NEXT JS Basics

brain dump and active recall

server side rendering sets up all the content before the client even loads the website

getStaticProps is a function that allows you to manage data that'll be server side rendered

sanity.io is a content manager

project structure

any folder with a route.js or page.js file is routable, and only the content returned by the route.js or page.js will be publically accessible.

prefixing a folder name with an underscore will prevent the folders from being publically accessible, even with a route.js or page.js in them. these are called private folders, you may or may ot necessarily use them

module path aliases are the GOAT cause you can actualy perform much cleaner imports without the whole"../../../..an-eternity-later", and simply do a "@/components/component-name.js"

there are a couple of ways to organize your code in the project directory

1. you organize your components and lib folders outside of the app directory and the app directory has route-able folders only
2. you organize your components and lib folders inside the app directory as top level folders and then make route-able folders within the app directory as well

the guy i'm learning from prefers option 2, so i'm just gonna go w that.

some core concepts

1. **Static Rendering:** static rendering is compiling and building the project and all the code is rendered at build time. this can then be cached and provided to the user via a cdn, reducing latency. but keep in mind, this is un-changing, static content
2. okay so the data being fetced from fetch requests will be cached by default unless you specifically opt-out. you can however use the revalidate option to re-render the data statically during revalidation. for e.g., you can set a revalidate fetch request every 60s , or something like that.
3. **Dynamic Rendering:** During static rendering, if there's a dynamic function or a dynamic fetch request, all the content from the route will be switched to being rendered dynamically at request time (as opposed to build time, where you statically render). any cached data can still be reused during dynamic rendering

   ▼ more on this

   This statement means that even when Next.js switches to dynamic rendering (rendering content at request time rather than build time), it can still utilize previously cached data to improve performance.

   To explain further:

   - In Next.js, static rendering happens at build time, creating pre-rendered HTML that can be cached and served quickly.
   - When dynamic functions are encountered, the rendering process switches to dynamic rendering, which generates content at the time of the user's request.
   - However, this doesn't mean all data needs to be fetched fresh. Any data that was previously cached from earlier fetch requests can still be reused during this dynamic rendering process, improving efficiency.
   - This gives you the best of both worlds - dynamic content generation when needed, while still leveraging cached data where possible to maintain performance.

first two are default behaviours the third can be opted-in