividad.

- ✓ Expande tu componente Cart.js con el desglose de la compra y actualiza tu CartWidget.js para hacerlo reactivo al contexto

Recomendaciones

- ✓ 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.
- ✓ De no haber ítems muestra un mensaje, de manera condicional, diciendo que no hay ítems y un react-router Link o un botón para que pueda volver al Landing (ItemDetailContainer.js) para buscar y comprar algo.

Actividad Nº 6

Cart View

Recomendaciones

- ✓ CartWidget.js.
- ✓ Ahora debe consumir el CartContext y mostrar en tiempo real (aparte del ícono) qué cantidad de ítems están agregados (2 camisas y 1 gorro equivaldrían a 3 items).
- ✓ El cart widget no se debe mostrar más si no hay items en el carrito, aplicando la técnica que elijas (dismount, style, etc).
- ✓ Cuando el estado interno de ItemDetail tenga la cantidad de ítems solicitados mostrar en su
- ✓ lugar un botón que diga “Terminar mi compra”

Actividad Nº 6

Cart View

Ejemplo inicial.

```
const Cart = () => {
    //accede al contexto con el hook useContext

     return (
          //recorre el estado global con un map y renderiza
          //nombre del producto, cantidad de items agregados, precio por item
          //importe total por producto (para lo cual necesitarás agregar una función
global
          //en el contexto
     );

export default Cart;
```

Actividad Nº 6

Cart View

Ejemplo inicial.

```
const CartWidget = () => {
    //accede al contexto con el hook useContext
    const ctx = useContext(CartContext);

    return (
        <Badge badgeContent={ctx.calcItemsQty()} color="secondary">
            <ShoppingCartOutlined />
        </Badge>
    );
}

export default CartWidget;
```

👀 calcItemsQty() es una función global del contexto que retorna la cantidad de items en el carrito
👀 Badge y ShoppingCartOutlined son componentes de MUI utilizados para el ejemplo. No es necesario que uses MUI. Puedes utilizar cualquier otro framework de CSS o aplicar tus propios estilos CSS puros.

CLASE 13

Firebase I

Actividad Nº 7

Proyecto Final

Item Collection I

Descripción de la actividad.

- ✓ Conecta tu nueva ItemCollection de google Firestore a tu ItemListContainer y ItemDetailContainer

Recomendaciones

- ✓ Conecta tu colección de firestore con el listado de ítems y con el detalle de ítem.
- ✓ Elimina los async mocks (promises) y reemplazalos por los llamados de Firestore.
- ✓ Si navegas a /item/:id, debe ocurrir una consulta de (1) documento.
- ✓ Si navegas al catálogo, debes consultar (N) documentos con un query filtrado, implementando la lógica de categorías y obteniendo el id de categoría del parámetro de react-router :categoryId.

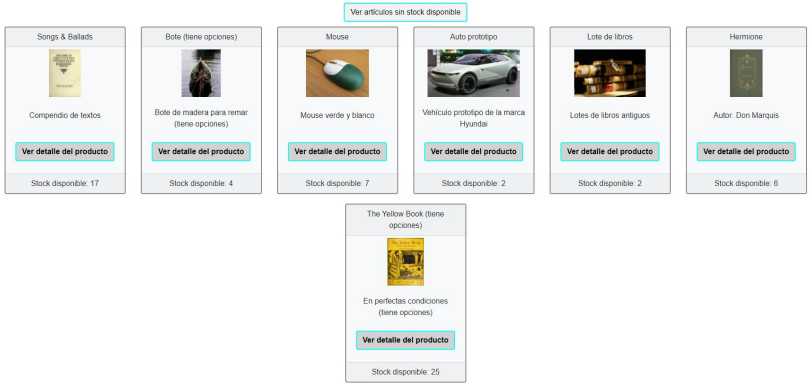
Actividad Nº 7

Item Collection I

Ejemplo inicial:

Desde aquí podrás ver un listado de Electrónica

Desde aquí podrás ver un listado de todas las categorías



CODERHOUSE

CLASE 14

Firebase II

Actividad Nº 8

Proyecto Final

Item Collection II

Descripción de la actividad.

- ✓ Crea tu colección de órdenes.

Recomendaciones

- ✓ Utiliza las operaciones de inserción para insertar tu orden en la colección y dale al user su id de orden auto-generada
- ✓ Crea los mappings para poder grabar un objeto del formato { buyer: { name, phone, email }, items: [{id, title, quantity, price}], date, total }
- ✓ Pista: Puedes controlar los stocks con multi-gets utilizando los itemld de tu cart.

Actividad Nº 8

Item Collection II

Ejemplo inicial.

👀 Crea el objeto ORDER con la información solicitada:

```
let order = {
  buyer: {
    name: "An Ecommerce Client",
    email: "client@coderhouse.com",
    phone: "123456789"
  },
  total: //utiliza una función global para calcular el importe total de la orden
  items: //mapea tu carrito para agregar aquí solo los datos solicitados de cada producto
  date: serverTimestamp()      //método de firebase para asignar la fecha y hora del servidor
};
```

👀 Luego deberás crear una función que agregue ese objeto a un documento en una collección ORDERS
Guíate de la documentación de Firestore 👉 https://firebase.google.com/docs/firestore/manage-data/add-data

Errores Frecuentes

Errores frecuentes

A continuación encontrarás ejemplos de errores que se cometen al generar un código y su respectiva solución.

Recuerda revisar este material con frecuencia para poder evitarlos y realizar buenas prácticas en el desarrollo del proyecto final.

Error #1

```
import ItemList from './ItemList';
import { useEffect, useState } from 'react';
import { useParams } from 'react-router';
import { firestoreFetch } from '../utils/firestoreFetch';

const ItemListContainer = () => {
    const [datos, setDatos] = useState([]);
    const { idCategory } = useParams();

    useEffect(() => {
        firestoreFetch(idCategory)
            .then(result => setDatos(result))
            .catch(err => console.log(err));
    }, [datos]);
```

¡Error muy grave!
Colocar el estado del componente como dependencia del useEffect().

Esto hace que la consulta hacia FireStore se ejecute muchas veces, terminando rápidamente con nuestra cuota gratuita de consultas en ese servicio.

Error #1

```
import ItemList from './ItemList';
import { useEffect, useState } from 'react';
import { useParams } from 'react-router';
import { firestoreFetch } from '../utils/firestoreFetch';

const ItemListContainer = () => {
    const [datos, setDatos] = useState([]);
    const { idCategory } = useParams();

    useEffect(() => {
        firestoreFetch(idCategory)
            .then(result => setDatos(result))
            .catch(err => console.log(err));
    }, [idCategory]);
```

Solución

La dependencia del useEffect() debe ser el idCategory que es recibido desde la URL

Error #2

```jsx
import Item from "./Item";
import { ProductsContainer } from './styledComponents';

const ItemList = ({ items }) => {
    return (
        <ProductsContainer>
        {
            items.map(item => <Item ... />)
        }
        </ProductsContainer>
    );
}

export default ItemList;
```

¡Error grave!
Mapear una props o un estado sin verificar si contiene elementos.

CONSECUENCIA:
Si la props o el estado es un array vacío, la aplicación dejará de funcionar (error en tiempo de ejecución)

CODERHOUSE

Error #2

```jsx
import Item from "./Item";
import { ProductsContainer } from './styledComponents';

const ItemList = ({ items }) => {
    return (
        <ProductsContainer>
        {
            items.length > 0
            ? items.map(item => <Item ... />)
            : <p>Cargando...</p>
        }
        </ProductsContainer>
    );
}

export default ItemList;
```

Solución

Verificar la longitud del array antes de aplicar el MAP

Error #3

```
1  import { Link, Router } from 'react-router-dom';
2  import { useEffect } from 'react';
3
4  const Item = ({ id, title, stock, price, pictureUrl }) => {
5      console.log(id);
6      return (
```

Error leve

Importar elementos y no usarlos (en el ejemplo: Router, useEffect y la props title no se usan luego en el código)

Dejar console.log() innecesarios en el proyecto final.

Error #3

```
1  import { Link, Router } from 'react-router-dom';
2  import { useEffect } from 'react';
3
4  const Item = ({ id, title, stock, price, pictureUrl }) => {
5      console.log(id);
6      return (
```

Solución

Eliminar las importaciones y/o declaraciones innecesarias. Eliminar los console.log() innecesarios.