بسم الله الرحمن الرحیم

**Nextjs**

**What is Nextjs ?**

**it is a framework that build on top of the react that gives you flexibility of building scalable apps by allowing you to render content on the server.**

در ری اکت همانطور که می دانیم تمامی محتوای برنامه سمت client ، رندر می شود اما درNext میتوانیم سمت سرور رندر کنیم.

**مزایای Next js :**

1. حالت‌های مختلف رندر شدن :

* Static site generation : سمت client
* Server side rendering (SSR) : سمت سرور
* Incremental site Regeneration : ترکیبی از کلاینت و سرور.

2 – Performance :

* Code Spliting : برنامه ی ما را به chunk های کوچکتری تقسیم می کند پس ما فقط chunk های مورد نظرمان را Load می کنیم.
* Minifying Files: در هنگام لود شدن حجم فایل ها را کم می کند که در ری اکت به سادگی قابل انجام شدن نیست.
* Image Optimizing: نکست یک کامپوننتی دارد به نام Next Image که این برای رندر کردن تصاویر است که به صورت ریسپانسیو عمل می کند.
* Pre-Fetching assets: در هنگام اسکرول کردن به پایین اطلاعات شروع به fetch کردن می شود و به صورت یکجا fetch نمی شود.

3 – File Based Routing : Routing در نکست بر اساس فایل ها خواهد بود برخلاف ری اکت که باید از ابزار React-router-dom استفاده می کردیم.

4 – ابزار های بسیار خوب برای SEO که باعث بالا امدن رنک صفحات در گوگل است.

5 – Serverless functoins :‌فایل های داخل دایرکتوری api ، بدون نیاز به هیچ کانفیگی در بک اند اجرا می شود. فرض کنیم که یه فایل hello.js در داخل این دایرکتوری موجود است زمانی که سرور شروع به کار می کند این فایل را هم اجرا می کند و زمانی که اجرایش تمام شود دوباره سرور shut down می کند. پس می توانیم بدون اینکه دانشی از بک اند داشته باشیم کدی را بنویسیم که با اجرای سرور اجرا شود. به این معنا هم است که نیازی نیست که سرور همیشه روشن باشد هر وقت خودش بخواهد اجرا شود سرور را روشن می کند و اجرا می شود.

با دستور npx-create-next-app یک پروژه ی next ساخته می شود و با دستور npm run dev این پروپه اجرا می شود.

در داخل فولدر pages اگر هر فایلی را قرار بدهیم به صورت خودکار برایش یک route درست می‌شود.

فایل ابتدایی که با بالا امدن سرور ان رندر می شود فایل index.js است.

اگر کامپوننتی داشته باشیم که بخواهیم در تمامی صفخات رندر شود باید ان را در فایل \_app.js قراردهیم . مثل فوتر و هدر

Global.css : کانفیگ های کلی که روی کل فایل های css و پروژه اعمال می شود و میدانیم که فایل های css باید به صورت ماژولار باشد.

نکست یک قابلیتی دارد به نام fast refresh که با کوچکترین تغییر در فایل های کد سریع سایت اپدیت می شود.

یک نکته ی css : میتوانیم در css هم یک فیچر را به صورت var مقدار دهی کنیم به این صورت:

Color : var(--textColor). We should use – before name.

مقدار دهی این متغیر ها را هم می توانیم در جایی بگذاریم که در همه جا اعمال شود که دقیقا فایل app.js است. در این فایل فایل css ای را import کردیم به نام global.css پس در این فایل global.css این متغیر ها را مقدار دهی می کنیم.

نکست یک تگ Head دارد که در ان می توانیم به صورت مستقیم title صفحه را عوض کنیم و سختی های javascript و react را ندارد و دیگر نیازی نیست که به فایل index.html برویم و انجا را تغییر دهیم. پس این تگ <Head></Head> دقیقا معادل همان تگ های head در index.html است که کار ما را راحت تر کرده است.

در نکست Routing به سه دسته تقسیم می شود. 1- index routes 2 – nested Routes 3 – dynamic Routes .

IndexRoot : دقیقا همین Routing ای که با قرار دادن فایل ها در پوشه pages اعمال می شود. چون با اجرا شدن سرور در ابتدا فایل index.js بالا می اید به این روش index routing می گویند که نام default route هم دارد.

Nested Route : این هم شبیه بالایی است. تفاوتش را متوجه نشدم !!!!!

Dunamic route : Route هایی که با براکت معرفی می شوند که مثلا داخل براکت product\_id قرار می دهند

هر فایلی که در دایرکتوری pages می گذاریم برای تبدیل شدن به Route باید دو شرط را ارضا کند.

1 – Page Needs to be a React Component

2 – Component needs to be exported by default.

برای اینکه Routing ما به صورت index انجام شود حتما باید به اسم فایل توجه داشته باشیم چرا که نام فایل دقیقا به عنوان pathname در location ما حضور خواهد داشت. مثلا اگر اسم فایل باشد MohammadReza ان وقت با اجرا کردن پروژه به این صورت صفحه MohammadReza را می بینیم.

Localhost:3000/MohammadReza

یعنی دقیقا نام فایل را به عنوان pathName داریم.

Dynamic Routing : به این صورت:

Localhost:3000/stores/[id]

نکته مهمممم:

اگر در داخل pages یک پوشه دیگری قرار دادیم برای اینکه فایل js داخل این پوشه اجرا شود باید default Route باشد یعنی اینکه مثلا اگر اسم فایل store.js است باید ان را به index.js تبدیل کنیم چون که index.js ، default Route خواهد بود.

Dynamic Routing : اگر داخل pages یک دایرکتوری جدید با هر اسمی مثلا coffee-stores بسازیم و داخل این دایرکتوری از داینامیک روتینگ استفاده کنیم (منظور از داینامیک روتینگ این است که در داخل ان دایرکتوری یک فایل js بسازیم که اسم آن به صورت [name] باشد) روند path سایت به این صورت خواهد شد.

Localhost:3000/coffee-stores/Golestan

یعنی اسم دایرکتوری به عنوان ثابت کار می اید و name می تواند هر اسمی باشد.

حالا اگر بخواهم name موجود در url را در دست داشته باشم و استفاده کنم باید از hook استفاده کنم. Next js یک hook دارد به نام useRouter که این از next/router ، import می شود.

Import {useRouter} from “next/router”

Const Router = useRouter();

Console.log(Router);

Localhost:3000/coffee-store/bestCoffee

یک ابجکتی را برمی گرداند که در فیلد query یک ابجکت دیگری داریم که به صورت زیر است

query:{id : “bestCoffee”}

که کلمه ی bestCoffee دقیقا url ما خواهد بود و id دقیقا نام فایل جاوا اسکریپت ما خواهد بود و اگر نام فایل را تغییر دهیم این هم قطعا تغییر می کند (چون اسم فایل [id].js بود.)

در HTML برای رفتن به صفحه ای از تگ a استفاده می کردیم که این کار باعث می شد که زمانی که روی آن دکمه کلیک کنیم کلا صفحه رفرش می شد اما در اینجا همانند ری اکت می توانیم از Link استفاده کنیم با این تفاوت که اولا از next/link باید import شود ثانیا دیگر to:{pathname:”path”}} ندارد بلکه href دارد.

از تگ لینک تنها برای جابه جا شدن در داخل خود Application استفاده می کنیم یعنی مثلا وسط کار خواستیم بریم به سایت مثلا ورزش 3 لزومی نداره که از تگ link استفاده کنیم و از a استفاده کنیم بهتره.

Next/link ، property های بینظیر دیگه ای هم داره که خیلییی بدرد می خوره مثلا prefetch و scroll و ...

برای استفاده از عکس حتما باید از کامپوونت Image داخل خود next استفاده کرد.

Import Image from “next/image”

که این تگ Image ، سورس داره که این سورس به صورت پیش فرض از public است یعنی اگر فایل های عکس داخل public باشد دیگه در داخل src نیازی نیست بزنیم src = “public/…..” یعنی فرض بر اینه که توی public هست.

افزودن فونت به پروژه:

برای این کار ابتدا باید به سایت google font برویم و در انجا فونت مورد نظر خود را دانلود کنیم و زمانی که دانلود کردیم فایلی به صورت زیپ دریافت می کنیم و باید ان فایل زیپ را به پوشه public پروژه اضافه کنیم و سپس باید ان را در همان جا extract کنیم و فونت هایی که می خواهیم را در پوشه ای به نام fonts نگه داریم. حال میخواهیم فونت ها را در همه جای پروژه داشته باشیم به همین دلیل از next js document باید استفاده کنیم. در سایت nextjs/advanced-features به قسمت custom-document می رویم که در این قسمت یک قطعه کدی داریم ان را کپی میکنیم و در پوشه ی pages فایلی به نام \_document.js ایجاد می کنیم و انجا paste می کنیم و در داخل تگ هدر باید فونت ها را قرار بدهیم به صورت زیر و در داخل global.css در ویژگی font-family قبل از هر فونتی که انجا ذکر شده است فونت IBMPlexSans را بنویسیم به این معنا که اگر این فونت ها نبودند به مابقی فونت ها بپرداز.

کدی که باید در داخل تگ هدر بزنیم:

<Head>

<link rel=”preload”

href=”/fonts/IBMPlexSana-Bold.ttf”

as = “font”

crossOrigin=”anonymous”>

</link>

</Head>

در داخل تگ هدر هر چدر فونت خواستیم می توانیم قرار بدهیم مثلا اینجا من برای Bold قرار دارم می توانیم علاوه بر این مثلا semi-Bold و Reqular هم قرار داد.

برای تست کردن اینکه ایا در پروژه فونت ها اعمال شده است یا نه باید در تب network ببینیم که ریکوئست آن وجود دارد یا نه.

Seo در Nextjs :

زمانی که ما چیزی را در گوگل سرچ می کنیم گوگل سه کار انجام می دهد.

* Crawling
* Inexing
* Ranking

در مرحله ی اول گوگل تمام صفحات وب را یکبار بررسی می کند تا بررسی کند محتوی سرچ شده در چه سایتی است، keywords ، descriptions و .... را سرچ می کند.

در مرحله ی دوم گوگل تمامی اطلاعات سایت های مرتبط پیدا شده را انالیز می کند (indexing ) و اطلاعات بدست آمده را در یک data base خیلی بزرگی به نام google index ذخیره می کند.

در مرحله آخر گوگل بر مبنای محتوی سرچ شده و اطلاعات بدست آمده توسط خودش محتوی سایت ها را رتبه بندی می کند تا بتواند اطلاعات سودمندتری را به شما بدهد

موارد مهم برای SEO :

* Semantics : تگ هایی مثل <a> ، <p> ، <h> و ...
* Meta tags : title ، description و ..
* Image : alt tag

سه بحث مهم در SEO وجود دارد به نام های click through rate و Bounce rate و Dwell Time

مورد اول:

شاخصی است که نشان میدهد که کاربر ها چقدر کلیک کردن تا به سایت شما وارد شوند. هر چقدر این مورد بیشتر باشد بهتر است.

مورد دوم : یعنی کاربر به سایت شما وارد می شود و بدون اینکه چرخی در سایت شما بزند از ان خارج می شود. هر چقدر این مورد کمتر باشد بهتر است.

مورد اخر هم یعنی اینکه کاربر چقدر زمان را در سایت شما سپری می کند. و هر چقدر که بیشتر باشد بهتر است.

موضوع مهم بعدی که در Next js وجود دارد و در ری اکت وجود ندارد و وجه تمایز بسیار خوبی است در مقایسه با ری اکت موضوع Pre-rendering است.

در ابتدا کاربر یک ریکوئست به سرور ما می دهد و سرور ما یک فایل static HTML را ارسال می‌کند. مرور گر ان فایل را دریافت میکند و فایل JS آن را فقط دانلود می کند و رندر نمی کند و فقط فایل HTML را به کاربر نمایش می‌دهد یعنی کاربر هیج interaction ای با سایت نمی تواند داشته باشد چون js نداریم. در مرحله ی بعد آن js لود می شود و در مرحله ی بعد کاربر می تواند تعاملش را با سرور انجام دهد. اما در ری اکت زمانی که ریکوئست به سمت سرور زده می شود و سرور فایل js و html را ارسال می کند اما هردو فایل فقط دانلود می شود و رندر نمیشود یعنی کاربر یک صفحه ی خالی می بیند. در مرحله ی بعد کد js لود می شود و این کد js کد HTML را رندر می کند یعنی در اخرین مرحله کد HTML را مشاهده می کنیم. خب این نشان میدهد که زمان بیشتری را سپری می کند تا کاربر بتواند سایت را ببیند.

اگر یک سایتی که با next زده شده را بالا بیاریم در view page source مشاهده می کنیم که بعضا کد html وجود دارد (با توجه به پیاده سازی پروژه) یعنی کد html قابل رویت است چون که ابتدا در next js کد html رندر میشود اما js ندارد ولی وقتی همان کد را با react بزنیم می بینیم که در page source فقط یک div با ایدی root داریم و دیگه کد html نداریم و ان چیزی که ما مشاهده می کنیم نتیجه ی لود شدن js است چون ابتدا js لود می شود و سپس html رندر می شود. در SEO چون باید content را داشته باشیم در نکست چون کد html را داریم پس سئو بهتری داریم بر خلاف ری اکت.

متد های رندر شدن در نکست (data fetching):

* Static Generation (SSG)
* Incremental site regeneration(ISR)
* Server side Rendering (SSR)

SSG : فرایندی است که در آن کامپایل کردن و رندر کردن برنامه در زمان biuld اتفاق می افتد و خروجی ما شامل فایل های استاتیک HTML و js است. یعنی در همان لود اولیه کل content را داشته باشیم.

ISR : این روش از مزیت static استفاده میکند و زمانی که کاربر ریکوئست میزند از کش نسخه‌ی قبلی سایت را نمایش می دهد و بعد از آن یک سری تغییرات را اعمال می کند یعنی کاربر در ابتدا نسخه قدیمی سایت را می بیند اگر تغییری داشته باشد خب سایت اپدیت می شود اگر هم نداشته باشد که هیچی

SSR : تمام rendering فایل ها در سمت سرور انجام میشود یعنی مرورگر کاری نمی کند درخواست را به سرور می دهد و سرور کل content را رندر می کند و به مرورگر میدهد که این از لحاظ سئو و سرعت نسبت به client side rendering بهتر است.

در SSG ، performance و load صفخات ما خیلی خوب است. در این روش از پروژ بیلد گرفته می‌شود و برای کاربر هایی که فاصله ی دوری تا سرور دارند چند cdn قرار می دهد و بیلد پروژه را در این cdn ها قرار می دهد و کاربر از این cdn ها دریافت میکند که باعث افزایش performance و سرعت لود می شود.

در این روش اگر data ی خارجی نداشته باشیم (مثلا از رابطی چیزی نگیریم) ، pre-render میشود وفایل HTML تحویل کاربر داده می شود مثل about us . اما اگر دیتای خارجی داشته باشیم ابتدا دیتا از رابط ها گرفته می شود و سپس رندر می شود. مثل product page

ISR : در این روش سایت بعد از یک مدتی اپدیت می شود یعنی به این صورت که در ریکوئست اول stale data will render اما در ریکوئست دوم fresh data will render .

SSR : در این روش به ازای هر ریکوئست یک HTML جدید رندر می شود و از قبل هم چیزی pre-render نشده. Performance در این روش نسبت به دو روش بالاتر کمتر است و همچنین امکان استفاده از cdn وجود ندارد چون به ازای هر ریکوئست یک html جدید داریم و نیازی به ذخیره نداریم. مثل سایت های خبری که نیاز به رندر شدن جدید دارد.

نکته ی css : اگر بخواهیم به یک المان دو کلاس بدهیم باید از یک پکیجی استفاده کنیم به نام classnames که با دستور npm install classnames نصب می شود.

در پروژه ای که ما داریم کار می کنیم تمامی عکس ها از دامین unsplash.cm است و نمیتوانیم این imgUrl ها را به درستی به src های <Image /> بدهیم به همین دلیل در فایل next.config.js دو خط کد زیر را وارد می کنیم به جای خط کد موجود

module.exports = {

images:{  
 domains = [“images.unsplash.com”],

},

};

پیاده سازی به روش های rendering :

SSG : اگر از تابع getStaticProps در هر صفحه ای استفاده کنیم ان صفحه در زمان ساخت pre-render می شود.

export async function getStaticProps(context) {

return {

props: {}, // will be passed to the page component as props

}

}

و این props ای که return می شود به عنوان ورودی کامپوننت می توانیم به ان دسترسی داشته باشیم و این تابعper page است یعنی به ازای هر page از این تابع استفاده می کنیم.

مورد بالا per-page بود اما اگر از dynamic route استفاده کنیم چی؟؟

برای این کار باید ازgetStaticPath استفاده کنیم.(این مورد هم برای page است نه کامپوننت)

شروط استفاده از getStaticPath : 1 – page باشد 2 – از getStaticProps استفاده کرده باشیم.

export async function getStaticPaths() {

return {

paths: [

{ params: { ... } }

],

fallback: true // false or 'blocking'

};

}

این مورد حتما باید با getStaticProps استفاده شود و اصلا نباید از getServerSideProps استفاده شود.

در داخل props موجود در ورودی تابع getStaticProps یک فیلدی هست به نام params که از ان برای استفاده و استخراج کردن محتویات داخل url استفاده می شود.

export async function getStaticProps(props) {

const params = props.params;

// params contain url properties for example => params.id return id that // it is showed in url

return {

props: {}, // will be passed to the page component as props

}

}

در تابع getStaticPaths اگر فیلد fallBack برابر با false باشد یعنی اینکه اگر هر صفحه ی دیگری که در فیلد paths نباشد وارد url شود صفحه ی 404 را می دهد.

در فیلد paths:[] دقیقا باید ابجکت هایی باشد که اسم params داشته باشند که هر params باید فیلد id:”id\_Number” فیلد id حتما باید string باشد