React'te Düşünmek

React, bize göre, JavaScript ile büyük ve hızlı Web uygulamaları oluşturmanın en önde gelen yoludur. Bizim için Facebook ve Instagram'ın geliştirilmesinde çok etkili oldu.

React'in en harika yanlarından biri de, uygulamaları oluştururken size kazandırdığı bakış açısıdır. Bu dökümanda, React'i kullanarak arama özelliği olan bir ürün tablosu oluşturmanın düşünce sürecinde size yol göstereceğiz.

* Bir Taslakla Başlayın

Zaten bir JSON API'ımızın ve tasarımcımızdan gelen bir taslağımızın olduğunu hayal edin. Taslak bunun gibi gözüküyor:

JSON API'ımız aşağıdakine benzeyen bir veri dönüyor:

```
[
  {category: "Sporting Goods", price: "$49.99", stocked: true, name: "Football"},
  {category: "Sporting Goods", price: "$9.99", stocked: true, name: "Baseball"},
  {category: "Sporting Goods", price: "$29.99", stocked: false, name: "Basketball"},
  {category: "Electronics", price: "$99.99", stocked: true, name: "iPod Touch"},
  {category: "Electronics", price: "$399.99", stocked: false, name: "iPhone 5"},
  {category: "Electronics", price: "$199.99", stocked: true, name: "Nexus 7"}
];
```

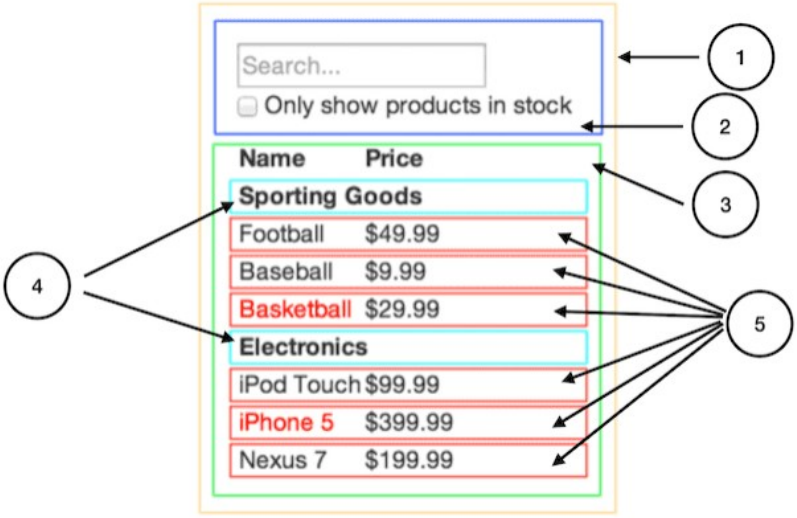
Adım 1: Kullanıcı Arabirimini Bileşen Hiyerarşisine Bölün

Yapmak isteyeceğiniz ilk şey, taslaktaki her bileşenin (ve alt bileşenlerin) etrafina kutular çizip, her birisine

isimler vermektir. Eğer bir tasarımcıyla çalışıyorsanız, bunu zaten yapmış olabilirler; o zaman gidip onlarla konuşun! Photoshop'taki katman isimleri React bileşenlerinin isimleri olabilir!

Ama nelerin kendi başına birer bileşen olacağına nasıl karar vereceksiniz? Yeni bir nesne ya da fonksiyon oluşturup oluşturmayacağınıza karar vermek için yine aynı teknikleri kullanın. Bu tekniklerden biri, tek sorumluluk ilkesidir; yani bir bileşen ideal olarak sadece tek bir şey yapmalıdır. Bileşen büyüdüğü taktirde, daha küçük alt bileşenlere ayrılmalıdır.

Çoğu zaman kullanıcıya bir JSON veri modeli göstereceğiniz için, modeliniz doğru inşa edildiyse, kullanıcı arayüzünüzün (ve dolayısıyla bileşen yapınızın) güzel bir şekilde eşleşeceğini göreceksiniz. Bunun nedeni, kullanıcı arabirimi ve veri modellerinin aynı bilgi mimarisine bağlı kalma eğiliminde olmasıdır. Bu yüzden kullanıcı arayüzünüzü bileşenlere ayırma işi genellikle önemsizdir. Arayüzünüzü sadece, her birisi veri modelinizin bir parçasını temsil edecek şekilde, bileşenlere bölün.



Burada uygulamamızın beş tane bileşeni olduğunu göreceksiniz. Her bileşenin temsil ettiği verileri italik halde aşağıda belirttik. Resimdeki sayılar aşağıdaki sayılara karşılık gelmektedir.

1-FilterableProductTable (turuncu): örnek uygulamanın tamamını içerir.

2-SearchBar (mavi): bütün kullanıcı girdilerini alır.

3-ProductTable (yeşil): kullanıcı girdisine bağlı olarak veri koleksiyonunu görüntüler ve filtreler.

4-ProductCategoryRow (turkuaz): her bir kategori için bir başlık gösterir.

5-ProductRow (kırmızı): her bir ürün için bir satır gösterir.

ProductTable'a bakarsanız, tablo başlığının ("Name" ve "Price" etiketlerini içeren kısım) kendi bileşeni olmadığını göreceksiniz. Bu bir tercih meselesi ve her iki şekilde de yapılmasını savunan bir argüman var. Bu örnekte, tablo başlığını ProductTable'ın bir parçası olarak bıraktık. Çünkü tablo başlığı, ProductTable'ın

sorumluluğunda olan veri koleksiyonunu render etme işleminin bir parçasıdır. Yine de, eğer bu başlık giderek karmaşık bir hale gelirse (Mesela sıralama özelliği ekleseydik), kendi ayrı ProductTableHeader bileşenini yapmak kesinlikle daha mantıklı olurdu.

Şimdi, taslağımızdaki bileşenleri belirlediğimize göre, onları bir hiyerarşiye göre düzenleyelim. Bu kolay. Taslakta başka bir bileşen içinde görünen bileşenler, hiyerarşideki bir alt eleman olarak görünmelidir:

.FilterableProductTable

.SearchBar

.ProductTable

.ProductCategoryRow

.ProductRow

Adım 2: React'te Statik Versiyonunu Oluşturun

Artık bileşen hiyerarşisine sahip olduğunuza göre, uygulamanızı hayata geçirme vakti geldi. Bunun en kolay yolu, veri modelinizi alıp, kullanıcı arayüzünü oluşturan; ancak etkileşimli (interaktif) olmayan bir sürüm oluşturmaktır. Bu noktada en iyi yaklaşım bu süreçleri birbirinden ayırmaktır. Çünkü statik bir versiyonu oluşturmak çok fazla yazmayı ama çok az düşünmeyi gerektirir. Bunun yanında etkileşimli (interaktif) versiyonunu yapmak çok daha fazla düşünmeyi ama daha az yazmayı gerektirir. Neden böyle olduğunu göreceğiz.

Veri modelinizi render eden bir statik versiyonu yapmak için, diğer bileşenleri kullanan ve prop'lar aracılığıyla veri ileten bileşenler oluşturmak isteyeceksiniz. prop'lar bileşenler arasında yukarıdan aşağıya veri iletmenin bir yoludur. Eğer state konseptine aşinaysanız, bu statik versiyonu oluşturmak için state'leri hiçbir şekilde kullanmayın. State konsepti sadece etkileşim, yani zaman içinde değişen verilerin olduğu durumlar, için ayrılmıştır. Buna, uygulamanın statik bir sürümü olduğundan, ihtiyacınız yoktur.

Yukarıdan aşağıya veya aşağıdan yukarıya oluşturabilirsiniz. Diğer bir deyişle, oluşturmaya hiyerarşide daha yukarıdaki (örneğin, FilterableProductTable) veya daha aşağıdaki (ProductRow) bileşenler ile başlayabilirsiniz. Daha basit örneklerde, yukarıdan aşağıya gitmek genellikle daha kolaydır ve daha büyük projelerde, aşağıdan yukarıya gitmek ve de bileşenleri oluşturdukça testler yazmak daha kolaydır.

Bu adımın sonunda, veri modelinizi oluşturan yeniden kullanılabilir bileşenlerden oluşan bir kütüphaneye sahip olacaksınız. Bileşenler yalnızca render() metotlarına sahip olacaktır, çünkü bu, uygulamanızın statik bir sürümüdür. Hiyerarşinin en üstündeki bileşen (FilterableProductTable) veri modelinizi bir prop olarak alır. Temel veri modelinizde değişiklik yaparsanız ve tekrar root.render()'ı çağırırsanız, kullanıcı arayüzü güncellenecektir. Arayüzünüzün nasıl güncellendiğini ve, karmaşık bir şey olmadığından, nerede değişiklik yapılacağını görmek kolaydır. React'in tek yönlü veri akışı (ayrıca tek yönlü bağlama olarak da bilinir) her şeyi modüler ve hızlı tutar.

Kısa bir araya girme: Prop'lar vs State

React’ta iki tür “model” datası vardır: props ve state. Bunlar arasındaki farkı anlamak önemlidir.

Adım 3: UI State’inin Minimal (ancak eksiksiz) Temsilini Belirleme

Kullanıcı arayüzünüzü etkileşimli hale getirmek için, temel veri modelinizde değişiklikleri tetikleyebilmeniz gerekir. React bunu state ile kolaylaştırmaktadır.

Uygulamanızı doğru bir şekilde oluşturmak için, öncelikle uygulamanızın ihtiyaç duyduğu minimum değişken state kümesini düşünmeniz gerekir. Burada anahtar kelime TEK: Tekrar Etme Kendini (DRY: Don’t Repeat Yourself) dir. Uygulamanızın ihtiyaç duyduğu state’in mutlak asgari temsilini belirleyin ve talep üzerine ihtiyacınız olan her şeyi hesaplayın. Örneğin; bir YAPILACAKLAR listesi oluşturuyorsanız, sadece YAPILACAKLAR listesini tutan bir dizi saklayın; listedeki madde sayısı için ayrı bir state değişkeni tutmayın. Bunun yerine, listedeki madde sayısını render etmek istediğinizde, sadece YAPILACAKLAR dizisinin uzunluğunu alıp kullanın.

Örnek uygulamamızdaki sahip olduğumuz tüm veri parçalarına bakalım:

- Orijinal ürün listesi

- Kullanıcının girdiği arama metni

- Checkbox’ın değeri

- Filtrelenmiş ürün listesi

Her birini gözden geçirelim ve hangisinin state’e dahil olduğunu bulalım. Bunun için, her veri parçasına dair üç soru sorun:

1-Üst elemandan prop’lar aracılığıyla mı iletilmiş? Öyleyse state’e ait değildir.

2-Zaman içerisinde değişiklik göstermiyor mu? Öyleyse state’e ait değildir.

3-Bileşeninizdeki herhangi başka bir state’e veya prop’a göre hesaplayabiliyor musunuz? Öyleyse state’e ait değildir.

Orijinal ürün listesi prop olarak iletildiği için state’e ait değildir. Arama metni ve checkbox, zaman içerisinde değiştikleri ve başka bir şey üzerinden hesaplanamadıkları için state’e ait gibi duruyorlar. Ve son olarak, filtrelenmiş ürün listesi de; orijinal ürün listesi, arama metni ve checkbox ın değerine göre hesaplanabileceği için, state’e ait değildir.

Sonuç olarak, state’imiz aşağıdaki gibidir:

- Kullanıcının girdiği arama metni

- Checkbox’ın değeri

Adım 4: State'inizin Nerede Yaşaması Gerektiğini Belirleyin

Evet, uygulama state'inin asgari düzeydeki setinin ne olduğunu belirledik. Şimdi, hangi bileşenin bu state'i değiştirdiğini veya sahip olduğunu belirlememiz gerekir.

Unutmayın: React'in tüm olayı, bileşen hiyerarşisinde yukarıdan aşağı doğru tek yönlü veri akışıdır. Hangi bileşenin hangi state'e sahip olması gerektiği hemen belli olmayabilir. Bu, yeni gelenlerin anlaması için en zor kısımdır. Bu yüzden, bunu anlamak için şu adımları izleyin:

Uygulamanızdaki her state parçası için:

- O state'e göre bir şeyler render eden her bileşeni belirleyin.

- Ortak bir sahip bileşen bulun. (Hiyerarşide, state'e ihtiyaç duyan bütün bileşenlerin üzerinde bulunan tek bir bileşen)

- Ya ortak bir sahip bileşen ya da hiyerarşide daha yüksekte bulunan bir bileşen state'e sahip olmalıdır.

- State'e sahip olması mantıklı olmayan bir bileşen bulamazsanız, yalnızca state'i tutması için yeni bir bileşen oluşturun ve onu hiyerarşide ortak sahip bileşeninin üzerindeki bir yere ekleyin.

Şimdi bu stratejiyi uygulamamız için hayata geçirelim:

- ProductTable ürün listesini state'e göre filtrelemeli ve SearchBar arama metnini ve Checkbox'ın durumunu göstermelidir.

- Ortak sahip bileşen FilterableProductTable bileşinidir.

- Filtre metninin ve Checkbox değerininin FilterableProductTable içinde yaşaması mantıklıdır.

Harika; state'imizin FilterableProductTable bileşeninde yaşamasına karar verdik. İlk olarak, uygulamanın başlangıç state'ini belirlemek için this.state = {filterText: '', inStockOnly: false} nesne özelliğini constructor'a ekleyin. Devamında, filterText ve inStockOnly değerlerini ProductTable ve SearchBar bileşenlerine prop olarak iletin. Son olarak, ProductTable'daki satırları filtrelemek ve form alanlarının değerlerini SearchBar'da ayarlamak için bu prop'ları kullanın.

Uygulamanızın nasıl davranacağını görmeye başlayabilirsiniz: filterText'i ball olarak ayarlayın ve uygulamanızı yenileyin. Veri tablosunun doğru bir şekilde güncellendiğini göreceksiniz.

Adım 5: Ters Veri Akışı Ekleyin

Şimdiye kadar, hiyerarşide yukarıdan aşağı akan prop'ların ve state'in bir fonksiyonu olarak

doğru şekilde işleyen bir uygulama geliştirdik. Şimdi diğer tarafa doğru akan verileri destekleme zamanı: Hiyerarşide derinlerde bulunan form bileşenlerinin FilterableProductTable'da state 'i güncellemesi gerekir.

React, programınızın nasıl çalıştığını anlamayı kolaylaştırmak için bu veri akışını açık bir hale getirir, ancak geleneksel iki yönlü (yukarıdan aşağı ve aşağıdan yukarı) veri akışından biraz daha fazla yazma gerektirir.

Eğer yukarıdaki örnekte metin kutusuna yazmayı ya da kutucuğu işaretlemeyi denediğiniz taktirde, React'in bunu yoksaydığını göreceksiniz. Bu kasıtlı olarak böyledir. Çünkü input un value prop'unu, her zaman FilterableProductTable bileşeninden gelen state'e eşit olacak şekilde ayarladık.

Ne olmasını istediğimizi düşünelim. Kullanıcı form'u her değiştirdiğinde, kullanıcının yaptığı değişikliği yansıtacak şekilde state'i güncellediğimizden emin olmak istiyoruz. Bileşenlerin yalnızca kendi state'lerini güncellemesi gerektiği için, FilterableProductTable bileşeni, SearchBar bileşenine, state'in her güncellenmesinde çağrılacak callback'ler iletir. Girdilerin (input) OnChange olayını, SearchBar bileşenini bilgilendirmek için kullanabiliriz. FilterableProductTable tarafından iletilen callBack'ler setState()'i çağıracak ve uygulama güncellenecektir.

Karışık görünmesine rağmen, aslında sadece birkaç kod satırından ibaret. Ve verilerinizin uygulama boyunca nasıl aktığı da gerçekten çok belirgindir.

Ve Bu Kadar

Umuyoruz ki bu bölüm, React ile bileşen ve uygulamalar oluştururken nasıl düşünmeniz gerektiği hakkında size bir fikir vermiştir. Normalde alıştığınızdan biraz daha fazla kod yazmanız gerekebilir. Ancak kodun yazıldığından çok daha fazla okunduğunu; ve bu modüler, açık kodun okumak için oldukça kolay olduğunu unutmayın. Büyük bileşen kütüphaneleri oluşturmaya başladığınızda, bu açıklık ve modülerliği takdir edeceksiniz ve yazdığınız kodu yeniden kullandıkça kod satırlarınız küçülmeye başlayacak