DOCUMENTACIÓN

Rescate Informático

Santiago Díaz

Contenido

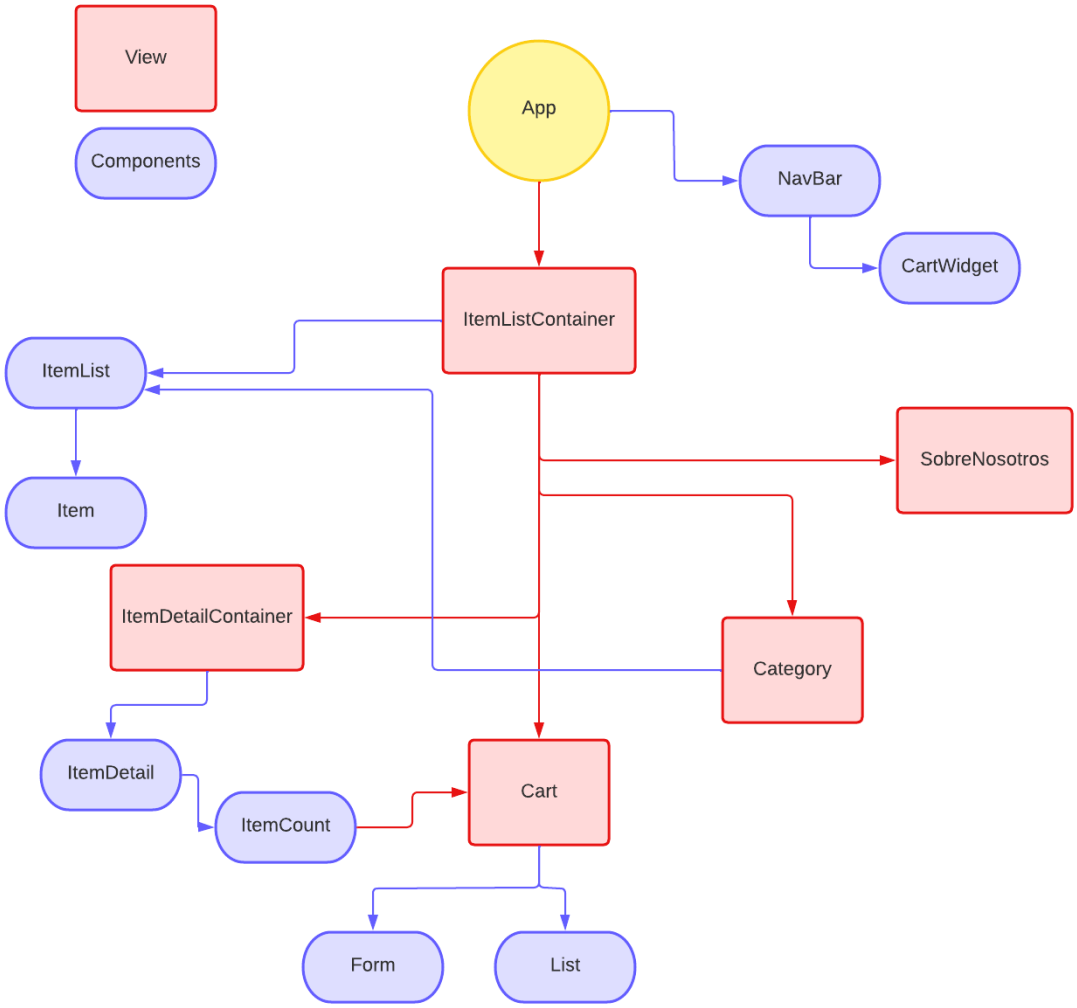
Introducción 2
Navegabilidad 2
Rescate Informático 3
App.js 3
ItemListContainer.js 4
ItemDetailContainer.js 5
Category.js 6
SobreNosotros.js 7
Cart.js 8
Aspectos Generales 9
Glosario 9
Conclusión 10
Feedback 10

Introducción

La App “Rescate Informático” se desarrolló durante el curso de “React Js” previsto por Coder House. Dicho curso tenia como objetivo final el crear una aplicación E-commerce aplicando los temas aprendidos. Rescate informático se enfoca en la venta de insumos computacionales variados.

Navegabilidad

A continuación, se muestra un mapa conceptual representando la estructura de la aplicación. Los elementos en rojo son correspondientes a la vista del usuario sobre una página de la aplicación. Los elementos en azul son los componentes que se utilizan dentro de las páginas de la aplicación con una funcionalidad ya sea estética como lógica. Las flechas rojas representan el movimiento dentro de la aplicación mientras que las flechas azules indican la pertenencia de un componente.



App.js

La aplicación comienza a la hora de crearla mediante la instalación y ejecución de React Js. Posee carpetas y archivos por defecto, los cuales se modifican a lo largo del desarrollo. El elemento “principal” es el archivo App.js al cual se le importan todos los componentes.

Dentro de App.js encontraremos el componente NavBar. El cual, al ser importado dentro de App, también se visualizará en las demás paginas ya que la aplicación implementa Simple-Page-Aplication[1] (SPA), o aplicación de página única.

App.js utiliza React Router Dom, el cual es una librería que nos va a permitir el navegar por las rutas de nuestra aplicación, para su correcta implementación se deberán importar los elementos a los que se visualizarán a la hora de movernos dentro de la aplicación. Estos son ItemListContainer.js como el elemento por defecto, ItemDetailContainer.js que nos permite ver el detalle del producto, Category.js que filtra los productos según la categoría a la que pertenecen, SobreNosotros.js que es una vista con información del local para el cual está hecha la aplicación, y, por último, Cart.js el cual visualiza la lista de productos que desea comprar el usuario.

Finalmente se importa el elemento CartProvider.js el cual es una forma de pasar datos que pueden considerarse globales a un árbol de componentes sin la necesidad de utilizar props[2] o redux. En este caso, CartContext posee la variable ItemsCart la cual almacena los ítems que el usuario desea comprar.

```jsx
//Style
import './App.css';
//Components
import NavBar from "./components/NavBar/NavBar";
//View
import ItemListContainer from './view/ItemListContainer/ItemListContainer';
import ItemDetailContainer from './view/ItemDetailContainer/ItemDetailContainer';
import Cart from "./view/Cart/Cart"
import SobreNosotros from './view/SobreNosotros/SobreNosotros';
import Category from "./view/Category/Category";
//React-Router-Dom
import {BrowserRouter, Routes, Route} from 'react-router-dom';
//Context
import { CartProvider } from './CartContext';



function App() {


  return (
    <CartProvider>
      <div className="App">
        <BrowserRouter>
          <header className="header">
            <NavBar />
          </header>
          <Routes>
            <Route path='/' element={<ItemListContainer />} />
            <Route path='/category/:categoryId' element={<Category />} />
            <Route path='/detail/:id' element={<ItemDetailContainer />} />
            <Route path='/sobre-nosotros' element={<SobreNosotros />} />
            <Route path='/cart' element={<Cart/>} />
          </Routes>
        </BrowserRouter>
      </div>
    </CartProvider>
  );
}

export default App;
```

ItemListContainer.js

Este elemento seria el “Inicio” de nuestra aplicación, es decir, cuando nosotros la abrimos, su página principal es esta. A ItemListContainer.js se le importan los componentes de firebase, el cual es “una plataforma para el desarrollo de aplicaciones web y aplicaciones móviles” que ofrece varias herramientas como firestore, la cual nos permite crear una base de datos en forma de array de objetos y forma la base de nuestra lista de productos.

ItemListContainer.js utiliza además dos componentes, cuyos nombres son Loader.js que es un componente de MUI[3] que sirve para cargar la animación de progreso de carga mientras la aplicación obtiene los datos de Firestore de manera sincrónica, y ItemList.js el cual recibe los datos obtenidos de la nube y realiza un renderizado de todos los productos obtenidos.

Encontramos dos funciones sencillas. La primera es LocalCart la cual recibe un array del LocalStorage[4] y lo guarda en la variable el Context. La segunda se utiliza para obtener los datos de los productos que se encuentran en la nube y guardarlos localmente, una vez realizado esto, verifica si hay elementos en el LocalStorage, de haberlos, los agrega al carrito, y por ultimo desactiva el componente de Loader.

```jsx
import React, { useState, useEffect, useContext } from 'react';
//Component
import ItemList from '../../components/ItemList/ItemList';
import Loader from '../../components/Loader/Loader';
//Style
import "./ItemListContainer.css"
//Firebase
import { collection, query, getDocs } from "firebase/firestore";
import { db } from '../../FireBaseConfig';
//Context
import { CartContext } from '../../CartContext';
//Img
import Banner from "../../components/assets/logos/Titulo-removebg-preview.png"



const ItemListContainer = () => {

        //Variables
        const[items, setItems] = useState([])
        const[loading, setLoading] = useState(true)
        const [itemsCart, setItemsCart] = useContext(CartContext);

        //Functions
        const localCart = () =>{
                var item = JSON.parse(localStorage.getItem("carrito"))
                setItemsCart(item)
        }
        const getItems = async () => {
                const q = query(collection(db, 'stock-computadoras'));
                const docs = [];
                const querySnapshot = await getDocs(q);
                querySnapshot.forEach((doc) => {
                        docs.push({...doc.data(), id: doc.id})
                });
                setItems(docs);
                if(itemsCart.length === 0){
                        if(localStorage.getItem("carrito")){
                                localCart();
                        }
                }
                setLoading(false);
        };
        useEffect(() => {
                getItems();
        })

        return (
                <div>
                        {
                                loading
                                ?
                                <div className='UserSection'>
                                        <Loader />
                                </div>
                                :
                                <div>
                                        <img src={Banner} alt="banner"/>
                                        <section className='seccion'>
                                                <h2 className='fw-bold text-success'>¿Tenés algún problema con tu dispositivo electrónico?</h2>
                                                <h2 className=''>Somos expertos en soluciones informáticas</h2>
                                                <h3 className=''>Empresa de informática con más de 20 años en el rubro, dedicada a brindar soporte técnico y asesoramiento a empresas y particular
                                        </section>
                                        <h2 className='fw-bold text-success mt-5 mb-0'>Nuestros Productos</h2>
                                        <div className="UserSection container">
                                                <div className='row justify-content-md-center'>
                                                        <ItemList Items = {items} className="col-6"/>
                                                </div>
                                        </div>
                                </div>
                        }
                </div>
        )
}

export default ItemListContainer
```

ItemDetailContainer.js

El elemento en cuestión es muy similar a ItemListContainer.js. Poseemos dos componentes. El primero es Loader.js, anteriormente mencionado, y el segundo es ItemDetail.js, Este último sirve para renderizar un solo producto al clickear sobre el mediante el filtrado por id de la base de datos y enviándonos a la ruta “detail/:id”, donde “:id” es un valor dinámico el cual se reemplazará por el id del producto.

Poseemos una sola función la cual filtra los datos del producto seleccionado y se los envía a ItemDetail.js.

```jsx
import React, { useState, useEffect } from 'react';
//Component
import ItemDetail from '../../components/ItemDetail/ItemDetail'
import Loader from '../../components/Loader/Loader';
//React-Router-Dom
import { useParams } from 'react-router-dom';
//Firebase
import { collection, query, where, getDocs, documentId } from "firebase/firestore";
import { db } from '../../FireBaseConfig';
//Style
import "./ItemDetailContainer.css";

const ItemDetailContainer = () => {

        //Variables
        const[items, setItems] = useState([])
        const[loading, setLoading] = useState(true)
        const { id } = useParams();


        //Functions
        useEffect(() => {
                const getItems = async () => {
                        const q = query(collection(db, 'stock-computadoras'), where(documentId(), '==', id));
                        const docs = [];
                        const querySnapshot = await getDocs(q);
                        querySnapshot.forEach((doc) => {
                                docs.push({...doc.data(), id: doc.id})
                        });
                        setItems(docs[0]);
                };
                getItems();
                setTimeout(() => {
                        setLoading(false);
                }, 1000);
        }, [id])




        return (
                <div>
                        {
                                loading
                                ?
                                <div className='UserSection'>
                                        <Loader />
                                </div>
                                :
                                <div>
                                        <div className='contenedor'>
                                                <ItemDetail {...items}/>
                                        </div>
                                </div>
                        }
                </div>
        )
}

export default ItemDetailContainer
```

Category.js

El elemento Category.js es idéntico a ItemDetailContainer.js, con la única diferencia de que este no filtra los productos por su id, sino que lo hace por la categoría a la que pertenecen. Esta puede variar entre periféricos, componentes o accesorios.

La ruta a la que los envía es “/category/:categoryId”.

```jsx
import React, { useState, useEffect } from 'react';
//Components
import ItemList from '../../components/ItemList/ItemList';
import Loader from '../../components/Loader/Loader';
//React-Router-Dom
import {useParams} from 'react-router-dom'
//Firebase
import { collection, query, where, getDocs } from "firebase/firestore";
import { db } from '../../FireBaseConfig';
//Style
import "./Category.css"


const Category = () => {

        //Variables
        const[items, setItems] = useState([])
        const[loading, setLoading] = useState(true)
        const { categoryId } = useParams()

        //Functions
        useEffect(() => {
                const getItems = async () => {
                const q = query(collection(db, 'stock-computadoras'), where('category', '==', categoryId));
                const docs = [];
                const querySnapshot = await getDocs(q);
                querySnapshot.forEach((doc) => {
                        docs.push({...doc.data(), id: doc.id})
                });
                setItems(docs);
                setLoading(false);
        };
        getItems();
        }, [categoryId])




        return (
                <div>
                        {
                                loading
                                ?
                                <div className='UserSection'>
                                        <Loader />
                                </div>
                                :
                                <div className="UserSection container">
                                        <div className='row justify-content-md-center'>
                                                <ItemList Items = {items} className="col-6"/>
                                        </div>
                                </div>
                        }
                </div>
        )
}

export default Category
```

SobreNosotros.js

SobreNosotros.js no posee ningún componente ni función en especial. Su propósito es mejorar la experiencia de usuario y realismo al proyecto mediante el uso de información del local en el cual está basado.

Se produce al inspirarse en otros sitios web de empresas similares.

```jsx
import React from 'react';
//Img
import marcas from "../../components/assets/Logos/marcas.jpg";
//Style
import "./SobreNosotros.css";


const SobreNosotros = () => {
    return(
        <div className='mt-4'>
            <h2 className='fw-bold text-success'>Nosotros</h2>
            <p className='fs-3 mx-2'>Somos una empresa dedicada a resolver los problemas tecnológicos. Poseemos un amplio stock de insumos computacionales para ofrecerle la mejor calida
            <h2 className='fw-bold text-success'>Local</h2>
            <p className='fs-4'>Nuestro local se encuentra en San Rafael, Mendoza.</p>
            <iframe src="https://www.google.com/maps/embed?pb=!1m18!1m12!1m3!1d3283.239971896295!2d-68.355643685175 79!3d-34.6233755660447!2m3!1f0!2f0!3f0!3m2!1i1024!2i768!4f13.1!3m3!1m2
            <h2 className='fw-bold text-success'>Contacto</h2>
            <p>Teléfono celular: 0260 456-2627</p>
            <p>Mail: diazpablo23@hotmail.com</p>
            <h2 className='fw-bold text-success'>Nuestras marcas</h2>
            <img src={marcas}  alt='marcas' />
        </div>
    )
}

export default SobreNosotros
```

Cart.js

El último elemento que compone la vista del usuario es el Cart.js. Este posee dos componentes. El primero es List.js el cual renderiza los productos del Local Storage si es que los hay, o de no ser así, visualiza los productos que el usuario haya agregado. El segundo componente es Form.js el cual implementa Formik[5] para validar los datos del comprador y luego subirlos a la base de datos en firestore.

Una vez realizada una compra, además de subir los datos del comprador y los productos seleccionados, también se realizan otras funciones, como vaciar el carrito, el local storage y editar la cantidad de productos disponibles en stock, además ofrece la opción de eliminar elementos por su id.

```jsx
import React, { useContext} from 'react';
//Context
import { CartContext } from '../../CartContext';
//Component
import List from '../../components/List/List'
import Form from "../../components/Form/Form"
//React-Router-Dom
import { Link } from 'react-router-dom';


const Cart = () => {

    //Variables
    const [itemsCart, setItemsCart] = useContext(CartContext);

    //Functions
    const clearCart=()=>{
        setItemsCart([])
        localStorage.clear()
    }
    const mapping = itemsCart.map((prod) => <List key = {prod.id} data = {prod}/>)


    return (
        <div>
            {itemsCart.length > 0 ? (
                        <div>
                            <div>{mapping}</div>
                            <div>
                            {function() {
                                let total = 0;
                            for (let i = 0; i<itemsCart.length; i++){
                                total = total + itemsCart[i].total
                            }
                            return <p>Total a pagar: ${total}</p>
                            }
                            }</div>
                            <button onClick={clearCart} className="btn btn-danger">Vaciar Carrito</button>
                            <Form/>
                        </div>
                    ) : (
                        <div><p>No hay items en el carrito</p>
                        <Link to="/" className="nav-item nav-link active fs-3 text-success">
                        Visitar stock
                        </Link></div>
                    )}
        </div>
    )
}

export default Cart
```

Aspectos Generales

Los que menciono como elementos de visualización del usuario no dejan de ser componentes, me gusta realizar una diferenciación y por eso es que también se encuentran en carpetas distintas.

Los componentes que utilizo a lo largo del proyecto son componentes funcionales, a los cuales le encontré una sintaxis más cómoda y sencilla en comparación con los componentes de clase.

El archivo .env sirve para guardar variables de entorno y no exponer los datos sensibles al público, como es un proyecto de prueba .env no se encuentra dentro del gitignore.

Firestore dejó de funcionar debido a un exceso de cuota de lectura, por eso es que tuve que realizar otro proyecto con otro mail, los datos del FireBaseConfig se encuentran dentro de un txt llamado firebase.

No explique en profundidad todas las funciones y componentes que se utilizan a lo largo del proyecto para que la documentación no resulte extensa.

Glosario

1-Single-Web-Aplication: es una aplicación web o es un sitio web que cabe en una sola página con el propósito de dar una experiencia más fluida a los usuarios

2-Props: Argumentos pasados entre componentes de React.

3-MUI: Material-UI es una biblioteca de código abierto que implementa el lenguaje visual de «materiales» de Google en sus componentes React. Ofrece la capacidad de combinar su biblioteca de interfaz de usuario, con el marco front-end de React.

4-Local Storage: LocalStorage y sessionStorage son propiedades de HTML5 (web storage), que permiten almacenar datos en nuestro navegador web.

5-Formik: es una librería de código abierto que sirve para la gestión de formularios, que nos proporciona las herramientas suficientes para que podamos crearlos en nuestras aplicaciones de una manera rápida, sencilla y eficiente.

Conclusión

La realización del proyecto y el curso personalmente se sintieron desafiantes y exigentes en comparación con otros cursos como desarrollo web y JavaScript. Se nota un gran cambio en cuanto a la sintaxis y manejo de los archivos y elementos del proyecto. A pesar de su dificultad, una vez captada la lógica básica de esta biblioteca/framework se vuelve cómoda gracias a la utilización de los componentes y demás herramientas como firestore o context.

Feedback

Coder House: Refiriéndonos al armado del curso, se siente un tanto incomodo la disposición de los desafíos, en el sentido de que es difícil cumplir con todos en base a un solo proyecto, y peor aun cuando debemos entregar dos desafíos semanales provocando que el proyecto esté lleno de componentes que se podrían juntar con otros u obviar para ahorrar código. Esto se puede solucionar permitiendo que los alumnos decidan si quieren realizar la entrega del desafío sobre su proyecto final o una aplicación distinta que solo cumpla con la consigna.

Profesor: El profesor dispuesto por Coder House para este curso tuvo un desempeño excelente. Explicaciones concisas, muestra de recursos varios para la facilidad de los estudiantes y el aspecto que deseo destacar personalmente es el de hablar sobre las entrevistas técnicas. Al comentarnos sobre su experiencia y hablarnos de conceptos teóricos nos pudo dar un vistazo de la situación a la que nos podemos enfrentar a la hora de conseguir trabajo.

Tutores: Globalmente se nota disposición y buena onda durante las clases por parte de ellos para resolver las dudas que surgen, pero hay que mejorar con las respuestas fuera de las clases. En varias ocasiones realice consultas sobre problemas y si, me contestaron dentro de las horas hábiles dispuestas por Coder, pero no fue una respuesta solucionando mi consulta, sino postergándola, esto me retraso en varias ocasiones obligándome a buscar respuesta por otro lado. No veo mal el tener que solucionar las cosas por mi cuenta, pero se supone que ellos deben estar para eso, porque de no hacerlo, simplemente toman asistencia. Para solucionar esto, Coder House debería asignar menos cantidad de alumnos a cada tutor, o que estos realmente dispongan del tiempo necesario para realizar un trabajo eficiente.

Alumnos: El compañerismo en clases virtuales es algo difícil de conseguir, personalmente solo me enfoco en el profesor y la clase, pero un aspecto a mejorar debe ser la utilización del chat durante la clase solo para desembocar dudas, y hablando de estas, es notorio la falta de habilidad que poseemos para comunicarle al profesor o tutores cual es nuestro problema, esto viene de la mano con una carencia de habilidades blandas y lenguaje técnico-teórico.

Santiago Díaz: Personalmente siento que carezco de conocimientos teóricos provocando errores en la comunicación y búsqueda de información. Hablando del proyecto, lo siento un tanto desprolijo debido a la cantidad de componentes que tuve que crear para completar los desafíos. Siento que debo practicar y estudiar mucho mas React Js y todo lo que este implica para obtener una mayor fluidez a la hora de programar.