

React JS

MANUAL DE DESAFÍOS



	FORMULARIO	
Clase 1	Formato: HTML + CSS + JS. Sugerencias: elegir la temática para el Proyecto Final y realizar las tareas en base a ello.	GENÉRICO (NO ENTREGABLE)
Consigna: Crea un formulario de contacto con nombre, apellido y un botón. Utiliza HTML, CSS y JS para simular su funcionamiento. Al presionar el botón debes emitir una alerta con el nombre concatenado con el apellido.		
Aspectos a incluir en el entregable: <ul style="list-style-type: none"> - Carpeta con 3 archivos: index.html, styles.css y script.js - En el index crea una estructura básica de un documento HTML con su body, head e importa los estilos y los scripts. 		

Desafío 1	CREAR LA APP UTILIZANDO EL CLI	
Clase 2	Formato: Link al repositorio de Github	ENTREGABLE
<p>Pre-requisitos: Es preferible que se tenga descargado node.js antes del comienzo de la clase para evitar inconvenientes de conexión durante el stream</p> <p>Consigna: Crea una aplicación utilizando el CLI con el nombre de tu tienda y ejecuta los comandos necesarios para instalar React, configurarlo y visualizarlo en tu navegador.</p> <p>Aspectos a incluir en el entregable:</p> <ul style="list-style-type: none"> - Link al repositorio en Github con tu app creada con CRA - Ten en cuenta haber descargado node.js y tener lista la herramienta de tu preferencia para Git (consola, GUI, etc.) 		

Desafío 2	MENÚ E-COMMERCE	
Clase 3	Formato: Link al último commit de tu repositorio en Github	ENTREGABLE
Consigna: En el directorio de tu proyecto, crea una carpeta dentro de src llamada components que contenga a NavBar.js para crear una barra de menú simple		
Aspectos a incluir en el entregable Crea una carpeta dentro de src llamada components que contenga a NavBar.js para crear una barra de menú simple, que tenga: <ul style="list-style-type: none">- Brand (título/nombre de la tienda)- Un listado de categorías clickeables (p)- Incorpora alguna librería de estilos con bootstrap/materialize u otro de tu preferencia (opcional).		
Ejemplo inicial: <pre>function App() { return <> // Desarrolla tu implementación de un navbar dentro del componente NavBar.js <NavBar /> <h2>Las ofertas de la semana</h2> </>; }</pre>		

Desafío 3	<i>Crea tu landing</i>	
Clase 4	Formato: Link al último commit de tu repositorio en Github	ENTREGABLE
Consigna: <ol style="list-style-type: none"> 1. Crear componente CartWidget.js con un ícono y ubicarlo en el navbar. Agregar algunos estilos con bootstrap/materialize u otro. 2. Crear componente contenedor ItemListContainer.js con una prop greeting y mostrar el mensaje dentro del contenedor con el styling integrado. 		
Aspectos a incluir en el entregable: <ul style="list-style-type: none"> - Parte 1: Crea un componente CartWidget.js que haga rendering de un icono Cart e inclúyelo dentro de NavBar.js para que esté visible en todo momento - Parte 2: Crea un componente ItemListContainer. Impórtalo dentro de App.js y abajo de NavBar.js 		
Ejemplo inicial: <pre>(1) function NavBar() { return <> // Customiza tu NavBar como prefieras <h3>TU_MARCA</h3> <CartWidget /> </>; } </NavBar> (2) function ItemListContainer() { // Incluye aquí el rendering de algún texto o título provisional que luego reemplazaremos por nuestro catálogo }</pre>		

	CLICK TRACKER	
Clase 5	Formato: stackblitz	GENÉRICO (NO ENTREGABLE)
Consigna: Crear en stackblitz un componente que registre qué cantidad de veces lo clickeamos, y lo muestre en pantalla en conjunto con la fecha/hora del último click, usando la librería Date de js		

Desafío 4	CONTADOR CON BOTÓN	
Clase 5	Formato: Link al último commit de tu repositorio en Github	ENTREGABLE
Consigna: Crea un componente ItemCount.js que debe estar compuesto de un botón y controles para incrementar y decrementar la cantidad requerida de ítems.		
Aspectos a incluir en el entregable: <ul style="list-style-type: none"> - Componente ItemCount.js con los respectivos controles de la consigna 		
A tener en cuenta: <ul style="list-style-type: none"> - 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 clicar el botón se debe ejecutar onAdd con un número que debe ser la cantidad seleccionada por el usuario 		

- Detalle importante: 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, sólo a propósitos de prueba. Después lo sacaremos de aquí y lo incluiremos en el detalle del ítem

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

	<i>MOCK ASYNC SERVICE</i>	
Clase 6	Formato: JSBIN ú otro IDE online	GENÉRICO (NO ENTREGABLE)
Consigna: Crear en JSBIN una promesa que resuelva en tres segundos un array de objetos de tipo producto. Al resolver, imprimirlos en consola <pre>{ id: string, name: string, description: string, stock: number }</pre>		

Desafío 5	<i>CATÁLOGO CON MAPS Y PROMISES</i>	
Clase 6	Formato: Link al último commit de tu repositorio en Github	ENTREGABLE
Consigna: 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 segs para emular retrasos de red		
Aspectos a incluir en el entregable: <ul style="list-style-type: none"> - 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 del desafío 3) - Implementa un async mock (promise): Usa un efecto de montaje para poder emitir un llamado asincrónico a un mock estático de datos que devuelva un conjunto de item { id, title, description, price, pictureUrl } en dos segundos, para emular retrasos de red 		

Ejemplo inicial:

```
function Item({ item }) {  
  // Desarrolla la vista de un ítem donde item es de tipo  
  // { id, title, price, pictureUrl }  
}  
  
function ItemList({ items }) {  
  // Desarrolla la vista utilizando un array de items y un map  
}
```

	<i>FETCH API-Call</i>	
Clase 7	Formato: Stackblitz	GENÉRICO (NO ENTREGABLE)
Crea en Stackblitz una app de React que al iniciar (utilizando un mount effect) utilice Fetch API para mostrar un listado de productos consumidos de la API de Mercadolibre y muestre los nombres de los primeros diez (¡si quieres mapear más datos hazlo!)		

Desafío 6a	<i>DETALLE DE PRODUCTO - A</i>	
Clase 7	Formato: Link al último commit de tu repositorio en Github	ENTREGABLE
Consigna: Crea tu componente ItemDetailContainer con la misma premisa que ItemListContainer.		
Aspectos a incluir en el entregable: <ul style="list-style-type: none">- Al iniciar utilizando un efecto de montaje, debe llamar a un async mock (promise) que en 2 segundos le devuelva un (1) ítem y lo guarde en un estado propio.		
Ejemplo inicial: <pre>const.getItems = () => { /* Esta función debe retornar la promesa que resuelva con delay */ } function ItemDetailContainer() { // Implementar mock invocando a.getItems() y utilizando el resolver then return /* JSX que devuelva un ItemDetail (desafío 6b) */ }</pre>		

Desafío 6b	DETALLE DE PRODUCTO - B	
Clase 7	Formato: Link al último commit de tu repositorio en Github	ENTREGABLE
Consigna: Crea tu componente ItemDetail.js		
Aspectos a incluir en el entregable: <ul style="list-style-type: none"> - ItemDetail.js, que debe mostrar la vista de detalle de un ítem incluyendo su descripción, una foto y el precio 		
Ejemplo inicial: <pre>function ItemDetail({ item }) { return <> ... // Desarrolla la vista de detalle expandida del producto con su imagen título, descripción y precio ... </>; }</pre>		

	AGREGA UN ROUTER A TU APP	
Clase 8	Formato: -	GENÉRICO (NO ENTREGABLE)
En tu aplicación, instala react-router-dom, agrégala al root de tu app y configura tus rutas apuntando a tu Home.		

	CONFIGURAR ROUTING (PRIMERA ENTREGA DEL PROYECTO)	
Clase 8	Formato: Link al último commit de tu repositorio en GitHub + Gif mostrando la navegabilidad por la app	ENTREGA INTERMEDIA
Consigna: Configura en App.js el routing usando un BrowserRouter de tu aplicación con react-router-dom		
Aspectos a incluir en el entregable: <ul style="list-style-type: none">- Rutas a configurar<ul style="list-style-type: none">- <code>'/'</code> navega a <code><ItemListContainer /></code>- <code>'/category/:id'</code> <code><ItemListContainer /></code>- <code>'/item/:id'</code> navega a <code><ItemDetailContainer /></code>- Links a configurar<ul style="list-style-type: none">- Clickear en el brand debe navegar a <code>'/'</code>- Clickear un Item.js debe navegar a <code>/item/:id</code>- Clickear en una categoría del navbar debe navegar a <code>/category/:categoryId</code>- Para finalizar integra los parámetros de tus async-mocks para reaccionar a <code>:itemId</code> y <code>:categoryId</code> utilizando efectos y los hooks de parámetros que vimos en clase! Si te encuentras en una categoría deberías poder detectar la navegación a otra categoría y volver a cargar los productos que correspondan a dicha categoría		

>>Además:

Deberás corroborar que tu proyecto cuente con:

1.

Navbar con cart

2.

Catálogo

3.

Detalle de producto

Incluir:

-

Archivo readme.md

A tener en cuenta: en la Rúbrica de Evaluación (ubicada en la carpeta de la camada) encontrarás un mayor detalle respecto a qué se tendrá en cuenta para la corrección.

Importante: La entrega intermedia no supone la realización de un archivo aparte o extra; marca que en este momento se hará una revisión más integral

	CREA UNA MÁSCARA DE INPUT	
Clase 9	Formato: Stackblitz	GENÉRICO (NO ENTREGABLE)
<div>En stackblitz crea un input de texto que no permita el ingreso de vocales, cancelando su evento onKeyDown en los keys adecuados</div> <div>Pista: el synthetic event de keydown tiene varias propiedades, usa las herramientas de desarrollador e investiga cuál te puede dar la información de la tecla ;)</div>		

Desafío 7	SINCRONIZAR COUNTER	
Clase 9	Formato: Link al último commit de tu repositorio en Github	ENTREGABLE

Consigna:

Importa el ItemCount.js del desafío N° 4 en el counter ItemDetail.js y configura el evento de compra

Aspectos a incluir en el entregable:

- 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
 - Cuando el estado interno de **ItemDetail** tenga la cantidad de ítems solicitados mostrar en su lugar un botón que diga **“Terminar mi compra”**
 - El botón de terminar mi compra debe poder navegar a un componente vacío por el momento en la ruta ‘/cart’

Ejemplo inicial:

```
function ItemDetail({ item }) {  
  onAdd(quantityToAdd) {  
    // Hemos recibido un evento del ItemCount  
  }  
  return <>  
  
    ...  
    <ItemCount > // Configura las props de ItemCount  
    <Button>Termina tu compra</Button> // Oculto hasta que el usuario seleccione la cantidad y clickee en comprar  
  
</>;  
  // Utilizar dentro de itemdetail el counter  
}
```

	CREA TU CONTEXTO	
Clase 10	Formato: -	GENÉRICO (NO ENTREGABLE)
<p>En tu proyecto en src/context/ crea un Contexto llamado cartContext.js cuyo valor default sea [], e importalo como provider de tu app con value []</p> <p><i>Pista: Los pasos son los mismos que en las slides, pero deberás exportar tu context creado para poder usarlo en App.js ;)</i></p>		

Desafío 8	CART CONTEXT	
Clase 10	Formato: Link al último commit de tu repositorio en Github.	ENTREGABLE
<p>Consigna: Implementa React Context para mantener el estado de compra del user</p>		
<p>Aspectos a incluir en el entregable:</p> <ul style="list-style-type: none">- CartContext.js con el context y su custom provider (Impórtalo en App.js)- Al clicar comprar en ItemDetail se debe guardar en el CartContext el producto y su cantidad en forma de objeto { item: {} , quantity }- Detalle importante: CartContext debe tener la lógica incorporada de no aceptar duplicados y mantener su consistencia.- Métodos recomendados:<ul style="list-style-type: none">- 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		

	CREA UN LOADER COMPONENT	
Clase 11	Formato: -	GENÉRICO (NO ENTREGABLE)
<p>En tu proyecto en src/context/ crea un Contexto llamado cartContext.js cuyo valor default sea [], e importalo como provider de tu app con value []</p> <p><i>Pista: Los pasos son los mismos que en las slides, pero deberás exportar tu context creado para poder usarlo en App.js ;)</i></p>		

Desafío 9	CART VIEW	
Clase 11	Formato: Link al último commit de tu repositorio en Github.	ENTREGABLE
<p>Consigna: Expande tu componente Cart.js con el desglose de la compra y actualiza tu CartWidget.js para hacerlo reactivo al contexto</p>		
<p>Aspectos a incluir en el entregable:</p> <ul style="list-style-type: none"> - Cart.js <ul style="list-style-type: none"> - 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.
- 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)

	CREA TU ITEM COLLECTION	
Clase 12	Formato: -	GENÉRICO (NO ENTREGABLE)
<p>Configura tu cuenta de Firebase y crea un cloud firestore con una nueva colección de items con los siguientes atributos (como mínimo): categoryId, title, description, image, price, stock</p> <p><i>Extra-mile: Si estás optando por el challenge-extra opcional crea también tu colección de categorías dinámicas (id, key y nombre)</i></p>		

Desafío 10	ITEM COLLECTION	
Clase 12	Formato: Link al último commit de tu repositorio en Github.	ENTREGABLE
<p>Pre-requisitos: Es preferible que se tenga acceso a una cuenta de google antes de la clase (creando una nueva o usando la suya personal) para optimizar el tiempo del módulo.</p> <p>Consigna: Conecta tu nueva ItemCollection de google Firestore a tu ItemListContainer y ItemDetailContainer</p> <p>Aspectos a incluir en el entregable:</p> <ul style="list-style-type: none"> - 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 		

	MODELA TUS ORDERS	
Clase 13	Formato: Trabajo en clase	GENÉRICO (NO ENTREGABLE)
Usa tu tus items del cart para modelar tu orden al siguiente formato: { buyer: { name, phone, email }, items: [{id, title, price}], total }, si todavía no creaste el formulario de compra puedes usar un objeto hardcodedo de tipo { name, phone, email }		

Desafío 11	ITEM COLLECTION II	
Clase 13	Formato: Link al último commit de tu repositorio en Github.	ENTREGABLE
Consigna: Conecta Firestore a tu APP para escrituras.		
Aspectos a incluir en el entregable: <ul style="list-style-type: none"> - 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, price}], date, total } 		
<i>Pista: Puedes controlar los stocks con multi-gets utilizando los itemId de tu cart</i>		

OPTIMIZACIÓN Y DETECCIÓN DE OPORTUNIDADES

Workshop **Formato:** Ejercicio en clase

GENÉRICO (NO ENTREGABLE)

Si bien funciona, en la siguiente pieza de código hay al menos 8 oportunidades de mejora. Encuéntralas, corrígelas e impleméntalas. Muchas de estas mejoras refieren a código redundante o oportunidades de reutilización de código.

Copia y pega el siguiente código en una sesión de react [stackblitz](https://stackblitz.com)

```
// Inicio del código
import React, { useState } from "react";

export default function App() {
  const [name, setName] = useState("");
  const [surname, setSurname] = useState("");
  const [age, setAge] = useState("");

  function onChangeName(evt) {
    setName(evt.target.value);
  }

  function onChangeSurname(evt) {
    setSurname(evt.target.value);
  }

  function onChangeAge(evt) {
    setAge(evt.target.value);
  }
}
```

```

}

function onSubmit() {
  console.log(`Your name is ${name} ${surname} and you have ${age} years`);
}

return (
  <>
    <div>
      <>
        <div style={{ display: "flex", marginBottom: 8 }}>
          <label style={{ marginRight: 4 }}>Nombre</label>
          <input type="text" onChange={evt => onNameChange(evt)} />
        </div>
      </>
      <>
        <div style={{ display: "flex", marginBottom: 8 }}>
          <label style={{ marginRight: 4 }}>Apellido</label>
          <input type="text" onChange={evt => onSurnameChange(evt)} />
        </div>
      </>
      <>
        <div style={{ display: "flex", marginBottom: 8 }}>
          <label style={{ marginRight: 4 }}>Edad</label>
          <input type="text" onChange={evt => onAgeChange(evt)} />
        </div>
      </>
    </>
  </>
)

```

```
{
  <button
    disabled={! (name !== "" && surname !== "" && age !== "") }
    onClick={evt => onSubmit(evt)}
    style={{}}
  >
    Crear orden
  </button>
}
</div>
</>
);
}
```