### ****State Management Nedir?****

React Native gibi modern UI framework’lerinde uygulamanın **state** (durum) yapısı, bir bileşenin belirli bir zaman dilimindeki veri veya durumu ifade eder. Örneğin, bir formdaki kullanıcının girdiği veriler, giriş yapmış bir kullanıcının bilgileri veya bir sayfadaki liste öğeleri gibi dinamik veriler, state ile temsil edilir. State management ise bu verilerin ve durumların nasıl yönetildiği, saklandığı ve kullanıldığı süreci ifade eder.

Her bileşen kendi local state'ine sahip olabilir. Ancak, uygulama büyüdükçe ve bileşenler arasında veri paylaşımı gerektiğinde bu durum karmaşıklaşır. State management, bu büyüyen yapıyı daha etkili ve düzenli bir şekilde kontrol altında tutmak için kullanılır. Verilerin nasıl ve ne zaman güncelleneceğini, hangi bileşenlerin bu verilere erişebileceğini ve verilerin global mi yoksa local mi tutulacağını belirlemek için çeşitli state management yöntemleri vardır.

### ****State Management Hangi Soruna Çözüm Sunar?****

React Native'de bileşenler arası veri paylaşımı ihtiyacı arttıkça, her bileşenin local state kullanarak veri saklaması karmaşıklığa yol açabilir. Bu durumda ortaya çıkan temel sorunlar şunlardır:

1. **Bileşenler Arasında Veri Paylaşımı**: Bir uygulamanın farklı bileşenleri aynı veriyi kullanmak zorunda olabilir. Bu durumda, veriyi üst bileşenlerde saklayıp alt bileşenlere prop olarak geçmek gerekebilir. Ancak bu yöntem büyük projelerde sürdürülebilir olmaz.
2. **State’in Merkezi Olarak Yönetilmesi**: Uygulamanın her bir bileşeni kendi state'ini yönettiğinde, veriler farklı yerlerde dağınık olarak tutulabilir. Bu, verilerin yönetimini ve uygulamanın bakımını zorlaştırır.
3. **Global State İhtiyacı**: Kullanıcı giriş bilgileri, alışveriş sepeti gibi uygulamanın genelinde her yerden erişilmesi gereken veriler olduğunda, bu verilerin merkezi bir yerden yönetilmesi daha uygundur. Bu sayede farklı bileşenler arasında veri geçişi yapmak zorunda kalınmaz.

### ****State Management Yöntemleri****

Bu sorunları çözmek için **state management** kullanılır. İşte başlıca yöntemler:

1. **Local State**: Bir bileşen içinde sadece o bileşene özgü verileri yönetmek için kullanılan state. Örneğin, bir input alanının içindeki değeri tutmak.
2. **Context API**: React'in içinde gelen, daha basit projelerde state’i bileşenler arasında paylaşmak için kullanılan bir yöntemdir. Context API, prop drilling sorununu ortadan kaldırır. Bir kez global state oluşturulur ve bu state, herhangi bir bileşenden erişilebilir hale gelir.
3. **Redux**: Redux, React uygulamalarında en popüler state management araçlarından biridir. Redux, global bir state sunarak, uygulamanın herhangi bir yerinde tutarlı ve merkezi bir veri yönetimi sağlar. Redux, tek bir “store” içinde verileri saklar ve bileşenler bu store’a bağlanarak verilere erişir veya güncelleyebilir.

* **Action**: Kullanıcı etkileşimleri veya dış veri kaynaklarından gelen olaylar.
* **Reducer**: Gelen action’a göre store'daki veriyi günceller.
* **Store**: Uygulamanın merkezi veri deposu.

Redux, daha karmaşık ve büyük projelerde yaygın olarak tercih edilir çünkü uygulamanın tamamını kapsayan bir state yönetimi sağlar.

### ****Prop Drilling Nedir?****

Prop drilling, bir bileşenin bir veriyi alt bileşenlere iletmek için prop kullanması ve bu işlemi zincirleme bir şekilde yapmasıdır. Örneğin, bir uygulamada en üst bileşende bir kullanıcı bilgisi saklıysa ve bu bilgiyi en alt bileşene kadar iletmek istiyorsak, bu veri her bileşen tarafından bir sonraki bileşene prop olarak geçirilir.

#### **Prop Drilling Sorunu**

Küçük bir projede prop drilling ile veri taşımak nispeten basittir. Ancak uygulama büyüdükçe ve bileşen hiyerarşisi derinleştikçe, prop drilling şu sorunları doğurur:

* **Gereksiz Karmaşıklık**: Bir veri, sadece en alt bileşende kullanılacaksa bile üst düzey bileşenler gereksiz yere bu veriyi prop olarak almak ve geçmek zorunda kalır. Bu durum, kodun okunabilirliğini ve bakımını zorlaştırır.
* **Kırılganlık**: Bileşenler arasındaki bu bağımlılık, bileşen yapısında değişiklikler yapıldığında veri akışının kırılmasına neden olabilir. Ayrıca, her bileşenin gereksiz yere prop alması, bileşenlerin tekrar kullanılabilirliğini azaltır.

#### **Prop Drilling’e Çözüm Olarak State Management**

Prop drilling sorununu çözmek için state management yöntemleri devreye girer. **Context API**, **Redux** gibi araçlar sayesinde, veriyi üst bileşenlerden alt bileşenlere manuel olarak prop geçirerek iletmek yerine, merkezi bir state sağlayarak herhangi bir bileşenin ihtiyaç duyduğu veriye doğrudan erişmesini sağlarız. Bu, uygulamanın daha modüler ve sürdürülebilir olmasını sağlar.

### ****State Management Ne Zaman Kullanılır?****

State management, aşağıdaki durumlarda kullanılmalıdır:

1. **Global Veri Yönetimi**: Kullanıcı bilgileri, alışveriş sepeti gibi uygulamanın her yerinde erişilmesi gereken veriler olduğunda state management kullanmak en mantıklı çözümdür.
2. **Prop Drilling’i Önlemek**: Eğer bir veriyi üst bileşenlerden alt bileşenlere çok sayıda ara bileşen aracılığıyla aktarmanız gerekiyorsa, prop drilling'den kaçınmak için state management kullanarak veriyi merkezi bir yerden yönetmek daha verimli olacaktır.
3. **Karmaşık Uygulamalar**: Bir uygulamanın birçok bileşeni veri paylaşımına ihtiyaç duyuyorsa ve her bileşenin state'ini tek tek yönetmek zorsa, global state management kullanmak gereklidir. Redux veya MobX gibi araçlar, bu tür uygulamalarda büyük kolaylık sağlar.

### ****Sonuç****

State management, modern uygulamaların büyümesiyle birlikte, bileşenler arasındaki veri akışını düzenli ve sürdürülebilir bir şekilde yönetmenin en etkili yoludur. Prop drilling, veri geçişini manuel olarak yönetmek zorunda olduğunuzda ortaya çıkan bir sorundur ve bu sorunu çözmenin en iyi yolu, state management araçlarını kullanarak verileri merkezi bir yerden yönetmektir. Bu sayede uygulamalar daha kolay yönetilebilir, test edilebilir ve modüler hale gelir.