**React Native'de useMemo ve useCallback Hook'ları**

React Native'de performans optimizasyonu yapmak, özellikle büyük ve karmaşık uygulamalarda önemlidir. useMemo ve useCallback, gereksiz bileşen yeniden render'larını önleyerek uygulamanızı optimize etmenize yardımcı olan iki önemli hook'tur.

**useMemo Nedir?**

useMemo bir memoization (ön bellekleme) tekniği sağlar. Yani, pahalı işlemler (hesaplamalar veya veri işlemleri) yapıldığında, sonucu bir kez hesaplar ve gelecekteki render işlemlerinde yeniden hesaplamaktan kaçınır. Sadece bağımlılıkları (dependencies) değiştiğinde yeniden hesaplama yapılır.

**Kullanım Amacı**

useMemo, referans tipi (objeler, diziler vs.) değişkenlerde kullanılan hesaplamaları optimize eder. Normalde, her render işleminde JavaScript'te referanslar farklı olarak kabul edilir, bu da bileşenin yeniden render edilmesine neden olabilir. useMemo, bu sorunu çözmek için kullanılır.

**Nerede Kullanılır?**

Eğer bir obje veya dizi her render işleminde yeniden hesaplanıyorsa (referansları değişiyorsa) ve bu hesaplamalar pahalıysa (useMemo ile bu hesaplamalar sadece bağımlılık değiştiğinde yapılır).

Örneğin, filtrelenmiş bir liste veya karmaşık matematiksel işlemleri useMemo ile optimize edebilirsiniz.

### useCallback Nedir?

useCallback, bir fonksiyonu **memoize** ederek, her render işleminde yeniden oluşturulmasını önler. Eğer bir bileşene (örneğin bir Button bileşeni) prop olarak bir fonksiyon geçiriyorsanız ve bu fonksiyon her seferinde yeniden oluşturuluyorsa, bileşen gereksiz yere yeniden render edilebilir. useCallback ile bu durumu önleyebilirsiniz.

#### Kullanım Amacı

useCallback, bir fonksiyonun bağımlılıklarına bağlı olarak yeniden tanımlanmasını kontrol eder. Bu sayede, her render işleminde yeni bir fonksiyon oluşturulmasından kaçınılır. Performans iyileştirmesi sağlar.

**Nerede Kullanılır?**

Eğer bir fonksiyonu prop olarak alt bileşenlere geçiriyorsanız ve bu alt bileşenin her seferinde yeniden render edilmesini istemiyorsanız.

Örneğin, bir butona tıklanmasıyla bir sayacın artırılması işlemi gibi durumlarda gereksiz render'ları önlemek için useCallback kullanabilirsiniz.

### React.memo ile Kullanım

React.memo, bir bileşeni prop'lar değişmediği sürece yeniden render etmeyen bir "Higher Order Component" (yüksek seviyeli bileşen) sağlar. useMemo ve useCallback ile birlikte React.memo, bileşenlerinizde performans iyileştirmesi sağlar.

**Özetle;**

React Native uygulamalarında performans sorunlarıyla karşılaştığımızda, useMemo ve useCallback hook'larını kullanarak bileşenlerin gereksiz yere yeniden render edilmesini önleyebilir ve uygulamamızı optimize edebiliriz. useMemo, pahalı işlemleri sadece gerekli olduğunda tekrar hesaplar, useCallback ise fonksiyonların her render'da yeniden oluşturulmasını engeller. Bu iki hook, React.memo ile birlikte kullanıldığında, özellikle büyük ve kompleks uygulamalarda performans artışı sağlayacaktır.

**RTK Query Nedir?**

RTK Query (Redux Toolkit Query), Redux Toolkit ile gelen bir veri getirme ve önbellekleme çözümüdür. API isteklerini yönetmek, durum yönetimi (state management) ve veri önbellekleme işlemlerini kolaylaştırır. RTK Query, Redux’a dayalıdır ve uygulamanın geri kalanıyla entegre bir şekilde çalışır.

**RTK Query ile Normal Query Yöntemleri Arasındaki Farklar**

Query yöntemleri genelde API'lerden veri almak için kullanırız. Örneğin, fetch veya Axios gibi kütüphanelerle API istekleri yaparız. Ancak, bu yöntemlerde veri yönetimi ve durum kontrolü manuel olarak yapıyoruz. Yani isteklerin başarılı olup olmadığını, verinin yüklenip yüklenmediğini ya da hataları kontrol etmek için ekstra kod yazmamız gerekiyor.

RTK Query ise bu işlemleri otomatik olarak yönetir ve birkaç temel fark sunar:

**RTK Query'nin Avantajları:**

**Durum Yönetimi ve API İsteklerinin Entegrasyonu:** RTK Query, Redux’un gücünü kullanarak API isteklerini doğrudan Redux store içinde yönetir. API isteği yaptığımızda veriler otomatik olarak state'e yazılır, başka yerlerde kullanmak için ek bir state yönetimi gerekmez.

**Otomatik Önbellekleme (Caching):** RTK Query, verileri otomatik olarak önbelleğe alır. Aynı veriyi tekrar istemek yerine, daha önce alınan ve önbellekte bulunan verileri kullanır. Bu sayede aynı veriyi tekrar tekrar çekmek zorunda kalmıyoruz, bu da performansı artırır.

**İstek Durumu Yönetimi:** İsteklerin durumu (başarılı, başarısız, yükleniyor vb.) RTK Query tarafından otomatik olarak yönetilir. Bu sayede her bir API isteği için ayrı bir loading durumu kontrolü veya hata yönetimi yazmamıza gerek kalmaz.

isLoading, isSuccess, isError gibi hazır durumlar ile API'nin durumu hakkında bilgi sahibi oluruz.

**Refetching (Yeniden Veri Çekme):** RTK Query, belirli koşullara göre veriyi yeniden çekebilir. Mesela, bir veri güncellendiğinde ya da bir kullanıcı başka bir ekrana geçtiğinde veriyi tekrar getirmek için refetch işlemi yapabiliriz.

**Automatik Cache Expiration:** Verilerin ne kadar süreyle önbellekte kalacağı, RTK Query ile ayarlanabilir. Bu sayede eski verilerin güncel kalmasını sağlayabiliriz.

**Performans Optimizasyonu:** RTK Query, veri güncellemelerini ve istekleri minimal performans etkisiyle gerçekleştirir. Aynı endpoint'e yapılan fazla istekler tek bir istekte birleştirilebilir ve böylece fazla API yükü önlenir.

### ****RTK Query Kullanımı****

#### **Önce API tanımlarız.** createApi fonksiyonuyla yeni bir API oluştururuz ve endpoints altında farklı veri istekleri tanımlarız.

#### **Daha sonra store entegrasyonu yapmamız gerek. A**PI'yi Redux store'a entegre etmek için, API'nin reducer'ını ve middleware'ini store'a ekliyoruz.

#### **Artık veri çekebilir ve bileşenlerde kullanabiliriz. Bunun** için useGetUsersQuery gibi hook'ları kullanabiliriz. Bu hook, verilerin durumu (loading, success, error) ve verilerin kendisini yönetir.

**Özetle;**

React ve Redux uygulamalarında API isteklerini çok daha kolay ve verimli bir şekilde yönetmek için geliştirilmiş bir araçtır. **Fetch** veya **Axios** gibi geleneksel veri çekme yöntemleri genellikle manuel durum yönetimi, önbellekleme ve hata yönetimi gerektirir. Bu süreçler zaman alıcı olabilir ve fazladan kod yazmayı gerektirir. RTK Query ise bu görevleri otomatikleştirir ve geliştiricilere daha temiz ve bakımı kolay bir yapı sunar.