Taaghche Homework Assignment Document

پروژه مصاحبه استخدامی طاقچه: جزییات در front end home work
فهرست:

تکنولوژی های استفاده شده

در این پروژه از Typescript, Next js, Redux, Tailwind Css, next-pwa استفاده شده است.

دلیل انتخاب typescript: در پروژه هایی که قابلیت scale شدن دارند بهترین انتخاب تایپ اسکریپت هست چرا که جلوی باگ های کوچک را خود به خود میگیرد و اجازه فرستادن پروژه ای که احتمال باگ های کوچک دارد را به پروداکشن نمیدهد. و در مرحله های بعدی develop به دلیل وجود تایپ ها مشکلی در یادآوری آنها نخواهیم داشت.

دلیل انتخاب Next js: در پروژه هایی که سامانه بسته نیستند و یا از نظر بیزنس بخشی از لید های خود را از طریق گوگل میگیرند مشخصا نیاز به SEO داریم. و ما میدانیم که react به دلیل کلاینت ساید بودن به ما seo خوب نمیدهد. در نتیجه ما نیازمند SSR هستیم. و یکی از انتخاب های خوب ما فریم ورک Next است.

دلیل انتخاب Redux: در پروژه های بزرگ و یا Scalable ما نیازمندی تمیز و یکجا بودن لایه دیتای خود را احساس میکنیم و معماری Flux این امر را برای ما بسیار راحت کرده است. یکی از لایبرری هایی که به ما راحتی اجرای فلاکس را ارائه میدهد ریداکس است. همچنین داشتن کامیونیتی بزرگ دلیل دوم انتخاب این تکنولوژی است.

دلیل انتخاب Tailwind Css: مشکل reusable و maintainable نبودن کلاس ها و همچنین لود بیش از حد css توسط مرورگر ها مشکلی است که باعث سردرد برنامه نویسان وب شده. Tailwind با داکیومنت قوی خود و همچنین قابلیت بیلد شدن کلاس هایی که در کد استفاده شدند به ما اجازه scale و همچنین گرفتن performance بهتر را میدهد.

دلیل انتخاب next-pwa: در دنیای امروز pwa بودن یک وب اپ تقریبا نیازی اساسی است. Next-pwa در این امر با قابلیت های caching و راحتی استفاده از آن به ما یاری میکند.

فولدر بندی و نام گذاری ها

میدانیم که یک شکل بودن فولدر بندیها در امر maintain کردن یک code base به ما کمک میکند. در این پروژه علاوه بر typescript template که خود nextjs به ما ارائه میدهد سوای پیج های ما یک فولد Core داریم که کامپوننت ها, ریداکس, و utility های ما (hooks and etc.) در آن وجود دارد.
این فولدر شامل یک فایل Layout.tsx هست که در واقع layout مشترک صفحه های ماست.

در پوشه Components ما فولدر های component های ما وجود دارند و هر پوشه شامل index.tsx یعنی فایل اصلی خود کامپوننت و کامپوننت های زیرمجموعه خود با استراکچر نام خود کامپوننت برای فولدر و index.tsx فایل اصلی خود کامپوننت.
در پوشه redux ما یک پوشه Actions و یک پوشه Reducers وجود دارد. و فایل های constants برای baseUrl ما, store.ts برای استور ریداکس ما و dataMager.ts برای dataManager ما.
توضیحات مورد نیاز datamanager : کلاس Datamanager یک کلاس کاستوم هست که متد های کراد را در خود داراست. یک متد چک وجود دارد که کار اصلی آن فچ کردن دیتای مورد نظر است و پس از آن Route انتخابی برای فتچ را به عنوان تایپ action و دیتای ریسپانس را به عنوان payload به reducer dispatch میکند.
در فولدر Actions, فانکشن های dispatcher ما برای هر بخش از ریداکس ما وجود دارد. به طور مثال Actions/Products فقط دیسپچر های مربوط به محصولات را داراست. و مانند بقیه بخش های کدبیس ما هم نام بخش روی فولدر و فایل آن index.ts هست.
در فولدر Reducers هم مانند Actions نام بخش ها روی فولدر های درونی هست که هر فولدر شامل یک index.ts است.
نام گذاری آبجکت های action به این صورت است: در فولدر Action/(Part) به اینصورت میباشد:
[Part]Actions

و نام گذاری فانکشن های هر ریدیوسر به این صورت است:

Folder: Reducers/[Part]
Name of function: [Part]State

هر فولدر Actions و reducers خود شامل یک فایل index.ts است که در واقع ایندکس تمامی بخش های آن به اینصورت است:

Export {default as [Part]Actions/State} from ‘./[Part]’

همچنین هر فولدر دارای یک فایل برای تایپ های مورد نیاز میباشد.
تایپ های مورد نیاز برای reducer های ما در بقیه قسمت های اپلیکیشن میتواند استفاده شود.

فولدر Utils: دارای Custom Hook ها و فانکشن های محاسباتی مورد نیاز ماست

توضیح معماری انتخابی

معماری را ما به شکل یک سناریوی کلی برای دیتا درنظر میگیریم. در ابتدای امر برای هر صفحه ای که نیاز به seo داریم دیتا را از api به صورت getServerSideProps گرفته و در ریداکس server side خود dispatch میکنیم. کامپوننت های مورد نیاز که به آن بخش از state ما متصل هستند داده مورد نظر خود را در Server Side دریافت کرده و رندر میکنند. این امر باعث میشود تا ما فقط برای صفحه خود layout را نداشته باشیم و بخشی از داده را با استفاده از اصول seo رندر کرده و داشته باشیم. اما برای بقیه لود شدن دیتا و بخش های مختلف ما از client side استفاده میکنیم. چرا که همان دیتا برای گوگل کافیست و از طرفی هم ما در نظر بگیریم پلتفرم ما در حال حاضر تعداد user های live زیادی دارد و یا در آینده خواهد داشت, گذاشتن بار اضافی روی سرور نیاز نیست و مرورگر کاربر میتواند بار اکشن های کاربر را به دوش بکشد. پس برای این امر ما نیاز داریم تا ریداکس server side خود را با ریداکس client side خود merge کنیم. میتوانیم از hydration استفاده کرده و این امر را انجام دهیم. لایبرری next-redux-wrapper این کار را برای ما به شدت آسان کرده و برای این کار از این لایبرری استفاده میکنیم. حال که لایه ی داده ما از لایه ی لیوت ما جداست میتوانیم فیلتر های مختلف و یا عوض کردن datamodel را در reducer های خود انجام دهیم. درواقع کل بخش محاسبات و داده های ما در redux انجام میشود.

برنچ ها

برنچ main ما در ابتدا برای پوش کردن تمپلیت و استراکچر اولیه استفاده میشود در ادامه هر فیچر ما یک برنچ میشود اما به این صورت: هر فیچری که وابسته به فیچر دیگری میشود, branch آن از branch فیچر اصلی ساخته میشود و بعد از انجام شدن فیچر با branch مادر خود merge میشود در آخر پس از اینکه branch اصلی دارای همه فیچر های لازم آن ورژن است با برنچ QA مرج میشود تا تست ها گرفته شوند و باگ ها فیکس شوند یا پرفورمنس آن افزایش یابد. پس از اتمام کنترل های کیفی و تغییرات لازمه برنچ QA با Main مرج میشود تا همیشه ورژن stable در برنچ main حضور داشته باشد. حال میتوان از CI/CD استفاده کرد تا برنچ main ما درواقع برنچ پروداکشن ما باشد.

روش QA

در برنچ QA زمانی که هر فیچر تمام میشود ما آن فیچر را QA میکنیم. گوریلا تست میگیریم و هر مشکلی که دیده شد ابتدا فیکس میشوند. سپسerror ها و warning های موجود در کنسول browser را فیکس کرده و بعد warning ها و error های کنسول ترمینال را فیکس میکنیم. پس از آن برای مطمئن شدن اینکه اصول seo رعایت شدند و performance وب اپ خوب است تست lighthouse گرفته میشود و مشکلات آن هم فیکس میشوند.
(در این مرحله میتوانیم تست های جاواسکریپتی با jest و یونیت تست های متخلف را هم بگیریم به دلیل کمبود وقت انجام نشدند.)