Bileşenler ve Prop'lar

Bileşenler, kullanıcı arayüzünü ayrıştırarak birbirinden bağımsız ve tekrar kullanılabilen parçalar oluşturmanızı sağlar. Bu sayede her bir parçayı, birbirinden izole bir şekilde düşünerek kodlayabilirsiniz.

Bu sayfa, bileşenlerin ne olduğuna dair bir fikir edinmenizi sağlayacaktır. Bileşenler API dokümanını inceleyerek daha detaylı bilgi edinebilirsiniz.

Kavramsal olarak bileşenler, JavaScript fonksiyonları gibidir. Bileşenler, "props" adındaki girdileri opsiyonel olarak alırlar ve ekranda görüntülenecek React elementlerini geri döndürürler.

* Fonksiyon ve Sınıf Bileşenleri

Bir bileşen oluşturmak için en basit yol, bir JavaScript fonksiyonu yazmaktır:

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

Bu fonksiyon, girdi olarak "props" (properties) adındaki tek bir nesneyi aldığı ve geriye bir React elementi döndürdüğü için geçerli bir React bileşenidir. Bu tarz bileşenler, gerçekten de birer JavaScript fonksiyonları oldukları için adına "**fonksiyonel bileşenler**" denir.

Fonksiyon yerine, bir ES6 sınıfı kullanarak da React bileşeni oluşturabilirsiniz:

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

Üstteki her iki bileşen de React'in bakış açısından birbirine eşittirler.

Sınıf ve fonksiyon bileşenlerinin her birisi bazı ek özelliklere sahiptirler. Buna sonraki bölümlerde değineceğiz.

* Bir Bileşenin Render Edilmesi

Önceki bölümlerde, React elementi olarak sadece DOM elementlerini ele almıştık.

```
const element = <div />;
```

Ancak elementler, kullanıcı tanımlı bileşenler de olabilirler:

```
const element = <Welcome name="Sara" />;
```

React, kullanıcı tanımlı bir bileşeni gördüğü zaman, JSX özelliklerini ve alt elemanlarını bu bileşene tek bir nesne olarak aktarır. Bu nesneye "**props**" adı verilir.

Örneğin aşağıdaki kod, sayfada "Hello, Sara" mesajını görüntüler:

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
const root = ReactDOM.createRoot(document.getElementById('root'));
const element = <Welcome name="Sara" />;
root.render(element);
```

Bu örnekte, hangi olayların gerçekleştiğine bir bakalım:

1- <Welcome name="Sara" /> elementi ile birlikte **root.render()** fonksiyonunu çağırıyoruz.

2- Devamında **React, {name: 'Sara'}** prop'u ile Welcome bileşenini çağırıyor.

3- Welcome bileşenimiz, sonuç olarak geriye bir **<h1>Hello, Sara</h1>** elementi döndürüyor.

4- React DOM, **<h1>Hello, Sara</h1>** ile eşleşmek için, DOM'ı arka planda efektif bir şekilde güncelliyor.

> **Not: Bileşen isimlendirmelerinde daima büyük harfle başlayınız.**
> Çünkü React, küçük harfle başlayan bileşenlere DOM etiketleri gibi davranır. Örneğin `<div />`, bir HTML div etiketini temsil eder, fakat `<Welcome />` ise bir bileşeni temsil eder ve kodun etki alanında `Welcome`'ın tanımlı olmasını gerektirir.
>
> Bu isimlendirmenin nedeni hakkında detaylı bilgi edinmek için lütfen Derinlemesine JSX sayfasına bakınız.

* Bileşenlerden Kompozisyon Oluşturulması

Bileşenler, çıktılarında diğer bileşenleri gösterebilir. Bu sayede soyutlanan bir bileşen, herhangi bir ayrıntı düzeyinde tekrar kullanılabilir. Butonlar, formlar, diyaloglar, ekranlar ve daha nicesi React uygulamalarında yaygın bir şekilde bileşen olarak ifade edilebilirler.

Örneğin, Welcome'ı istediğimiz kadar görüntüleyecek bir App bileşeni oluşturabiliriz:

Welcome Component

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

App Component

```
function App() {
  return (
    <div>
      <Welcome name="Sara" />
      <Welcome name="Cahal" />
      <Welcome name="Edite" />
    </div>
  );
}
```

Genellikle, yeni React uygulamaları, en üstte bir tane App bileşeni içerirler. Ancak React'i mevcut uygulamanıza entegre ediyorsanız, Button gibi en küçük bileşenlerden başlayacak şekilde, basitten karmaşığa doğru ilerleyerek bileşen hiyerarşisini oluşturabilirsiniz.

* Bileşenlerin Çıkarılması

Büyük bileşenleri, sade ve yönetilebilir olması adına daha küçük bileşenlere bölebilirsiniz.

Örneğin aşağıdaki Comment bileşenini ele alalım:

```
function Comment(props) {
  return (
    <div className="Comment">
      <div className="UserInfo">
        <img className="Avatar"
```

```
        src={props.author.avatarUrl}
        alt={props.author.name}
      />
      <div className="UserInfo-name">
        {props.author.name}
      </div>
    </div>
    <div className="Comment-text">
      {props.text}
    </div>
    <div className="Comment-date">
      {formatDate(props.date)}
    </div>
  </div>
 );
}
```

Üstteki bileşen; author nesnesini, text metnini ve bir date tarihini prop olarak alır. Bu bileşen, bir sosyal medya sitesinde yorum kutucuğunun görüntülenmesini sağlar.

İç içe halde bulunan bu bileşenin üzerinde değişiklik yapmak zor olabilir. Ayrıca bünyesindeki DOM elementlerinin de tekrar kullanılabilirliği oldukça düşük seviyededir. Bu durumu çözmek için, kod içerisinden birkaç bileşen çıkarabiliriz.

Öncelikle Avatar bileşenini çıkaralım:

```
function Avatar(props) {
 return (
  <img className="Avatar"
    src={props.user.avatarUrl}
    alt={props.user.name}
```

```
    />
  );
}
```

Avatar bileşeninin, bir Comment bileşeni içerisinde render edildiğini bilmesi gerekli değildir. Bu nedenle Avatar bileşenini, gelecekte uygulamanın daha farklı yerlerinde de kullanma ihtimalimiz bulunduğundan dolayı, prop değişkenleri için author yerine user gibi daha genel bir isim verebiliriz.

Prop'lar isimlendirilirken, ilgili bileşenin hangi bileşen içerisinde kullanıldığını ele almak yerine, bileşeni bağımsız olarak ele almanız gerekmektedir.

Yaptığımız değişiklikle Comment bileşenini az bir miktar basitleştirmiş olduk:

```
function Comment(props) {
  return (
    <div className="Comment">
      <div className="UserInfo">
        <Avatar user={props.author} />
        <div className="UserInfo-name">
          {props.author.name}
        </div>
      </div>
      <div className="Comment-text">
        {props.text}
      </div>
      <div className="Comment-date">
        {formatDate(props.date)}
      </div>
    </div>
  );
}
```

Şimdi Avatar ile birlikte, kullanıcı adını da render edecek olan UserInfo bileşenini kod içerisinden çıkarabiliriz.

```
function UserInfo(props) {
  return (
    <div className="UserInfo">
      <Avatar user={props.user} />
      <div className="UserInfo-name">
        {props.user.name}
      </div>
    </div>
  );
}
```

Bu sayede Comment bileşeni daha da basitleşmiş hale geldi:

```
function Comment(props) {
  return (
    <div className="Comment">
      <UserInfo user={props.author} />
      <div className="Comment-text">
        {props.text}
      </div>
      <div className="Comment-date">
        {formatDate(props.date)}
      </div>
    </div>
  );
```

```
}
```

Bileşenlerin çıkarılması en başta angarya bir işlem gibi görünebilir. Fakat büyük çaplı uygulamalarda, tekrar kullanılabilir bileşenler içeren bir bileşen paletine sahip olmak oldukça faydalı hale gelecektir. Bileşen çıkarmanın genel mantığı aşağıdaki gibidir:

-Eğer kullanıcı arayüzündeki bir eleman (Button, Panel, Avatar) uygulama içerisinde birçok defa kullanılıyorsa,

-Eğer bir bileşen (App, FeedStory, Comment) oldukça karmaşık hale geldiyse,

Bu bileşen, içerisinden bileşenler çıkarmak için iyi bir adaydır diyebiliriz.

* Prop'lar Salt Okunurdur

Fonksiyon veya sınıf bileşeninden herhangi birini oluşturduğunuzda, bu bileşen kendi prop'larını asla değiştirmemelidir. Örneğin aşağıdaki sum fonksiyonunu ele alalım:

```
function sum(a, b) {
  return a + b;
}
```

Bu tarz fonksiyonlar, kendi girdi parametrelerini değiştirmedikleri ve her zaman aynı parametreler için aynı sonucu ürettiklerinden dolayı "pure" (saf) fonksiyonlardır.

Tam ters örnek verecek olursak, aşağıdaki fonksiyon impure'dür (saf değildir). Çünkü kendi girdi değerini değiştirmektedir:

```
function withdraw(account, amount) {
  account.total -= amount;
}
```

React, kod yazımında oldukça esnek olmasına rağmen, sadece bir tek kuralı şart koşmaktadır:

Bütün React bileşenleri yalın (pure) fonksiyonlar gibi davranmalı ve prop'larını asla değiştirmemelidirler.

Tabii ki kullanıcı arayüzleri dinamiktir ve zaman içerisinde değişiklik gösterir. Sonraki bölümde, "state" (durum) adındaki yeni konsepte değineceğiz. State bu kurala sadık kalarak; kullanıcı etkileşimleri, ağ istekleri ve diğer şeylerden dolayı zaman içerisinde değişen arayüzün görüntülenmesi için, React bileşenlerinin kendi çıktılarını değiştirebilmesine izin verir.