State'i Yukarı Taşıma

Çoğu zaman, birden çok bileşenin değişen aynı state'i yansıtması gerekmektedir. Bu durum için paylaşılan state'i en yakın ortak üst elemana taşımayı tavsiye etmekteyiz. Bunun nasıl çalıştığını görelim.

Bu bölümde, suyun belirli bir sıcaklıkta kaynayıp kaynayamayacağını hesaplayan bir sıcaklık hesaplayıcısı oluşturacağız.

BoilingVerdict diye isimlendirdiğimiz bir bileşenle işe başlayacağız. Bu bileşen celsius değerini bir prop olarak kabul edip bu değerin suyu kaynatmaya yeterli olup olmayacağını gösterecektir.

```
function BoilingVerdict(props) {
 if (props.celsius >= 100) {
  return <p>Su kaynar.</p>;
 }
 return <p>Su kaynamaz.</p>;
}
```

Bir sonraki adımda, Calculator diye isimlendirdiğimiz bir bileşen oluşturacağız. Bu bileşen sıcaklık değerini girmemizi sağlayacak bir <input> elemanına sahip olacak ve bu input elemanında this.state.temperature değeri tutulacaktır.

Bu bileşen ayrıca, <input>'ta tuttuğumuz değer için BoilingVerdict bileşenini de ekrana yansıtacaktır.

```
class Calculator extends React.Component {
 constructor(props) {
  super(props);
  this.handleChange = this.handleChange.bind(this);
  this.state = {temperature: ''};
 }
 handleChange(e) {
  this.setState({temperature: e.target.value});
 }
 render() {
```

```
    const temperature = this.state.temperature;
    return (
      <fieldset>
        <legend>Santigrat cinsinden sıcaklık girin:</legend>
        <input
          value={temperature}
          onChange={this.handleChange} />
        <BoilingVerdict
          celsius={parseFloat(temperature)} />
      </fieldset>
    );
  }
}
```

* İkinci Input Elemanını Ekleme

Santigrat input'una ek olarak Fahrenayt cinsinden de sıcaklık değeri girebileceğimiz bir input'a ve sıcaklık değerlerinin birbirleri cinsinden senkronize şekilde çalışabileceği yeni bir gereksinimimizin olduğunu düşünelim.

Öncelikle TemperatureInput bileşenini Calculator bileşeninden ayırarak işe başlayacağız. Santigrat birimi olan "c" veya Fahrenayt birimi olan "f" değerine sahip olacak scale prop'unu ekleyeceğiz:

```
const scaleNames = {
  c: 'Santigrat',
  f: 'Fahrenayt'
};


class TemperatureInput extends React.Component {
  constructor(props) {
    super(props);
```

```
    this.handleChange = this.handleChange.bind(this);
    this.state = {temperature: ''};
  }
  handleChange(e) {
    this.setState({temperature: e.target.value});
  }
  render() {
    const temperature = this.state.temperature;
    const scale = this.props.scale;
    return (
      <fieldset>
        <legend>{scaleNames[scale]} cinsinden sıcaklık girin:</legend>
        <input value={temperature}
               onChange={this.handleChange} />
      </fieldset>
    );
  }
}
```

Şimdi iki ayrı sıcaklık ölçü biriminden de değer girebilmek için Calculator bileşenini aşağıdaki gibi değiştirebiliriz:

```
class Calculator extends React.Component {
  render() {
    return (
      <div>
        <TemperatureInput scale="c" />
        <TemperatureInput scale="f" />
```

```
    </div>
   );
  }
}
```

Şu an sıcaklık değerini girebileceğimiz iki input'unuz var, ancak herhangi birinde sıcaklık değeri girdiğimizde diğer input güncellenmemektedir. Bu durum, değerleri birbiri cinsinden senkronize etme gereksinimimizi karşılamamaktadır.

Ayrıca Calculator bileşeninde BoilingVerdict bileşenini görüntüleyememekteyiz. Sıcaklık değeri TemperatureInput bileşeninde gizli olduğundan dolayı Calculator bileşeni girilen sıcaklığı bilmemektedir.

* Dönüştürme Fonksiyonlarını Yazma

Öncelikle, Santigrat-Fahrenayt ve Fahrenayt-Santigrat dönüşümlerini sağlayabilmek için iki ayrı fonksiyon yazacağız.

```
function toCelsius(fahrenheit) {
  return (fahrenheit - 32) * 5 / 9;
}


function toFahrenheit(celsius) {
  return (celsius * 9 / 5) + 32;
}
```

Bu iki fonksiyon, sıcaklık değerlerini birbiri cinsinden dönüştürmeyi sağlamaktadır.

String olan temperature değerini ve dönüştürücü (convert) fonksiyonunu argüman olarak alan, geriye string döndüren başka bir fonksiyon yazacağız. Bu fonksiyonu input'lardaki değerleri birbirleri cinsinden hesaplayabilmek amaçlı kullanacağız. Bu fonksiyon geçerli olmayan bir temperature değeri için geriye boş string döndürecektir. Geçerli bir değer girildiğinde ise virgülden sonra 3 basamağa yuvarlanmış bir string döndürecektir.

```
function tryConvert(temperature, convert) {
  const input = parseFloat(temperature);
  if (Number.isNaN(input)) {
    return '';
```

```
  }
  const output = convert(input);
  const rounded = Math.round(output * 1000) / 1000;
  return rounded.toString();
}
```

Örneğin, tryConvert('abc', toCelsius) geriye boş string döndürürken, tryConvert('10.22', toFahrenheit) ise geriye '50.396' değerini döndürecektir

* State'i Yukarı Taşıma

Şu anda, her iki TemperatureInput bileşeni birbirlerinden bağımsız olarak değerlerini kendi yerel state'lerinde tutmaktadır:

```
class TemperatureInput extends React.Component {
  constructor(props) {
    super(props);
    this.handleChange = this.handleChange.bind(this);
    this.state = {temperature: ''};
  }
  handleChange(e) {
    this.setState({temperature: e.target.value});
  }
  render() {
    const temperature = this.state.temperature;
    // ...
```

Ancak, bu iki girişin birbiriyle senkronize olmasını istediğimiz için Santigrat input'unu güncellediğimizde, Fahrenayt input'u dönüştürülen sıcaklığı yansıtmalıdır ya da aynı şekilde Fahrenayt input'unu güncellediğimizde Santigrat input'u dönüştürülen sıcaklığı yansıtmalıdır.

React'te state paylaşımı, state'i, ihtiyacı olan bileşenlerin en yakın ortak üst elemana taşıyarak gerçekleştirilir. Buna "lifting state up" yani state'i yukarı taşıma denir. Yerel state'i TemperatureInput'dan kaldırıp, Calculator bileşinine taşıyacağız.

Calculator bileşeni paylaşılan state'e sahip olacağı için, bu bileşen her iki input'ta geçerli sıcaklık değeri için "source of truth" yani gerçek veri kaynağı olacaktır. Bu şekilde Calculator, her iki input'a birbirleriyle tutarlı değerlere sahip olma talimatını verebilecektir. Her iki TemperatureInput bileşeninin prop'ları üst eleman olan Calculator bileşeninden geldiği için, her iki input değeri her zaman senkronize olacaktır.

Nasıl çalıştığını şimdi adım adım inceleyelim.

İlk olarak, TemperatureInput bileşeninde this.state.temperature'u this.props.temperature olarak değiştireceğiz. Şimdilik, this.props.temperature hali hazırda varmış gibi düşünelim, ancak ileride bunu Calculator bileşeninden göndermemiz gerekecek.

```
render() {
  // Önceki hali: const temperature = this.state.temperature;
  const temperature = this.props.temperature;
  // ...
```

Prop'ların salt okunur olduğunu biliyoruz. temperature yerel state'te bulunuyorken, TemperatureInput bileşeni bu değeri değiştirebilmek için this.setState()'i çağırabiliyordu. Ancak şimdi, temperature değeri üst elemandan prop olarak geldiği için TemperatureInput bileşeninin artık temperature'ın üzerinde bir kontrolü kalmadı.

React'te, bu durum genellikle bileşen oluşturulurken, "kontrollü" bileşen biçiminde yapılarak çözülür. DOM'da <input> öğesinin hem value hem de onChange prop'unu kabul etmesi gibi, oluşturduğumuz TemperatureInput bileşeni hem temperature hem de onTemperatureChange prop'larını üst eleman olan Calculator bileşeninden kabul alabilecektir.Şimdi, TemperatureInput kendi temperature'unu güncellemek ya da değiştirmek istediğinde this.props.onTemperatureChange'i çağıracaktır.

```
handleChange(e) {
  // Önceki hali: this.setState({temperature: e.target.value});
  this.props.onTemperatureChange(e.target.value);
  // ...
```

> **Not:**
> Bileşenlerdeki `temperature` veya `onTemperatureChange` prop isimlerinin özel bir anlamı bulunmamaktadır. Onlara herhangi farklı bir şey diyebilirdik, örneğin onları `value` ve `onChange` gibi genel konvansiyonla isimlendirebilirdik.

onTemperatureChange prop'u, Calculator bileşeni tarafından temperature prop'u ile birlikte verilecektir. Değer değişimini kendi yerel state'ini değiştirerek halledecek, böylece her iki input yeni değerlerle beraber gösterilecektir. Birazdan Calculator'un yeni implementasyonuna birlikte bakacağız.

Calculator'deki değişikliklere başlamadan önce, hızlı bir şekilde TemperatureInput bileşenimizdeki yaptığımız değişikliklere bakalım. Yerel state'i bileşenden kaldırdık ve this.state.temperature okumak yerine artık this.props.temperature okumaktayız. Ayrıca değer değiştirmek için this.setState() çağırmak yerine, Calculator tarafından sağlanan this.props.onTemperatureChange() çağırmaktayız:

```
class TemperatureInput extends React.Component {
 constructor(props) {
  super(props);
  this.handleChange = this.handleChange.bind(this);
 }
 handleChange(e) {
  this.props.onTemperatureChange(e.target.value);
 }
 render() {
  const temperature = this.props.temperature;
  const scale = this.props.scale;
  return (
   <fieldset>
    <legend>{scaleNames[scale]} cinsinden sıcaklık girin:</legend>
    <input value={temperature}
           onChange={this.handleChange} />
   </fieldset>
  );
 }
}
```

Şimdi Calculator bileşenimize dönebiliriz.

Input'un temperature ve scale değerlerini bileşenimizin yerel state'inde saklayacağız. Böylelikle input'larımızdan "state'i taşımış" olduk. Bu bize her iki input değeri için "source of truth" yani doğru veri kaynağını sağlamış olacaktır. "Source of truth" her iki input değerini senkron bir şekilde göstermek için bilmemiz gereken tüm verilerin asgari temsilidir.

Örneğin, Santigrat input elemanına 37 yazarsak, Calculator bileşenindeki state aşağıdaki gibi olacaktır:

```
{
  temperature: '37',
  scale: 'c'
}
```

Eğer daha sonra Fahrenayt input elemanını 212 olarak değiştirirsek, Calculator bileşinindeki state aşağıdaki gibi olacaktır:

```
{
  temperature: '212',
  scale: 'f'
}
```

Her iki input değerini de kaydedebilirdik ancak bu gereksiz görünmektedir. En son değiştirilen input değerini ve onun birimini kaydetmek yeterlidir. Daha sonra temperature ve scale değerlerine bağlı olarak diğerinin değerini hesaplayabiliriz.

Input'lardaki değerler senkron bir şekilde kalmaktadır, çünkü değerleri aynı state üzerinden hesaplanmaktadır:

```
class Calculator extends React.Component {
  constructor(props) {
    super(props);
    this.handleCelsiusChange = this.handleCelsiusChange.bind(this);
    this.handleFahrenheitChange = this.handleFahrenheitChange.bind(this);
    this.state = {temperature: '', scale: 'c'};
  }
  handleCelsiusChange(temperature) {
    this.setState({scale: 'c', temperature});
  }
  handleFahrenheitChange(temperature) {
    this.setState({scale: 'f', temperature});
  }
  render() {
```

```
    const scale = this.state.scale;
    const temperature = this.state.temperature;
    const celsius = scale === 'f' ? tryConvert(temperature, toCelsius) : temperature;
    const fahrenheit = scale === 'c' ? tryConvert(temperature, toFahrenheit) : temperature;
    return (
      <div>
        <TemperatureInput
          scale="c"
          temperature={celsius}
          onTemperatureChange={this.handleCelsiusChange} />
        <TemperatureInput
          scale="f"
          temperature={fahrenheit}
          onTemperatureChange={this.handleFahrenheitChange} />
        <BoilingVerdict
          celsius={parseFloat(celsius)} />
      </div>
    );
  }
}
```

Şimdi, hangi input'u değiştirdiğinizin önemi olmaksızın, Calculator bileşenindeki this.state.temperature ve this.state.scale güncellenmektedir. Input'lardan herhangi birisinin değeri olduğu gibi alınmaktadır, böylelikle kullanıcı girdisi korunmaktadır, ve diğer input değeri girilen değere göre yeniden hesaplanmaktadır.

Herhangi bir input'a değer girildiğinde ne olduğunu özetleyelim:

- React, DOM'daki <input> üzerinde onChange olarak belirtilen fonksiyonu çağırır. Bizim örneğimizde, TemperatureInput bileşenindeki handleChange metodudur.

- TemperatureInput bileşenindeki handleChange metodu this.props.onTemperatureChange()'i yeni girilen değerle çağırır. TemperatureInput bileşenindeki prop'lar, onTemperatureChange ile beraber, üst eleman olan Calculator tarafından verilmektedir.

- Calculator bileşeninde bulunan Santigrat cinsindeki TempratureInput bileşeninin onTemperatureChange fonksiyonunu handleCelsiusChange metodu olarak belirledik. Aynı şekilde Fahrenayt için ise handleFahrenheitChange olarak belirledik. Calculator'deki bu iki fonksiyondan herhangi biri, değişen input'a bağlı olarak çağrılır.

- Bu metodlarda, Calculator bileşeni React'e kendisini tekrar ekranda sunabilmek için this.setState()'i yeni input değeri ve input'un bağlı olduğu ölçüm birim değeri ile çağırır.

- React Calculator bileşeninin render metodunu çağırarak yeni UI'ın nasıl sunacağını öğrenir. Input'lardaki her iki değer kendi birimlerine göre tekrar hesaplanır. Sıcaklık dönüşümü bu adımda gerçekleşir.

- React her bir TemperatureInput bileşeninin render metodunu Calculator bileşeni tarafından belirlenen yeni prop'lar ile çağırır. Bu şekilde ekrana bu input'ları nasıl göstereceğini öğrenir.

- React BoilingVerdict bileşeninin render metodunu çağırır. Çağırırken Santigrat cinsinden olan değeri prop olarak bileşene gönderir.

- React DOM, DOM'da suyun kaynayacağını ya da kaynamayacağını gösteren bir mesajla günceller. Ayrıca input değerlerini güncel hesaplanan değerler ile ekrana yansıtır.

Her güncelleme aynı adımlardan geçer, böylece input'lar senkronize kalır.

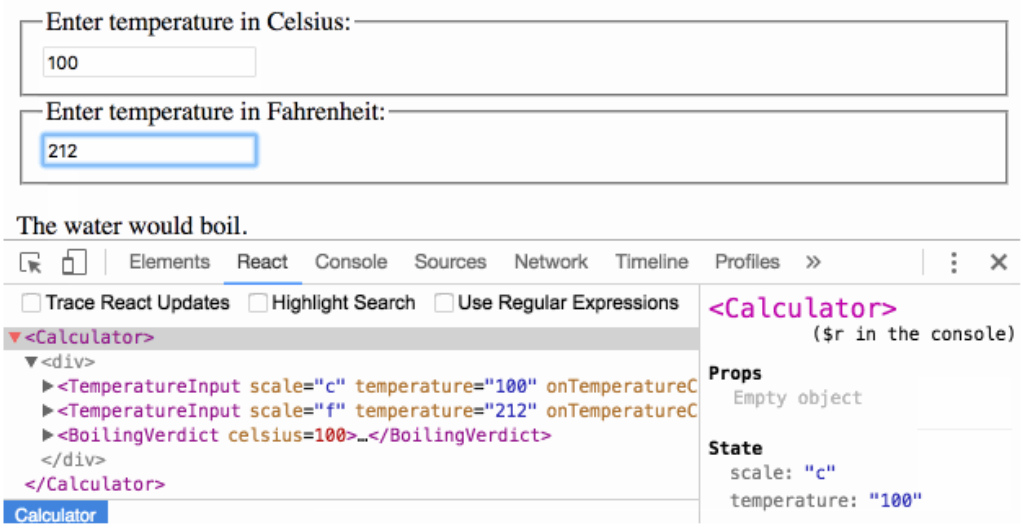
* Neler Öğrendik

React uygulamasında değişen veriler için tek bir gerçek veri kaynağı olmalıdır. Genelde, state onu kullanacak olan bileşene eklenir. Daha sonra, diğer bileşenlerde bu state'e ihtiyaç duyarsa state'i en yakın ortak üst elemana taşıyabilirsiniz. State'i farklı bileşenler arasında senkronize etmektense, yukarıdan-aşağıya veri akışı'ın kullanabilirsiniz.

State taşıma daha çok genel hatlarıyla kod yazmayı ve iki yönlü bağlama yaklaşımını gerektirmektedir. Bu işin getirisi hataları bulup ayıklamak için daha az iş gerektirmektedr. Herhangi bir state, bazı bileşenlerde "yaşadığından" ve bileşenler tek başına onu değiştirebildiğinden, hataların kapsam alanı büyük ölçüde azalmaktadır. Ek olarak, kullanıcı tarafından girilen input değerlerini reddetmek veya dönüştürmek için herhangi bir özel mantık uygulayabilirsiniz.

Eğer herhangi bir şey prop'tan veya state'ten türetilebilirse, büyük ihtimalle o state'te olmamalıdır. Örneğin, hem celsiusValue hem de fahrenheitValue tutmaktansa, sadece en son değiştirilen temperature ve onun scale'ini tutarız. Diğer input'un değeri böylelikle bu değerler ile render() metodunda hesaplanabilir. Bu bize, kullanıcı input girişinde herhangi bir hesaplama hassasiyetini kaybetirmeden yuvarlama işlemini diğer input'a uygulamamızı veya silmemizi sağlar.

Kullanıcı arayüzünde yanlış bir şey gördüğünüzde, React Developer Tools'u prop'ları incelemek ve state'i güncellemekle sorumlu olan bileşeni bulabilmek için kullanabilirsiniz. Bu, size hataları kaynağına kadar izleme olanağı sağlar:


Enter temperature in Celsius:
100
Enter temperature in Fahrenheit:
212
The water would boil.
Elements
React
Console
Sources
Network
Timeline
Profiles
Trace React Updates
Highlight Search
Use Regular Expressions
<Calculator>

<TemperatureInput scale="c" temperature="100" onTemperatureC
<TemperatureInput scale="f" temperature="212" onTemperatureC
<BoilingVerdict celsius=100>…</BoilingVerdict>

</Calculator>
Calculator
<Calculator>
($r in the console)
Props
Empty object
State
scale: "c"
temperature: "100"