Public route: به روت هایی گفته میشود که برای همه ی کاربران قابل دسترسی هستند.

Private route: به روت هایی گفته میشود که برای کاربرانی با شرط خاص قابل نمایش است و اگر کاربر ان شرط را نداشته باشد redirect میشود به یک صفحه ی دیگر.به این صورت عمل میکنیم که ابتدا یک تابع برای بررسی یک شرط مثلا لاگین بودن کاربر یا ادمین بودن کاربر مینویسیم و با شرط های تک خطی بررسی میکنیم اگر مقدار true از تابع برگشت محتوا را نشان میدهیم و در غیر اینصورت با استفاده از Navigate کاربر را به صفحه ی دیگری میفرستیم یا پیامی را به کاربر نشان میدهیم.

روش بهینه تر این است که یک کامپوننت با اسم دلخواه مثلا PrivateRoute بسازیم که در prop هایش children را دریافت کنیم که میشود همان محتوای بین کامپوننت تگ حالا همان شرط لاگین بودن را بررسی میکنیم و اگر true بود محتوای children را نمایش میدهیم و در غیر اینصورت redirect میکنیم حالا در فایلی که تمام روت ها را نوشتیم هر روتی که خواستیم Private باشد را بین تگ های باز و بسته ی PrivateRoute مینویسیم.

روش بهتر این است که کامپوننت privateRoute را به عنوان path \* بنویسیم و تمام روت های private را به صورت subRoute در children براش تعریف کنیم.در کامپوننت PrivateRoute هم به جای محتوای children باید Outlet را نمایش بدیم.

useNavigate: برای شرایطی که بخواهیم در داخل تابع با استفاده از شرط ها کاربر را redirect کنیم میتوانیم از این هوک استفاده کنیم.ایمپورت از react-router-dom یک متغیر با استفاده از این هوک تعریف میکنیم و سپس هر جا خواستیم روت مورد نظر را به آن میدهیم تا redirect کند.اگر ورودی را -1 بذاریم به صفحه قبل از خودش هر چی که باشد برمیگردد.هر عدد منفی که بذاریم به همان تعداد به روت قبلی میرود.ورودی دوم هم به صورت یک ابجکت میتواند تنظیماتی بگیرد مثلا اگر replace: true بذاریم وقتی ریدایرکت میوشد از history مرورگر هم حذف میشود.

useLocation: برای دسترسی به اطلاعات لوکیشن مانند pathname, search, hash و غیره بکار میرود.

Jsx format: در فایل هایی که کد jsx مینویسیم بهتر است از فرمت jsx استفاده کنیم مثلا test.jsx

Recharts: برای استفاده از نمودارها در react استفاده میشود که نمودار های متنوعی دارد. npm install recharts

Custome hook: بعضی وقتا توی پروژه نیاز به هوک هایی داریم که در ری اکت وجود ندارد در این شرایط میتوانیم هوک اختصاصی خودمان را بنویسیم که از تکرار کد جلوگیری کنیم یک تابع ساده با نام use\_\_\_ مینویسیم و عملکرد مد نظر مثل اعتبار سنجی ورودی کاربر را انجام میدهیم و نتیجه را return میکنیم. و در کدمان مانند بقیه هوک ها استفاده میکنیم.تمام هوک ها را در یک دایرکتوری با نام hooks مینویسیم و اگر نیاز بود در فایل های jsجدا مینویسیم و ایمپورت میکنیم. هوک باید مستقیماً در سطح کامپوننت اجرا شود (نه داخل تابع یا اکشن).

useMemo:فرض کنیم در یک کامپوننت یک تابع داریم که عملیاتی سنگین مثل fetch, loop را انجام میدهد در حالت عادی در هر بار رندر شدن این تابع اجرا میشود ولی با هوک useMemo میتوانیم بررسی کنیم اگر یک state خاص تغییر کند این تابع اجرا بشود.در ورودی اول یک تابع ارسال میکنیم که میخواهیم مقدار آن حفظ شود و در ورودی دوم هم داخل یک آرایه state هایی که با تغغیر آن میخواهیم تابع دوباره اجرا شود را مشخص میکنیم اگر مشخص نکنیم تابع در هر بار رندر اجرا میشود.خروجی useMemo یک مقدار است نه یک تابع پس هنگام استفاده از متغیری که با این هوک ساختیم نباید آن را کال کنیم.در شرایطی که داریم داخل یک کامپوننت یک state را با مقدار یک input , bind میکنیم با هربار تغییر در اینپوت اگر در کد jsx یک رندر روی مثلا پست ها داشته باشیم انجام میشود که در واقع اصلا نیازی نیست و فقط باید زمانی که state posts تغییر کند رندر انجام شود در این شرایط در کد jsx از هوک useMemo استفاده میکنیم و ورودی اول کد های مربوط به رندر و ورودی دوم مقدار استیت پست ها را مینویسیم.در child component ها هم اگر نیاز باید میتوانیم از useMemo استفاده کنیم چون با هربار رندر parent ان هم رندر میشود ولی میتوانیم با ممو تععین کنیم که اگر state خاصی تغییر کرد child دوباره رندر بشود.

Memo: برای جلوگیری از ریرندر کردن کامپوننت هایی که نیاز به رندر مجدد آن ها نیست استفاده میشود یعنی خارج از کامپوننت ها تعریف میشود.عملکرد آن مثل pureComponent است.

useCallback: عملکرد آن مانند ممو است با این تفاوت که به عنوان حروجی یک تابع برمیگرداند.زمانی که بخواهیم با ممو پروپ هایی که تابع هستند را برای ریرندر بررسی کنیم ممو به تنهایی کافی نیست چون با هربار رندر app تابع به عنوان تابع جدید شناخته میشود.در این شرایط تابع را با هوک useCallback مینویسیم تا ممو به درستی کار بکند.

### **۱.** useMemo **(هوک)**

* **هدف:** بهینه‌سازی محاسبات و جلوگیری از بازساختن مقادیر (اشیاء، آرایه‌ها و…)
* **کی استفاده کنیم؟**
* وقتی تابعی داریم که یک **محاسبه سنگین** (مثل فیلتر کردن بزرگ، تبدیل داده‌های پیچیده) انجام می‌دهد و نمی‌خواهیم در هر رندر اجرا شود.
* وقتی **یک شیء یا آرایه** را به عنوان prop به یک کامپوننت memo شده می‌فرستیم و نمی‌خواهیم در هر رندر reference جدیدی از آن ساخته شود (چون باعث رندر مجدد کامپوننت memo شده می‌شود).
* **نتیجه:** بازگرداندن یک **مقدار کش شده** از محاسبه.

### **۲.** useCallback **(هوک)**

* **هدف:** بهینه‌سازی توابع و جلوگیری از بازساختن reference توابع.
* **کی استفاده کنیم؟**
* وقتی **یک تابع** را به عنوان prop به یک کامپوننت memo شده می‌فرستیم. این کار باعث می‌شود کامپوننت فرزند بدون نیاز به رندر مجدد، از reference ثابت تابع استفاده کند.
* در مواقعی که فانکشن به عنوان dependency در useEffect یا useMemo دیگری استفاده می‌شود و می‌خواهیم از infinite loop یا رندرهای غیرضروری جلوگیری کنیم.
* **نتیجه:** بازگرداندن یک **تابع کش شده**.

### **۳.** memo **(کامپوننت HOC)**

* **هدف:** بهینه‌سازی رندر کامپوننت‌ها.
* **کی استفاده کنیم؟**
* برای **کامپوننت‌های تابعی** که می‌دانیم خروجی آن‌ها (UI) تنها در صورتی تغییر می‌کند که props آن‌ها تغییر کند.
* برای کامپوننت‌هایی که احتمالاً در یک لیست بزرگ قرار دارند و مرتباً رندر می‌شوند، اما props آن‌ها ثابت باقی می‌ماند.
* **نتیجه:** یک **کامپوننت رپ شده** که تنها در صورت تغییر props یا state داخلی‌اش رندر مجدد می‌شود.

**useRef:** 1-میتوان از ان برای ذخیره اطلاعات کامپوننت استفاده کرد ولی فرق آن با state این است که با تغییر آن رندر مجدد انجام نمیشود.یک متغیر میسازیم و مفدار اولیه را براش مشخص میکنیم.

2-برای دسترسی به المنت های داخل dom میتوانیم از useRef استفاده کنیم.به این صورت که یک متغیر با استفاده از این هوک میسازیم و سپس در هر المنت که بخواهیم به آن اشاره داشته باشد.در آن المنت از prop , ref استفاده میکنیم.یک ابجکت برمیگرداند که با .current به خود المنت دسترسی داریم.

نکته: setState در کامپوننت های تابعی یا کلاسی به صورت async عمل میکند.یعنی اگر یک تابع بنویسیم و بخواهیم که بعد از setState کردن کاری بکنیم نباید دقیقا بعد از همانsetState بنویسیم بلکه باید یک useEffect بنویسیم که با تغییر همان state مدنظر اجرا بشود.

Pure func in js: تابعی که با یک ورودی ثابت همیشه خروجی یکسانی را نمایش دهد همچنین برای محاسبه خروجی از ورودی خودش و متغیر های لوکال خودش استفاده کند را تابع خالص یا pure میگوییم.در واقع این توابع نباید side effect داشته باشند و چیزی خارج از خودشون رو تغییر بدن.به توابعی که هر یک از این شرایط را نداشته باشد impure func میگوییم.

Higher order functions : تابعی که یک تابع کال بک به عنوان ورودی بگیرد یا یک تابع را به عنوان خروجی برگرداند را تابع hof میگویند.

Higher order component: به کامپوننت های گفته میشود که در ورودی خودش یک کامپوننت دیگر را دریافت کنند و خروجی یک کامپوننت جدید را برمیگرداند که به ان prop و متود هایی اضافه کنیم که در چندین کامپوننت مشترک باشد و حتما باید pure باشند.معمولا با نام With آن ها را نام گذاری میکنیم.

useLayoutEffect:

profiler: برای بررسی کامپوننت ها از نظر performance و فهمیدن اینکه چه زمانی رندر ریرندر میشود استفاده میکنیم.به این صورت که کامپوننت یا تگ html مد نظر را داخل تگ باز و بسته ی Profiler قرار میدهیم.دو prop میگیرد که اولی id است که همان نام خود کامپوننتی که قصد بررسی آن را داریم میگذاریم از طریق آن متوجه میشویم که هر profiler مربوط به کدام کامپوننت است و دومی onRender که چندین ورودی میگیرد که میتوانیم از آن ها استفاده کنیم.

۱. **id:** اسم بخش پروفایل‌شده

۲. **phase:** نوع رندر (mount یا update)

۳. **actualDuration:** مدت زمان واقعی صرف‌شده برای همین رندر

۴. **baseDuration:** زمان لازم برای رندر کل subtree بدون کش

۵. **startTime:** زمان شروع رندر subtree

۶. **commitTime:** زمان ثبت تغییرات رندر در DOM

۷. **interactions:** مجموعه تعاملات (event, async) که باعث این رندر شدن

Profiler in devtools: در react devtools یک تب برای profiler وجود دارد که قابلیت هایی مانند نشان دادن هر بخشی که هنگام تغییر آپدیت میشود و همچنین نشان دادن دلیل آپدیت شدن هر کامپوننت را دقیق نمایش میدهد که این بخش ها در قسمت تنظیمات در دسترس هستند.

Uuid: نحوه ی نصب npm install uuid برای ایجاد ایدی یکتا استفاده میشود به این صورت که از uuid v4 رو ایمپورت میکنیم و هر جا نیاز به آیدی داشتیم استفاده میکنیم. Import {v4 as uuidv4} from uuid

useId: برای ساخت آیدی منحصربفرد از این هوک میتوانیم استفاده کنیم.ولی برای استفاده به عنوان key درmap زدن مناسب نیست و مقدار باید از خود داده ها برداشته شود.

Context: زمانی که بخواهیم اطلاعاتی را به یک کامپوننت از یک کامپوننت دیگر که والد مستقیم ان نیست بفرستیم باید از زنجیره ای از prop ها استفاده کنیم تا اطللاعات را در بین کامپوننت ها منتقل کنیم تا به کامپوننت مقصد برسد.ولی با استفاده از context میتوانیم اطلاعات را یکبار در آن ذخیره کنیم ودر هر کامپئننت فرزند که نیاز داشتیم از ان استفاده کنیم.

createContext: در پروژه یک دایرکتوری جدا میسازیم و برای هر context یک فایل جدا میسازیم یا همرو در یک فایل میسازیم. برای ساختن context که به نوعی یک مخزن است که اطلاعات در آن قرار داده میشود از هوک createContext استفاده میکنیم و در یک متغیر ذخیره میکنیم و حتما export میکنیم تا هر جا نیاز داشتیم استفاده کنیم.در کامپوننت والد کل آن را داخل تگ باز وبسته همان context که ساختیم قرار میدهیم به صورت contextvar.Provider و با prop , value مقدار آن را مشخص میکنیم .

useContext: در هر کامپوننتی که نیاز به اطلاعات context داشتیم آن را ایمپورت میکینم و داخل هوک useContext انرا مینویسیم تا مقدار ذخیره شده ان را بدست بیاریم و در متغیری ذخیره کنیم.

نکته: Provider رو به صورت hoc مینویسیم که داخل همان فایل context تعریف بشود.