IZDVOJENA ISKUSTVA ZAJEDNICE

Naučite Angular 2

Vaš brz, jasan vodič za izgradnju aplikacija pomoću Angulara 2

Pablo Deeleman

kompjuter biblioteka

[PACKT] PUBLISHING

IZDVOJENA ISKUSTVA ZAJEDNICE

Naučite Angular 2

Vaš brz, jasan vodič za izgradnju aplikacija pomoću Angulara 2

Pablo Deeleman

kompjuter biblioteka

[PACKT]
PUBLISHING
BIRMINGHAM - MUMBAI

Izdavač:
kompjuter biblioteka

Obalskih radnika 15, Beograd

Tel: 011/2520272

e-mail: kombib@gmail.com

internet: www.kombib.rs

Urednik: Mihailo J. Šolajić

Za izdavača, direktor:
Mihailo J. Šolajić

Autori: Pablo Deeleman

Prevod: Slavica Prudkov

Lektura: Miloš Jevtović

Slog : Zvonko Aleksić

Znak Kompjuter biblioteke:
Miloš Milosavljević

Štampa: „Pekograf“, Zemun

Tiraž: 500

Godina izdanja: 2016.

Broj knjige: 485

Izdanje: Prvo

ISBN: 978-86-7310-508-6

Learning Angular 2
by Pablo Deeleman

ISBN 978-1-78588-207-4

CIP - Каталогизација у публикацији
Народна библиотека Србије, Београд,
се добија на захтев

UVOD

Tokom proteklih godina Angular 1.x je postao jedan od najprisutnijih JavaScript radnih okvira za izgradnju vrhunskih veb aplikacija, malih i velikih. U jednom trenutku, njegovi nedostaci u pogledu performanse i skalabilnosti su postali previše istaknuti čim bi aplikacija postala veća i složenija. Tada je nastao Angular 2, potpuno prepisana verzija – ona ispunjava očekivanja modernih programera, koji zahtevaju brze performanse i odzive u svojim veb aplikacijama.

Angular 2 je dizajniran imajući u vidu moderne veb standarde. Omogućava potpunu fleksibilnost kada birate jezik, obezbeđujući punu podršku za ES6 i TypeScript, ali isto tako dobro funkcioniše sa današnjim ES5, Dart ili CoffeeScriptom. Njegove ugrađene funkcije dependency injection omogućavaju korisniku izgradnju skalabilnih i modularnih aplikacija pomoću izražajnog i jasnog koda, koji olakšava zadatke održavanja i maksimalno pojednostavljuje razvoj vođen testiranjem koda. Međutim, ono što stvarno ističe Angular 2 je neuporediv nivo brzine i performanse, zahvaljujući novom sistemu za detekciju promene koji je do pet puta brži od prethodne verzije. Jasniji prikazi i nenadmašna sintaksa kreiranja šabona usaglašena sa standardima ujedinjuju beskrajnu listu moćnih funkcija za izgradnju veb mobilnih i desktop aplikacija nove generacije.

Angular 2 će ostati u upotrebi i uvešće novinu u budućem kreiranju igara na način na koji su moderne veb aplikacije predviđene i razvijane. Međutim, zbog jedinstvenog dizajna i arhitekture, učenje Angulara 2 može izgledati zbunjujuće i teško za nove korisnike.

Cilj u ovoj knjizi je da korisnik izbegne previše informacija o API referencama i opisima radnog okvira, ali da se pruži praktični pristup, pomažući čitaocu da već od prvog dana nauči kako da iskoristi radni okvir za izgradnju važnih komponenata.

Ova knjiga pruža programerima kompletan vodič kroz ovu novu platformu i TypeScript sintakse, kreiranjem veb projekta od kraja ka početku, počevši od osnovnih koncepata i uzoraka komponenata i spajanja tih komponenata za igradnju složenijih funkcija u svakom poglavlju, dok ne pokrenemo kompletnu, testiranu, spremnu veb aplikaciju do kraja knjige.

ŠTA OBUHVATA OVA KNJIGA

U Poglavlju 1, „*Kreiranje prve komponente u Angularu 2*“, predstavljamo čitaocu veb komponente koje su gradivni blokovi svih Angular 2 aplikacija.

Poglavlje 2, „*Uvod u TypeScript*“, sadrži uputstva o sintaksi i posebnim funkcijama ovog super seta ECMAScripta 6, što je, u stvari, sintaksa po izboru Angular tima za izgradnju Angulara 2.

U Poglavlju 3, „*Implementiranje parametara i događaja u komponente*“, opisujemo kako se naše komponente ponašaju kao mašine stanja koje mogu da promene stanje primanjem podataka korišćenjem ulaznih parametara i emituju podatke kao događaje korišćenjem izlaznih parametara.

Poglavlje 4, „*Poboljšanje komponenata pomoću usmeravanja i komandi*“, sadrži kompletan vodič kroz ugrađena usmeravanja radnog okvira koja se koriste za sažetak izlaznih podataka u šablonima i kroz ugrađene komande koje obezbeđuju napredne funkcije za komponentu. Čitalac će, takođe, naučiti kako da kreira sopstvena usmeravanja i komande.

Poglavlje 5, „*Izgradnja aplikacije pomoću Angular 2 komponenata*“, posvećeno je rekapitulaciji onoga što smo naučili do sada da bi se obezbedilo dobro skaliranje Angular 2 projekata, bez obzira na njihovu veličinu, uz njihovo prilagođavanje kodiranju zajednice i konvencijama imenovanja.

U Poglavlju 6, „*Asinhroni servisi podataka u Angularu 2*“, čitalac uči kako da implementira i upotrebi HTTP konekcije, koristeći druge servise podataka sredstvima Http modula, pa može da kreira sopstvene klijente servisa podataka.

U Poglavlju 7, „*Usmeravanja u Angularu 2*“, predstavljamo ruter Angulara 2 i njegove ugrađene komande, obezbeđujući kompletan vodič kroz različite strategije za učitavanje komponenata iz ruta i rukovanje stanjima pomoću History API-ja.

U Poglavlju 8, „*Formulari i rukovanje proverom identiteta u Angularu 2*“, ilustrujemo različite strategije koje su na raspolaganju za izgradnju veb formulara pomoću Angulara 2, upravljanje dvosmernim vezivanjem podataka na ulaznim kontrolama i kreiranje složenih formulara i validacija.

Poglavlje 9, „*Animiranje komponenata pomoću Angulara 2*“, obuhvata aktuelno dostupne alatke i klase za implementiranje animacija u komponente, od čistih CSS animacija, kojima se rukuje pomoću Angular 2 komandi, do složenijih prelaza, kojima se rukuje samo pomoću JavaScripta, zahvaljujući graditeljima animacije u Angularu 2.

Poglavlje 10, „*Testiranje koda u Angularu 2*“, će provesti čitaoca kroz korake koji su potrebni za implementiranje osnove za zvučno testiranje u aplikaciju i generalne šablone za upotrebu testriranja koda u komponentama, komandama, usmeravanjima, rutama i servisima.

ŠTA VAM JE POTREBNO ZA OVU KNJIGU

Da biste izradili primere koji su sadržani u ovoj knjizi, potreban vam je veb pretraživač koji je ažuriran na najnoviju verziju. Preporučujemo da upotrebite Google Chrome ili Mozilla Firefox, mada je Angular 2 podržan u svim klasičnim pretraživačima.

Takođe vam je potreban softver terminala koji je instaliran na operativni sistem, pošto se mnogim operacijama rukuje pomoću npm komandi na konzoli. Zbog toga će biti potrebno da imate instalirane Node.js i npm na sistemu za pokretanje većine komandi konzole koje su spomenute u ovoj knjizi. Ostali potrebni moduli i procedure njihove instalacije će biti opisani u toku rada.

Takođe će vam biti potreban editor za tekst za kodiranje Angular 2 modula, mada će Poglavlje 1, „*Kreiranje prve komponente u Angularu 2*“, obezbediti detaljan opis svih dobrih IDE alternativa na tržištu za razvijanje Angular 2 aplikacija.

ZA KOGA JE OVA KNJIGA

Ova knjiga je namenjena veb programerima koji žele da kreiraju umetničke mobilne i desktop veb aplikacije nove generacije korišćenjem Angulara 2. Ova knjiga ne zahteva prethodno poznavanje Angulara 1.x ili 2, mada se pretpostavlja sveobuhvatno poznavanje JavaScripta. Knjiga je odlična za početnike u Angularu koji žele da nauče najbolje kroz jasna objašnjenja i definicije koncepata.

KONVENCIJE

U ovoj knjizi pronaći ćete veliki broj stilova teksta koji međusobno razlikuju različite vrste informacija. Evo i nekih primera ovih stilova i objašnjanja njihovog značenja.

Reči koda u tekstu, nazivi tabela baze podataka, nazivi direktorijuma, nazivi fajlova, ekstenzije fajla, nazivi putanja, kratki URL-ovi, korisnički unos i Twitter identifikatori su prikazani na sledeći način: „Kao rezultat ove akcije, pronaći ćete novi `tsconfig.json` fajl u osnovnom direktorijumu projekta, uključujući podešavanja koje zahteva TypeScript kompajler za transpilaciju koda komponente u čist ECMAScript 5 JavaScript kod koji aktuelni pretraživači mogu da čitaju“.

Blok koda je postavljen na sledeći način:

```
<body>
  <nav class="navbar navbar-default navbar-static-top">
    <div class="container">
      <div class="navbar-header">
        <strong class="navbar-brand">My Pomodoro Timer</strong>
      </div>
    </div>
  </nav>
  <pomodoro-timer></pomodoro-timer>
</body>
```

Kada želimo da privučemo pažnju na određeni deo bloka koda, relevantne linije ili stavke su ispisane masnim slovima:

```
<body>
  <nav class="navbar navbar-default navbar-static-top">
    <div class="container">
      <div class="navbar-header">
        <strong class="navbar-brand">My Pomodoro Timer</strong>
      </div>
    </div>
  </nav>
  <pomodoro-timer></pomodoro-timer>
</body>
```

Unosi ili ispisi komandne linije su napisani na sledeći način:

```
$ npm install angular2 es6-shim es6-promise reflect-metadata rxjs
zone.js --save
```

Novi termini i važne reči su napisane masnim slovima. Reči koje vidite na ekranu - na primer, u menijima ili okvirima za dijalog, biće prikazane u tekstu na sledeći način: „Odeljak Naučite daje nam pristup brzim uputstvima da bismo ubrzali podešavanje jezika“.

Upozorenja ili važne napomene će biti prikazani prikazati u ovakvom okviru.

Saveti i trikovi prikazani su ovako.

POVRATNE INFORMACIJE ČITALACA

Povratne informacije od naših čitalaca su uvek dobrodošle. Obavestite nas šta mislite o ovoj knjizi – šta vam se dopalo ili šta vam se možda nije dopalo. Povratne informacije čitalaca su nam važne da bismo kreirali naslove od kojih ćete dobiti maksimum.

Da biste nam poslali povratne informacije, jednostavno nam pošaljite e-mail na adresu feedback@packtpub.com i u naslovu poruke napišite naslov knjige.

Ako postoji tema za koju ste specijalizovani i zainteresovani ste da pišete ili sarađujete na nekoj od knjiga, pogledajte vodič za autore na adresi www.packtpub.com/authors.

KORISNIČKA PODRŠKA

Sada, kada ste ponosni vlasnik „Packt“ knjige, mi imamo mnogo štošta da vam ponudimo da bismo vam pomogli da dobijete maksimum iz svoje narudžbe.

Preuzimanje primera koda

Možete da preuzmete fajlove sa primerima koda za ovu knjigu sa GitHub veb sajta na adresi https://github.com/deeleman/learning-angular2.

Možete da preuzmete fajlove sa primerima koda za ovu knjigu sa vašeg naloga na adresi http://www.packtpub.com. Ako ste ovu knjigu kupili na drugom mestu, možete da posetite stranicu http://www.packtpub.com/support i registrujete se da biste e-mailom dobili fajlove.

Možete da preuzmete fajlove sa primerima koda prateći sledeće korake:

1. Prijavite se ili registrujte na našem veb sajtu, koristeći svoju e-mail adresu i lozinku.
2. Postavite kursor na karticu SUPPORT na vrhu stranice.
3. Kliknite na Code Downloads & Errata.
4. U polje Search unesite naslov knjige.
5. Izaberite knjigu za koju želite da preuzmete fajlove sa primerima koda.
6. Iz padajućeg menija izaberite mesto na kojem ste kupili knjigu.
7. Kliknite na Code Download.

Takođe možete da preuzmete fajlove koda tako ako kliknete na dugme Code Files na veb stranici knjige na veb sajtu Packt Publishing. Ovoj stranici možete da pristupite tako što ćete u polje Search uneti naziv knjige. Ne zaboravite da morate biti prijavljeni na vaš Packt nalog.

Kada su fajlovi preuzeti, ekstrahujte direktorijum, koristeći najnoviju verziju:

- WinRAR / 7-Zip za Windows
- Zipeg / iZip / UnRarX za Mac
- 7-Zip / PeaZip za Linux

Štamparske greške

Iako smo preduzeli sve mere da bismo obezbedili tačnost sadržaja, greške se dešavaju. Ako pronađete grešku u nekoj od naših knjiga – možda grešku u tekstu ili u kodu – bili bismo zahvalni ako biste nam to prijavili. Na taj način možete da poštedite čitaoce od frustracija i nama da pomognete da poboljšamo naredne verzije ove knjige. Ako pronađete neku štamparsku grešku, molimo vas da nas obavestite, tako što ćete posetiti stranicu http://www.packtpub.com/submit-errata, selektovati knjigu, kliknuti na link Errata Submission Form i uneti detalje o grešci koju ste pronašli. Kada je greška verifikovana, vaša prijava će biti prihvaćena i greška će biti aploudovana na naš veb sajt ili dodata u listu postojećih grešaka, pod odeljkom Errata za određeni naslov.

Da biste pregledali prethodno prijavljene greške, posetite stranicu https://www.packtpub.com/books/content/support i unesite naslov knjige u polje za pretragu. Tražena informacija će biti prikazana u odeljku Errata.

Piraterija

Piraterija autorskog materijala na Internetu je aktuelan problem na svim medijima. Mi u „Packtu“ zaštitu autorskih prava i licenci shvatamo veoma ozbiljno. Ako pronađete ilegalnu kopiju naših knjiga, u bilo kojoj formi na Internetu, molimo vas da nas o tome obavestite i pošaljete nam adresu lokacije ili naziv veb sajta da bismo mogli da podnesemo tužbu.

Molimo vas, kontaktirajte sa nama na adresi copyright@packtpub.com i pošaljite nam link ka sumnjivom materijalu.

Zahvalni smo vam na pomoći u zaštiti naših autora i mogućnosti da vam pružimo vredan sadržaj.

Pitanja

Možete da kontaktirate sa nama na adresi questions@packtpub.com ako imate bilo kakvih problema sa bilo kojim aspektom knjige i mi ćemo učiniti sve što je u našoj moći da rešimo takve probleme.

1

Kreiranje prve komponente u Angularu 2

Verovatno ste svesni da su moderni JavaScript veb radni okviri i biblioteke u frontend areni, osim ako ste bili „izgubljeni u svemiru“ poslednjih nekoliko godina. Čak smo dosegli stadijum u kojem se svakodnevno rađa novi radni okvir, što frontend programerima dodatno otežava da pažljivo procene da li ova nova moderna alatka za kodiranje dodaje dovoljno vrednosti da bi bili opravdani vreme i trud koji su potrebni za učenje i upotrebu u sledećem novom projektu.

Na kraju, šačica radnih okvira će dobiti veći značaj od drugih. Mi očigledno govorimo o radnim okvirima na strani klijenta koji će vam, verovatno, zvučati prilično poznato: Backbone, Ember, Knockout, Angular 1 i tako dalje.

Pošto se u svetu JavaScripta vodi bitka za prevlast, novi radni okviri, kao što su React ili Aurelia, su „ušli u igru“, favorizujući veb komponente i iskorišćavajući moć Shadow DOM-a kao kamen-temeljac u svojoj arhitekturi. Aplikacije koje su građene na ovaj način dokazano su modularne, skalabilne su i lake za održavanje i imaju nenadmašan nivo performanse.

Angular 1 je već prošao dug put od svog nastanka i njegove mane su postale previše „nametljive“ da bi dalje mogle da se previde. Bilo je vreme za nešto bolje, a jednostavna prerada osnove koda nije bila dovoljna. Bio je potreban ambiciozniji pristup i razvijen je Angular 2 – novi radni okvir, konstruisan od nule, koji u potpunosti obuhvata najnovije trendove u industriji. U „srcu“ dizajna Angular 2 ima veb komponente i koristi moć Shadow DOM-a da bi maksimirao prilagodljivost veb elemenata u odnosu na promene stanja.

Pored toga, Angular 2 pruža moderan sistem za detekciju promena u svakoj komponenti, koji je odgovoran za propagiranje vezivanja kroz stablo komponenata koje čine aplikacije.

Definisanje osobina Angulara 2 prevazilazi koncept da je on samo radni okvir veb komponenata, pošto njegove funkcije obuhvataju, uglavnom, sve što je potrebno u modernoj aplikaciji: interoperabilnost komponente, univerzalnu podršku za više platformi i uređaja, vrhunsku mašinu za dependency injection, fleksibilan, ali napredan mehanizam rutera sa podrškom za razdvajanje i komponentiranje definicija rute, napredne HTTP poruke i animaciju ili internacionalizaciju.

U ovom poglavlju ćete:

- naučiti zašto je Angular 2 jedinstven u poređenju sa svojim prethodnim verzijama
- naučiti kako se podešava okruženje koda za rad u Angularu 2 i TypeScriptu
- poboljšati izabrani IDE da biste stekili bolje iskustvo za kodiranje Angular 2 aplikacija
- igraditi prvu Angular 2 veb komponentu i naučiti kako da je ugradite na veb stranicu
- dodati osnovne funkcije za interaktivnost u veb komponentu
- otkriti neke osnovne pomagače za bolje formatiranje ispisa podataka

NOVI POČETAK

Kao što je ranije pomenuto, Angular 2 predstavlja potpuno prepisan Angular 1.x radni okvir, predstavljajući potpuno novu arhitekturu aplikacije izgrađenu od nule u TypeScriptu, striktnom super setu JavaScripta koji dodaje opciono statičko kucanje i podršku za interfejse i dekoratore.

Ukratko, Angular 2 aplikacije su zasnovane na dizajnu arhitekture koji se sastoji od veb komponenata povezanih međusobno pomoću njihovog posebnog I/O interfejsa. Svaka komponenta koristi potpuno obnovljeni mehanizam dependency injectiona. Da budem iskren, ovo je najjednostavniji mogući opis Angulara 2. Međutim, najjednostavniji projekat ikada kreiran u Angularu je isključen ovim definicijama osobina. Mi ćemo se fokusirati na učenje kako da izgradimo interoperabilne komponente i upravljamo dependency injectionom u sledećim poglavljima, pre prelaska na usmeravanja, veb formulare ili HTTP komunikaciju. To takođe objašnjava zašto nećemo kreirati eksplicitne reference za Angular 1.x u ovoj knjizi. Očigledno je da nema smisla gubiti vreme i trošiti stranice na reference ka nečemu što vam neće pružiti koristan uvid u temu; pretpostavljamo da ne znate ništa o Angularu 1.x, a i da ga poznajete, takvo znanje ovde nema nikakvu vrednost.

Veb komponente

Veb komponenta je koncept koji obuhvata četiri tehnologije dizajnirane da budu upotrebljene zajedno za izgradnju elemanata funkcije sa višim nivoom vizuelne izražajnosti i mogućnošću ponovne upotrebe, što dovodi do modularnije, dosledne veb aplikacije, koja je lakša za održavanje. Ove četiri tehnologije su sledeće:

- **šabloni** - Šabloni su delovi HTML-a koji strukturiraju sadržaj koji želimo da renderujemo.
- **prilagođeni elementi** - Ovi šabloni ne sadrže samo tradicionalne HTML elemente, već i prilagođene omotačke stavke koje obezbeđuju dalju prezentaciju elemenata ili API funkcionalnosti.
- **Shadow DOM** - Obezbeđuje karantin za kapsuliranje CSS pravila rasporeda i JavaScript ponašanja svakog prilagođenog elementa.
- **HTML importovanja** - HTML nije više ograničen samo za hostovanje HTML elemenata, već je moguće hostovanje i drugih HTML dokumenata.

Teoretski, Angular 2 komponenta je, u stvari, prilagođeni element koji sadrži šablon za hostovanje HTML strukture rasporeda elementa kojom upravlja oblasni CSS stil, kapsuliran unutar Shadow DOM kontejnera. Pokušajmo ovo da preformulišemo da bi bilo jasnije. Zamislite vrstu kontrole ulaznog opsega u HTML-u 5. To je koristan način da omogućite korisnicima upotrebu odgovarajuće kontrole prilikom unosa vrednosti u rasponu između dve unapred definisane granice. Ako ranije niste koristili ovu kontrolu, ubacite u prazan HTML šablon sledeći deo koda i učitajte ga u pretraživaču:

```
<input id=”mySlider” type=”range” min=”0” max=”100” step=”10”>
```

Videćete lepu kontrolu unosa koja sadrži horizontalni klizač u pretraživaču. Ispitujući takvu kontrolu pomoću programerskih alatki pretraživača, otkrićemo skriveni set HTML tagova koji nisu bili prisutni u vreme kada ste editovali HTML šablon. Tamo ćete pronaći primer Shadow DOM-a u akciji, sa stvarnim HTML šablonom, kojim upravlja njegov kapsulirani CSS sa naprednim funkcijama prevlačenja. Verovatno ćete se složiti da bi bilo dobro kada biste to mogli sami da kreirate. Pa, dobra vest je da Angular 2 obezbeđuje set alatki koje su potrebne za isporučivanje ove funkcije, pa možemo da kreiramo prilagođene (sopstvene) elemente (kontrole unosa, personalizovane tagove i samostalne vidžete) koji imaju unutrašnju HTML oznaku po izboru i sopstveni stil koji ne utiče na CSS stranice koji hostuje komponentu (ni CSS stranice ne utiču na stil).

Zašto je TypeScript iznad drugih sintaksi?

Angular 2 aplikacije mogu da budu kodirane na širokom rasponu jezika i sintaksi: ECMAScript 5, Dart, ECMAScript 6, TypeScript ili ECMAScript 7.

TypeScript je tipiziran super set ECMAScripta 6 (poznatog i kao ECMAScript 2015) koji se prevodi u čist JavaScript i podržan je u modernim operativnim sistemima. TypeScript ima dizajn orijentisan zvučnim objektom i podržava komentare, dekoratore i proveru tipa.

Razlog zbog kojeg smo izabrali (i očigledno preporučujemo) TypeScript kao izabranu sintaksu za instrukcije kako da razvijete Angular 2 aplikacije u ovoj knjizi je zasnovan na činjenici da je sam Angular 2 napisan na ovom jeziku. Dobro poznavanje TypeScripta će programeru pružiti ogromnu prednost pri razumevanju suštine radnog okvira.

Sa druge strane, vredi napomenuti da je podrška TypeScripta za komentare i introspekciju tipa veoma važna kada je u pitanju upravljanje zavisnostima i vezivanje tipa između komponenata korišćenjem minimalnog koda, kao što ćemo videti u nastavku ove knjige.

Na kraju krajeva, možete da kreirate Angular 2 projekte u čistoj ECMAScript 6 sintaksi ako tako želite. Čak i primeri koji su predstavljeni u ovoj knjizi mogu lako da budu prebačeni na ES6 uklanjanjem komentara tipa i interfejsa ili zamenom načina na koji se rukuje zavisnostima u TypeScriptu opširnijim načinom koji koristi ES6.

Da bismo skratili opise, mi ćemo predstaviti primere napisane u TypeScriptu i preporučujemo ovu sintaksu zbog njene bolje izražajnosti, koja je postignuta zahvaljujući komentarima tipa i jednostavnom načinu pristupa zavisnostima na osnovu introspekcije tipa.

PODEŠAVANJE RADNOG PROSTORA

Pre nego što započnemo implementaciju prve i sjajne Angular 2 komponente, potrebno je da postavimo sve alatke koje će nam biti potrebne za implementiranje softvera zasnovanog na TypeScriptu, ostavljajući module Angular 2 radnog okvira sa strane.

Prvo i najvažnije je da kreiramo direktorijum i dva puta proverimo da li je NPM CLI dostupan na sistemu i da li je pravilno ažuriran na najnoviju stabilnu verziju. Ako nije, otvorite stranicu `https://nodejs.org` i instalirajte najnoviji izvršni `Node.js`.

U vreme pisanja ove knjige Angular 2 radni okvir je u verziji Release Candidate 1, pa su, stoga, možda promenjeni zahtevi za izgradnju i upotrebu primera koji su predstavljeni u ovoj knjizi. Autor održava skladište koda na adresi https://github.com/deeleman/learning-angular2, gde možete da proverite najnoviju ažurnu verziju svakog primera iz ove knjige. Skladište je podeljeno na direktorijume poglavlja, a svaki direktorijum sadrži ažuriranu verziju projekta. Pogledajte skladište koda ako se suočite sa bilo kakvim problemima tokom instaliranja ili primene primera u knjizi.

Instaliranje zavisnosti

Prvo treba da instaliramo Angular 2 u radni prostor, uključujući i njegove ravnopravne zavisnosti. Angular 2 tim se potrudio da obezbedi da instalacija bude dovoljno modularna da omogući da dodamo samo ono što je potrebno da bi naši projekti postali manji ili više obimni, u zavisnosti od potreba.

Angular 2 ne dolazi u obliku jednog instalacionog paketa, već postoji više paketa, što pametnom programeru daje mogućnost da izabere samo one module koji su potrebni za njegov projekat, minimizirajući osnovu zavisnosti. Neki od ovih paketa, kao što su common ili core, potrebni su, bez obzira na vrstu projekta koji želimo da kreiramo. Drugi paketi, kao što je platform-browser-dynamic, vezani su za vrstu projekta i platformu na kojoj će se projekat koristiti. Ovde je prikazana lista (ne detaljna) najčešće upotrebljavanih paketa koji će vam biti potrebni u projektima:

- `@angular/core` - Ovo je najvažniji paket koji sadrži osnovu Angulara i njegove najčešće upotrebljavane elemente, kao što su komande i komponente. Često će biti potrebno da se oslonite na ovaj modul za importovanje osnovnih elemenata Angulara 2 u projekat.
- `@angular/common` - Retko će biti potrebno da eksplicitno importujete tokene iz ovog modula, ali vredi napomenuti da ovaj paket sadrži, osim ostalih važnih klasa, definicije svih komandi, servisa i usmeravanja koji su sadržani u Angularu 2.
- `@angular/compiler` - Isto važi kao i za paket common; retko ćete importovati tokene eksplicitno iz ovog modula, iako je ovaj paket odgovoran za kompajliranje HTML šablona i pretvaranje ovih šablona u kod koji može da renderuje ispis korisničkog interfejsa aplikacije.
- `@angular/platform-browser` - Ovaj modul sadrži klase i funkcije koje su potrebne za pisanje i interakciju sa DOM-om u kontekstu veb pretraživača. Ažuriranje naslova stranice ili konfigurisanje opcija za dodire ekrana su uobičajene akcije koje omogućava ovaj modul. Ovaj paket takođe sadrži funkcije koje su potrebne za kompajliranje šablona van mreže u proizvodnim okruženjima.

- `@angular/platform-browser-dynamic` - Mi ćemo se često oslanjati na ovaj modul u toku izrade primera u ovoj knjizi, pošto će nam obezbediti funkciju za samopodizanje koja je potrebna za pokretanje aplikacija u toku kreiranja.
- `@angular/http` - Ovo je Angular 2 HTTP klijent koji ćemo detaljno opisati u Poglavlju 6, „*Asinhroni servisi podataka u Angularu 2*“.
- `@angular/router` - Ovo je ugrađeni ruter Angulara 2, koji je prilikom pisanja ove knjige još uvek bio u Beta verziji.
- `@angular/router-deprecated` - Ovo je snimak prethodne verzije ugrađenog rutera Angulara 2, koji je dostupan da bi obezbedio kompatibilnost sa starijim postojećim aplikacijama. U Poglavlju 7, „*Usmeravanje u Angularu* 2“, detaljno ćemo opisati neke od ovih istaknutih razlika u odnosu na novi ruter koji je još uvek u fazi razvoja.

U vreme pisanja ove knjige ovo su sve različite nezavisne biblioteke koje su potrebne kao ravnopravne zavisnosti u Angular 2 projektima, pored Angular 2 modula:

- `es6-shim` - Ova biblioteka predstavlja ECMAScript 6 polifile kompatibilnosti za starije JavaScript mašine (uglavnom Microsoft Internet Explorer). Ova zavisnost je sada potrebna, zato što mnogi pretraživači i dalje ne pružaju podršku za funkcije ECMAScripta 6, ali se nadamo da će to uskoro biti promenjeno. Umesto ove biblioteke, neke druge implementacije koriste core-js standardnu biblioteku. Izaberite onu koja vam se najviše dopada, ali obratite pažnju da biblioteka pravilno programski popunjava osnovne ES2015 API-je koji su potrebni za Angular 2.
- `zone.js` - Ovo je polifil za Zone specifikaciju, koja se koristi za rukovanje detekcijama promena u Angular 2 aplikacijama.
- `reflect-metadata` - Ova biblioteka ima podršku za dekoratore u Angular 2 klasama i refleksiju meta podataka u komponentama. Dekoratore u akciji ćemo videti kasnije u ovom poglavlju i detaljnije ćemo opisati njihove različite vrste i implementaciju u Poglavlju 2, „*Uvod u TypeScript*“. Dekoratori su osnovni deo Angulara 2.
- `rxjs` - Ovu biblioteku je razvila kompanija „Microsoft Open Technologies, Inc.“. Kako navodi „Microsoft“, to je set biblioteka za kreiranje asinhronih programa, zasnovanih na događajima, koje koriste primetne kolekcije i kompoziciju Array#extras stila u JavaScriptu. Ukratko, RxJS je biblioteka za upravljanje klasom Observables, koja omogućava da se kreiraju aplikacije potpuno reaktivne za asinhrone promene stanja. Klasu Observables će u budućnosti standardizovati moderni pretraživači, pa ćemo do tada moći da ovladamo ovom zavisnošću.

Ove zavisnosti mogu se razviti bez prethodnog obaveštenja, pa, stoga, pogledajte GitHub skladište da biste pronašli najnoviju, ažuriranu listu zahteva.

Verovatno ćete biti iznenađeni količinom biblioteka koje su za Angular 2 potrebne i činjenicom da ove zavisnosti nisu deo samog Angular paketa. Razlog što nisu deo Angular paketa je taj što ovi rekviziti nisu specifični za Angular 2, već i za većinu modernih JavaScript aplikacija.

Uz sve ove zavisnosti i nezavisne biblioteke, možete da pokrenete sledeći set bash komandi u konzoli terminala kada kreirate direktorijum za projekat koji ćemo opisati u ovoj knjizi:

```
$ mkdir learning-angular2
$ cd learning-angular2
$ npm init
$ npm install @angular/common @angular/core @angular/compiler --save
$ npm install @angular/platform-browser @angular/platform-browser-
dynamic --save
$ npm install @angular/router @angular/router-deprecated --save
$ npm install @angular/http --save
$ npm install es6-shim reflect-metadata rxjs zone.js --save
```

Pored zavisnosti koje su prethodno izlistane, takođe treba da instaliramo systemjs, paket univerzalnog programa za učitavanje modula da bismo podržali učitavanje modula između jedinica koda kada su prevedene u ES5. Paket systemjs nije jedina dostupna opcija za upravljanje učitavanjem modula u Angularu 2. U stvari, možemo da ga zamenimo drugim paketom programa za učitavanje modula, kao što je WebPack (https://webpack.github.io/), mada svi primeri koji se nalaze u ovoj knjizi koriste SystemJS za rukovanje injektiranjem koda. Mi ćemo instalirati SystemJS, označavajući ga kao zavisnost programiranja, izvršavanjem sledeće komande:

```
$ npm install systemjs –save
```

Na kraju ćemo instalirati `Bootstrap` u aplikaciju da bismo lako mogli da kreiramo lep korisnički interfejs za primer aplikacije, koji ćemo izgraditi postepeno po poglavljima. Ovo nije zahtev Angulara 2, već posebna zavisnost projekta koji ćemo raditi u ovoj knjizi:

```
$ npm install bootstrap –save
```

Instalacija može da prikaže različita upozorenja, u zavisnosti od verzije svake ravnopravne zavisnosti koju zahteva Angular 2 u ovom trenutku, pa preporučujem da, u slučaju greške, koristite najnoviju verziju fajla `package.json`, koji je dostupan u skladištu koda za ovu knjigu na adresi https://github.com/deeleman/learningangular2/blob/master/chapter_01/package.json.

Preuzmite fajl u radni prostor direktorijuma i pokrenite komandu npm install. NPM će, umesto vas, automatski pronaći i instalirati sve zavisnosti.

Korisnici Mac OS-a koji nisu zatražili vlasnička prava za npm direktorijum koji se nalazi u /usr/local/bin/npm (ili /usr/local/npm za one korisnike koji koriste verziju operativnog sistema stariju od verzije Mac OS El Capitan) možda treba da izvrše komandu npm install, koristeći privilegije programa.

Instaliranje TypeScripta

Sada imamo kompletan set Angular 2 izvora i njihove zavisnosti, `Bootstrap` modul za ulepšavanje projekta i SystemJS za rukovanje učitavanjem modula i generisanjem paketa.

Međutim, TypeScript verovatno još nije dostupan na vašem sistemu. Instalirajmo ga i učinimo ga globalno dostupnim u okruženju da bismo kasnije mogli da iskoristimo njegov prikladan CLI za kompajliranje fajlova:

```
$ npm install -g typescript
```

Odlično! Skoro smo završili posao. Poslednji korak podrazumeva informisanje TypeScripta o načinu na koji želimo da upotrebimo kompajler unutar projekta. Da bismo to uradili, izvršićemo sledeću jednokratnu komandu:

```
$ tsc --init --experimentalDecorators --emitDecoratorMetadata
--target ES5 --module system --moduleResolution node
```

U suštini, upravo smo pokrenuli TypeScript projekat (koji je naš Angular 2 projekat) koji ima podršku za eksperimentalne dekoratore (kao što smo već rekli, to su nove funkcije u ES7-u i TypeScriptu koje Angular 2 intenzivno koristi) i podešen SystemJS kao standardni mehanizam za importovanje modula i zavisnosti između fajlova.

Kao rezultat ove akcije, pronaći ćemo novi `tsconfig.json` fajl u osnovnom direktorijumu projekta, uključujući i podešavanja koja su potrebna TypeScript kompajleru za prevođenje koda komponente na čist ECMAScript 5 JavaScript kod koji mogu da čitaju aktuelni pretraživači.

Ne zaboravite da pretraživači ne pružaju podršku za TypeScript ili ECMAScript 6, pa ćemo, stoga, prevesti kod na neku verziju JavaScripta koji je dobro podržan u pretraživačima.

Sledi primer takvog fajla:

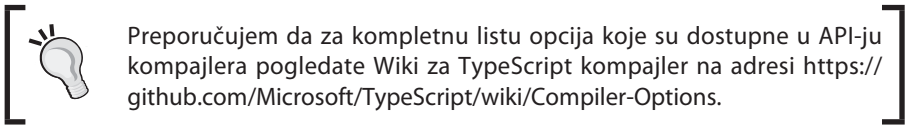
```
{
  "compilerOptions": {
    "experimentalDecorators": true,
    "emitDecoratorMetadata": true,
    "target": "es5",
    "module": "system",
    "moduleResolution": "node",
    "noImplicitAny": false,
    "outDir": "built",
    "rootDir": ".",
    "sourceMap": false
  },
  "exclude": [
    "node_modules"
  ]
}
```

Jednostavno, zar ne? Set parametara koji je uključen u manifest konfiguracije je dovoljno jasan, ali možemo da istaknemo tri interesantna parametra:

- `rootDir` ukazuje na direktorijum koji će kompajler upotrebiti za skeniranje TypeScript fajlova za kompajliranje (trenutno je osnovni direktorijum u našem primeru).
- `outDir` definiše gde će kompajlirani fajlovi biti pomereni, osim ako definišemo sopstvenu putanju, koristeći parametar --`outDir` u komandnoj liniji. Standardni direktorijum koji će kompajler upotrebiti je direktorijum `built`, koji je kreiran prilikom pokretanja na istoj lokaciji na kojoj se nalazi i fajl `tsconfig.json`.
- `sourceMap` podešava opcije mapiranja izvornog koda, što pomaže prilikom ispravljanja grešaka.

Promenite vrednost ovog parametra na `true` ako želite da fajlovi izvorne mape budu generisani prilikom pokretanja da biste mogli da pratite kod unazad do njegovog izvora, koristeći programerske alatke pretraživača u slučaju da se generiše izuzetak.

Pored ovih parametara, takođe može da se vidi da smo označili direktorijum `node_modules` kao isključen, što znači da će `tsc` komanda izostaviti taj direktorijum i sve njegove sadržaje prilikom transpilacije TypeScript fajlova u ES5 kroz stablo aplikacije.



Instaliranje TypeScript typingsa

Pored zavisnosti projekta, kao što su `Bootstrap` i zavisnosti samog Angulara 2, TypeScript zahteva neke dodatne biblioteke da bismo mogli da ga iskoristimo na najbolji mogući način. Konkretno, ES6 proširuje JavaScript okruženje metodima i API-jima, koji treba da budu opisani u TypeScript kompajleru, jer, u suprotnom, neće ih prepoznati kao deo sintakse i prilikom kompajliranja će biti prikazana greška. Kad god treba da damo instrukcije TypeScript kompajleru o JavaScript API-ju, bilo izvornom ili bilo kom drugom API-ju koji pripada nezavisnoj biblioteci, želećemo da upotrebimo fajl definicije TypeScript tipa.

Fajl definicije TypeScript tipa je, u stvari, fajl koji ima ekstenziju `d.ts` i sadrži TypeScript interfejse (više informacija pronaći ćete u Poglavlju 2, „*Uvod u TypeScript*“) da bismo mogli bolje da izvršimo proveru tipa u realnom vremenu i sprečimo greške kompajlera. Instaliranje fajlova definicije tipa u projektima nije teško i zahteva samo da alatka `typings` bude instalirana u okruženju. U stvari, potrebno je da instaliramo fajl definicije tipa da bismo obezbedili da TypeScript kompajler upozna najnoviju verziju ES6 API-ja. Dobra vest je da možemo da instaliramo alatku za upravljanje TypeScript definicijama direktno iz NPM registra, pa možemo da automatizujemo proces pretrage, instalacije i primene fajlova definicije tipa. Vratimo se sad u konzolu i nastavimo kod unosom sledećih komandi:

```
$ npm install -g typings
$ typings install es6-shim --ambient --save
```

Prvo ćemo instalirati alatku `typings` globalno, a zatim ćemo iskoristiti `typings` CLI za instaliranje fajla definicije `es6-shim` tipova u projekat, kreirajući fajl `typings.json` - on će čuvati reference u početnom izvoru za sve fajlove definicije tipa koje ćemo instalirati sada i kasnije. Kreiran je novi direktorijum, pod nazivom `typings`, koji sadrži potrebne fajlove definicije. Bez njih, osnovne funkcije ES6-a, kao što su novi funkcionalni metodi klase `Array`, ne bi bili dostupne.

Pre nego što nastavimo, potrebno je da izvršimo još jedan korak koji se odnosi na TypeScript `typings`. Kada instaliramo fajlove definicije tipa, CLI generiše dva façade fajla: `typings/main.d.ts` i `typings/browser.d.ts`. Međutim, trebalo bi da samo jedan fajl bude izložen TypeScript kompajleru, jer će, u suprotnom, podići izuzetak nakon pronalaska duplirane definicije tipa. Pošto mi kreiramo čeone aplikacije, upotrebićemo fajl browser.d.ts, a izostavićemo fajl main.d.ts i njegove povezane fajlove definicije, tako što ćemo ih označiti kao isključene u fajlu `tsconfig.json`:

```
{
  "compilerOptions": {
    "experimentalDecorators": true,
    "emitDecoratorMetadata": true,
    "target": "es5",
    "module": "system",
    "moduleResolution": "node",
    "noImplicitAny": false,
    "outDir": "built",
    "rootDir": ".",
    "sourceMap": false
  },
  "exclude": [
    "node_modules",
    "typings/main.d.ts",
    "typings/main"
  ]
}
```

Sa druge strane, preporučljivo je isključiti direktorijum `typings` iz distribucije projekta njegovim uključivanjem u fajlu `.gitignore`, isto kao što obično radimo sa direktorijumom `node_modules`. Želećemo da uključimo samo `typings.json` manifest kada distribuiramo aplikaciju, a zatim će svim procesima instalacije rukovati npm, pa je, stoga, veoma dobro uključiti instalaciju fajlova definicije tipa kao akcije kojima rukuje postinstall skript u fajlu `package.json`. Na ovaj način možemo da instaliramo npm zavisnosti i fajlove definicije u jednom koraku. Kod je sledeći:

```
"scripts": {
  "typings": "typings",
  "postinstall": "typings install"
},
```

Kada primenjujemo ovaj pristup, paket `typings` treba da bude uključen u fajl `package.json` kao deo razvojnih zavisnosti. Stoga ćemo reinstalirati ovaj paket, koristeći oznaku `--save-dev`. Pogledajte skladište koda za ovu knjigu na veb sajtu GitHub da biste preuzeli najnoviju verziju fajla `package.json` za ovo poglavlje.

HELLO, ANGULAR 2!

Kada je postavljen paket Angular 2 biblioteka i kada je dostupna puna podrška za TypeScript, vreme je da sve testiramo. Prvo ćemo kreirati prazan fajl, pod nazivom `hello-angular.ts` (.ts je izvorna ekstenzija za TypeScript fajlove), u osnovnom direktorijumu našeg radnog direktorijuma.

Ovde ćemo naići na prvu konvenciju kodiranja od mnogih koje ćemo opisati u ovoj knjizi - na imenovanje fajla. Fajlove modula ćemo imenovati korišćenjem karaktera povezanih crticom (kebab case). U Poglavlju 5, „Izgradnja aplikacije pomoću Angular 2 komponenata“, detaljnije ćemo opisati konvencije imenovanja i najbolju praksu za kodiranje Angular 2 aplikacija. Do tada, upotrebićemo, radi lakšeg učenja, neke nešablonske nazive, što će iskusniji korisnici uskoro i primetiti.

Sada otvorite fajl hello-angular.ts i napišite sledeći kod na njegovom vrhu:

```
import { Component } from '@angular/core';
import { bootstrap } from '@angular/platform-browser-dynamic';
```

Upravo ste importovali najosnovniji tip i funkciju koji će vam biti potrebni za izgradnju osnovne komponente u sledećem odeljku. Sintaksa importovanja će biti poznata onim čitaocima koji već poznaju ECMAScript 6 - čitaoci koji ga ne poznaju ne treba da brinu, jer ćemo ga opisati detaljnije u Poglavlju 2, „*Uvod u TypeScript*“.

TypeScript klase

Definišimo sada klasu:

```
class HelloAngularComponent {
  greeting: string;
  constructor() {
    this.greeting = 'Hello Angular 2!';
  }
}
```

ECMAScript 6 (kao i TypeScript) predstavio je klase kao jedan od osnovnih delova gradivnih blokova. Naš primer ima parametar polja klase pod nazivom `greetingm` koji je tipiziran kao string i popunjen je tekstualnim nizom unutar konstruktora, kao što možete da vidite u prethodnom kodu. Funkcija konstruktora je pozvana automatski kada je kreirana instanca klase i svaki parametar (kao i funkcija) treba da bude označen tipom koji predstavlja (ili se vraća u slučaju funkcija).

Ne brinite o svemu ovome sada. U Poglavlju 2, „*Uvod u TypeScript*“, pružićemo vam informacije koje su potrebne da biste bolje razumeli mehaniku TypeScripta. Sada ćemo da se fokusiramo na aktuelni raspored naše komponente. Verovatno ste primetili strukturu naziva, što odgovara još jednoj uobičajenoj konvenciji u Angularu 2. Svaku klasu ćemo definisati tako što ćemo prvi karakter svake reči ispisati velikim slovom (Pascal casing) i dodaćemo sufiks koji ukazuje na vrstu klase (bez obzira da li je to komponenta, komanda, usmeravanje...) - to je u ovom slučaju `Component`.

Predstavljanje dekoratora meta podataka

Klasa kontrolera koju smo upravo kreirali pruža nam mašineriju koja je potrebna za instanciranje objekta, izlažući parametar greeting, ali i dalje treba da primenimo neke dekoracije Angulara 2 da bismo objekat pretvorili u komponentu. Već smo importovali `Component` klasu meta podataka, a sada ćemo da pokrenemo klasu i da je dekorišemo na sledeći način:

```
@Component({
  selector: 'hello-angular',
  template: '<h1> {{greeting}} </h1>'
})

class HelloAngularComponent {
  greeting: string;
  constructor() {
    this.greeting = 'Hello Angular 2!';
  }
}
```

Dekorator je veoma interesantna eksperimentalna funkcija predstavljena u ECMAScriptu 7, a kasnije je prihvaćena i implementirana u TypeScript za dekorisanje klasa pomoću meta podataka. Postoji nekoliko vrsta dekoratora - svi su prepoznatljivi po simbolu @, koji je upotrebljen kao prefiks. Uvod u dekoratore ćemo vam predstaviti u Poglavlju 2, „*Uvod u TypeScript*“, aliopširniji prikaz logike osnove dekoratora nije tema ove knjige. Međutim, napredovanjem kroz poglavlja ove knjige bolje ćete ih upoznati.

U ovom primeru mi ukazujemo kompajleru da je klasa `HelloAngularComponent`, u stvari Angular 2, komponenta. Namenjeno je da komponentu kapsulira prilagođena komponeta `<hello-angular>` a svojstvo šablona definiše internu HTML strukturu naše komponente. Kao što smo već pomenuli, prilagođeni elementi koji kapsuliraju HTML šablone su osnova veb komponenata.

Kompajliranje TypeScripta u JavaScript prilagođen pretraživaču

Završili smo predstavljanje osnova TypeScripta, ali, nažalost, najverovatnije je da naš izabrani pretraživač neće podržavati TypeScript. Dakle, treba da kompajliramo izvorni kod u „dobar stari“ ECMAScript 5 JavaScript kod.

Dobra vest je da TypeScript CLI sadrži alatke za kompajliranje TypeScripta u JavaScript kod. Da biste kompajlirali kod, samo otvorite prozor terminala i ukucajte sledeću komandu na lokaciji fajla `hello-angular.ts`:

```
$ tsc --watch
```

Novi hello-angular.js fajl će biti prikazan unutar direktorijuma built (ili na putanji koju ste definisali u parametru outDir u fajlu `tsconfig.json`) i u fajlu je sadržana ECMAScript 5 verzija TypeScript koda koji smo upravo kreirali. Ovaj fajl već sadrži neki funkcionalan kod za implementiranje podrške za Metadata dekorator koji smo konfigurisali.

Vidite oznaku --watch u komandi. Ovaj parametar informiše kompajler da želimo da prevođenje bude automatski pokrenuto ponovo prilikom menjanja bilo kod fajla. Ne obraćajte pažnju na oznaku ako želite da kompajlirate kod samo jednom i nema potrebe da se prate promene koda.

Naša komponenta izgleda bolje i sada možemo da počnemo da je koristimo, ali i dalje treba da je ugradimo negde da bismo je videli „uživo“. Vreme je da definišemo HTML skript ili veb kontejner u koji ćemo postaviti komponentu.

HTML kontejner

Kreirajte HTML fajl u osnovnom direktorijumu radnog prostora i dodelite mu naziv `index.html`. Zatim ga popunite sledećim kodom:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Hello Angular 2!</title>
```

```
    <script src="node_modules/es6-shim/es6-shim.min.js"></script>
    <script src="node_modules/zone.js/dist/zone.js"></script>
    <script src="node_modules/reflect-metadata/Reflect.js"></
script>
    <script src="node_modules/systemjs/dist/system.js"></script>
    <script src="node_modules/rxjs/bundles/Rx.js"></script>

    <script src="systemjs.config.js"></script>

  </head>

  <body>

  </body>

</html>
```

Ovo je najosnovnija moguća verzija HTML kontejnera za Angular 2 aplikaciju koju smo mogli da smislimo. Ovaj kontejner je dobar, zato što će većim delom prezentacije logike i upravljanjem zavisnostima rukovati sama komponenta.

Međutim, u ovom šablonu dve stavke privlače našu pažnju. Sa jedne strane, pronašli smo blok skripta koji je umetnut. On sadrži sve ravnopravne zavisnosti koje su potrebne za programsko popunjavanje ES2015 (ES6) funkcija, Zone i Observables specifikacije, koje obezbeđuju punu podršku za dekoratore meta podataka, i, na kraju, ali ne i manje važno, implementirane funkcije za dinamičko učitavanje modula u aplikaciju.

Nemojte pokušavati da promenite sortiranje rasporeda u ovim blokovima koda ako ne želite da naiđete na neočekivane izuzetke.

Zatim ćemo predstaviti skriptove za umetanje u novom fajlu, pod nazivom systemjs.config.js. Ovaj fajl tek treba da napišemo, pa hajde da ga kreiramo u osnovnom direktorijumu projekta, koristeći sledeću implementaciju. Ovo podešavanje ćemo rastaviti na sledeće pasuse:

```
(function() {

  var pathMappings = {
    '@angular': 'node_modules/@angular',
    'rxjs': 'node_modules/rxjs',
  };

  var packages = [
```

```
      '@angular/common',
      '@angular/compiler',
      '@angular/core',
      '@angular/http',
      '@angular/platform-browser',
      '@angular/platform-browser-dynamic',
      '@angular/router',
      '@angular/router-deprecated',
      '@angular/testing',
      'rxjs',
      'built',
    ];

    var packagesConfig = {};

    packages.forEach(function(packageName) {
      packagesConfig[packageName] = {
        main: 'index.js',
        defaultExtension: 'js'
      };
    });

    System.config({
      map: pathMappings,
      packages: packagesConfig,
    });

  })();
```

Kao što možete da vidite, fajl `systemjs.config.js` sadrži primarno izraze trenutno pozvane funkcije. To znači da će ovaj blok koda biti izvršen čim ga pretraživač analizira. Pregledajte sada svaki deo ove rutine:

```
var pathMappings = {
  '@angular': 'node_modules/@angular',
  'rxjs': 'node_modules/rxjs',
};
```

Ovde smo definisali literal tipa objekta koji sadrži parove ključ/vrednost indeksa koji sadrži cele putanje. Mi ćemo upotrebiti ovaj literal tipa objekta za konfigurisanje alijasa putanje u SystemJS-u kasnije u ovom istom fajlu. Stoga, izvršavanje iskaza `import ,@angular/core‘` će dati instrukcije SystemJS-u da importuje osnovni paket iz direktorijuma `node_modules/@angular/core`. Međutim, paketi nisu importovani u celosti. Potrebna nam je tačka unosa za takav paket i, stoga, treba da informišemo SystemJS koji façade fajl bi trebalo da traži prilikom importovanja celog modula, što ćemo uraditi definisanjem niza modula, pa kreiranjem literala tipa objekta u kojem je putanja svakog modula ključ parametra koji sadrži glavnu tačku unosa i standardnu ekstenziju fajla za objekat konfiguracije za takve putanje:

```
var packages = [
  '@angular/common',
  '@angular/compiler',
  '@angular/core',
  '@angular/http',
  '@angular/platform-browser',
  '@angular/platform-browser-dynamic',
  '@angular/router',
  '@angular/router-deprecated',
  '@angular/testing',
  'rxjs',
  'built',
];

var packagesConfig = {};

packages.forEach(function(packageName) {
  packagesConfig[packageName] = {
    main: 'index.js',
    defaultExtension: 'js'
  };
});
```

Rezultirajuća promenljiva `packagesConfig` predstavlja prethodno spomenuti literal tipa objekta konfiguracije. Takođe uključuje glavni fajl unosa i podatke standardne ekstenzije za izgradnju direktorijuma, a to je direktorijum u kojem će TypeScript kompajler čuvati prevedene fajlove u toku kreiranja aplikacije.

Kada je postavljena ova konfiguracija, poslednji korak je konfigurisanje SystemJS-a korišćenjem mapiranja ovih putanja, tačke unosa i standardne ekstenzije fajla koje se očekuju za svaki paket predstavljen u prethodnim putanjama:

```
System.config({
  map: pathMappings,
  packages: packagesConfig,
});
```

Ovim su završena sva podešavanja koja su potrebna za SystemJS. Kada su ona postavljena, možemo da pokrenemo aplikaciju i iskoristimo ES2015 sintaksu za importovanje modula u kodu, što ćemo opisati u sledećim odeljcima.

Serviranje primera ove knjige

Pre nego što nastavimo naš primer, potreban nam je lokalni veb server za izvršavanje primera koji su sadržani u ovoj knjizi. Ako ste već koristili veb server koji možete da konfigurišete da ukazuje na radni direktorijum, izostavite sledeći odeljak. Ako nemate veb server, podesite ga u radnom prostoru. Da bismo vam olakšali posao, preporučujemo da instalirate veoma moćan i jednostavan lite-server modul iz NPM-a:

```
$ npm install -g lite-server
```

Zatim, možete da pokrenete veb server pomoću live-reloading funkcije, tako što ćete pokrenuti sledeću komandu u terminalu nakon što otvorite direktorijum projekta:

```
$ lite-server
```

Nakon izvršenja prethodne komande, prozor pretraživača će biti otvoren i prikazaće radni direktorijum. U zvaničnom skladištu NPM modula možete da vidite sve dostupne opcije (https://github.com/johnpapa/lite-server).

Preporučljivo je instalirati lite-server paket, zajedno sa paketima typescript i concurrently, kao zavisnosti instalirane upotrebom oznake --save-dev flag. Na taj način možete istovremeno da pokrenete TypeScript kompajler u watch režimu i lokalni server pomoću jedne komande koja može da bude obuhvaćena u start skriptu fajla package.json. Zatim, možete odmah da započnete tako što ćete pristupiti radnom direktorijumu i izvršiti komandu npm start. Skladište koda za ovu knjigu na sajtu GitHub implementira ovaj pristup, pa slobodno pozajmite primer fajla package.json.

Spajanje

Naš HTML fajl je sada spreman za hostovanje Angular 2 komponente. Da bismo dodali Angular 2 komponentu, potrebno je ponovo da editujemo šablon i dodamo prilagođeni element, koristeći isti naziv taga koji smo definisali u parametru selector za komponentu. Zatim ćemo importovati falj koji sadrži deklaraciju klase komponente, a za to ćemo upotrebiti API SystemJS-a. Pogledajte ove promene u sledećem primeru:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Hello Angular 2!</title>
    <script src="node_modules/es6-shim/es6-shim.min.js"></script>
    <script src="node_modules/zone.js/dist/zone.js"></script>
    <script src="node_modules/reflect-metadata/Reflect.js"></
script>
    <script src="node_modules/systemjs/dist/system.js"></script>
    <script src="node_modules/rxjs/bundles/Rx.js"></script>

    <script src="systemjs.config.js"></script>
      // Here we import the component module
      // with no file extension
      System.import ('built/hello-angular');
    </script>
  </head>
  <body>
    <!-- This is our custom element tag -->
    <hello-angular></hello-angular>
  </body>
</html>
```

Sada možemo da kreiramo elemente koji će renderovati šta god definišemo u njima. Hajde da kreiramo stranicu na veb serveru i vidimo je u akciji, tako što ćemo otvoriti stranicu http://localhost:3000/ (ili na kojem god hostu ili portu je pokrenut veb server).

Nažalost, ako ponovo učitamo pretraživač, ništa se neće desiti i videćemo samo praznu stranicu. Razlog za prikaz prazne stranice je što i dalje treba da podignemo komponentu da bismo je pokrenuli na HTML stranici.

Hajde da se vratimo fajlu komponente `hello-angular.ts` i dodamo finalnu liniju koda:

```
import { Component } from '@angular/core';
import { bootstrap } from '@angular/platform-browser-dynamic';

@Component({
```

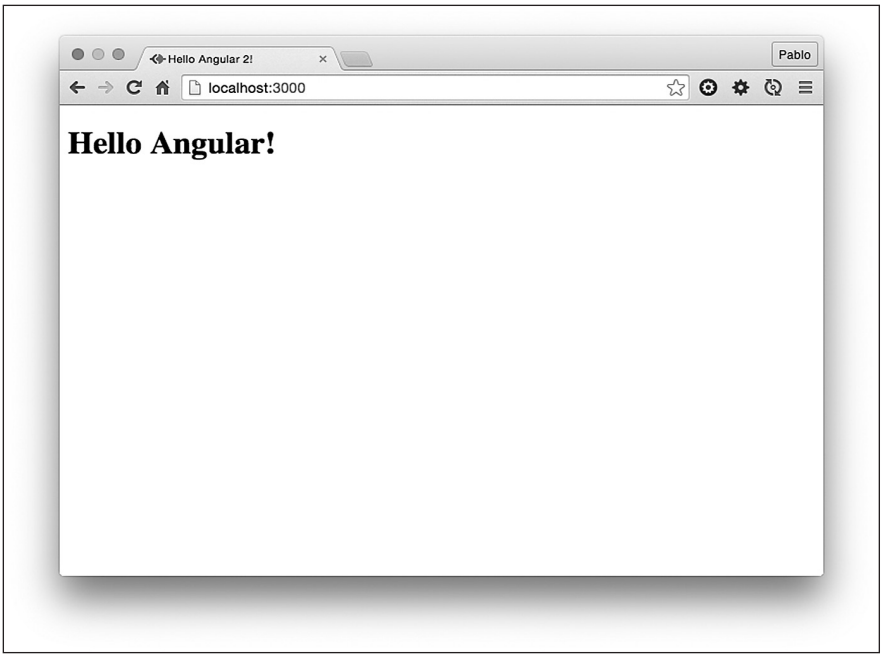
```
  selector: 'hello-angular',
  template: '<h1> {{greeting}} </h1>'
})
class HelloAngularComponent {
  greeting: string;
  constructor() {
    this.greeting = 'Hello Angular 2!';
  }
}

bootstrap(HelloAngularComponent); // Component is bootstrapped!
```

Komanda `bootstrap` instancira klasu kontrolera koju prosleđujemo kao argument i koristi je za postavljanje konstrukcije za celu aplikaciju. U suštini, metod ponovnog podizanja pokreće sledeće akcije:

- Analizira komponentu koja je konfigurisana kao prvi argument i proverava njenu vrstu.
- Pretražuje DOM nakon elementa upoređivanjem tagova selektora komponente.
- Kreira podređeni injektor koji će rukovati injektiranjem zavisnosti u konkretnoj komponenti i svim podređenim komandama (uključujući komponente koje su takođe komande) koje takva komponenta može imati (veoma slično grananju stabla).
- Kreira novi Zone. Mi nećemo opisati Zone u ovoj knjizi, ali recimo samo da su Zone zadužene za upravljanje mehanizmom za detekciju promena svake instance ponovno podignute komponente na izolovani način.
- Kreira Shadow DOM kontekst u prilagođenom elementu koji je identifikovan selektorom komponente i renderuje se unutar HTML-a koji je definisan u šablonu komponente.
- Klasa kontrolora komponente je instancirana odmah i pokreće se mašinerija za detekciju promena. Sada, kada imamo postavljena rezervisna polja za Shadow DOM, provajderi podataka su pokrenuti i podaci su injektirani na mesto na koje su zatraženi.

Kasnije u ovoj knjizi ćemo opisati kako možete da iskoristite komandu `bootstrap` za prikaz informacija o ispravljanju grešaka ili kako provajderi aplikacije mogu globalno da budu poništeni kroz celu aplikaciju, pa, stoga, dependency injector koji je kreiran u Angularu 2 bira odgovarajuću zavisnost na odgovarajućem mestu.

Nadamo se da ste pokrenuli TypeScript kompajler u `watch` režimu. Ako niste, izvršite komandu `tsc` da biste preveli kod u ES5 i ponovo učitajte pretraživač. Možemo biti zadovoljni našim renderovanim sadržajem u prvoj Angular 2 komponenti.



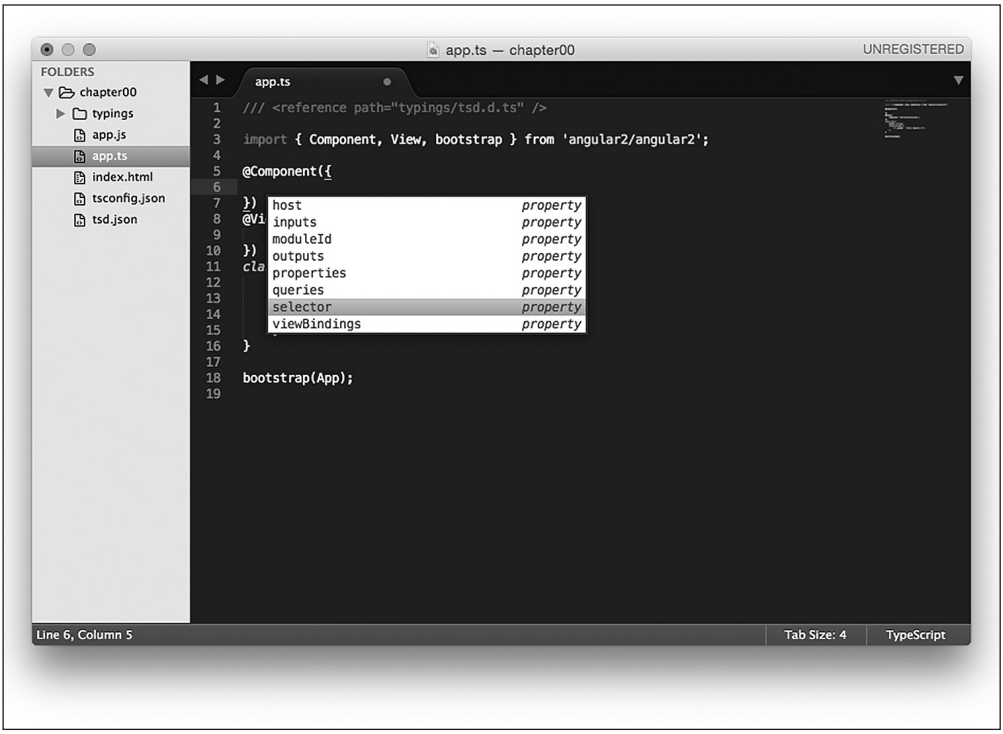
UNAPREĐENJE NAŠEG IDE-A

Pre nego što nastavimo naše „putovanje“ kroz Angular 2, vreme je da pregledamo IDE. Naš omiljeni editor koda može postati nenadmašan „saveznik“ kada je reč o preduzimanju agilnog toka rada koji podrazumeva TypeScript kompilaciju u vreme pokretanja, statičnu proveru tipa i introspekciju, završavanje koda i vizuelnu pomoć za ispravljanje grešaka i izgradnju aplikacije. Hajde sada da opišemo i neke od osnovnih editora koda i pregledamo kako svaki od njih može da nam pomogne prilikom kreiranja Angular 2 aplikacija. Ako ste zadovoljni pokretanjem kompilacije TypeScript fajlova iz komandne linije i ne želite da imate vizuelnu pomoć, slobodno preskočite sledeći odeljak. Ako želite vizuelnu pomoć, pređite na sledeći odeljak koji će vam otkriti IDE po vašem izboru.

Sublime Text 3

Ovo je verovatno jedan od najčešće upotrebljavanih editora koda, mada je u poslednje vreme izgubio malo na popularnosti, jer korisnici se radije opredeljuju za neke druge, novije editore, kao što je GitHubov Atom. Ako je to vaš omiljeni editor, pretpostavljamo da je već instaliran na vaš sistem i da imate i Node (što je očigledno, jer inače ne biste instalirali TypeScript pomoću NPM-a). Da biste obezbedili podršku za editovanje TypeScript koda, treba da instalirate „Microsoftov“ TypeScript plugin, koji je dostupan na adresi https://github.com/Microsoft/TypeScriptSublime-Plugin. Pogledajte ovu stranicu da biste naučili kako da instalirate plugin i da biste pregledali sve prečice i mapiranja ključeva.

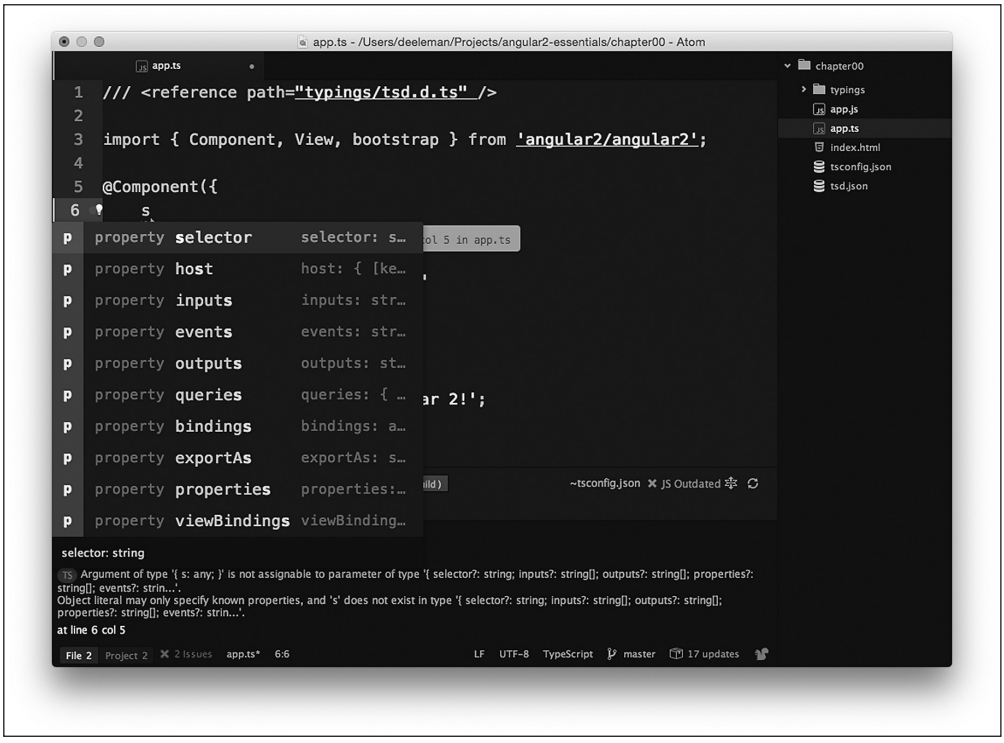
Kada je plugin uspešno instaliran, potrebna vam je samo prečica Ctrl + razmaknica da biste prikazali smernice koda na osnovu introspekcije tipa (vidite sledeću sliku). Pored toga, možete da pokrenete proces izgradnje i kompajlirate fajl u JavaScript u kojem radite, tako što ćete pritisnuti funkcijski taster F7. Izveštaj o greškama u kodu u realnom vremenu je još jedna moderna funkcija koju možete da uključite iz menija komande.



Atom

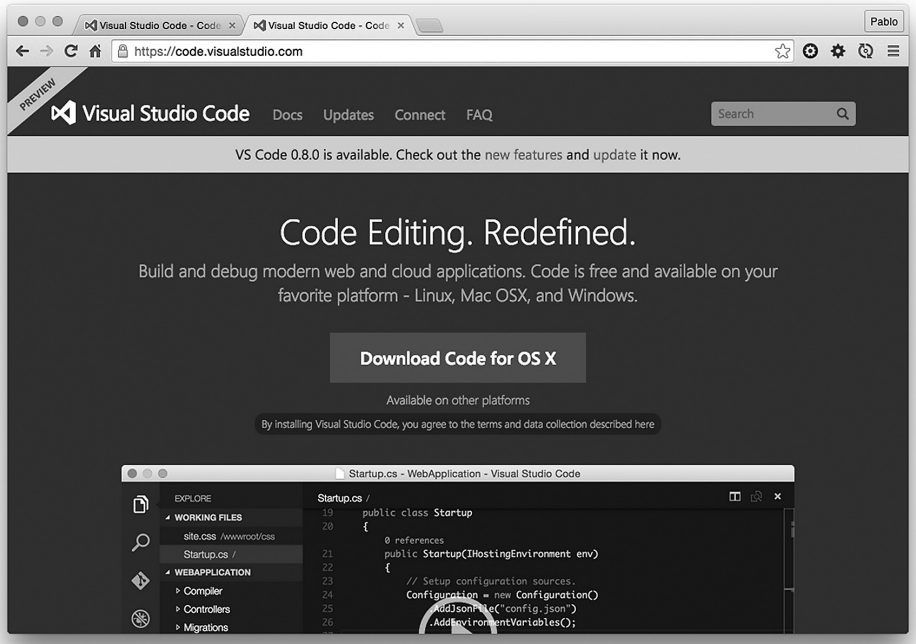
„GitHub“ je proizveo Atom, veoma prilagodljivo okruženje koje je lako za instaliranje novih paketa, što ga je učinilo omiljenim IDE-om za mnogo korisnika. Vredi napomenuti da su primeri koda koji su obezbeđeni za ovu knjigu, u stvari, kodirani korišćenjem samo Atoma.

Da bismo olakšali upotrebu TypeScripta prilikom kodiranja Angular 2 aplikacija, treba da instaliramo Atom TypeScript paket. Možemo da ga instaliramo pomoću APM CLI-a ili možemo jednostavno da upotrebimo ugrađeni instaler paketa. Funkcije koje su uključene su, uglavnom, iste koje smo imali u Sublime editoru nakon instaliranja Microsoft paketa: automatske smernice za kod, statična provera tipa, introspekcija koda ili automatska izgradnja prilikom snimanja. Pored toga, ovaj paket uključuje i odgovarajući ugrađeni `tsconfig.json` generator.



Visual Studio Code

Visual Studio Code je relativno novi editor koda koji je kreirao „Microsoft“ i koji je ozbiljan takmac u svetu Angulara 2, uglavnom zbog odlične podrške za TypeScript. TypeScript je bio, u većoj meri, projekat kojim je upravljao „Microsoft“, pa ima smisla da je jedan od njihovih popularnih editora kreiran sa ugrađenom podrškom za ovaj jezik. To znači da su uključene sve odlične funkcije koje će nam možda biti potrebne, uključujući označavanje sintakse i grešaka i automatsku izgradnju.



WebStorm

WebStorm je odličan editor koda, koji je kreirao „IntelliJ“. On je odličan izbor za kodiranje Angular 2 aplikacija zasnovanih na TypeScriptu. IDE ima ugrađenu podršku za TypeScript, zbog čega možemo da započnemo kreiranje Angular 2 komponenata već od prvog dana. WebStorm takođe implementira ugrađeni transpiler sa podrškom za pregled fajla, pa možemo da kompajliramo TypeScript kod u čist osnovni oblik JavaScripta, bez oslanjanja na bilo koji nezavisni plagin.

Korišćenje Gulpa sa drugim IDE-ovima

Možda vaš IDE nije ovde izlistan i ne želite sada da menjate vaš omiljeni editor koda, zbog čega nećete moći, zbog nekog razloga, da automatizujete TypeScript kompilaciju za projekat. Ili možda ne želite da se mučite sa TypeScript komandama na konzoli.

Ako možete da automatizujete TypeScript kompilaciju za projekat, slobodno pređite na sledeći odeljak. Međutim, ako ne želite da se mučite sa TypeScript komandama na konzoli, ne brinite. Pomoći će vam moderni JavaScript pokretači zadataka. Hajde da izaberemo Gulp (http://gulpjs.com) i vidimo kako možemo da kreiramo izuzetno jednostavan skript za automatizaciju TypeScript kompilacije u projektu.

Prvo ćemo nastaviti instalaciju zavisnosti. U suštini, instaliraćemo Gulp, a zatim `gulp-typescript`, kompajler TypeScripta za gulp sa postupnom podrškom za kompilaciju. U prozoru konzole ukucajte sledeće komande:

```
$ npm install -g gulp
$ npm install gulp gulp-typescript --save-dev
```

Kreiraćemo JavaScript fajl pod nazivom `gulpfile.js` u osnovnom direktorijumu projekta, koji ima sledeći sadržaj:

```
var gulp = require('gulp');
var ts = require('gulp-typescript');
var tsProject = ts.createProject('tsconfig.json');

gulp.task('build', function() {
  var tsResult = tsProject.src().pipe(ts(tsProject));
  return tsResult.js.pipe(gulp.dest('./built'));
});
```

Biće nam potreban fajl `tsconfig.json` u osnovnom direktorijumu projekta, pa će Gulp zadatak moći da preuzme parametre kompilacije iz ovog fajla. Od sada možemo da pokrenemo izgradnju, obrađujući fajlove koji su izlistani u parametru array za fajl `tsconfig.json` izvršavanjem sledeće komande:

```
$ gulp build
```

Nažalost, plagin `gulp-typescript` ne podržava pregled fajla, pa, ako želimo da pokrenemo proces izgradnje automatski uvek kada se promeni TypeScript fajl, potrebno je da se oslonimo na Gulpov izvorni metod `watch`. Potrebno je samo da dodamo sledeći blok koda na kraj fajla `gulpfile.js`:

```
gulp.task('watch', ['build'], function() {
  gulp.watch('./**/*.ts', ['build']);
});
```

Sada možemo da pokrenemo proces izgradnje i pregledamo promene fajla, tako što ćemo izvršiti sledeću komandu:

```
$ gulp watch
```

„ZARONIMO DUBLJE“ U ANGULAR 2 KOMPONENTE

Do sada smo prešli dug put da bismo naučili kako da kodiramo osnovnu šemu skriptovanja Angular 2 komponente. Međutim, pre nego što nastavimo sa apstraktnijim temama, dodajmo komponenti još funkcija i pregledajmo uobičajene osobine Angular aplikacija i komponenata.

Poboljšanje produktivnosti

Ponekad nam je potrebna pomoć da bismo se bolje fokusirali, posebno kada izvršavamo previše apstraktne zadatke koji zahtevaju dodatnu pažnju. Široko prihvaćen pristup je Pomodoro tehnika, u kojoj sklapamo listu zadataka, pa delimo listu na stavke koje treba da izvršimo, a koji ne zahtevaju više od 25 minuta za njihovo izvršenje. Kada izaberemo bilo koju od tih stavki, fokusiraćemo se da je izvršimo, bez ometanja, za 25 minuta pomoću tajmera. Više informacija o ovoj tehnici možete da nađete na stranici http://pomodorotechnique.com.

U ovoj knjizi kreiraćemo glavnu komponentu koja predstavlja ovu funkciju i popunićemo komponentu dodatnim funkcijama i stavkama korisničkog interfejsa koje su obuhvaćene unutar sopstvenih komponenata. Upotrebićemo Pomodoro tehniku da bismo započeli kreiranje Pomodoro tajmera.

Metodi komponente i ažuriranja podataka

Kreirajte novi pomodoro-timer.ts fajl u istom direktorijumu i popunite ga sledećim osnovnim implementacijama veoma jednostavne komponente. Ne brinite zbog složenosti, jer ćemo svaku promenu koja je izvršena pregledati i opisati nakon sledećeg bloka koda:

```
import { Component } from '@angular/core';
import { bootstrap } from '@angular/platform-browser-dynamic';

@Component({
  selector: 'pomodoro-timer',
  template: '<h1> {{ minutes }}:{{ seconds }} </h1>'
})
class PomodoroTimerComponent {
  minutes: number;
```

```
    seconds: number;

    constructor() {
      this.minutes = 24;
      this.seconds = 59;
    }
}

bootstrap(PomodoroTimerComponent);
```

Naša nova komponenta, da budemo iskreni, ne razlikuje se mnogo od one koju smo prethodno imali. Ažurirali smo nazive da bismo ih učinili jasnijim, a zatim smo definisali dva polja za parametre, statično tipizirana kao brojevi u klasi `PomodoroTimerComponent`. Ova polja su renderovana u prikazu i obuhvaćena su unutar elementa <h1>. Sada otvorite fajl index.html i zamenite prilagođeni element <hello-angular></hello-angular> novim <pomodoro-timer></pomodoro-timer> tagom. Možete da duplirate fajl index.html i da ga snimite pod drugim nazivom ako ne želite da izgubite HTML stranu naše moderne „Hello World“ komponente.

Napomena o imenovanju prilagođenih elemenata

Selektori u Angularu 2 su osetljivi na veličinu slova. Kao što ćete videti kasnije u ovoj knjizi, komponente su podsetovi komandi koje mogu da podržavaju veliki raspon selektora. Kada kreiramo komponente, potrebno je da podesimo sopstven naziv taga u parametru selector, koristeći konvenciju imenovanja u kojoj se koriste crtice između reči. Kada renderujemo takav tag u prikazu, uvek bi trebalo da ga zatvorimo kao element koji nije prazan. Dakle, pravilno je <customelement></custom-element>, dok će <customelement /> podići izuzetak. Na kraju da naglasimo da određeni uobičajeni nazivi ispisani Camel Case načinom mogu dovesti do konflikta u Angular 2 implementaciji, pa izbegavajte nazive kao što su AppView ili AppElement.

Želećemo da ažuriramo referencu u bloku `System.import(...)` u fajlu `index.html` da ukazuje na našu novu komponentu:

```
System.import('built/pomodoro-timer')
      .then(null, console.error.bind(console));
```

Sada je pravi trenutak da istaknemo da je import metod SystemJS-a asinhron i da vraća rezultat kada je modul uspešno učitan. Možemo da iskoristimo ovaj rezultat za prikaz eventualnih poruka o greškama u konzoli, što će biti veoma korisno kad god treba da ispravimo grešku u kodu. Kasnije u ovoj knjizi ćemo opisati kako da to uradite.

Ako otvorite prozor pretraživača i učitate ovaj fajl, videćete prikaz brojeva koji su definisani u klasi komponente. Međutim, mi želimo više od samog prikazivanja brojeva, zar ne? Mi želimo, u stvari, da brojevi prikazuju vreme tajmera, a to ćemo postići predstavljanjem ovih promena. Hajde da prvo predstavimo funkciju koju možemo da ponavljamo da bismo ažurirali tajmer. Dodajte sledeću funkciju iza funkcije konstruktora:

```
tick(): void {
  if (--this.seconds < 0) {
    this.seconds = 59;
    if (--this.minutes < 0) {
      this.minutes = 24;
      this.seconds = 59;
    }
  }
}
```

Kao što vidite, funkcije u TypeScriptu treba da budu označene vrstom vrednosti koju vraćaju, ili označene sa `void` u slučaju da ne vraćaju vrednost. Naša funkcija procenjuje aktuelnu vrednost za `minutes` i `seconds`, pa smanjuje njihove vrednosti ili ih samo resetuje na početne vrednosti. Zatim je pozvana ova funkcija svake sekunde pokretanjem intervala vremena iz klase konstruktora:

```
constructor() {
  this.minutes = 24;
  this.seconds = 59;
  setInterval(() => this.tick(), 1000);
}
```

Ovde smo prvi put u našem kodu upotrebili funkciju arrow (takođe poznatu i kao lambda funkcija, fat arrow), novu sintaksu za funkcije koju je uveo ECMAScript 6, koji ćemo detaljnije opisati u Poglavlju 2, „*Uvod u TypeScript*“. Funkcija `tick` je takođe, označena kao private, pa ne može biti ispitana ili izvršena van instance objekta `PomodoroTimerComponent`.

Do sada je sve u redu - imamo radni Pomodoro tajmer koji odbrojava od 25 minuta do 0 minuta, a zatim se ponovo pokreće. Problem je što se kod replicira. Hajde da malo raščlanimo sve da bismo sprečili dupliranje koda.

```
constructor() {
  this.resetPomodoro();
  setInterval(() => this.tick(), 1000);
}

resetPomodoro(): void {
  this.minutes = 24;
```

```
    this.seconds = 59;
  }

  private tick(): void {
    if (--this.seconds < 0) {
      this.seconds = 59;
      if (--this.minutes < 0) {
        this.resetPomodoro();
      }
    }
  }
```

Objasnili smo pokretanje (i resetovanje) minuta i sekundi unutar funkcije `resetPomodoro`, koja je pozvana prilikom instanciranja komponente ili kada brojač dostigne nula minuta. Međutim, sačekajte malo! Prema navodima o Pomodoro tehnici, korisnici koji je primenjuju smeju da se odmore između intervala ili čak mogu i da prave pauze u slučaju neočekivanih smetnji. Potrebno je da obezbedimo neku vrstu interaktivnosti da bi korisnik mogao da pokrene, zaustavi na određeno vreme i ponovo pokrene aktuelni Pomodoro tajmer.

Dodavanje interaktivnosti u komponentu

Angular 2 obezbeđuje vrhunsku podršku za događaje korišćenjem deklarativnog interfejsa koji podseća na interfejs u radnim okvirima, kao što je React. Hajde da prvo modifikujemo definiciju šablona:

```
@Component({
  selector: 'pomodoro-timer',
  template: `
    <h1> {{ minutes }}:{{ seconds }} </h1>
    <p>
      <button (click)="togglePause()">
        {{ buttonLabel }}
      </button>
    </p>
  `
})
```

Upotrebili smo višelinijski tekstualni niz. ECMAScript 6 je predstavio koncept nizova šablona koji su literali niza sa podrškom za ugrađene izraze, umetnuta vezivanja teksta i višelinijski sadržaj. Detaljnije ćemo ih opisati u Poglavlju 2, „*Uvod u TypeScript*“.

U međuvremenu, fokusirajmo se na činjenicu da smo predstavili novu vrstu HTML-a koja sadrži dugme za rukovanje događajem, koje „osluškuje“ događaje klika i izvršava metod `togglePause` prilikom klika na dugme. Ovaj (click) atribut možda još nikada ranije niste videli, čak i ako u potpunosti odgovara W3C standardima. Opisaćemo ovaj atribut detaljnije u Poglavlju 3, „*Implementiranje parametara i događaja u komponente*“. Fokusirajmo se sada na metod `togglePause` i na novo vezivanje `buttonLabel`. Prvo modifikujmo parametre klase da izgledaju kao u sledećem primeru:

```
class PomodoroTimerComponent {
  minutes: number;
  seconds: number;
  isPaused: boolean;
  buttonLabel: string;

  // … Rest of code will remain as it is below this point
}
```

Predstavili smo dva nova polja. Prvo polje `buttonLabel` sadrži tekst koji će kasnije biti prikazan na novokreiranom dugmetu. Polje `isPaused` je nova promenljiva koja će pretpostavljati vrednost `true`/`false`, u zavisnosti od stanja tajmera. Dakle, potrebno nam je mesto za prekopčavanje vrednosti ovog polja. Hajde da kreiramo metod `togglePause` koji smo ranije pomenuli:

```
togglePause(): void {
  this.isPaused = !this.isPaused;
  // if countdown has started
  if (this.minutes < 24 || this.seconds < 59) {
    this.buttonLabel = this.isPaused ? 'Resume' : 'Pause';
  }
}
```

U suštini, metod `togglePause` samo menja vrednost polja `isPaused` na suprotnu vrednost, a mi ćemo, u zavisnosti od te nove vrednosti i da li je tajmer pokrenut (što znači da bi svaka vremenska promenljiva imala vrednost nižu od početne vrednosti) ili ne, dodeliti dugmetu drugačiji naziv.

Sada treba da pokrenemo ove vrednosti; izgleda da ne postoji bolje mesto za to. Dakle, pokrenuta funkcija `resetPomodoro` je mesto na kojem će promenljive uticati na stanje klase:

```
resetPomodoro(): void {
  this.minutes = 24;
  this.seconds = 59;
  this.buttonLabel = 'Start';
  this.togglePause();
}
```

Izvršavanjem funkcije `togglePause` resetovaćemo Pomodoro tajmer da bismo se uverili da će se, uvek kada on dostigne stanje u kojem zahteva da bude resetovan, njegovo ponašanje promeniti na suprotno stanje od prethodnog. Ostaje nam još samo jedno podešavanje u metodu kontrolera koje rukuje odbrojavanjem:

```
private tick(): void {
  if (!this.isPaused) {
    this.buttonLabel = 'Pause';

    if (--this.seconds < 0) {
      this.seconds = 59;
      if (--this.minutes < 0) {
        this.resetPomodoro();
      }
    }
  }
}
```

Očigledno je da ne želimo da tajmer nastavi da odbrojava kada je „stavljen na pauzu“, pa ćemo obuhvatiti ceo skript u uslove. Pored toga, želećemo da prikažemo različit tekst na dugmetu uvek kada brojač nije na pauzi i kada stigne do kraja odbrojavanja. Zaustavljanje i resetovanje Pomodoro tajmera na početne vrednosti je očekivano ponašanje, što potvrđuje potrebu za funkcijom `togglePause` unutar objekta `resetPomodoro`.

Poboljšanje ispisa podataka u prikazu i sređivanje korisničkog interfejsa

Do sada smo ponovo učitali pretraživač i poigrali se novokreiranom funkcijom za prekopčavanje. Međutim, postoji još nešto što zahteva malo sređivanja: kada je brojač sekundi manji od 10, prikazuje jednocifrenu vrednost, umesto uobičajenog dvocifrenog broja, koji, obično, možemo da vidimo na digitalnim satovima. Srećom, Angular 2 implementira set deklarativnih pomagača koji formatiraju ispis podataka u šablonima. Nazivamo ih usmeravanjima i detaljnije ćemo ih opisati u Poglavlju 3, „I*mplementiranje parametara i događaja u komponente*“. Za sada ćemo predstaviti samo usmeravanje `number` u šablonu komponente i konfigurisaćemo ga da formatira ispis `seconds`, tako da se prikazuju dve cifre svo vreme dok se vrši odbrojavanje. Ažurirajmo šablon na sledeći način:

```
@Component({
  selector: 'pomodoro-timer',
    template: `
      <h1> {{ minutes }}:{{ seconds | number: '2.0' }} </h1>
```

```
        <p>
          <button (click)="togglePause()">
            {{ buttonLabel }}
          </button>
        </p>
      `

})
```

U suštini, dodali smo naziv usmeravanja u interpolirano povezivanje u šablonu, koje je razdvojeno simbolom vertikalne crte (|).Ponovo učitajte šablon i videćete kako se sekunde uvek prikazuju dvocifrenim brojem, bez obzira na vrednost koju tajmer prikazuje.

Kreirali smo potpuno funkcionalan Pomodoro tajmer vidžet, koji možemo da upotrebimo ponovo ili da ga ugradimo u složenije aplikacije. U Poglavlju 5, „*Izgradnja aplikacije pomoću Angular 2 komponenata*“, vodićemo vas kroz proces ugrađivanja i ugnežđivanja komponenata u kontekst većih stabala komponente.

U međuvremenu ćemo malo da sredimo korisnički interfejs da bi komponenta izgledala lepše. Već smo vam predstavili atribut klase u tagu dugmeta kao očekivanu implementaciju `Bootstrap` CSS radnog okvira u projekat. Sada treba da importujemo stilove koje smo preuzeli putem NPM-a prilikom instaliranja zavisnosti za projekat. Otvorite fajl `pomodoro-timer.html` i dodajte sledeći blok koda na kraj `<HEAD>` elementa:

```
<link rel="stylesheet" href="node_modules/bootstrap/dist/css/
bootstrap.min.css">
```

Sada ćemo da ulepšamo korisnički interfejs ubacivanjem lepog zaglavlja stranice ispred komponente:

```
<body>
  <nav class="navbar navbar-default navbar-static-top">
    <div class="container">
      <div class="navbar-header">
        <strong class="navbar-brand">My Pomodoro Timer</strong>
      </div>
    </div>
  </nav>
  <pomodoro-timer></pomodoro-timer>
</body>
```

Podešavanje komponente dugmeta pomoću `Bootstrap` klase dugmeta će dugmetu dati osobenost, obuhvatiće ceo šablon u centralnom kontejneru i dodaće lepu ikonicu na vrh, što će se odlično prilagoditi korisničkom interfejsu. Ažurirajmo šablon na sledeći način:

```
<div class="text-center">
  <img src="assets/img/pomodoro.png" alt="Pomodoro">
  <h1> {{ minutes }}:{{ seconds  | number: '2.0' }} </h1>
  <p>
    <button (click)="togglePause()"
      class="btn btn-danger">
      {{ buttonLabel }}
    </button>
  </p>
</div>
```

Za ikonicu smo izabrali bitmap ikonicu koja prikazuje paradajz. Možete da upotrebite bilo koju bitmapiranu sliku po vašem izboru ili možete da izostavite ikonicu za sada, čak i ako će vam biti potrebna u narednim poglavljima. Ovako izgleda Pomodoro tajmer aplikacija nakon implementiranja ovih vizuelnih podešavanja:

Preuzimanje primera koda

Možete da preuzmete fajlove sa primerima koda za ovu knjigu sa GitHub veb sajta na adresi https://github.com/deeleman/learning-angular2.
Možete da preuzmete fajlove sa primerima koda za ovu knjigu sa vašeg naloga (ukoliko ga imate) na adresi http://www.packtpub.com. Ako ste ovu knjigu kupili na drugom mestu, možete da posetite stranicu http://www.packtpub.com/support i registrujete se da biste e-mailom dobili fajlove.
Možete da preuzmete fajlove sa primerima koda praćenjem sledećih koraka:

- Prijavite se ili se registrujte na našem veb sajtu, koristeći svoju e-mail adresu i lozinku.
- Postavite kursor na karticu SUPPORT na vrhu stranice.
- Kliknite na Code Downloads & Errata.
- U polje Search unesite naslov knjige.
- Izaberite knjigu za koju želite da preuzmete fajlove sa primerima koda.
- Iz padajućeg menija izaberite mesto na kojem ste kupili knjigu.
- Kliknite na Code Download.

Takođe možete da preuzmete fajlove koda, tako što ćete kliknuti na dugme Code Files na veb stranici knjige na veb sajtu Packt Publishing. Ovoj stranici možete da pristupite tako što ćete u polje Search uneti naziv knjige. Ne zaboravite da morate biti prijavljeni na vaš Packt nalog.
Kada su fajlovi preuzeti, ekstrahujte direktorijum, koristeći najnoviju verziju:

- - WinRAR / 7-Zip za Windows
- - Zipeg / iZip / UnRarX za Mac
- - 7-Zip / PeaZip za Linux

REZIME

U ovom poglavlju videli smo veb komponente u skladu sa modernim veb standardima i prikazali smo način na koji Angular 2 komponente obezbeđuju jednostavan i jasan API za izgradnju sopstvenih komponenata. Opisali smo TypeScript i neke osnovne osobine njegove sintakse kao pripremu za Poglavlje 2, „*Uvod u TypeScript*“. Objasnili smo kako se podešava radno okruženje i gde da potražimo zavisnosti koje su potrebne da bismo uveli TypeScript u projekat i upotrebili Angular 2 biblioteke u projektu predstavljanjem uloga svake zavisnosti u aplikaciji.

Prva komponenta nam je dala mogućnost da opišemo formu klase kontrolera koja sadrži polja parametara, konstruktor i pomoćne funkcije. Opisali smo i zašto su komentari meta podataka toliko važni u kontekstu Angular 2 aplikacija za definisanje načina na koji će se komponenta integrisati u HTML okruženju. Sada znate kako da upotrebite alatke veb servera i poboljšate editore koda da biste olakšali sebi posao prilikom kreiranja Angular 2 aplikacija, iskorišćavajući introspekciju tipa i proveru tipa u toku rada. Naša prva veb komponenta ima sopstveni šablon, a takvi šabloni hostuju vezivanje parametara deklarativno u formi interpolacija promenljivih koje formatiraju usmeravanja. „Osluškivači“ događaja vezivanja su sada jednostavniji i njihova sintaksa odgovara standardima.

U sledećem poglavlju ćemo detaljno opisati sve funkcije TypeScripta koje treba da znate da biste ubrzali rad u Angularu 2.