REACT JSX'e Giriş

Aşağıdaki değişken tanımını ele alalım:

```
const element = <h1>Hello, world!</h1>;
```

Burada acayip bir şekilde yazılan söz dizimi ne bir string ne de HTML'e aittir.

Bu söz dizimi JSX olarak adlandırılır ve JavaScript'in bir söz dizimi uzantısıdır. Arayüzün nasıl görünmesi gerektiğini tanımlamak için, React ile birlikte JSX'i kullanmanızı tavsiye ederiz. JSX, bu bağlamda size kullanıcı arayüzü oluşturmayı sağlayan bir şablon dili gibi görünebilir. Fakat JavaScript'i tüm gücüyle kullanmanızı sağlayacak yeteneklerle donatılmıştır.

JSX, React elementleri oluşturmanızı sağlar. Sonraki bölümde bu elementlerin nasıl DOM'a render edileceğine değineceğiz. Aşağıdaki bölümlerde, JSX'e başlangıç yapabilmeniz için gerekli bilgiler mevcuttur.

Neden JSX?

React, render edilecek kısımların yer aldığı kodlar ile diğer arayüz kodlarının birbirinden ayrılmasını teşvik eder. Diğer arayüz kodlarına örnek verecek olursak: onClick gibi olayların nasıl işleneceği, state'in zaman içerisinde nasıl değiştirileceği ve gösterim için verilerin nasıl hazırlanacağıdır.

HTML ve JavaScript kodlarının ayrı dosyalarda tutularak teknolojilerin birbirinden yapay bir şekilde ayrılması yerine React, hem HTML hem de JavaScript kodu barındıran ve birbirine gevşek bir şekilde bağlı olan bileşenler (components) sayesinde ilgili işlerin ayrılmasını sağlar. İlerleyen bölümlerde bileşenlere tekrar değineceğiz. Fakat hala HTML kodlarının JavaScript içerisine konulması sizi rahatsız ediyorsa bu video sizi ikna edecektir.

React, JSX kullanımını zorunlu tutmaz. Fakat birçok geliştirici, JavaScript kodu içerisinde arayüz ile ilgili çalışırken JSX'in kullanılmasının, görsel anlamda yardımcı olduğunu düşünüyor. Ayrıca JSX, React için daha anlaşılır hata ve uyarı mesajlarının görüntülenmesini sağlıyor.

Bu kısımda anlaştıysak, artık JSX ile React kullanımına geçebiliriz.

JSX İçerisinde JavaScript Kodlarının Kullanımı

Aşağıdaki örnekte ilk satırda name değişkenini tanımlıyoruz. Ardından bu değişkeni süslü parantezler ile sarmalayarak JSX kodu içerisinde kullanıyoruz:

```
const name = 'Josh Perez';

const element = <h1>Hello, {name}</h1>;
```

JSX'te süslü parantezler arasına dilediğiniz herhangi bir JavaScript ifadesini yazabilirsiniz. Örneğin, 2 + 2, user.firstName, veya formatName(user) gibi JavaScript ifadelerini kullanabilirsiniz.

Aşağıdaki örnekte, bir JavaScript fonksiyonun çağrısının sonucu JSX içerisine gömülmektedir. Yani formatName(user), <h1> elemanının içerisine konulmaktadır.

```
function formatName(user) {
  return user.firstName + ' ' + user.lastName;
}
const user = {
  firstName: 'Harper',
  lastName: 'Perez'
};
const element = (
  <h1>
    Hello, {formatName(user)}!
  </h1>
);
```

Okunabilirliği arttırmak için JSX kodunu birkaç satır halinde yazdık. Buradaki gibi, JSX kodunu birçok satır halinde yazarken, kodu parantezler ile sarmalamanızı öneririz. Bu sayede otomatik olarak noktalı virgül eklenmesi ile oluşan birçok hatanın önüne geçebilirsiniz.

* JSX de bir JavaScript İfadesidir

Oluşan derlemenin ardından JSX ifadeleri, sıradan JavaScript fonksiyon çağrılarına dönüşür ve bu fonksiyonlar JavaScript nesnelerini işleyecek şekilde çalışırlar.

Bu sayede if ifadelerini ve for döngülerini JSX içerisinde kullanabilir, değişkenlere atama yapabilir, fonksiyona parametre olarak geçebilir ve fonksiyondan geri döndürebilirsiniz:

```
function getGreeting(user) {
  if (user) {
    return <h1>Hello, {formatName(user)}!</h1>;
```

```
  }
  return <h1>Hello, Stranger.</h1>;
}
```

JSX ile Özelliklerin Tanımlanması

Bir HTML elemanı için string ifadelerini çift tırnak içerisinde atayabilirsiniz:

```
const element = <a href="https://www.reactjs.org"> link </a>;
```

Ayrıca bir JavaScript ifadesini, elemanın özelliği olarak tanımlamak için süslü parantezler ile sarmalayabilirsiniz:

```
const element = <img src={user.avatarUrl}></img>;
```

Bir JavaScript ifadesini, herhangi bir özellik içerisine yazarken çift tırnak kullanmayınız. String için çift tırnak, JavaScript ifadeleri için süslü parantezler kullanmalısınız. Aynı özellik için hem çift tırnak hem de süslü parantez kullanmayınız.

> **Uyarı:**
> JSX ifadeleri, HTML'den ziyade JavaScript'e daha yakındırlar. Bu nedenle React DOM, özellik isimlendirme için HTML'deki gibi bir isimlendirme yerine `camelCase` isimlendirme standardını kullanmaktadır.
>
> Örneğin JSX içerisinde `class` özelliği `className`, ve `tabindex` özelliği de `tabIndex` olarak yazılmalıdır.

JSX ile Alt Elemanların Tanımlanması

Eğer bir HTML etiketinin içeriği boş ise, XML'deki gibi /> kullanarak etiketi kapatabilirsiniz:

```
const element = <img src={user.avatarUrl} />;
```

JSX etiketleri alt elemanlar da içerebilir:

```
const element = (
  <div>
    <h1>Hello!</h1>
    <h2>Good to see you here.</h2>
  </div>
);
```

* JSX, Injection Saldırılarını Engeller

JSX'te kullanıcı girdisini koda direkt olarak gömmek güvenlidir:

```
const title = response.potentiallyMaliciousInput;

const element = <h1>{title}</h1>;

// Bu kullanım güvenlidir:
```

Çünkü varsayılan olarak React DOM, render işlemi öncesinde gömülen değerlerdeki <, & gibi bazı özel karakterleri < ve & olacak şekilde dönüştürür. Böylece uygulama içerisinde, kullanıcının yazabileceği kötü amaçlı kodların enjekte edilmesi engellenmiş olur. Render işlemi öncesi her şey string ifadeye dönüştürüldüğünden dolayı, XSS saldırıları engellenmiş olur.

* JSX, JavaScript Nesnelerini Temsil Eder

Babel derleyicisi, JSX kodlarını React.createElement() çağrılarına dönüştürür.

Bu nedenle aşağıdaki iki kod örneği de aynı işlemi gerçekleştirir:

```
const element = (
  <h1 className="greeting">
    Hello, world!
  </h1>
);

const element = React.createElement(
  'h1',
  {className: 'greeting'},
  'Hello, world!'
);
```

React.createElement() çağrısı, hatasız kod yazmanız için size yardımcı olacak birtakım kontrolleri gerçekleştirir. Aslında yaptığı şey, aşağıdaki gibi bir nesne oluşturmaktadır:

```
// Not: bu yapı basitleştirilmiştir

const element = {
  type: 'h1',
```

```
  props: {
    className: 'greeting',
    children: 'Hello, world!'
  }
};
```

Bu nesnelere “React elementleri” adı verilir. Bunu, ekranda görmek istediğiniz kullanıcı arayüzünün kodlar ile tasvir edilmesi gibi düşünebilirsiniz. React, bu nesneleri okuyarak DOM’ı oluşturur ve arayüzü günceller.

Sonraki bölümde, React elementlerinin DOM’a render edilmesi işlemini daha detaylı bir şekilde ele alacağız.

> **İpucu:**
> ES6 ve JSX kodlarının uygun şekilde renklendirilmesi için, kod editörünüzde “Babel” dil tanımlamalarını kullanmanızı öneririz.