Source code github: <https://github.com/adrianhajdin/ecommerce_sanity_stripe>

npm install 相關: <https://ithelp.ithome.com.tw/articles/10191783>

legacy-peer-deps相關: <https://juejin.cn/post/6971268824288985118>

css/justify-content: space-between: <https://w3c.hexschool.com/flexbox/4a029043>

React Roadmap: <https://roadmap.sh/roadmaps> <https://www.figma.com/community/file/1143555742445608897>

Javascript陣列處理方法filter(), find(), map().. : <https://www.casper.tw/javascript/2017/06/29/es6-native-array/>

Stripe金流, 通過Payoneer: <https://reurl.cc/yMDqX2>

HTTP method: GET, POST: <https://reurl.cc/AO6Qkj>

Next.js路由: <https://www.nextjs.cn/docs/api-routes/introduction>

async: <https://ithelp.ithome.com.tw/articles/10230214>

res.json(): <https://developer.mozilla.org/zh-CN/docs/Web/API/Response/json>

API, fetch, chart.js(cdn) : <https://ithelp.ithome.com.tw/articles/10230303>

Axios: <https://ithelp.ithome.com.tw/articles/10279366>

var, let, const: <https://chinese.freecodecamp.org/news/javascript-var-let-and-const/>

wireframe: <http://jinjin.mepopedia.com/~jinjin/ui/ui-05.html>

user flow: <https://reurl.cc/W18QdO>

RESTful API: <https://reurl.cc/GEaN23>

**前置-node module套件**

1. npx create-next-app , 選項y, 名稱ecommerce
2. 到source github找package.json, 安裝dependencies項目: <https://github.com/adrianhajdin/ecommerce_sanity_stripe/blob/main/package.json>
3. npm install --legacy-peer-deps
4. npm install --legacy-peer-deps --save next-sanity-image
5. npm install --legacy-peer-deps --save-dev @babel/preset-react

**前置-sanity**

1. npm install -g @sanity/cli
2. sanity init --coupon javascriptmastery2022
3. 安裝sanity至Next.js project裡, 設定sanity資料庫名稱, sanity帳號對連
4. sanity docs 查看sanity 文檔, sanity manage 查看sanity後台連線數據.
5. sanity start查看sanity 資料庫數據
6. 編輯schema, 創造資料庫分類. cd sanity\_ecommerce/schemas, 新增products.js, banner.js檔案
7. 設定product.js 與banner.js 的分類, 再import至schema.js裡面.
8. sanity start, 進去資料庫, 到product分類新增一個headphones, image取至tutorial連結的assets.

**前端-首頁架構**

1. ./page/index.js, 刪除裡面所有內容, 重建, rafce. (安裝ES7套件(vs code))
2. 根目錄, 建立.babelrc檔案(轉譯ES5至ES6), .eslintrc裡寫入next/babel(偵錯工具來檢查 JavaScript 靜態程式碼的語法和風格).
3. "presets": [["@babel/preset-react", {"runtime": "automatic"}]]
4. 在app.js, 加入import React from 'react'
5. Index.js 先設頁面架構, 更動styles/globals.css. 在index.js 裡面各項標籤中加入與globals.css連結

**前端-首頁裡的各項components**

1. 在根目錄新增components資料夾, 再新增HeroBanner.jsx, Cart.jsx, FooterBanner.jsx, HeroBanner.jsx, Layout.jsx, Navbar.jsx, Product.jsx,裡面輸入rafce 另外新增index.js, 把上述各項component導入./components/index.js
2. 到pages/index.js, 把<HeroBanner/>, <FooterBanner/>加進去.
3. 編輯HeroBanner.jsx

**Sanity資料-前端網頁 連結**

1. 根目錄, 創建lib資料夾, 再創建client.js檔案
2. 編輯client.js檔案, 導入@sanity/client, 新增函數export const client = sanityClient ({})
3. 開啟新terminal, cd sanity-ecommerce, sanity manage. 參照manage裡面的設定值,寫入const client裡各個參數: projectId, dataset, apiVersion, useCdn, token. token值要利用.env寫來保密. <https://www.sanity.io/docs/js-client>
4. Index.js, import {client} from ‘../lib/client’, 底下寫入getServerSideProp = async() => {}, 連結sanity數據顯示.
5. Index.js裡, 寫入{console.log(bannerData)}, 到網頁F12測試bannerData是否掛上.
6. Index.js裡, 寫入<HeroBanner heroBanner = {bannerData.length && bannerData[0]}/>, 再至HeroBanner.js, 參數導入heroBanner, 寫入各個{heroBanner.smallText},{heroBanner.midText}…等資料庫項目, 顯示至前端網頁.
7. 圖片顯示: import { urlFor } from '../lib/client', 寫入const builder = ImageUrlBuilder(client); export const urlFor = (source) => builder.image(source); <https://www.sanity.io/docs/image-url>

**主頁中間顯示<Product />**

1. Index.js裡, 於下方排列顯示產品位置寫入{products?.map((product) => <Product key={product.id} product = {product}/>)}
2. Product.jsx, 導入{product:{name, price, image}}, 把各項數據顯示在Product.jsx前端上

**主頁下方顯示<FooterBanner />**

1. Index.js, 寫入<FooterBanner footerBanner = {bannerData && bannerData[0]}/>
2. 編輯FooterBanner.jsx, import{Link}, import{urlFor}, 在const FooterBanner = () ={}裡的參數導入{footerBanner: { discount, largeText1, largeText2, saleTime, smallText, midText, desc, product, buttonText, image }}.
3. css排版, 並連結參數, 將資料顯示於前端FooterBanner.jsx

**Layout.jsx**

1. 架構Layout包\_app.js包index.js
2. import Head from 'next/head', import Navbar from './Navbar', import Footer from './Footer'; 配置<Head>/<title>, <header>/<Navbar>, <main>/{children}, <footer>/<Footer>
3. \_app.js, 用<Layout></Layout>去包<Component {...pageProps} />, 回到Layout.jsx, 參數加入{children}, <main></main>裡加入{children}

**Navbar.jsx**

1. import Link, import { AiOutlineShopping} from ‘react-icons/ai’
2. 編排Navbar, 新增左邊logo, 與右邊購物車圖示<AiOutlineShopping/>連結

**Footer.jsx**

1. import {AiFillInstagram, AiOutlineTwitter} from "react-icons/ai", 排版.

**建立分頁-產品細項-prouduct-[slug].js**

1. 到page資料夾, 建立product資料夾, 裡面再建立[slug].js檔案
2. [slug].js, []代表裡面的檔名slug是動態的.
3. rafce, 排版, import {client, urlFor} from '../../lib/client', 到index.js底下的getServerSideProps, 複製過來貼到[slug].js底下.用來連結sanity的資料>. <https://nextjs.org/docs/basic-features/data-fetching/get-server-side-props>
4. 把getServerSideProps改成getStaticProps (根據Next.js 官方說明，如果需要每次使用者瀏覽網頁時，伺服器都能呼叫API，將最新的資料都注入到HTML，則可以選擇使用 getServerSideProps 。 否則，如果不在意使用者是否拿到最新的資料，可以考慮使用 getStaticProps)
5. 在getStaticProps的參數()加上{params:{slug}}, 用來連結[slug].js的動態檔名, 與連結動態網址(從首頁選擇任一產品圖片, 即連結到任一產品細項的網址).
6. getStaticProps裡的const query調整成: const query = `\*[\_type == "product" && slug.current == '${slug}'][0]`
7. 再加上const productsQuery = '\*[\_type == "product"]' , const products = await client.fetch( productsQuery), 更動props為props:{product, products}
8. 在const ProductDetails = (), 參數裡加上{product, products}
9. 寫入getStaticPaths = async() => {}
10. 排版左方description(名稱, 星星, Details, 價格, 數量, 按鈕), 下方推薦商品: {products.map((item)=>(<Product key={item.id} product={item}/> ))}
11. 排版左下方小圖, 設置useState: const [index, setIndex] = useState(0), 並把setIndex整併至小圖顯示區域程式碼: {image?.map((item, i)=>(<img key={i} src={urlFor(item)} className={i === index? "small-image selected-image" : "small-image"} onMouseEnter={()=>setIndex(i)}/>))}

**各式功能程式碼, createContext, useContext**

1. 根目錄, 創建context資料夾, 再創建StateContext.js,
2. import React, { createContext, useContext, useState, useEffect} from 'react'; 使用Context可以避免的组件的层层props嵌套的问题。
3. 先設定購物車功能, useState/showCart, cartItems 價格,數量功能, ueseState/Price, Quantities, 購物車數量功能, useState/qty
4. 設定<Context.Provider>, 用參數{children}/return內的{children}-到\_app.js用<stateContext></stateContext>包住<Layout></Layout>相扣 , StateContext.js的return()裡設定value值:showCart, CartItems…
5. \_app.js, <Layout>裡再增加<Toaster>
6. StateContext.js裡, 寫入增加+數量功能:const incQty =( ) =>{ } , const decQty = ()=> {}, 再把incQty, decQty增加到const StateContesxt裡的<context.Provider/>的value裡.
7. 新增 const Context = createContext(), export const useStateContext = () => useContext(Context);
8. 到[slug].js, import {useStateContext} from "../../context/StateContext", const { decQty, incQty, qty } = useStateContext(), 把{decQty}, {incQty}, {qty}代入到功能區裡.
9. StateContext.js, 寫入”產品新增購物車功能”, const OnAdd = (product, quantity)=>{}, 到<Contex.Provider>內的value, 新增OnAdd, 再到[slug].js的<button className='add-to-cart'>中, 加入onClick={()=>onAdd(product, qty)

**購物車功能Cart.jsx**

1. 到Navbar.jsx, import {cart} , import {useStateContex}, 寫入const {showCart, setShowCart, totalQuantities } = useStateContex()
2. <button type='button' className='cart-icon'>裡加上onClick={()=> setShowCart(true)}, <span className=’cart-item-qty’>裡加上{totalQuantities},下方寫入{showCart && <Cart/>}
3. 編輯Cart.jsx, 帶入多個import: useRef, Link, AiOutline, TiDeleteOutline, toast, useStateContext, urlFor.
4. 寫入const{}= useStateContex().
5. 編輯版面, cart-container, cart-heading, empty-cart. 編輯購物車物品選項欄, product-container, cart-prouduct-image, item-desc, flex top, flex bottom, quantity desc, minus, num, plus, remove-item. cart-buttom, total, btn-container.
6. 至StateContext.js, 寫入toggleCartItemQuanitity, onRemove等函式, 設定到value裡, 再代入到Cart.jsx, const{toggleCartItemQuanitity, onRemove, totalPrice, totalQuantities} = useStateContext(), 並設定至各類button的onClick裡.

**信用卡付款, Stripe**

1. cd pages/api, 創建stripe.js, cd lib, 創建getStripe.js檔案,寫入const stripe = new Stripe();
2. 登入stripe網站, 查找Publishable Key, Secret Key, 寫入.env, 再帶入new Stripe()裡
3. In Next.js, 每個檔案必須要有自己的handler, 在stripe.js檔案裡,寫入export default async function handler(req, res){if{ }else{ }},裡面的寫法參照: <https://stripe.com/docs/checkout/quickstart?client=next>, parameters: <https://stripe.com/docs/api/checkout/sessions/object>
4. 至Cart.jsx, button(pay with Stripe), 設定onClick={handleCheckout}, 在檔案const Cart=()={}裡寫入handlecheckout = async =() =>{}. 先至lib, 創建getStripe.js檔案.寫入const getStripe =()=>{}, 參照<https://stripe.com/docs/checkout/quickstart> .回到Cart.js, import getStripe from ‘../lib/getStripe.js’. 寫入handlecheckout = async () =>{}參照: <https://stripe.com/docs/payments/quickstart> (next.js特別的地方是可以在前端用await叫調動出後端的東西)
5. 在stripe.js 的if(req, res === ‘POST’){ 後面一行寫入console.log(req.body), 從vs code backend來測試送到購物車的資料是否有送出往Stripe的伺服器.
6. 更改line\_items為line\_items: req.body.map((item)=>{}, 列舉購物車產品內容.
7. 測試stripe testing功能.

**商品交易成功頁面-success.js**

1. cd pages, 創建success.js, rafce. Import const {setCartItems, setTotalPrice, setTotalQuantities} = useStateContext(); 寫入useEffect(() => { localStorage.clear(); setCartItems([]); setTotalPrice(0); setTotalQuantities(0); runFireworks();}, []).
2. 排版success.js.
3. cd lib, 創建utils.js, 寫入煙花特效公式, 參照(FireWork): <https://www.kirilv.com/canvas-confetti/>, 回到success.js, import {runFireworks} from '../lib/utils'; useEffect裡加入runFireworks();
4. 到stripe.js, 將 cancel\_url: `${req.headers.origin}/?success=true`, 改成cancel\_url: `${req.headers.origin}/success`

**部屬Vercel**

1. afasf