**Documentation**

Documenting coding process for ecommerce application built using next.js, sanity.

1. First create next.js app
   1. npx create-next-app
   2. install all dependencies, use – legacy-peer-deps for resolving the package version conflict
   3. npm run dev 🡪 to start the next.js app
2. create another folder (sanity\_ecommerce) within the next.js created application
   1. After that, change to same directory
   2. npm install -g @sanity/cli
   3. if any conflict use npm install –legacy-peer-deps -g @sanity/cli
   4. then, sanity init
   5. data configuration -> y
   6. chose template
   7. change in gitignore of outer folder( change /node moduler to node module)
3. In sanity\_ecommerce setup the database schema in Schema folder (if you chose empty schema while installing the sanity)
   1. Make schema (for necessary documents) for banner and products
   2. Import it into the index.js and include it in array.
4. In front end clear global.css, home module.css content
   1. Paste necessary css from github
   2. Create new folder components
      1. Create necessary file like cart.jsx, footer.jsx ,and so on
      2. Type rafce(shortcut for template) type in each folder
      3. Create index.js within the component to export the all the component in efficient manner
         1. Export {default as Footer} from ‘./Footer’
         2. In pages -> index.js -> include-> import {Product, FooterBanner, MainBanner) from ‘../..components
5. Connect with sanity database
   1. First create library folder
      1. Within that create client.js and include necessary code to connect (refer from project)
      2. Import in index.js
   2. Difference between react and next.js
      1. In react we use life cycle hook (useEffect to fetch the data), but in next.js we use getServerSideProps which return the data to the render of same file
6. Passing data to children
   1. Pasing banner data to MainBanner component, as well as Products, FooterBanner components
   2. In mainbanner, displaying the fetch data
7. Next step
   1. Jsx in Layout component after populating in footer components
   2. Import Layout in \_app.js and include other element within layout.
   3. Whatever pass to the <component/> will be accessible by Layout component by keyword (children) as props
   4. Write jsx in navbar and footer component
8. Create folder (product) in pages
   1. Save file name as [slug].js in product 🡪 it causes file to be dynamic
   2. It is file based routing in next.js – we didn’t have to implement any kind of library like react-router-dom, we just created new folder with specific file name and can immediately start creating the jsx and logic for that component, it especially trigger with <Link>
9. In [slug].js use getStaticProps
   1. getStaticProps is a method in Next.js, a popular React-based web framework, that allows you to fetch data at build time and pass it as props to your React components. It's typically used when you have data that doesn't change frequently and you want to pre-render your pages to improve performance.
   2. When you use getStaticProps, Next.js will generate a static HTML version of your page with the data you fetched, and serve that HTML to users when they request the page. This means that the page will load faster and be more SEO-friendly, since search engines can index the static HTML.
   3. You should use getStaticProps when:
      1. You have data that doesn't change frequently
      2. You want to pre-render your pages for performance and SEO benefits
      3. You're building a static site or a site with dynamic content that can be pre-rendered
   4. You should also use getStaticPaths
      1. specify dynamic routes that should be pre-rendered at build time using
      2. When you use dynamic routes in Next.js, you may want to pre-render all possible variations of the dynamic route ahead of time, instead of waiting until the user requests the page. This can improve performance and ensure that all variations of the dynamic route are available for search engines to index.
   5. Write all necessary jsx for UI
10. Create context folder in ecommerce for state management
    1. Create file StateContext
    2. Import createContext from react
    3. Initialize like Context = createContext()
    4. Wrap return{} with <Context.Provider>
    5. Pass all the usestate to value ={{}}
    6. Import toaster, stateContext in \_app.js
    7. Wrap the <layout> with <stateContext>
    8. Include <toaster> inside layout.
11. In context -> statecontext -> write onAdd and toggleCartItem Quantity
    1. onAdd to increase or decrease the quantity of product that are added on each product before added to the cart
    2. toggleCartIItemQuantity to specifically increase or decrease the quantity of the product in the cart component
12. Stripe
    1. Get secret key and public key from stripe and paste in .env file
    2. Create new file (stripe.js) in pages -> api
    3. Import and inititalize stripe while passing secret key from .env
    4. Docs -> click -> application payment -> next.js -> copy demo code -> paste in stripe.jsx
    5. Shipping rates -> create free and fast shipping rates copy ids in pasted code in stripe.(refer the project)
    6. Create new file in library(getStripe), which create promise -> import in cart.js -> create response