AngularJS

Burak Tokak

İçindekiler

1

AngularJS Nedir, Neden ve Nasıl Kullanılır?

Bu Bölümde

Bu bölümde, AngularJS'in diğer teknolojiler arasındaki konumuna bakacağız. Şimdiye kadar framework kullanmamış okurlarımız için, framework kullanma nedenlerimizi ve kitap devamında yaygın olarak kullanacağımız bazı anahtar kelimelerin anlamlarından bahsedeceğiz.

GİRİŞ

AngularJS, Google tarafından geliştirilmesi denetlenen, açık kaynak kodlu ve bağımsız geliştiricilerin önerileri ile iyileştirilip/geliştirilen bir JavaScript MVC veya MVVM Framework'üdür. İstatistiksel olarak en çok kullanılan, JavaScript'e MVC yapısı sağlayan kütüphanelerin başında gelir.

AngularJS, başarısını tek sayfa web uygulamalarında yakalamıştır. Geliştiricilere sağladığı etkili araçlar ile tek sayfa web uygulamalarında ve özellikle Cordova gibi web teknolojileri ile çoklu platform mobil uygulama geliştirmemizi sağlayan framework'ler üzerinde hem zaman hem de verimlilik olarak büyük artılar sağlamaktadır.

ANGULARJS ÖĞRENMEK İÇİN BİLMEMİZ GEREKEN TEKNOLOJİLER

AngularJS üzerinde geliştirme yapmak için öncelikli olarak bilmemiz gereken şeylerin başında basit HTML ve DOM yapısı geliyor diyebiliriz. Buna ek olarak AngularJS, JavaScript ile uygulanması nedeni ile Expression, Obje ve modül sistemleri olarak JavaScript'in temel syntax'ını kullanmaktadır. Yapacağımız çoğu mantıksal ve navigatif işlem için de JavaScript kullanacağımızdan orta seviyenin üstünde yalın JavaScript bilgisi gereklidir.

Tüm bunlara ek olarak belirtmeyilim ki, özelliklerinin tamamı olmasa da orta seviyede CSS bilgisi ve yapısının nasıl çalıştığı konusunda bilgi sahibi olunmalıdır.

> **DİKKAT** Şahsen, AngularJS gibi yalın JavaScript ve temel DOM manipülasyonu mantığı üzerinde büyük bir değişiklik yapan bir kütüphane kullanmaya başlamadan önce yalın JavaScript'in veya jQuery gibi bir kütüphanenin, DOM ile nasıl iletişim kurduğunu ve manipülasyonlar yaptığını en azından mantık olarak kavramış olmanızı şiddetle öneriyorum.

Yeni başlayan çoğu geliştiricilerin yaptığı en büyük hatalardan birisinin, kullandığı programlama dilinin kendi akış mantığını öğrenmeden ve çalışma kalıplarını çözmeden, bu teknoloji için geliştirmiş mantığı tamamen değiştiren bir kütüphane ya da framework kullanmaya başlıyor olmalarıdır. Bu durum hem programlama mantığına hem de framework'lerin oluşturulma amacına ters düşmektedir.

Ek olarak JavaScript fonksiyonları, **event** (*olay*) ve **hata raporlama** sistemleri konusunda ileri seviye bilgi sahibi olmanız, AngularJS'de yolunuzu bulmanıza büyük bir katkı sağlayacaktır.

AngularJS Çalışma Mantığı ile İlgili Bilinmesi Gerekenler

MVC

MVC (Model-View-Controller) yapısı, en çok kullanılan yazılım tasarımlarından birisidir. Bu tasarımın ana mantığı, sistemi görünüm (*view*), kontrol eden (*controller*) ve işleyen (*model*) bileşenlerine ayırarak tasarlamak ve programlamaktır.

MVVM

MVVM (Model-View-ViewModel) yazılım tasarımı, çift taraflı bağlamayı mümkün kılan, uygulamanın veri-mantık kısmı ile görünüm (view) kısmının senkron bir şekilde sunulmasını sağlar. MVVM'de, görünümdeki bir objeye bağlanan bir değişken, logic (mantık) tarafında değiştiğinde bu değişiklik otomatik olarak görünümdeki objenin de değişmesini sağlar. Bu sayede geliştiririci, görünümün ne zaman güncellenmesi gerektiği ile değil, asıl önemli olan mantık kısmı ile ilgilenecektir.

Çift Taraflı Bağlama (Two-Way Data Binding)

AngularJS'in en yenilikçi özelliklerinin başında gelen çift taraflı veri bağlama, MVVC tasarımının bir ürünüdür. Uygulamanın mantık tarafındaki `scope` objesi tarafından tutulan alt objeler, yararlı olarak değiştirildiğinde bu değişiklik görünümü (HTML'de scope verisinin bağlandığı elementi)'de etkiler veya değiştirir.

Scope

Scope AngularJS'de çift taraflı veri bağlama yapmamızı sağlayan, görünüm (*view*) tarafının erişilebilirliği olan değişkenlerin sunulduğu servise AngularJS için `scope` veya `$scope` servisi diyoruz. Akılda kalması için görünüm tarafından kullanılabilen değişken skopu olarak da tanımlayabiliriz.

> **NOT** Kitabımızın devamında `$scope` servisinden daha ayrıntılı bir şekilde bahsedeceğiz.

DOM

DOM (Document Object Model), tarayıcıda görsel olarak sunulan elementlerin özellik ve aksiyonların bir obje olarak kabul edildiği HTML, CSS ve JavaScript'in arasında bir protokol oluşturan modelin genel adıdır. DOM modeline göre her HTML elementi, CSS veya JavaScript üzerinde seçilebilir birer objedir. Tüm bunlara ek olarak DOM objeleri, hiyerarşik olarak birbiri ile alakadardır.

MEAN Stack

AngularJS teknolojisinin de içinde bulunduğu, en popüler ve çok kullanılan web teknolojisi istifidir. MEAN, veritabanı olarak MongoDB, web sunucusu olarak Express, Front-end teknolojisi olarak AngularJS, Backend teknolojisi olarak Node.js kullanmaktadır.

Özellikle MongoDB'nin sağladığı JSON formatındaki verinin Node.js ve AngularJS (JavaScript) tarafından rahat ve verimli bir şekilde okunuyor olması ve bu yapının yatayda genişlemesi gereken büyük çaplı projeler için kullanılması açısından son zamanlarda büyük bir kullanım alanı bulunmaktadır. Çoğu web üzerinde ürün çıkartan teknoloji start-up'larında bu istif popüler olarak kullanılıyor.

Tek Sayfa Web Uygulaması

Tek sayfa web uygulamaları, özellikle AngularJS gibi işlemlerini tek HTTP isteği yaptıktan sonra sayfadaki görünüm elementlerini değiştiren framework'lerin kullanımı ile popülerleşmiştir. Sayfada gerçekleştirilecek çoğu mantıksal işlemlerin bir backend dili yerine client üzerinde çalışan JavaScript tarafından gerçekleştirilen bir uygulama geliştirme tarzıdır denilebilir.

Tek sayfa web uygulamalarında, sunucudan alınması gereken yeni verilerin iletişimi genellikle backend tarafından üretilen verinin HTTP aracılığı ile bir RESTful API tarafından AngularJS'e geçirilmesi sayesinde gerçekleştirilir.

NOT

RESTful API, sunucuda backend tarafından oluşturulmuş ve HTTP aracılığı ile web uygulamalarına veritabanından istenilen verinin sunulmasını için kullanılan bir API katmanı olarak düşünülebilir. Bir Restful API, gelen istek karşılığı JSON verisi döner ve bu verinin Front-end tarafında kullanılması gerekir. Daha ayrıntılı bir açıklamayı kitabımızın ilerleyen bölümlerinde ve `$http` servisi konusunda değineceğiz.

Ek olarak tek sayfa uygulaması olmasına rağmen, linkler üretebilen AngularJS uygulamaları da kitabımızın devamında bahsedeceğimiz bir konu olacak.

Modüler Programlama

Programlama ve genel proje yönetimini kolaylaştırmak veya çalışılan proje üzerinde birden fazla kişinin çalışması gerektiği durumlara uygun programlama yaklaşımıdır. Buna göre uygulamanın-projenin birbirinden ayrılabilir bölümleri modüllere ve bu modüller de alt modüllere ayırılarak geliştirme ortamı daha sürdürülebilir ve test edilebilir hale getirilir.

3

AngularJS'de Modül ve Kontrol Metodları

Bu Bölümde

Kitabımızın ilk iki bölümü ile AngularJS'in çalışma mantığı ve basit birkaç uygulama oluşturma konusunda temel bir bilgi tabanı oluşturduk. İlk iki bölüm boyunca kitabımızdaki uygulamalara bir modül atamamıştık.

Bu bölümde, AngularJS uygulamalarımız için bir modül oluşturup bu modül sayesinde uygulamamıza kontrol fonksiyonları, direktif tanımlayıcılar ve servisler atamayı öğreneceğiz.

MODÜL KAVRAMI VE UYGULAMA MODÜLLERİ

Modüller, bir AngularJS uygulamasını tanımlamak için kullanılan objelerdir denilebilir. Bir modül, uygulamalar için uygulamanın farklı bölümlerini içinde barındıran bir taşıyıcı olarak düşünülebilir. Ayrıca bir modülün en büyük görevi, uygulama kontrol fonksiyonlarını taşımasıdır.

> **NOT** AngularJS'de modül sistemi olmasının en büyük faydasının dış modülleri rahat şekilde kendi uygulama modülümüzle birleştirmeyi sağlamak olduğunu söyleyebiliriz.

Uygulama içerisinde kullanılan her kontrol metodu, bir modüle ait veya diğer bir deyiş ile bir modül tarafından taşınıyor olmalıdır.

ANGULARJS UYGULAMA MODÜLÜ OLUŞTURMAK

Bir uygulama modülü oluşturmak için `angular.module()` fonksiyonunu kullanıyoruz. Modüle verdiğimiz ismi `ng-app` direktifi ile uygulamanın çalışacağı taşıyıcı elementine belirtiyoruz.

Bir örnek ile görelim;

HTML Kodu

```
<!DOCTYPE html>
<html>
<meta charset="utf-8"/>
<script src="angular.min.js"></script>
<title>Angular Modüller ve Kontrol Fonksiyonları</title>
</head>
  <div ng-app="ngUygulamam" ng-init="degisken='Merhaba Dünya'">
    {{degisken}}
  </div>
  <script type="text/javascript">
  var uygulama = angular.module("ngUygulamam", []);
  </script>
</body>
</html>
```

Bu örnek, sayfaya beklendiği gibi `"Merhaba Dünya"` yazacaktır. Modülü oluşturmak ve bu modülü taşıyıcınıza bağlamak bu kadar basit. Dikkatinizi çektiyse, oluşan modül objesini ayrıca `uygulama` değişkenine atadık. Bundan sonra bu modüle kontrol metodu, direktifler, filtreler, servisler ve daha fazlasını **modüler** bir şekilde ekleyebiliriz.

NOT

`module()` metodu, modül ismi ve bu modülün çalışması için bağmlı olduğu (*dependency*) modüllerin bir array'ini (`[modul1, modul2]` vb.) olmak üzere 2 parametre alır. Bağımlılık olayının çalışma mantığına kitabımınızın devamında tekrar göz atacağız.

AngularJS Uygulamalarında Kontrol Metodu

Kontrol metodu, uygulamanın kök taşıyıcı HTML elementine `ng-controller` direktifi ile bağlanan ve oluşturduğu uygulama yüklendiğinde çağırılan fonksiyonu (*constructor*) ile, uygulamada kullanılacak değişken veya metodları `$scope` alt objesi aracılığı ile içerisinde oluşturan ve `View` kullanımına sunan AngularJS uygulamasının metodudur.

`controller`, aslına bakılırsa oluşturduğumuz `uygulama` objesinin bir metodudur. Basit bir örnek ile kullanımını açıklamaya çalışalım;

HTML Kodu

```
<div ng-app="ngUygulamam" ng-controller="ilkKontrolMetodum">
  {{degisken}}
</div>
<script type="text/javascript">
  var uygulama = angular.module("ngUygulamam", []);
  uygulama.controller('ilkKontrolMetodum', ['$scope', function($scope) {
    $scope.degisken = 'Meeerhabaaa Dünnyaaa!!';
  }]);
</script>
```

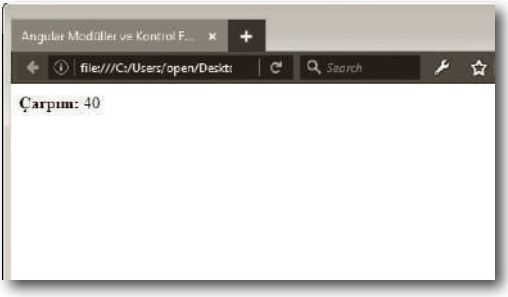
Örneğimizde ekranda `"Meeerhabaaa Dünnyaaa!!"` çıktısını göreceğiz. `uygulama` objesinin `controller` metodu ile `$scope` objesini fonksiyona geçirerek bu obje üzerinde yeni bir değişken oluşturup, yazıyı görünüm tarafına bağlamak suretiyle expression yansıması olarak `ng-app` taşıyıcı elementinin içerisinde yazdırdık.

Bu örnekte fonksiyon ile geçirilen `$scope` objesi, görünüm tarafından erişilebilen bir obje olarak düşünülebilir. Başlangıçta çağırılan bu kontrol metodu ile, `$scope` objesine görünümde kullanmak üzere değişkenler ve yeni metodlar tanımlayabiliyoruz.

Yeni bir örnek ile, $scope objesi üzerinde birden fazla değişken tanımlayıp bunları görünümde yansıtalım.

HTML Kodu

```
<div ng-app="ngUygulamam" ng-controller="ngKontrol">
  <b>Çarpım:</b> {{sayi1 * sayi2}}
</div>
<script type="text/javascript">
  var uygulama = angular.module("ngUygulamam", []);
  uygulama.controller('ngKontrol', ['$scope', function($scope) {
    $scope.sayi1 = 8;
    $scope.sayi2 = 5;
  }]);
</script>
```



Örneğimizde 2 farklı değişken, $scope üzerinden alınarak expression yansıması olarak sayfada çarpıldı.

HTML Kodu

```
<div ng-app="ngUygulamam" ng-controller="ngKontrol">
  <b>Çarpım:</b> {{ sayiArray[1] * sayiArray[3]}}
</div>
<script type="text/javascript">
  var uygulama = angular.module("ngUygulamam", []);
  uygulama.controller('ngKontrol', ['$scope', function(uygulamaObj) {
    uygulamaObj.sayiArray = [10, 5, 8, 9];
  }]);
</script>
```

6

AngularJS'de JSON Verileri ve Veri İletişimi

Bu Bölümde

Kitabımızın önceki bölümlerinde `$http` ile XHR isteklerini nasıl isteyeceğimizden bahsetmiştik. Fakat JSON verisini hem almak hem de işlemek konusunda çoğu geliştirici sıkıntı yaşayabiliyor.

Kitabımızın bu bölünde, `$http` servisi ile edindiğimiz JSON verilerini uygulamamızda en verimli ve kolay şekilde nasıl kullanacağımız konusu üzerine duracağız. Ve birkaç uygulama örneği ile uygulamalar içinde nasıl kullanıldığı konusunda bilgi sahibi olmanızı sağlayacağız.

JSON NEDİR? NOTASYON GÖSTERİMİ VE KURALLARI

JSON, birbirinden bağımsız olarak geliştirilmiş programlama dilleri ve teknolojileri arasında karşılıklı olarak verilerin anlaşılmasını sağlamak için belirlenmiş bir obje notasyonudur. 2 farklı obje tabanlı teknoloji arasında verilip alınan JSON verisi iki programlama dili tarafında da yazıldığında encode (çevirme) ve decode (çevirilmiş olanı anlama) fonksiyonları yardımı ile verimli bir veri iletişimi sağlanır.

NOT

JSON, (JavaScript Object Notation) kelimelerinin kısaltılmasından oluşur. Her ne kadar JavaScript üzerinde tanımlanmış objelerin text notasyonu olarak kullanılmaya başlansa bile, back-end teknolojilerinin obje yapılarını da bu notasyona indirgeyebiliyoruz.

JavaScript Obje Notasyonunun syntax'ını şu basit örnekler ile anlatalım;

Örneğin "sebze" ile ifade edebileceğimiz bir değişkenimiz olduğunu var sayalım ve bu değişkeni "patates" değerine eşitlemiş olalım, bu tip bir verinin JSON çıktısı şöyle olur;

JSON

```
{
  "sebze":"Patates"
}
```

NOT

JSON ile dönen değer bir obje veya bir array olabilir. Buna göre {} gösterimi ile başlayıp biten bir JSON karakter kümesi, objeyi temsil ederken (üstteki örnekte olduğu gibi) [] gösterimi ile başlayıp biten bir karakter kümesi, bir array'i temsil eder.

Örneğin bu sebzenin 100 gramındaki kalori, protein vb. gibi değişkenleri de bu objenin içine ekleyelim;

JSON

```
[
  "patates",
  'domates",
  "patlıcan"
]
```

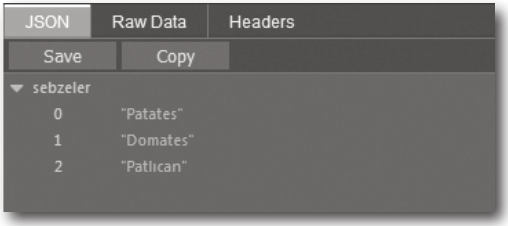
Gördüğünüz gibi, objenin alacağı birkaç değişkeni tanımlarken virgüller ile ayırıyoruz.

DİKKAT

Dikkat ederseniz, değer karşılığı string olan veriler için " " kullanırken, sayısal değerler için yalnızca `say1` değerini verdik. Burada dikkat edilmesi gereken, sayının " " ile verildiği durumlarda notasyon işlendiğinde sayı ile (sayıya cast edilmezse) aritmetik işlemler yapamayız.

JSON

```
{
  "sebzeler": ["Patates", "Domates", "Patlıcan"]
}
```



JSON gösterimi ile kodlanmış bir obje içerisinde değişken olarak objeler bulundurabilirsiniz. Sonraki örneğimizde bunu gösterelim;

JSON

```
{
    "sebzeler":[
        {"isim":"Patates","kalori":77,"pro":2,"yag":0.1,"karb":17},
        {"isim":"Domates","kalori":18,"pro":0.9,"yag":0.2,"karb":3.9},
        {"isim":"Patlıcan","kalori":25,"pro":2,"yag":0.2,"karb":6}
    ]
}
```

İki örnek önce gösterdiğimiz JSON ile göstilen objeyi, `sebzeler` isimli bir `array` içindeki elemanlara attık. Bu JSON gösteriminin şöyle bir hiyerarşik çıktısı olacak.

9

AngularJS'de Olay Tanımlayıcıları

Bu Bölümde

Bir önceki bölümümüzde, AngularJS tarafından form ve input'lar için sunulan özellikleri inceledik.

Bu bölümde, JavaScript olay tanımlayıcılarının AngularJS direktifleri üzerinde nasıl kullanıldığını ve AngularJS değişkenlerinin olaylara nasıl bağlanacağı üzerine küçük birkaç bilgi vereceğiz.

JAVASCRIPT OLAYLARI NEDİR?

Çoğu okuyucunun bilgi sahibi olduğunu düşündüğüm JavaScript olayları kullanıcının aksiyonlarına bir referanstır. Kullanıcının yaptığı; bir DOM objesine tıklama, üzerine gelmesi, klavyede bir tuşa basmak veya sayfanın yüklenmesi, JavaScript olayı olarak örnek verilebilir.

DOM (*Document Object Model*) elementleri, olayları dinler ve olayın ateşlenmesi durumunda, dinlediği olaya özel fonksiyon atanmış ise bu fonksiyonu çağırır. Örneğin, bir elemente tıklanması durumununda tetiklenecek olacak fonksiyonu şöyle atıyoruz:

HTML Kodu

```
<div onclick="tetiklenecekFonksiyon()">Tıkla!</div>
```

> **NOT** Buraya kadar bahsettiğimiz konulardan çoğumuzun bilgisi var. Fakat AngularJS kullanırken aklımıza şöyle bir soru gelebilir. Peki AngularJS fonskiyonlarını bu olay tanımlayıcıları ile nasıl kullanacağım?

AngularJS, kendi fonksiyonlarını olay tanımlayıcılarına bağlamak için kendi direktiflerine (nitelik tanımlayıcılarına) sahiptir. Örneğin, AngularJS kontrol metodum üzerinde tutulan bir fonksiyonu, bir DOM elementine tıkladığımda tetiklemek istersem şu yapıyı kullanmam gerekecek:

HTML Kodu

```
<div ng-click="tetiklenecekAngularFonksiyonu()">Tıkla!</div>
```

> **NOT** Anlattıklarımıza ek olarak, tüm JavaScript olayları için bir AngularJS direktifi olduğunu ve bağına `ng-` eki getirerek kullandığımızı da not olarak ekleyelim.

AngularJS Olay Tanimlayicilarinin Kullanimi

Kullanımı çok basit olan, olay tanımlayıcısı atama olayını basit bir tıklama örneği ile gösterelim:

HTML Kodu

```
<div ng-app="ngUygulamam" ng-controller="ngKontrol">
  {{ yazi }}
  <hr>
  <button ng-click="tiklandi()">Tıkla</button>
</div>
<script type="text/javascript">
var uygulama = angular.module("ngUygulamam", []);
  uygulama.controller("ngKontrol", function($scope){
     $scope.yazi = "Butona Tıklayın!";

    $scope.tiklandi = function(){
      $scope.yazi = "Merhaba Dünya!!";
    }
  });
</script>
```

Bu örneğimizde, kitabımızın diğer bölümlerinde de gösterdiğimiz fakat mantığını açıklamadığımız görünüme fonksiyon bağlama olayının nasıl gerçekleştiğini daha net bir şekilde görebilirsiniz.

HTML Kodu

```
<div ng-app="ngUygulamam" ng-controller="ngKontrol">
  {{ yazi }}
  <hr>
```

13

AngularJS'de Template Kavramı ve Kullanımı

Bu Bölümde

Önceki bölümlerde de bahsettiğimiz gibi, AngularJS'in en önemli özelliklerinin başında direktifleri ile HTML'i zenginleştirip, bu angular-özel element ve direktifler ile JavaScript tarafında basit bir geliştirme arayüzü oluşturmasıydı.

Kitabımızın bu bölümünde bu özellikleri daha modüler şekilde kullanmamıza ön ayak olan template kavramını inceleyeceğiz.

TEMPLATE KAVRAMI VE MODÜLER YAPISI

Template (şablon), HTML kodları ve angular-özel yapılardan (direktifler, expression belirticileri, filtreler, form kontrol metodları) oluşmuş genelgeçer bir veri yapılandırıcısı olarak tanımlanabilir.

Template'ler genellikle kullanılmak (include edilmek) üzere bir HTML dosyasında tutulur. (sayfa.html) Bahsi geçen template bir özel direktife `template` olarak atanarak kullanılmaktadır.

Örneğin; Bir kütüphanenin kitaplarını sunan uygulama oluşturuyorken kitapların sıralanmasını sağlayan yapıyı bir template (listele.html), bir kitabın ayrıntılarını gösterdiğimiz sayfa yapısını bir template olarak (ayrinti.html) düşünebiliriz.

NEDEN TEMPLATE (ŞABLON) KULLANIYORUZ?

Template, tek sayfa uygulamalarını geliştirken daha düzenli ve modüler bir geliştirme ortamı oluşturur. En büyük artısının bu olduğunu söyleyebiliriz.

Buna ek olarak bir template, uygulama içerisinde birden fazla kez kullanılabilir. Aynı HTML kodunu tekrar tekrar kullanmak yerine, oluşturulmuş bir template üzerinden tek satır kod ile dahil edebilirsiniz.

TEMPLATE KULLANIMI VE UYGULAMADAKİ YERİ

Template kullanırken, programlama sırasında kodun hangi bölümünün bir template oluşturacağını genel olarak düşünmeden kararlaştırmak kötü bir işleyiş olabilir. Bunun yerine oluşturacağınız uygulamanın birbirinden ayrılacak alakası olmayacak veya birden fazla kez kullanılacak bölümlerinin belirlenmesi çalışma akışınızı daha verimli hale getirecektir.

Çok basit bir örnek ile, bir template dosyasını direktifler yardımı ile uygulamamıza nasıl eklediğimizi inceleyelim;

index.html Dosyası İçeriği

HTML Kodu

```
<div ng-app="ngUygulamam" ng-controller="ngKontrol">
    Normal İçerik
    <hr/>
    <div sayfa></div>
</div>
<script type="text/javascript">
    var uygulama = angular.module("ngUygulamam", []);
```

```
    uygulama.controller("ngKontrol", function($scope) {
        $scope.sayfaBaslik = "Deneme Başlık";
        $scope.sayfaIcerik = "Lorem ipsum dolor sit amet.";
    });
    uygulama.directive("sayfa", function() {
        return {
            templateUrl: 'sayfa.html'
        };
    });
</script>
```

sayfa.html Dosyası İçeriği

HTML Kodu

```
<h2>{{sayfaBaslik}}</h2>
<p>{{sayfaIcerik}}</p>
```

Sayfa Yapısı:

```
/Ana Klasör
-index.html
-sayfa.html
-angular.min.js
```

Bu örnekte, aynı klasörde bulunan *index.html* ve *sayfa.html* dosyaları içeriğini görüyoruz. Burada, kendi oluşturduğumuz sayfa direktifi ile, *sayfa.html* template'ini bu direktife yükletiyoruz.