۱.مدیریت State تقریبا در هر برنامه‌ای، سخت‌ترین کار است. دو راه برای مدیریت استیت موجود. یکی استفاده از قابلیت مدیریت state به صورت built-in. که در واقع هدف اصلی Context جلوگیری از prop-drilling است. در سناریوهای متفاوت‌تر برای جهت دسترسی‌های آسان‌تر کتابخانه‌های متنوعی ساخته شدند که مدیریت را از جهاتی آسان‌تر می‌کنند. بعضی از این کتابخانه‌ها عبارتند از : Redux,MobX, Akita, Recoil,Zustand... که این همان روش دوم محسوب می‌شود.

به‌طور معمول برای انتقال استیت و مقادیر، از props استفاده و مقادیر را از بالا به پایین منتقل می‌کنیم. چه زمانی این مورد می‌تواند مشکل‌ساز شود؟ هنگامی که نیاز است تا تمام کامپوننت‌ها از یک استیت استفاده کنند. به این دلیل که ما می‌بایست از بالاترین کامپونت به پایین‌ترین، مقادیر را از راه props منتقل کنیم که کار اشتباهی‌ست!

پس Context چیست؟

یکی از ویژگی‌های ری‌اکت است که به ما این اجازه را می‌دهد که داده را بین کامپوننت‌ها تبادل و به اشتراک بگذاریم. مثلا فرض کنید یک تابع دارید که می‌توانید به آن یک محصول پاس بدهید و این تابع این محصول را به سبد خرید اضافه می‌کند و همچنین می‌تواند یک لیست محصولات را نیز برگرداند!

چه زمانی از Context استفاده کنیم؟

هنگامی که نیاز است یک استیت را همه جا استفاده کنیم. به عبارتی یک استیت سراسری (Global) باشد!

نمونه استیت‌هایی که نیاز است سراسری باشند مثل دسترسی‌ها، تم (مثلا برای حالت‌های لایت و دارک مود) و سایر.

معایب Context:

می‌دانیم زمانی که مقدار استیت و یا props تغییر کند، خود کامپوننت و فرزندانش مجدد رندر می‌شوند. پس فرض کنید یک Context داریم که بالای بالاترین کامپونت قرار دارد. حال فرض کنید که ۱۰۰۰ کامپونت داریم، مثلا یک جدول بزرگ، در این صورت ممکن است سرعت و کیفیت به شدت کاهش یابد.

پس خوب است که تا جای ممکن از Context برای استیت‌هایی که زیاد آپدیت می‌شوند استفاده نکنیم و خوب است که دقت کنیم که چه زمانی نیاز است از Context استفاده کنیم.

.۲

۱- خوانایی فانکشنال کامپوننت ها بالاتر از کلاس کامپوننت هاست (چون همون توابع ساده جاواسکرپیتی هستند)
۲- تست کردن فانکشنال کامپوننت ها ساده تر از کلاس کامپوننت هاست.

۳- کدهای نهایی ترجمه شده (و حتی سورس کدها) با فانکشنال خیلی کوتاه تر میشه.

۴- استفاده از فانکشنال کامپوننت ها به رعایت **Best Practice** ها کمک میکنه. یکی از مهمترین **Best Practice** های ری اکت جداسازی کامپوننت های نمایشی از **container** هاست. استفاده از فانکشنال کامپوننت ها به رعایت این موضوع کمک زیادی میکنه (وقتی کامپوننتی بنویسیم که داخلش **setState** نداشته باشیم)

۵- درنهایت تیم ری اکت اعلام کرده تغییرات ورژن ۱۷ مبتنی بر استفاده از فانکشنال کامپوننت ها برای افزایش کارایی و سرعت ری اکت هست و توصیه کرده تا جایی که مجبور نشدیم از کلاس کامپوننت ها استفاده نکنیم.

۳.هوک (Hook) تابع خاصی است که با استفاده از آن می‌توانیم به کامپوننت‌ها، استیت (State) و منطق اضافه کنیم. نام‌گذاری هوک‌ها با use آغاز می‌شود.

با استفاده از هوک، می‌توانید از ویژگی‌هایی که ری‌اکت دراختیارتان قرار داده استفاده کنید، مانند نگه‌داری مقادیر، انجام کارهای خاص در زمان اولین اجرا، یا انجام عملی خاص در زمانی که یک متغیر تغییر کرد. برخی از هوک‌هایی که ری‌اکت در اختیار ما قرار داده به شرح زیر می‌باشند:

- useState
- useEffect
- useRef
- Etc..

useState
این هوک برای نگه‌داشتن مقدارهای کامپوننت استفاده می‌شود.این هوک دو مقدار به ما می‌دهد. مقدار اول، خود استیت و مقدار دوم، تابع مربوطه که با دادن مقدار جدید به این تابع، استیت، این مقدار جدید را می‌گیرد.

useEffect
هوکِ **useEffect** یک تابع و یک آرایه را به عنوان ورودی می‌گیرد و هر زمان که مقادیر داخلی آرایه تغییر پیدا کنند، آن **useEffect** صدا زده می‌شود. اگر آرایه خالی باشد فقط اولین بار وقتی کامپونت رندر

می‌شود صدا زده می‌شود. اگر هم آرایه‌ای را به آن پاس ندهیم، هر موقع کامپوننت رندر یا رندر مجدد شود، آن تابع صدا زده می‌شود.

همچنین از قبل نیز می‌دانیم که زمانی که مقدار استیت تغییر می‌کند، کامپوننت مجددا رندر می‌شود.

همیشه پس از رندر `useEffect` نکته مهمی که نیاز است به آن توجه کنیم این است که شدن و نمایش داده شدن در صفحه
صدا زده می‌شود