

Redux kütüphanesinin kullanımını ve uygulama durumunu (application state) yönetme yöntemlerini açıklamaktadır.

## Uygulama Durumu ve Bileşen Durumu

- **Application State:** Uygulamanın tamamı hakkında bilgi tutar ve herhangi bir bileşenden erişilebilir.
- **Component State:** Sadece belirli bir bileşenle sınırlıdır ve o bileşenin durumunu yönetir.

## Redux Kütüphanesinin Amacı

- Redux, uygulama durumunu yönetmek için kullanılır ve bileşenlerin kendi durumlarını yönetmek yerine merkezi bir yapı sağlar.
- Uygulama büyündükçe, Redux kullanmak daha faydalı hale gelir.

## Redux Store

- **Store:** Verileri saklayan ve yöneten bir veri yapısıdır. Redux Store, "single source of truth" (tek gerçek kaynak) olarak adlandırılır.
- Redux, durumu global olarak yönetir, böylece uygulamanın herhangi bir yerinden güncellemler yapılabilir.

## Redux Kullanmanın Faydaları

- Kod karmaşıklığını azaltır ve okunabilirliği artırır.
- Uygulama durumunu tahmin etmeyi kolaylaştırır.
- Performansı optimize eder, çünkü yalnızca gerekli kısımlar güncellenir.

## Redux'un Temel Kavramları

- **Action:** Durum güncellemesi için bir eylem nesnesi oluşturur. Eylem, durumu nasıl değiştireceğini belirtmez.
- **Store:** Uygulamanın tüm mevcut durumlarını saklar. Durum ağaçları (state tree) kullanarak yönetilir.

## Redux Mimarisi

- **Reducer:** Durumu güncellemek için bir fonksiyon yazılır. Reducer, eylem nesnesi ve mevcut durum nesnesini alarak yeni durumu hesaplar.
- **Dispatching:** Eylem nesnesi, bir eylem oluşturucu (action creator) fonksiyonu tarafından oluşturulur ve store'a gönderilir.

## Uygulama Örneği

- E-ticaret uygulamasında, kullanıcı "sepete ekle" seçeneğini seçtiğinde bir eylem başlatılır. Bu işlem, sepetteki öğe sayısını artırmak için bir eylem nesnesi oluşturur.

## Önemli Terimler

- **Action Object:** Eylemi tanımlayan nesne.
- **Payload:** Durumu güncellemek için gereken ek verileri saklar.
- **Pure Function:** Reducer, aynı girdilerle her zaman aynı çıktıyı döndüren saf bir fonksiyondur.

## Senkron ve Asenkron İşlemler

- **Senkron (Synchronous):** İki işleminden biri tamamlanmadan diğerini başlayamaz. Örneğin, bir işlem başlarsa, ikinci işlem ilk işlem bitene kadar bekler.
- **Asenkron (Asynchronous):** İşlemler paralel olarak çalışır. Bu, bir işlemin devam ederken diğerinin de başlayabileceği anlamına gelir.

## Asenkron İşlemlerin Zorlukları

- Kullanıcı etkileşimleri sırasında, işlemlerin sıralı bir şekilde işlenmesi gerekebilir. Örneğin, bir alışveriş sepetine iki ürün eklemek için önce ürün sayısını girmek gereklidir.

## Redux ve Asenkron İşlemler

- Redux, senkron bir yapı gerektirir. Ancak, bazı durumlarda asenkron işlemler gereklidir, örneğin, sunucudan veri almak.
- **Middleware:** Asenkron verilerle etkileşimde bulunmak için kullanılır. Middleware, eylemleri (actions) yakalar ve senkron bir akış sağlamak için geciktirir.

## Thunk ve Saga

- **Thunk:** Eylem oluşturucuların (action creators) fonksiyonlar döndürmesine olanak tanır. Bu fonksiyonlar, asenkron işlemler yapabilir. Thunk, basit uygulamalar için uygun ama ölçeklenebilirlik sorunları vardır.
- **Saga:** ES6'da tanıtılan jeneratör fonksiyonlarını kullanır. Saga, asenkron işlemleri yönetmek için daha karmaşık bir yapıdır ve test etme ve hata ayıklama için birçok özellik sunar.

Redux ve veri akışı konularını ele alıyor. Aşağıda önemli noktaları ve terimleri bulabilirsiniz:

## State Change (Durum Değişikliği)

- React uygulamalarında, bir bileşenin durumu (state) değiştiğinde, React DOM'u yeniden render eder.
- Uygulama büyündükçe, durumu yönetmek zorlaşır ve bileşenler arasında veri transferi karmaşık hale gelebilir.

## Redux'un Temel Bileşenleri (Core Components of Redux)

- **Central Store (Merkezi Depo):** Uygulamanın tüm durumunu tutar.
- **Actions (Eylemler):** Bileşenlerden gönderilen bilgi paketleridir. Genellikle iki özelliği sahiptir: **type** (tür) ve **payload** (yük).

- **Reducer (Azaltıcı)**: Eski durumu ve eylemi alarak güncellenmiş bir durum döndüren basit bir işlevdir.
- **Subscription (Abonelik)**: Durum güncellendiğinde bileşenlerde tetiklenir.

## Veri Akışı (Data Flow)

- React-Redux uygulamalarında veri akışı tek yönlüdür. Kullanıcı etkileşimleri, eylem yaratıcılarının (action creators) eylem göndermesine yol açar.
- Eylem gönderildiğinde, kök azaltıcı (root reducer) tarafından alınır ve tüm azaltıcılara iletilir.

## Tek Yönlü Veri Akışının Avantajları (Advantages of One-Way Data Flow)

- Uygulama büyükçe durumu yönetmek daha kolay hale gelir.
- Kullanıcı arayüzünde (UI) yapılan eylemler ile durumun güncellenmesi ayrıldığında, yönetim daha basit olur.

## Redux Toolkit Nedir?

- Redux Toolkit (RTK), Redux geliştirmeyi basitleştirmek ve daha verimli hale getirmek için Redux ekibi tarafından sağlanan resmi bir pakettir.
- RTK, yaygın Redux görevlerini kolaylaştırın ve boilerplate kodunu azaltan yardımcı programlar içerir.

## Redux Toolkit'in Temel Bileşenleri

- **configureStore**: Redux mağazasını kurmak için kullanılan bir fonksiyondur. Bu fonksiyon, yaygın middleware'leri (örneğin, **Redux Thunk**) ve **Redux Devtools Extension**'ı ayarlamak için kullanılır.
- **createSlice**: Geliştiricilerin durumu güncellemek için slice reducer'ları tanımlamasına olanak tanır. Bu, durumu doğrudan değiştirme endişesi olmadan reducer mantığını yazmayı kolaylaştırır.

## Store ve Slice İlişkisi

- **Store**: Uygulamanızın tam durum ağacını tutan tek bir JavaScript nesnesidir. Durum güncellemleri **dispatch(action)** ile yapılır.
- **Slice**: Uygulama durumunun bir parçasını ve onu güncelleme mantığını temsil eder. Her slice, bir reducer, action creators ve bir başlangıç durumu içerir.

## Örnek Uygulama

- Bir e-ticaret uygulamasında, ürün miktarını artırma, toplam fatura tutarını hesaplama ve kazanılan süper paraları gösterme gibi bileşenler bulunmaktadır.
- **ProductQuantity.jsx**: Ürün miktarını gösterir ve artırma/azaltma butonları içerir.
- **CartValue.jsx**: Toplam fatura tutarını hesaplar.

- **CounterSlice.jsx**: Bir Redux slice'ı tanımlar ve iki reducer (increment ve decrement) içerir.