نمایش داده ها

برای نمایش داده ها می توانید کنترل های موجود در قالب HTML را به ویژگی های یک کامپوننت Angular مقید کنید.

در این صفحه کامپوننتی را به همراه فهرستی از هیروها ایجاد می کنید. سپس فهرستی از اسامی هیروها و به صورت شرطی پیغامی را در زیر این فهرست نمایش می دهید.

رابط کاربری به صورت زیر در می آید:

مثال موجود در این لینک تمامی سینتکس ها و تکه کدهای توضیح داده شده در این صفحه را تبیین می کند.

نمایش ویژگی های کامپوننت به کمک میانگیری (interpolation)

ساده ترین راه برای نمایش یک ویژگی کامپوننت این است که اسم این ویژگی را از طریق میانگیری مقید کنید. در میانگیری شما باید اسم ویژگی را در قالب view و داخل دو آکولاد قرار دهید: {{myHero}}.

برای ایجاد یک پروژه ی جدید به نام displaying-data از دستورالعمل های بخش «ابتدای کار» پیروی کنید.

فایل app.component.html را پاک کنید، زیرا در این مثال به آن نیاز ندارید.

سپس با تغییر قالب و بدنه ی کامپوننت، فایل app.component.ts را اصلاح کنید.

بعد از انجام این کارها، نتیجه ی نهایی به صورت زیر خواهد بود:

src/app/app.component.ts

```
import { Component } from '@angular/core';

@Component({
selector: 'app-root',
template: `
<h1>{{title}}</h1>
<h2>My favorite hero is: {{myHero}}</h2>
`
})
export class AppComponent {
title = 'Tour of Heroes';
myHero = 'Windstorm';
}
```

شما دو ویژگی title و myHero را به کامپوننتی که قبلا خالی بود اضافه کردید.

قالب با استفاده از میانگیری دو آکولادی، دو ویژگی کامپوننت را نمایش می دهد:

src/app/app.component.ts (template)

```
template: `
 <h1>{{title}}</h1>
 <h2>My favorite hero is: {{myHero}}</h2>
 `
```

قالب یک رشته ی چند خطی داخل کاراکتر (`) ECMAScript 2015 است. با کمک این کاراکتر که البته با کاراکتر نقل قول (') تفاوت دارد، می توانید رشته ای را در چندین خط ایجاد کنید تا HTML تان خواناتر شود.

Angular به صورت خودکار از کامپوننت، ویژگی های title و myHero را دریافت می کند و این مقادیر را داخل مرورگر درج می کند. هر زمان که این ویژگی ها تغییر کنند، Angular نمایش را به روز رسانی می کند.

به بیان دقیق تر، نمایش مجدد بعد از برخی از رویدادهای ناهمگام مربوط به view مانند فشردن کلید، اتمام زمان تایمر یا پاسخ به یک درخواست HTTP، رخ می دهد.

توجه داشته باشید که برای ایجاد نمونه ای از کلاس AppComponent نیازی به فراخوانی new ندارید. بلکه Angular نمونه ای را برای شما ایجاد می کند. چگونه؟

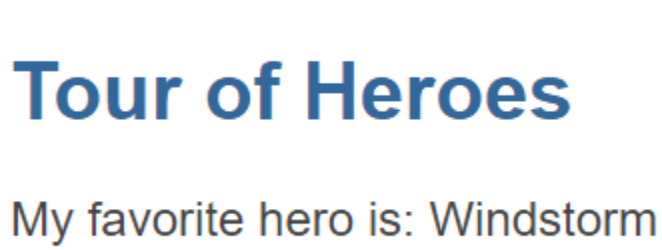
به این صورت که CSS selector موجود در دکوراتور Component@ عنصری به نام <app-root> را تعیین می کند. این عنصر نقش یک نگهدارنده در بدنه ی فایل index.html را ایفا می کند.

src/index.html (body)

```
<body>
  <app-root></app-root>
</body>
```

زمانی که به کمک کلاس AppComponent (در main.ts) به بوت استرپ می پردازید، Angular در index.html به دنبال یک <app-root> می گردد. بعد از آن که آن را پیدا می کند، نمونه ای از AppComponent را می سازد و آن را داخل تگ <app-root> رندر می کند.

حالا پس از اجرای برنامه، باید بتوانید عنوان و اسم هیرو را مشاهده کنید:



در بخش های زیر به برخی از گزینه های کد نویسی در نرم افزار می پردازیم.

قالب درون خطی یا فایل قالب

قالب کامپوننت خود را می توانید در یکی از دو مکان ذیل ذخیره کنید. می توانید آن را با استفاده از ویژگی template به صورت درون خطی تعریف کنید و یا آن را درون یک فایل HTML مجزا تعریف کنید و پس از آن قالب خود را با استفاده از ویژگی templateUrl مربوط به دکوراتور Component@ به متادیتای کامپوننت پیوند بزنید.

اینکه بین این دو کدام را انتخاب کنید بستگی به سلیقه ی شخصی، شرایط و سیاست های سازمانی دارد. در این برنامه از HTML درون خطی استفاده شده است، زیرا قالب آن کوچک است و دموی آن بدون وجود فایل های اضافی HTML ساده تر می شود.

در هر دو سبک بیان شده، مقیدسازی داده های قالب به صورت یکسانی به ویژگی های کامپوننت دسترسی دارند.

به صورت پیش فرض، دستور Angular CLI ، [`ng generate component`](cli/generate) کامپوننت ها را به کمک یک فایل قالب ایجاد می کند. این کار را می توانید به صورت زیر لغو کنید:

```
ng generate component hero –it
```

Constructor یا مقداردهی اولیه به متغیر

هرچند که در این مثال برای دادن مقدار اولیه به کامپوننت ها از تخصیص دهی متغیر استفاده شده است، اما شما می توانید به جای آن با استفاده از یک constructor ویژگی های برنامه را اعلان و مقداردهی اولیه کنید:

```
export class AppCtorComponent {
 title: string;
 myHero: string;

 constructor() {
  this.title = 'Tour of Heroes';
  this.myHero = 'Windstorm';
 }
}
```

در این برنامه برای مختصر شدن برنامه از سبک کوتاه تری از تخصیص دهی متغیر استفاده شده است.

نمایش یک ویژگی آرایه به کمک ngFor*

برای نمایش فهرستی از هیروها، کار خود را با اضافه کردن آرایه ای از اسامی هیروها به کامپوننت شروع کنید و پس از آن myHero را به گونه ای مجددا تعریف کنید که اولین اسم موجود در آرایه باشد.

```
src/app/app.component.ts (class)
export class AppComponent {
 title = 'Tour of Heroes';
 heroes = ['Windstorm', 'Bombasto', 'Magneta', 'Tornado'];
 myHero = this.heroes[0];
}
```

حالا برای نمایش هر یک از آیتم های موجود در فهرست heroes در قالب از دستورالعمل ngFor استفاده کنید.

src/app/app.component.ts (template)

```
template: `
 <h1>{{title}}</h1>
 <h2>My favorite hero is: {{myHero}}</h2>
 <p>Heroes:</p>
 <ul>
  <li *ngFor="let hero of heroes">
   {{ hero }}
  </li>
 </ul>
`
```

در این رابط کاربری از فهرست نامرتب HTML به همراه تگ های <ul> و <li> استفاده شد است. ngFor* موجود در عنصر <li> دستورالعمل «تکرار کننده »ی Angular محسوب می شود. ngFor* این عنصر (و فرزندان آن) را به عنوان «قالب تکرار کننده» نشان می کند.

src/app/app.component.ts (li)

```
<li *ngFor="let hero of heroes">
 {{ hero }}
</li>
```

یادتان نرود که حتما در ngFor* از کاراکتر (*) استفاده کنید. این کاراکتر بخش جدایی ناپذیری از سینتکس است. برای اطلاعات بیشتر به صفحه ی سینتکس قالب مراجعه کنید.

به دابل کوتیشن hero موجود در ngFor دقت کنید. این دستورالعمل مثالی از یک متغیر ورودی قالب است. برای دریافت اطلاعات بیشتر در ابطه با متغیرهای ورودی قالب به قسمت میکروسینتکس صفحه ی سینتکس قالب مراجعه کنید.

Angular به ازای هر یک از آیتم های موجود در این فهرست <li> را تکثیر می کند و در حلقه ی تکرار فعلی، متغیر hero را بر روی آیتم (خود هیرو) تنظیم می کند. Angular از این متغیر به عنوان زمینه ی میانگیری داخل دو آکولاد استفاده می کند.

در این حالت ngFor در حال نمایش یک آرایه است، اما می تواند آیتم ها را به ازای هر شیء قابل تکراری تکرار کند.

حالا هیروها در لیست نامرتبی نمایش داده می شوند.

ایجاد کلاسی برای داده ها

کد این برنامه داده ها را به صورت مستقیم داخل کامپوننت تعریف می کند. درست است که این کار، بهترین راهکار نیست، اما در نسخه ی دموی ساده ی ما این کار ایرادی ندارد.

در حال حاضر، binding مربوط به آرایه ای از رشته ها است. در برنامه های واقعی اغلب binding ها مربوط به اشیاء تخصصی تری است.

برای آن که این binding را به گونه ای تبدیل کنید که از اشیاء تخصصی تری استفاده کند، آرایه ی اسامی هیروها را به آرایه ای از اشیاء Hero تبدیل کنید. برای انجام این کار به یک کلاس Hero نیاز دارید:

```
ng generate class hero
```

به کمک کد زیر کلاسی را به همراه یک constructor و دو ویژگی id و name ایجاد می کنید.

```
src/app/hero.ts

export class Hero {
 constructor(
  public id: number,
  public name: string) { }
}
```

ممکن است در ظاهر این کلاس ویژگی ای نداشته باشد، اما اینطور نیست. اعلان پارامترهای constructor از میانبر تایپ اسکریپت بهره می برد.

پارامتر اول را در نظر بگیرید:

```
src/app/hero.ts (id)

public id: number,
```

این سینتکس مختصر کارهای زیادی انجام می دهد:

- یک پارامتر constructor و نوع آن را اعلان می کند.
- ویژگی همگانی همین اسم را اعلان می کند.
- این ویژگی را به کمک آرگومان متناظر و در زمان ایجاد نمونه ای از کلاس، مقداردهی اولیه می کند.

استفاده از کلاس Hero

ویژگی AppComponent.heroes پس از وارد کردن کلاس Hero می تواند آرایه ی نوع داری از اشیاء Hero را برگشت دهد:

```
src/app/app.component.ts (heroes)

heroes = [
 new Hero(1, 'Windstorm'),
 new Hero(13, 'Bombasto'),
 new Hero(15, 'Magneta'),
 new Hero(20, 'Tornado')
];
```

```
myHero = this.heroes[0];
```

بعد از انجام این کار، قالب را به روز رسانی کنید. قالب در حال حاضر name و id هیرو را نمایش می دهد. قالب را به گونه ای تنظیم کنید که تنها ویژگی name هیرو را نمایش دهد.

src/app/app.component.ts (template)

```
template: `
 <h1>{{title}}</h1>
 <h2>My favorite hero is: {{myHero.name}}</h2>
 <p>Heroes:</p>
 <ul>
  <li *ngFor="let hero of heroes">
   {{ hero.name }}
  </li>
 </ul>
`
```

با این که تغییری در نمایش ایجاد نشده است، اما کد واضح تر به نظر می رسد.

نمایش شرطی به کمک NgIF

در برخی مواقع یک برنامه باید یک view و یا بخشی از view را تنها تحت شرایط مشخصی نمایش دهد.

بیایید مثال را به گونه ای تغییر دهیم که اگر تعداد هیروها بیشتر از 3 عدد بود، پیامی را نمایش دهد.

دستورالعمل Angular nglf عنصری را بر اساس شرط صحیح / نادرست درج و یا حذف می کند. برای آن که این موضوع را در عمل مشاهده کنید، پاراگراف زیر را در پایین قالب خود اضافه کنید:

src/app/app.component.ts (message)

```
<p *nglf="heroes.length > 3">There are many heroes!</p>
```

یادتان نرود که حتما در nglf* از کاراکتر (*) استفاده کنید. این کاراکتر بخش جدایی ناپذیری از سینتکس است.

برای اطلاعات بیشتر به قسمت nglf صفحه ی سینتکس قالب مراجعه کنید.

ظاهر و رفتار عبارت قالب موجود در دابل کوتیشن "*nglf="heroes.length > 3 تا حد زیادی شبیه به تایپ اسکریپت است. زمانی که فهرست هیروهای کامپوننت بیش از 3 آیتم داشته باشد، Angular این پاراگراف را به DOM اضافه می کند و در نهایت این پیام نمایش داده می شود؛ اما اگر تعداد آیتم ها کوچک تر مساوی 3 باشد، Angular از این پاراگراف صرف نظر می کند و هیچ پیامی نمایش داده نمی شود. برای اطلاعات بیشتر به قسمت عبارت های قالب در صفحه ی سینتکس قالب مراجعه کنید.

Angular کاری به پنهان یا آشکار کردن این پیام ندارد؛ بلکه صرفا عنصر این پاراگراف را از DOM حذف و یا به آن اضافه می کند. این کار مخصوصا در پروژه های بزرگ که بخش های بزرگی از HTML به همراه مقیدسازی های داده ای زیاد، حذف و اضافه می شوند، باعث بهبود عملکرد می شود.

حالا برنامه را امتحان کنید. با توجه به اینکه این آرایه 4 آیتم دارد، پیام بالا باید ظاهر شود. به app.component.ts برگردید و یکی از عناصر را از آرایه ی هیرو حذف کنید و یا آن را به یک کامنت تبدیل کنید. بعد از انجام این کار، مرورگر به صورت خودکار رفرش می شود و پیام ناپدید می شود.

خلاصه

حالا شما باید چگونگی استفاده از موارد زیر را یاد گرفته باشید:

- میانگیری به همراه دو آکولاد، جهت نمایش یک ویژگی کامپوننت.
- ngFor جهت نمایش آرایه ای از آیتم ها.
- یک کلاس تایپ اسکریپت، جهت شکل دادن به داده های مدل کامپوننت خود و نمایش ویژگی های این مدل.
- nglf جهت نمایش شرطی بخشی از HTML بر اساس یک عبارت بولی.

در نهایت کد شما به صورت زیر در می آید:

src/app/app.component.ts

```
1. import { Component } from '@angular/core';
2.
3. import { Hero } from './hero';
4.
5. @Component({
6. selector: 'app-root',
7. template: `
8. <h1>{{title}}</h1>
```

```
<h2>My favorite hero is: {{myHero.name}}</h2>
<p>Heroes:</p>
<ul>
<li *ngFor="let hero of heroes">
{{ hero.name }}
</li>
</ul>
<p *ngIf="heroes.length > 3">There are many heroes!</p>
`
})
export class AppComponent {
title = 'Tour of Heroes';
heroes = [
new Hero(1, 'Windstorm'),
new Hero(13, 'Bombasto'),
new Hero(15, 'Magneta'),
new Hero(20, 'Tornado')
];
myHero = this.heroes[0];
}
```

src/app/hero.ts

```
export class Hero {
 constructor(
  public id: number,
  public name: string) { }
}
```

src/app/app.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppComponent } from './app.component';

@NgModule({
imports: [
BrowserModule
],
declarations: [
AppComponent
```

```
12. ],
13. bootstrap: [ AppComponent ]
14. })
15. export class AppModule { }
```

main.ts

```
import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app/app.module';
import { environment } from './environments/environment';

if (environment.production) {
 enableProdMode();
}

platformBrowserDynamic().bootstrapModule(AppModule);
```