TECH STACK

NextJS
TailwindCSS
Firebase

Explicación del Gráfico

CÍRCULOS, RECTÁNGULOS, LÍNEA DIVISORIA, Y CAPAS ROJA Y AZUL

1- Los círculos y los rectángulos son Componentes, pero los dividimos en 2 categorías
2- Los círculos representan rutas en NextJS. Las rutas en NextJS no están especificadas con React router, sino que basta con incluirlas en la carpeta de Pages.
3-Las rutas tienen nombres de archivo que comienzan en minúscula, para que así coincidan con la ruta concreta en la url
4- Los rectángulos representan componentes reciclables, que son hijos de las rutas.
5- Los componentes también se pueden dividir entre componentes de front-end, o de back-end (NextJS da la posibilidad de combinar ambas facetas en el mismo proyecto).
6- El Componente central es _app.js, que incluye como hijos a todo el resto de componentes.
7. Rodeamos _app.js con una capa de datos de Redux, representada en el gráfico por una línea roja
8. Rodeamos _app.js con un objeto AuthProvider, que vamos a utilizar para autenticar la cuenta del login, representada en el gráfico con una línea azul

FLOW BÁSICA DEL APP

=> Comenzamos en el HomePage, que se aloja en index.js
=> En el componente Home, obtenemos el objeto que contiene todos los productos, proveniente de una API externa
=> Los productos se los pasamos como props a ProductStore, que a su vez, se los pasa a cada uno de los Productos individuales y los renderiza en pantalla.
=> Cada producto individual tiene un botón, y al clickar, inyectamos dicho producto en nuestro array de items (basket), gracias al set up inicial de redux.

SET UP INICIAL DE REDUX

1- Iniciamos el store de redux, escuchando a basketSlice
2- basketSlice es la única capa de datos que establecemos en nuestra aplicación
3- como estado inicial , tenemos un array de productos (basket), que en principio está vacío
4- también establecemos 2 reducers para poder cambiar este array, añadiendo productos, o borrándose del array.

=>La ruta Home renderiza el componente reutilizable Header. Dicho componente consume el estado del basket (redux), y muestra en el carrito de la compra la longitud de dicho array.
=>En Header, también añadimos la funcionalidad de crear una sesión de login

SET UP INICIAL DE LOGIN

1- En la ruta [..nextauth].js configuramos los Providers de autenticación que va a soportar nuestro app. Por simplicidad, en el nuestro solo vamos a incluir el proveedor de google.
2- En nuestras variables de entorno tenemos almacenadas el GOOGLE_ID, Y EL GOOGLE_SECRET, necesario para crear una sesión con google.

Google cloud platform => Api y servicios => credenciales
https://console.cloud.google.com/apis/credentials/oauthclient/433778320447-6q6bsucuqbhh304ua74hvo1cscpaml01.apps.googleusercontent.com?hl=es&project=ecommerce-next-tailwind

=> en Header, ya podemos importar el hook useSession de next/auth-Client, y utilizar el objeto session para renderizar condicionalmente, o bien session.user.name para dar la bienvenida al usuario.
=>también podemos importar {Signin y Signout} de next/auth-Client, y cuando clickeamos en el div, ejecutaremos alternativamente las dos funciones para salir de nuestra sesión, o volver a entrar.

=>en Header, al clickear el icono del carrito, accedemos a la siguiente ruta, checkout.
=>en checkout , reciclamos el componente Header
=>en checkout consumimos el estado actual del basket, y renderizamos los productos que tenemos, con el formato de checkout
=> cada producto en el componente CheckoutProduct, tiene 2 botones. El botón addItemToBasket, inyecta en redux la acción addToBasket. El “payload” es el producto en cuestión que estamos añadiendo.
=>El botón “removeItemFromBasket”, inyecta en redux la acción removeFromBasket, con el id del producto como “payload”.
=> los reducers addToBasket y removeFromBasket escuchan la acción y el payload, y ejecutan la actualización del estado inicial de redux.

REMOVEFROMBASKET REDUCER - CÓMO FUNCIONA

1. el reducer “removeFromBasket”, escucha la acción y recibe el id como parte del “payload”
2. mediante el método findIndex, encuentra el índice en el array de productos en el que se encuentra el producto que queremos borrar
3. saca una copia del array basket
4. si ha encontrado el índice , entonces borra el objeto de ese indice del array que ha copiado (new Basket)
5. asigna newBasket (ya sin el producto) al array state.items.

=> la ruta checkout además de renderizar checkoutProduct, también crea una sesión en Stripe (createCheckoutSession), al clickar en el botón de “Proceed to checkout”

CONFIGURACIÓN INICIAL DE STRIPE

=> Para iniciar stripe, debemos abrir una cuenta, y obtener las llaves pública y privada
https://dashboard.stripe.com/test/apikeys

=>Añadirlas como variables de entorno en nuestro fichero
=>en next.config.js debemos incluir la llave pública de stripe

=>la función createCheckoutSession nos va a crear el backend para crear una sesión de checkout en la ruta /api/create-checkout-session
=>en la misma función, vamos a redireccionar al usuario hacia la página de checkout automática de la que nos provee Stripe
=> en dicha página vamos a ver, a la izquierda, los productos que vamos a comprar con su subtotal, y a la derecha , los datos de la tarjeta que debemos rellenar.
=> Al rellenar, le damos a su botón, y quedamos a la espera de que se resuelva el pago.

=>Necesitamos configurar un webhook para que escuche el momento en el que el pago esté procesado - funciona de manera inversa a un API

https://stripe.com/docs/webhooks/test

=>Lo mejor es crear un emulador, que va a escuchar el momento en el cuál esté listo el pago.
=> Configuramos webhook. No es fácil configurarlo - los pasos principales son

- conectar firebase con el backend
- hacer operaciones de CRUD para pasar la orden a nuestra base de datos de firebase

https://console.firebase.google.com/u/0/project/ecommerce-next-tailwind/firestore/data~2Fusers~2Fjavascriptdenoobapro@gmail.com~2Forders

=> Para crear el emulador, seguimos las instrucciones de la documentación

- Instalamos el Stripe CLI
- en la terminal, tecleamos stripe listen --forward-to localhost:3000/api/webhook, y la dejamos corriendo.
- Nos facilita una clave temporal del emulador, que incluiremos en nuestras variables de entorno

=> En el momento que se ha resuelto el pago, se dispara el evento de Stripe, ,y lo capta el webhook.

=> simultáneamente, redirigimos la ruta a success

RUTA SUCCESS

=> en la ruta create-checkout-session, es donde redireccionamos al usuario a la ruta success, en caso de que el pago haya sido exitoso, o de vuelta a checkout en caso de que haya habido algún fallo.

=>es una ruta intermedia que nos redirige a ORDERS, esta vez si, con el router que nos facilita next/router.

RUTA ORDERS

=> En orders, tenemos getServerSideProps.
=>Si no tenemos una sesión abierta, no nos retorna nada.
=> Leemos las órdenes que hay en Firebase (CRUD)
=>Utilizarmos los datos de firebase y los combinamos con los de Stripe
=>Lo pasamos como props a la ruta orders
=>en la ruta orders, lo renderizamos en pantalla,