

# React ile statik sayfa görüntüleme

Üstün Özgür

2016-12-16 Cuma

# React ile statik sayfa görüntüleme

## *render* metodu

- render'da bir JSX ifadesi döneriz
- Sıradan HTML elemanları ya da birleşik (kompozit) React bileşenleri

# JSX Hakkında

- Bir HTML ya da XML sürümü değil
- Normal JavaScript'e dönüştürülür
- `<div ad="foo" soyad="bar">içerik</div>`
- `React.createElement('div', ({ad: "foo", soyad: "bar"}, "icerik"))`
- Özellikler *props* adı verilen bir nesnede (object) toplanır
- Çocuklar sonraki argümanlar olarak geçirilir
- `fonksiyon(props, çocuklar)`
- İç içe geçebilir:: Birleşik (kompozit) bileşenler

# Daha karmaşık bir örnek

## JSX

```
<div ad="foo">  
  <span>Bar</span>  
  <span>Baz</span>  
</div>
```

## JS

```
React.createElement('div', {ad: 'foo'},  
  React.createElement('span', {}, "Bar"),  
  React.createElement('span', {}, "Baz"))
```

# JSX: 5 Dakika Verin

- JS içinde HTML mi?
- HTML içinde JS mi?
- Aslında JS ve HTML karışımı değil, sadece JS

# Ana Özellikler

- Sadece tek bir eleman dön
  - Bir fonksiyonun tek bir çıktısı olması gibi
- Bütün elemanlar kapanmalı ya da kendiliğinden kapanır olmalı
  - `<div></div>`, `<img/>`, `<input/>`
- JSX ifadeleri koymak için süslü parantezler
  - `<div>{2 + 3}</div>` Geçerli JSX

# Ana Özellikler

```
var icTaraf = <div>Merhaba</div>
```

```
var disTaraf = <div>{icTaraf} {icTaraf}</div>
```

- String birleşimi değil. icTaraf herhangi bir değişken gibi

```
var icerikler = [];
```

```
icerikler.push(icTaraf);
```

```
icerikler.push(icTaraf);
```

```
var disTaraf = <div>{icerikler}</div>
```

# Ana Özellikler

```
var isimler = ["Ali", "Ahmet", "Mehmet"];

var icEleman = isimler.map(function (ad) {
    return <div>Merhaba {ad}</div>;
});

var disEleman = <div>{icEleman}</div>
```



# Ana Özellikler

```
var icerik;  
  
if (this.props.renk == "kirmizi") {  
    icerik = <div>Renk kırmızı.</div>  
}  
  
dis = <div>{icerik}</div>
```

# Ana Özellik

- If ifadeleri statement'tır, expression değildir
- Ama üç terimliler expression:

```
a == 3 ? "a 3" : "a 3 değil"
```

## Örnek

```
outer = <div>{this.props.renk == "kırmızı" ?  
<div>Renk kırmızı</div>: null}</div>
```

```
outer = <div>{this.props.color == "kırmızı" &&  
  <div>Renk kırmızı</div>}</div>
```

- Okumayı zorlaştırabilir

# JSX içinde CSS

- JSX aslında arka planda sadece JS

## class

- class kelimesi JS içinde rezerve (anahtar kelime)
- Bunun yerine className kullanılır
- `<div class="foo bar">` suna dönüşür: `<div className="foo bar">`

## Stiller

- HTML'deki stiller string'dir. CSS'te, JS'teki nesnelere benzer bir sözdizim kullanılır.
- JSX'te stiller gerçek JS nesneleridir.

## JSX içinde CSS (devam)

```
<div style="color: red; background-color: yellow">
```

- String bir JS nesnesine dönüştürülür

```
{color: 'red', backgroundColor: 'yellow'} <div  
style={{color: 'red', backgroundColor: 'yellow'}}>
```

- Dikkat: İki tane { : Birincisi JSX için, ikincisi JS nesnesi için
- Tire yok: Bunun yerine sonraki harfi büyük harfe çevir  
background-color yerine backgroundColor

# JSX içinde CSS (devam)

- JSX'te sayılar: CSS'te çoğu birim px'tır, bu yüzden otomatik olarak eklenir

`fontSize: '12px'`

ifadesi şu şekilde yazılabilir

`fontSize: 12`

## JSX içinde CSS (devam)

- Her zaman beklediği gibi çalışmaz. Örneğin lineHeight için ana birim 'px' değil 'em'dir

lineHeight: 18 ifadesi lineHeight: "18em" anlamına gelir

Şunu yazın:

```
lineHeight: '18px'
```

# Var olan bir HTML'i React'e çevirmek

- 1 App adında bir React bileşeni oluşturun. Hazır HTML'i alın ve render metodunda bunu dondurun.
- 2 Gerekli CSS değişikliklerini yapın. class: className. stiller: nesne
- 3 Bütün etiketlerin (taglerin) doğru kapandığından emin olun.
- 4 App.js dosyasını HTML'den çağırın, HTML'e React bileşeninizin yerleştirileceği ana HTML bileşenini koyun ve React bileşenini 'React.render' ile yerleştirin (mount). (Doğrudan body'ye de yerleştirilebilir.)
- 5 2. ve 3. adımlar için dönüştürücüyü (transpiler) izleme (watch) modunda çalıştırın. Hata vermeyene kadar değişiklikleri yapın.

# JSX'i JavaScript'e Çevirmek İçin Araçlar

- Dönüştürücü (Transpilation)
- **react-tools**'u şununla yükleyin: `npm install -g react-tools`
- İki mod: Tek dosya girişi ya da klasör girişi



# Basit Bir Örnek

```
var MerhabaDunya = React.createClass({  
  render: function () {  
    return <div>Merhaba Dünya</div>  
  }  
})
```

HTML kısmında ana bileşen için bir içerici (container)

```
<div id="app"></div>
```

Yerleştirmek (monte) için:

```
ReactDOM.render(<MerhabaDunya/>, document.getElementById('ap
```

# Alıştırma 1: React ile Başlangıç: 5-10 dakika

İpucu: Makefile dosyasına bakabilirsiniz.

a - create-react-app ile yeni proje oluşturun.

b - 3 farklı yöntemle ekrana HelloWorld yazdırın:

React.createClass, function, class HelloWorld

c - JSX kullanmadan React.createElement kullanarak oluşturun.

- JSX'te JavaScript expressionlarini embed edebiliriz.

```
function formatName(user) {  
  return user.firstName + ' ' + user.lastName;  
}
```

```
const user = {  
  firstName: 'Harper',  
  lastName: 'Perez'  
};
```

```
const element = (  
  <h1>  
    Hello, {formatName(user)}!  
  </h1>  
);
```

# Daha Karmaşık bir örnek: ToDo Uygulaması

- `examples/02_todo_props/mockup.html` alıp React'e çevirin.
- Çevrimiçi HTML-JSX dönüştürücü :  
`https://facebook.github.io/react/html-jsx.html`
- Komut satırı versiyonu:  
`https://www.npmjs.com/package/htmltojsx`

## Alıştırma 2:

a - 02\_convert\_mockup/mockup.html'da verilen HTML'i React'e çevirin. Dikkat: Burada iki farklı ana kısım var, o yüzden iki ayrı kök eleman (node) olacak.

b - ToDo uygulamasını küçük bileşenlere parçalayın. Şu adlarda bileşenler oluşturun: SearchBar, Todos, TodoItem, Footer.