Angular

Kuvar

Više od 80 praktičnih recepata koje bi svaki programer trebalo da zna

Muhammad Ahsan Ayaz

kompjuter biblioteka

Packt›

Angular

KUVAR

Više od 80 praktičnih recepata koje bi svaki programer trebalo da zna

Muhammad Ahsan Ayaz

kompjuter biblioteka

Packt›

Izdavač:

kompjuter biblioteka

Obalskih radnika 4a, Beograd

Tel: 011/2520272

e-mail: kombib@gmail.com

internet: www.kombib.rs

Urednik: Mihailo J. Šolajić

Za izdavača, direktor:
Mihailo J. Šolajić

Autor: Muhammad Ahsan Ayaz

Prevod: Biljana Tešić

Lektura: Miloš Jevtović

Slog: Zvonko Aleksić

Znak Kompjuter biblioteke:
Miloš Milosavljević

Štampa: „Pekograf“, Zemun

Tiraž: 500

Godina izdanja: 2021.

Broj knjige: 546

Izdanje: Prvo

ISBN: 978-86-7310-569-7

Angular Cookbook

Muhammad Ahsan Ayaz

ISBN 978-1-83898-943-9

Mojoj majci Zahidi Khatoon i mom ocu Muhammadu Ayazu za njihove molitve i požrtvovanost i za to što su mi poslužili kao primer odlučnosti tokom mog odgoja. Mojoj Nani (baki) Aarif-un-Nisa Begum za tone molitava za moj uspeh. I mojoj supruzi Najli Obaid, koja mi je pružila mnogo ljubavi tokom našeg zajedničkog životnog puta.

Muhammad Ahsan Ayaz

PREDGOVOR

Poznajem Ahsana i radim sa njim više od 10 godina. On je jedan od vodećih stručnjaka globalne Angular zajednice. Veoma je strastven u služenju zajednici programera i ima uticaja na svet, tako što obučava pojedince da pomognu da svet postane bolje mesto, zahvaljujući razvoju softvera. Održao je veliki broj govora i sesija širom sveta o JavaScriptu, Angularu i veb razvoju, a aktivno kreira video tutorijale koje postavlja na svoj YouTube kanal i piše članke na svojoj veb stranici. Ova knjiga je njegov najnoviji napor da ostavi dobar utisak i da obuči što više ljudi i da im pomogne da nauče Angular i da sazrevaju kao Angular programeri.

Ahsan ovom sveobuhvatnom knjigom vodi čitaoce na „putovanje“ razumevanja osnovnih koncepata Angulara i načina implementacije jediničnih i end-to-end testova u Angular aplikacijama. Recepti obuhvataju obrasce koji su vođeni šablonima i reaktivne obrasce, do istraživanja kako da kreirate svoje prilagođene kontrole obrazaca. Štaviše, nećete samo učiti o „stvarima“ kao što su Angular animacije, Angular ruter i upravljanje stanjem pomoću NgRx-a, već ćete i „zaroniti“ u neke neverovatne alatke i API-e pomoću Angular CDK-a. I, na kraju, ali ne najmanje važno, izradićete, koristeći Angular, nešto o čemu softverska industrija godinama „govori“ - progresivne veb aplikacije (PWA - Progressive Web App).

Ukratko, Ahsan je „pretočio“ godine učenja i iskustva u kreiranje ove knjige. Knjiga vam obezbeđuje doživljaj scenarija iz stvarnog života i njihovih tehničkih rešenja u obliku recepata. Sadržaj ove knjige je praktičan, precizan i dobro objašnjen.

Pošto poznajem Ahsana veoma dugo, mogu reći da on posvećuje svu svoju snagu i sposobnost da, kada odluči da uradi nešto, uradi to najbolje što može, a ova knjiga nije izuzetak. A zahvaljujući receptima, izvornom kodu i načinu na koji vas Ahsan vodi kroz sadržaj ove knjige, sigurno ćete mnogo naučiti i bićete bolji softverski inženjer i Angular programer.

Najla Obaid

poslovni analitičar u kompaniji „IOMechs“

O AUTORU

Muhammad Ahsan Ayaz je Angular programer i Syncron softverski arhitekta. Predaje programiranje širom sveta poslednjih osam godina pomoću članaka, video sadržaja, individualnog mentorstva i tehničkih govora na različitim globalnim konferencijama. Razvio je nekoliko biblioteka i plaginova koje koriste stotine hiljada programera, uključujući ingx-device-detector, koji ima više od pet miliona instalacija i više od 2.000 GitHub projekata koji ga koriste. On piše na blogu https://ahsanayaz.com i ima YouTube kanal na adresi https://ahsanayaz.com/youtube, na kojem redovno postavlja video tutorijale za razvoj veb aplikacija i mobilnih aplikacija. Osim što voli programiranje, Ahsan voli da putuje i da igra video igre sa više igrača. Živi sa suprugom u Švedskoj.

Zahvaljujem se ljudima koji su mi bliski i koji su me podržavali svih ovih godina, posebno mojim roditeljima Zahidi Khatoon i Muhammadu Ayazu i mojoj supruzi Najli. Takođe se zahvaljujem čitaocima ove knjige i drugih knjiga koje ću pisati u budućnosti i ljudima koji me prate na mojim profilima na društvenim mrežama i koji su pretplaćeni na moj YouTube kanal.

O RECENZENTU

Pawel Czekaj je diplomirao računarske nauke. Ima 12 godina iskustva kao frontend programer. Trenutno radi kao vodeći razvojni programer u kompaniji „Ziflow Ltd“. Specijalizovan je za radne okvire AngularJS, Angular, Amazon Web Services, Auth0, NestJS i druge. Trenutno izrađuje rešenja za proveru na nivou preduzeća, u potpunosti zasnovana na Angularu.

„PACKT“ TRAŽI AUTORE KAO ŠTO STE VI

Ako ste zainteresovani da postanete autor za „Packt“, prijavite se na stranicu authors.packtpub.com. Sarađujemo sa hiljadama programera i tehničkih profesionalaca da bismo im pomogli da podele svoje mišljenje sa globalnom tehničkom zajednicom. Možete da podnesete osnovnu prijavu, da se prijavite za specifičnu temu za koju tražimo autore ili da pošaljete neke svoje ideje.

Predgovor

Angular je jedan od najpopularnijih radnih okvira u svetu za izradu ne samo veb aplikacija, već i mobilnih i desktop aplikacija. Ovaj radni okvir koji obezbeđuje Google koriste milioni aplikacija. Iako je radni okvir dobro prilagođen za bilo koju skalu primene, preduzeća posebno vole Angular zbog njegove dogmatike i zbog doslednog eko-sistema koji uključuje sve alatke koje su vam potrebne za kreiranje aplikacija zasnovanih na veb tehnologijama.

Dok je učenje osnovnih tehnologija, poput JavaScripta, HTML-a i CSS-a, apsolutno neophodno za napredak veb programera, kada je reč o radnom okviru, učenje osnovnih koncepata samog radnog okvira je takođe važno. Kada je reč o Angularu, možemo učiniti neverovatne „stvari" pomoću veb aplikacija, upoznavajući i koristeći prave alatke u Angular eko-sistemu. Sada „u igru" ulazi ova knjiga.

Kome je namenjena ova knjiga

Ova knjiga je napisana za srednje i napredne Angular programere da bi usavršili svoje veštine razvoja Angulara pomoću recepata koje mogu lako pratiti, mogu eksperimentisati pomoću njih i mogu uvežbavati svoje varijacije. Nećete učiti samo iz recepata, već i iz stvarnih projekata iz realnog života povezanih sa receptima. Dakle, u ovim receptima i projektima postoji mnogo skrivenih „dragulja" za vas.

Šta obuhvata ova knjiga

U Poglavlju 1, „Pobednička komunikacija komponenata“, objašnjene su različite tehnike koje se koriste za implementaciju komunikacije između komponenata u Angularu. Obuhvaćeni su i dekoratori, usluge i hooks „životnog ciklusa“ `@Input()` i `@Output()`. Sadrži i recept za kreiranje dinamičke Angular komponente.

Poglavlje 2, „Razumevanje i korišćenje Angular direktiva“, sadrži uvod u Angular direktive i neke recepte koji koriste Angular direktive, uključujući direktive atributa i strukturalne direktive.

Poglavlje 3, „Magija injektovanja zavisnosti u Angularu“, sadrži recepte koji obuhvataju opcione zavisnosti, konfigurisanje tokena za injektovanje i upotrebu metapodataka `providedIn: 'root'` za Angular servise, provajdere vrednosti i provajdere klasa.

Poglavlje 4, „Razumevanje Angular animacija“, sadrži recepte za implementaciju animacija sa više stanja, neverovatnih animacija, animacija ključnih kadrova i animacija za prebacivanje između ruta u vašim Angular aplikacijama.

Poglavlje 5, „Angular i RxJS - kombinovane fenomenalnosti“, sadrži recepte za RxJS instance i statičke metode. Takođe sadrži neke recepte za upotrebu operatora `combateLatest`, `flatMap` i `switchMap` i neke savete i trikove za korišćenje RxJS strimova.

Poglavlje 6, „Upravljanje reaktivnim stanjem pomoću NgRxa“, sadrži recepte koji se odnose na čuvenu NgRx biblioteku i njene osnovne koncepte. Obuhvata osnovne koncepte, kao što su NgRx akcije, reduktori, selektori i efekti, i bavi se korišćenjem paketa, kao što su `@ngrx/storedevtools` i `@component/store`.

U Poglavlju 7, „Razumevanje Angular navigacije i usmeravanja“, istražujemo recepte za sporo učitane rute, čuvare ruta, strategije za učitavanje ruta unapred i neke zanimljive tehnike koje će se koristiti u Angular ruteru.

Poglavlje 8, „Savladavanje Angular obrazaca“, sadrži recepte za obrasce zasnovane na šablonima, za reaktivne obrasce, za obrasce za validaciju, za obrasce za testiranje i za kreiranje kontrole obrazaca.

Poglavlje 9, „Angular i Angular CDK“, ima mnogo odličnih Angular CDK recepata, uključujući recepte za virtuelno pomeranje, za navigaciju tastaturom, za overlay API, za clipboard API, za CDK drag and drop, za CDK stepper API i za CDK textfield API.

Poglavlje 10, „Pisanje jediničnih testova u Angularu pomoću Jesta“, sadrži recepte za jedinično testiranje pomoću Jesta, istraživanje globalnih simulatora u Jestu, simuliranje servisa/podređenih komponenata/kanala, korišćenje Angular CDK komponentnih testiranja i objekte Observable za jedinično testiranje.

Poglavlje 11, „E2E testiranja u Angularu pomoću Cypressa“, sadrži recepte za E2E testiranje pomoću Cypressa u Angular aplikacijama. Obuhvata obrasce za validaciju, čekanje na XHR pozive, simuliranje HTTP odgovora na pozive, korišćenje pratećih paketa pomoću Cypressa i korišćenje uređaja za testiranje u Cypressu.

Poglavlje 12, „Optimizacije performansi u Angularu“, sadrži neke odlične tehnike za poboljšanje performansi aplikacije Angular korišćenjem strategije otkrivanja promena OnPush, odloženo učitavanje ruta funkcija, odvajanje detektora promena od komponente, korišćenje veb workera u Angularu, korišćenje čistih kanala, dodavanje budžetiranja zasnovanog na performansama u Angular aplikaciju i korišćenje analizatora `webpack-bundle`.

Poglavlje 13, „Izrada PWA-ova pomoću Angulara“, sadrži recepte za kreiranje PWA-ova pomoću Angulara. Obuhvata navođenje boje teme za PWA, korišćenje tamnog režima uređaja, obezbeđivanje prilagođenog upita za instalaciju PWA, keširanje zahteva unapred pomoću Angularovog workera servisa i korišćenje App Shella.

Izvucite maksimum iz ove knjige

Recepti u ovoj knjizi izrađeni su pomoću radnog okvira Angular v12, a Angular prati semantičku verziju za svoja izdanja. Pošto se Angular stalno poboljšava radi stabilnosti, Angular tim je obezbedio predvidljiv ciklus izdanja za ažuriranja. Učestalost izdanja je sledeća:

- glavno izdanje na svakih šest meseci
- jedno do tri manja izdanja za svako glavno izdanje
- izdanje „zakrpa“ i predizdanje (sledeće ili rc) skoro svake nedelje

Izvor:https://angular.io/guide/releases#release-frequency

SOFTVER KOJI JE OBUHVAĆEN U KNJIZI	ZAHTEVI OPERATIVNOG SISTEMA
Angular	Windows, macOS ili Linux
TypeScript 4.2.4+	
ECMAScript 11	

Ako koristite digitalnu verziju ove knjige, savetujemo vam da sami otkucate kod ili pristupite kodu iz GitHub spremišta knjige (link je dostupan u sledećem odeljku). Na taj način ćete izbeći moguće greške koje se odnose na kopiranje i „lepljenje“ koda.

Kada završite čitanje knjige, obavezno pošaljite post na adresu https://ahsanayaz.com/twitter da biste mi predočili vaše povratne informacije o knjizi. Osim toga, kod koji ste dobili uz ovu knjigu možete da izmenite po svojoj želji, da ga postavite u svoje GitHub spremište i da ga podelite. Pobrinuću se da ga ponovo objavim :)

Preuzimanje datoteka primera koda

Možete da preuzmete datoteke sa primerima koda za ovu knjigu na adresi https://bit.ly/2XMANkH

Kod, ako postoji njegovo ažuriranje, biće ažuriran u GitHub spremištu: https://bit.ly/3Ejt4dg

Na raspolaganju su vam i drugi paketi kodova iz našeg bogatog kataloga knjiga i video zapisa dostupnih na adresi https://github.com/PacktPublishing/.

Preuzimanje kolornih slika

Takođe smo obezbedili PDF datoteku koja sadrži kolorne snimke ekrana i dijagrama koji su upotrebljeni u knjizi. Možete da preuzmete ovu datoteku sa adrese: https://bit.ly/3bdgfoA

Upotrebljene konvencije

U ovoj knjizi upotrebljen je veliki broj konvencija za tekst.

`CodeInText` ukazuje na reči koda u tekstu, u nazivima tabele baze podataka, u nazivima direktorijuma, u nazivima fajlova, u ekstenzijama fajlova, putanjama, lažnim URL-ovima, korisničkim unosima i Twiter postovima. Primer je sledeći: „Sada ćemo premestiti kod iz datoteke `the-amazing-listcomponent.html` u datoteku `the-amazing-list-item.component.html` radi označavanja stavke."

Blok koda je postavljen na sledeći način:

```
 openMenu($event, itemTrigger) {
     if ($event) {
       $event.stopImmediatePropagation();
     }
     this.popoverMenuTrigger = itemTrigger;
     this.menuShown = true;
   }
```

Kada želimo da privučemo vašu pažnju na određeni deo bloka koda, relevantne linije ili stavke će biti ispisane zadebljanim slovima:

```
.menu-popover {
  ...
  &::before {...}

  &--up {
    transform: translateY(-20px);
    &::before {
      top: unset !important;
      transform: rotate(180deg);
      bottom: -10px;
    }
  }

  &__list {...}
}
```

Zadebljana slova Novi termini, važne reči ili reči koje vidite na ekranu - na primer, u menijima ili okvirima za dijalog, biće prikazani u tekstu. **zadebljanim slovima**. Na primer: „Primetićete da ne možete videti ceo sadržaj unosa - ovo je pomalo dosadno i u najboljim okolnostima, jer sadržaj ne možete zaista pregledati pre nego što pritisnete dugme **Action**.“

Saveti ili trikovi se prikazuju ovako.

Kontaktirajte sa nama

Povratne informacije naših čitalaca su uvek dobrodošle.

Opšte povratne informacije - Ako imate pitanja o bilo kojem aspektu ove knjige, pošaljite nam e-poruku na adresu kombib@gmail.com.

Štamparske greške - Iako smo preduzeli sve mere da bismo obezbedili tačnost sadržaja, greške su moguće. Ako pronađete grešku u nekoj od naših knjiga – u tekstu ili u kodu, bili bismo zahvalni ako biste nam to javili. Na taj način možete da pomognete drugim čitaocima da izbegnu frustracije, a nama da poboljšamo sledeće verzije ove knjige. Ako pronađete grešku, molimo vas da nas o tome obavestite na email kombib@gmail.com.

Piraterija - Ako pronađete ilegalnu kopiju naših knjiga u bilo kojoj formi na Internetu, molimo vas da nas o tome obavestite i da nam pošaljete adresu lokacije ili naziv veb sajta. Pošaljite nam poruku na adresu kombib@gmail.com i pošaljite nam link ka sumnjivom materijalu.

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

Podelite svoje mišljenje

Voleli bismo da saznamo vaše mišljenje nakon što pročitate „Angular kuvar“! Kliknite ovde da biste direktno pristupili stranici Amazon pregleda https://packt.link/r/1838989439 za ovu knjigu i napišite vaše povratne informacije.

Vaša recenzija je važna za nas i tehničku zajednicu i pomoći će nam da se uverimo da isporučujemo sadržaj odličnog kvaliteta.

Postanite član Kompjuter biblioteke

Kupovinom jedne naše knjige stekli ste pravo da postanete član Kompjuter biblioteke. Kao član možete da kupujete knjige u pretplati sa 40% popustai učestvujete u akcijama kada ostvarujete popuste na sva naša izdanja. Potrebno je samo da se prijavite preko formulara na našem sajtu. Link za prijavu: http://bit.ly/2TxekSa

Skenirajte QR kod
registrujte knjigu
i osvojite nagradu

1

Pobednička komunikacija komponenata

U ovom poglavlju upoznaćete komunikaciju komponenata u Angularu. Naučićete različite tehnike za uspostavljanje komunikacije između komponenata i koja je tehnika pogodna u određenoj situaciji. Takođe ćete naučiti kako da kreirate dinamičku Angular komponentu.

Slede recepti koje ćemo obuhvatiti u ovom poglavlju:

- komunikacija komponenata pomoću komponenata `@Input(s)` i `@Output(s)`
- komunikacija komponenata pomoću servisa
- korišćenje settera za presretanje izmena svojstava ulaza
- korišćenje `ngOnChangesa` za presretanje izmena svojstava unosa
- pristup podređenoj komponenti u nadređenom šablonu pomoću promenljivih šablona
- pristup podređenoj komponenti u klasi nadređenih komponenata pomoću `ViewChilda`
- kreiranje vaše prve dinamičke komponente u Angularu

Tehnički zahtevi

Zbog recepata u ovom poglavlju proverite da li su na vašem računaru instalirani Git i Node.js. Takođe morate imati instaliran paket `@angular/cli`, koji možete da instalirate pomoću komande `install -g @angular/cli` iz vašeg terminala. Kod za ovo poglavlje možete pronaći na adresi https://github.com/PacktPublishing/Angular-Cookbook/tree/master/chapter01.

Komunikacija komponenata pomoću komponenata @Input i @Output

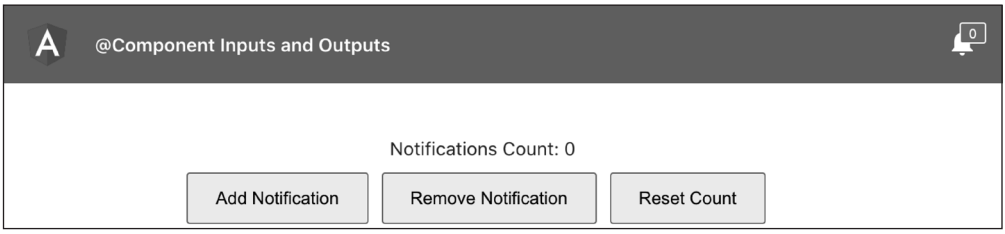
Prvo ćete koristiti aplikaciju sa nadređenom komponentom i dve podređene komponente. Zatim ćete koristiti Angular dekoratore `@Input` i `@Ouput` da biste uspostavili komunikaciju između njih, koristeći atribute i `EventEmittere`.

Priprema

Projekat koji ćemo koristiti nalazi se u `chapter01/start_here/cc-inputs-outputsu` unutar kloniranog spremišta:

1. Otvorite projekat u Visual Studio Codeu.
2. Otvorite terminal i pokrenite `npminstall` da biste instalirali zavisnosti projekta. Nakon toga, pokrenite `ng serve -o`.

   Sada bi trebalo da se otvori aplikacija u novoj kartici pregledača i trebalo bi da vidite sledeće:



Slika 1.1 Aplikacija cc-inputs-outputs koja je pokrenuta na adresi http://localhost:4200

Kako to uraditi...

Sada imamo aplikaciju koja sadrži komponente `AppComponent`, `Notifications-ButtonComponent` i `NotificationsManagerComponent`. Iako je `AppComponent` nadređena komponenta druge dve pomenute komponente, ne postoji apsolutno nikakva komunikacija između njih za sinhronizaciju vrednosti broja obaveštenja. Uspostavićete odgovarajuću komunikaciju između njih pomoću sledećih koraka:

1. Premestite promenljivu `notificationsCount` iz komponente `NotificationsManagerComponent` i smestite je u komponentu `AppComponent`. Da biste to uradili, samo kreirajte svojstvo `notificationsCount` u datoteci `app.component.ts`:

```
export class AppComponent {
  notificationsCount = 0;
}
```

2. Zatim, konvertujte svojstvo notificationsCount u datoteci notifications-manager.component.ts u komponentu @Input(). Tu komponentu preimenujte u count i zamenite njenu upotrebu na sledeći način:

```
import { Component, OnInit, Input } from '@angular/core';
@Component({
  selector: 'app-notifications-manager',
  templateUrl: './notifications-manager.component.html',
  styleUrls: ['./notifications-manager.component.scss']
})
export class NotificationsManagerComponent implements
OnInit {
  @Input() count = 0
  constructor() { }
  ngOnInit(): void {
  }
  addNotification() {
    this.count++;
  }
  removeNotification() {
    if (this.count == 0) {
      return;
    }
    this.count--;
```

```
  }
  resetCount() {
    this.count = 0;
  }
}
```

3. Ažurirajte datoteku `notifications-manager.component.html` da biste koristili `count`, umesto `notificationsCount`:

```
<div class="notif-manager">
  <div class="notif-manager__count">
    Notifications Count: {{count}}
  </div>
  ...
</div>
```

4. Zatim, prosledite svojstvo `notificationsCount` iz datoteke `app.component.html` elementu `<pp-notifications-manager>` kao unos:

```
<div class="content" role="main">
  <app-notifications-manager
    [count]="notificationsCount">
  </app-notifications-manager>
</div>
```

Sada možete testirati da li se vrednost ispravno prosleđuje iz datoteke `app.component.html` u `app-notifications-manager` dodeljivanjem vrednosti `notificationsCountu` u datoteci `app.component.ts` kao `10`. Videćete da će u komponenti `NotificationsManagerComponent` početna prikazana vrednost biti `10`:

```
export class AppComponent {
  notificationsCount = 10;
}
```

5. Sada kreirajte `@Input()` u datoteci `notifications-button.component.ts` pod nazivom `count`:

```
import { Component, OnInit, Input } from '@angular/core';
...
export class NotificationsButtonComponent implements
OnInit {
```

```
  @Input() count = 0;
  ...
}
```

6. Prosledite `notificationsCount` u `<pp-notifications-button>` iz datoteke `app.component.html`:

```
<!-- Toolbar -->
<div class="toolbar" role="banner">
  ...
  <span>@Component Inputs and Outputs</span>
  <div class="spacer"></div>
  <div class="notif-bell">
    <app-notifications-button
    [count]="notificationsCount">
    </app-notifications-button>
  </div>
</div>
...
```

7. Koristite unos `count` u datoteci `notifications-button.component.html`, zajedno sa ikonom zvona za obaveštenje:

```
<div class="bell">
  <i class="material-icons">notifications</i>
  <div class="bell__count">
    <div class="bell__count__digits">
      {{count}}
    </div>
  </div>
</div>
```

Sada bi trebalo da vidite i vrednost `10` za broj ikona zvona za obaveštenje.

Ako promenite broj dodavanjem/uklanjanjem obaveštenja iz komponente `NotificationsManagerComponent`*, broj na ikoni zvona za obaveštenja se neće promeniti.*

8. Da bismo preneli promenu iz `NotificationsManagerComponenta` na `NotificationsButtonComponent`, sada ćemo koristiti Angular komponente `@Output`. Koristite `@Ouput` i `@EventEmitter` iz `'@angular/corea'` unutar datoteke `notifications-manager.component.ts`:

```
import { Component, OnInit, Input, Output, EventEmitter }
from '@angular/core';
...
export class NotificationsManagerComponent implements
OnInit {
  @Input() count = 0
  @Output() countChanged = new EventEmitter<number>();
  ...
  addNotification() {
    this.count++;
    this.countChanged.emit(this.count);
  }
  removeNotification() {
    ...
    this.count--;
    this.countChanged.emit(this.count);
  }
  resetCount() {
    this.count = 0;
    this.countChanged.emit(this.count);
  }
}
```

9. Zatim ćemo u datoteci `app.component.html` preslušati prethodno emitovani događaj iz komponente `NotificationsManagerComponent` i u skladu s njim ažurirati svojstvo `notificationsCount`:

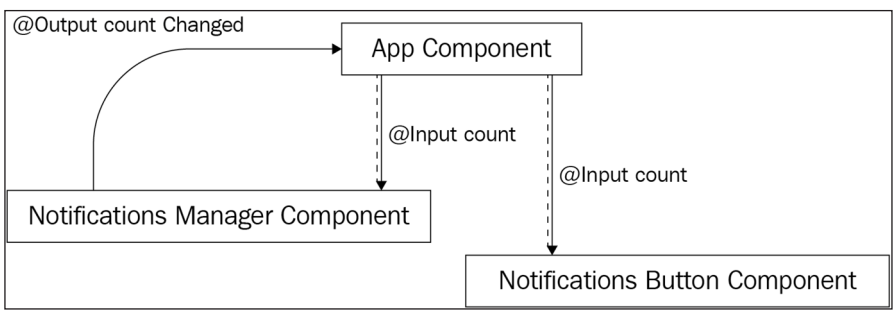
```
<div class="content" role="main">
  <app-notifications-manager
  (countChanged)="updateNotificationsCount($event)"
  [count]="notificationsCount"></app-notifications-
  manager>
</div>
```

10. Pošto smo prethodno preslušali događaj `countChanged` i pozvali metod `updateNotificationsCount`, moramo da kreiramo ovaj metod u datoteci `app.component.ts` i u skladu s njim da ažuriramo vrednost svojstva `notificationsCount`:

```
export class AppComponent {
  notificationsCount = 10;
  updateNotificationsCount(count: number) {
    this.notificationsCount = count;
  }
}
```

Kako to funkcioniše...

Da bi komponente komunicirale pomoću komponenata `@Input` i `@Output`, tok podataka će se uvek kretati **od** *podređenih komponenata* **do** *nadređene komponente*, pri čemu se može obezbediti nova (ažurirana) vrednost kao *ulaz* traženim podređenim komponentama. Dakle, `NotificationsManagerComponent` emituje događaj `countChanged`. `AppComponent` (kao nadređena komponenta) osluškuje događaj i ažurira vrednost `notificationsCount`, koja automatski ažurira svojstvo `count` u komponenti `NotificationsButtonComponent`, jer se `notificationsCount` prosleđuje kao broj `@Input()` komponenti `NotificationsButtonComponent`. Na sledećoj slici prikazan je ceo proces.



Slika 1.2 Kako komunikacija komponenata funkcioniše pomoću ulaza i izlaza

Takođe pogledajte

- Kako komuniciraju Angular komponente? https://www.thirdrocktechkno.com/blog/how-angular-components-communicate
- „ComponentCommunication in Angular“, Dhananjay Kumar: https://www.youtube.com/watch?v=I8Z8g9APaDY

Komunikacija komponenata pomoću servisa

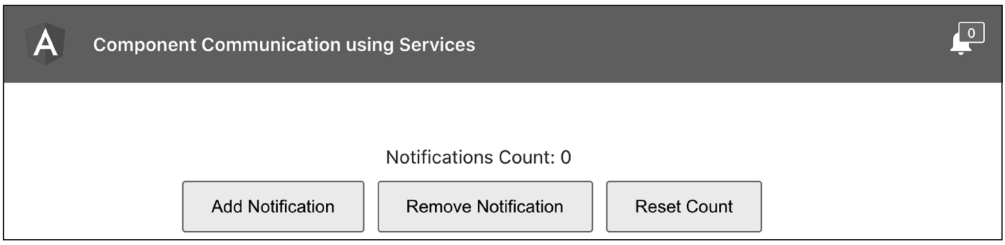
U ovom receptu ćete prvo koristiti aplikaciju sa nadređenom i podređenom komponentom. Zatim ćete koristiti Angular servis za uspostavljanje komunikacije između njih. Koristićemo strimove `BehaviorSubject` i `Observable` za komunikaciju između komponenata i servisa.

Priprema

Projekat za ovaj recept nalazi se u `chapter01/start_here/cc-servicesu`:

1. Otvorite projekat u Visual Studio Codeu.
2. Otvorite terminal i pokrenite `npminstall` da biste instalirali zavisnosti projekta.
3. Nakon toga, pokrenite `ng serve -o`.

Sada bi trebalo da se otvori aplikacija u novoj kartici pregledača i aplikaciju bi trebalo da vidite kao na sledećoj slici.



Slika 1.3 Aplikacija cc-services koja je pokrenuta na adresi http://localhost:4200

Kako to uraditi...

Slično prethodnom receptu, imamo aplikaciju sa komponentama `AppComponent`, `NotificationsButtonComponent` i `NotificationsManagerComponent`.

`AppComponent` je nadređena komponenta drugim dvema komponentama koje smo prethodno pomenuli i moramo da uspostavimo odgovarajuću komunikaciju između njih pomoću sledećih koraka:

1. Kreirajte novu fasciklu u projektu `chapter10/start_here/cc-services/src/app`, koji je nazvan `services`. Ovde ćemo smestiti naš novi servis.
2. Iz terminala uđite u projekat, tj. u `chapter01/start_here/cc-services`, i kreirajte novi servis, koji se zove `NotificationService`, na sledeći način:

```
ng g service services/Notifications
```

3. Kreirajte strim `BehaviorSubject` sa nazivom `count` u datoteci `notifications.service.ts` i inicijalizujte ga pomoću vrednosti `0`, jer `BehaviorSubject` zahteva početnu vrednost:

```
import { Injectable } from '@angular/core';
import { BehaviorSubject } from 'rxjs';
@Injectable({
  providedIn: 'root'
})
export class NotificationsService {
  private count: BehaviorSubject<number> = new
  BehaviorSubject<number>(0);
  constructor() { }
}
```

Imajte na umu da je strim `BehaviorSubject` svojstvo `private` i da ćemo ga kasnije ažurirati samo unutar servisa, koristeći metod `public`.

4. Sada kreirajte `Observable` pod nazivom `count$`, koristeći metod `.asObservable()` u `countBehaviorSubjectu`:

```
import { Injectable } from '@angular/core';
import { BehaviorSubject, Observable } from 'rxjs';
...
export class NotificationsService {
  private count: BehaviorSubject<number> = new
  BehaviorSubject<number>(0);
  count$: Observable<number> = this.count.asObservable();
```

```
  ...
}
```

5. Konvertujte svojstvo `notificationsCount` u datoteci `notifications-manager.component.ts` u strim `Observable`, koji se zove `notificationsCount$`. Injektujte `NotificationsService` u komponentu i dodelite broj (`count$`) servisa za Observable promenljivoj komponente `notificationsCount$`:

```
import { Component, OnInit } from '@angular/core';
import { Observable } from 'rxjs';
import { NotificationsService } from '../services/
notifications.service';
...
export class NotificationsManagerComponent implements
OnInit {
  notificationsCount$: Observable<number>;
  constructor(private notificationsService:
  NotificationsService) { }

  ngOnInit(): void {
    this.notificationsCount$ = this.notificationsService.
    count$;
  }
  ...
}
```

6. Komentarišite kod koji za sada ažurira broj obaveštenja (na to ćemo se vratiti kasnije):

```
...
export class NotificationsManagerComponent implements
OnInit {
  ...
  addNotification() {
    // this.notificationsCount++;
  }
  removeNotification() {
    // if (this.notificationsCount == 0) {
    //   return;
```

```
    // }
    // this.notificationsCount--;
  }
  resetCount() {
    // this.notificationsCount = 0;
  }
}
```

7. Koristite Observable `notificationsCount$` u datoteci `notifications-manager.component.html` sa kanalom `async` da biste prikazali njegovu vrednost:

```
<div class="notif-manager">
  <div class="notif-manager__count">
    Notifications Count: {{notificationsCount$ | async}}
  </div>
  ...
</div>
```

8. Sada na sličan način injektujte `NotificationsService` u datoteku `notifications-button.component.ts`, unutar komponente `NotificationsButtonComponent` kreirajte strim `ObservablenotificationsCount$` i dodelite mu Observable broj servisa (`count$`):

```
import { Component, OnInit } from '@angular/core';
import { NotificationsService } from '../services/
notifications.service';
import { Observable } from 'rxjs';
 ...
export class NotificationsButtonComponent implements
OnInit {
  notificationsCount$: Observable<number>;
  constructor(private notificationsService:
  NotificationsService) { }

  ngOnInit(): void {
    this.notificationsCount$ = this.notificationsService.
    count$;
  }
}
```

9. Koristite Observable `notificationsCount$` u datoteci `notifications-button.component.html` sa kanalom `async`:

```
<div class="bell">
  <i class="material-icons">notifications</i>
  <div class="bell__count">
    <div class="bell__count__digits">
      {{notificationsCount$ | async}}
    </div>
  </div>
</div>
```

Ako sada „osvežite“ aplikaciju, trebalo bi da vidite vrednost `0` i za komponentu menadžera obaveštenja i za komponentu dugmeta za obaveštenja.

10. Promenite početnu vrednost za `countBehaviorSubject` u `10` i pogledajte da li se ta promena odražava na obe komponente:

```
...
export class NotificationsService {
  private count: BehaviorSubject<number> = new
  BehaviorSubject<number>(10);
  ...
}
```

11. Kreirajte metod `setCount` u datoteci `notifications.service.ts` da biste mogli da ažurirate vrednost `countBehaviorSubject`:

```
...
export class NotificationsService {
  …
  constructor() {}
  setCount(countVal) {
    this.count.next(countVal);
  }
}
```

12. Sada, kada imamo postavljen metod setCount, upotrebićemo ga u datoteci notifications-manager.component.ts da bismo ažurirali njegovu vrednost na osnovu klikova na dugme. Da bismo to učinili, moramo da pribavimo najnoviju vrednost za Observable `notificationsCount$`, a zatim da izvršimo neku radnju. Prvo ćemo da kreiramo metod `getCountValue` unutar komponente `NotificationsManagerComponent` na način koji je prikazan u nastavku i koristićemo `subscribe` zajedno sa operatorom `new` u Observableu `notificationsCount$` da bismo dobili najnoviju vrednost:

```
...
import { first } from 'rxjs/operators';
...
export class NotificationsManagerComponent implements
OnInit {
  ngOnInit(): void {
    this.notificationsCount$ = this.notificationsService.
    count$;
  }
  ...
  getCountValue(callback) {
    this.notificationsCount$
      .pipe(
        first()
      ).subscribe(callback)
  }
  ...
}
```

13. Sada ćemo koristiti metod `getCountValue` u okviru metoda `addNotification`, `removeNotification` i `resetCount`. Moraćemo da prosledimo funkciju povratnog poziva iz ovih metoda u metod `getCountValue`. Prvo ćemo upotrebiti metod `addNotification`:

```
import { Component, OnInit } from '@angular/core';
import { Observable } from 'rxjs';
import { NotificationsService } from '../services/
notifications.service';
import { first } from 'rxjs/operators';

...
```

```
export class NotificationsManagerComponent implements
OnInit {
  ...
  addNotification() {
    this.getCountValue((countVal) => {
      this.notificationsService.setCount(++countVal)
    });
  }
  ...
}
```

Zahvaljujući prethodnom kodu, već bi trebalo da vidite obe komponente koje pravilno odražavaju ažurirane vrednosti uvek kada kliknete na dugme **AddNotification**.

14. Sada ćemo implementirati istu logiku za `removeNotification` i `resetCount`:

```
...
export class NotificationsManagerComponent implements
OnInit {
  ...
  removeNotification() {
    this.getCountValue((countVal) => {
      if (countVal === 0) {
        return;
      }
      this.notificationsService.setCount(--countVal);
    })
  }
  resetCount() {
    this.notificationsService.setCount(0);
  }
}
```

Kako to funkcioniše...

BehaviorSubject je poseban tip Observablea koji zahteva početnu vrednost koju mogu koristiti mnogi pretplatnici. U ovom receptu kreiramo `BehaviorSubject`, a zatim `Observable`, koristeći metod `.asObservable()` u `BehaviorSubjectu`. Iako smo mogli koristiti `BehaviorSubject`, zajednica koristi pristup `.asObservable()`.

Nakon što smo kreirali Observable `count$` u `NotificationsServiceu`, injektujemo `NotificationsService` u naše komponente i dodeljujemo Observable `count$` lokalnom svojstvu komponenata. Zatim se pretplaćujemo na ovo lokalno svojstvo (koje je Observable) direktno u šablonu `NotificationsButtonComponent(html)` i u šablonu `NotificationsManagerComponent`, koristeći kanal `async`.

Kad god treba da ažuriramo vrednost za Observable `count$`, koristimo metod `setCountNotificationsService` da bismo ažurirali stvarnu vrednost za `BehaviorSubject`, koristeći metod `.next()`. Ovo automatski emituje ovu novu vrednost pomoću Observablea `count$` i ažurira prikaz novom vrednošću u obe komponente.

Takođe pogledajte

- Subjects iz zvanične dokumentacije za RxJS: https://www.learnrxjs.io/learn-rxjs/subjects
- Poređenje `BehaviorSubjecta` i `Observablea` na sajtu StackOverflow:https://stackoverflow.com/a/40231605

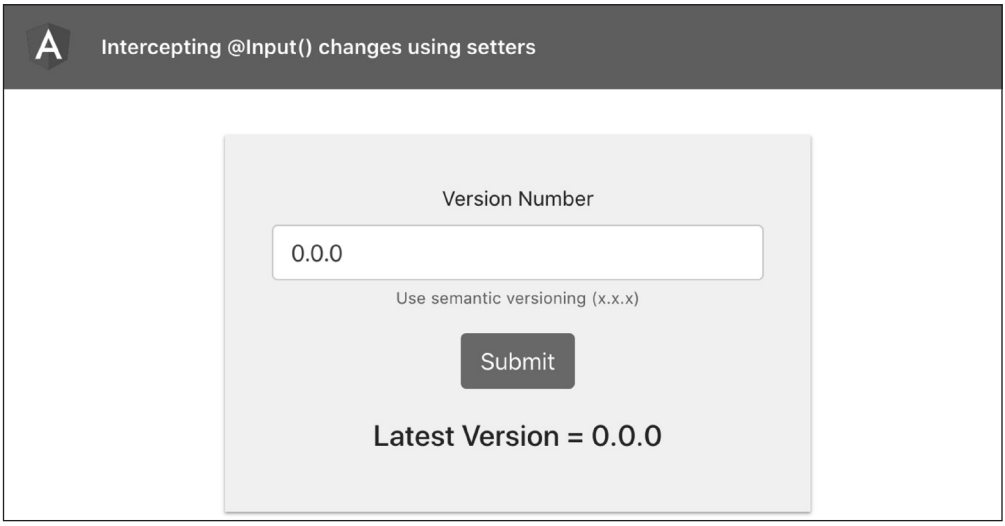
Korišćenje settera za presretanje izmena svojstava ulaza

U ovom receptu ćete naučiti kako da presretnete promene u komponenti `@Input` koja je prosleđena iz nadređene komponente i da izvršite neke radnje u ovom događaju. Presrešćemo ulaz `vName` koji je prosleđen iz nadređene komponente `VersionControlComponent` u podređenu komponentu `VcLogsComponent`. Koristićemo settere za generisanje evidencije kad god se vrednost `vName` promeni i prikazaćemo te evidencije u podređenoj komponenti.

Priprema

Projekat za ovaj recept nalazi se u `chapter01.start_here/cc-settersu`:

1. Otvorite projekat u Visual Studio Codeu.
2. Otvorite terminal i pokrenite `npminstall` da biste instalirali zavisnosti projekta.
3. Nakon toga, pokrenite `ng serve -o`. Ovo bi trebalo da otvori aplikaciju u novoj kartici pregledača i aplikaciju bi trebalo da vidite na sledeći način:



Slika 1.4 Aplikacija cc-setters koja je pokrenuta na adresi http://localhost:4200

Kako to uraditi...

1. Prvo ćemo niz evidencija u komponenti `VcLogsComponent` za skladištenje svih evidencija koje ćemo kasnije prikazati pomoću našeg šablona kreirati na sledeći način:

```
export class VcLogsComponent implements OnInit {
  @Input() vName;
  logs: string[] = [];
  constructor() { }
...
}
```

2. Sada ćemo kreirati HTML u kojem ćemo prikazivati evidencije. Dodaćemo kontejner evidencija i stavke evidencije pomoću sledećeg koda u datoteku `vc-logs.component.html`:

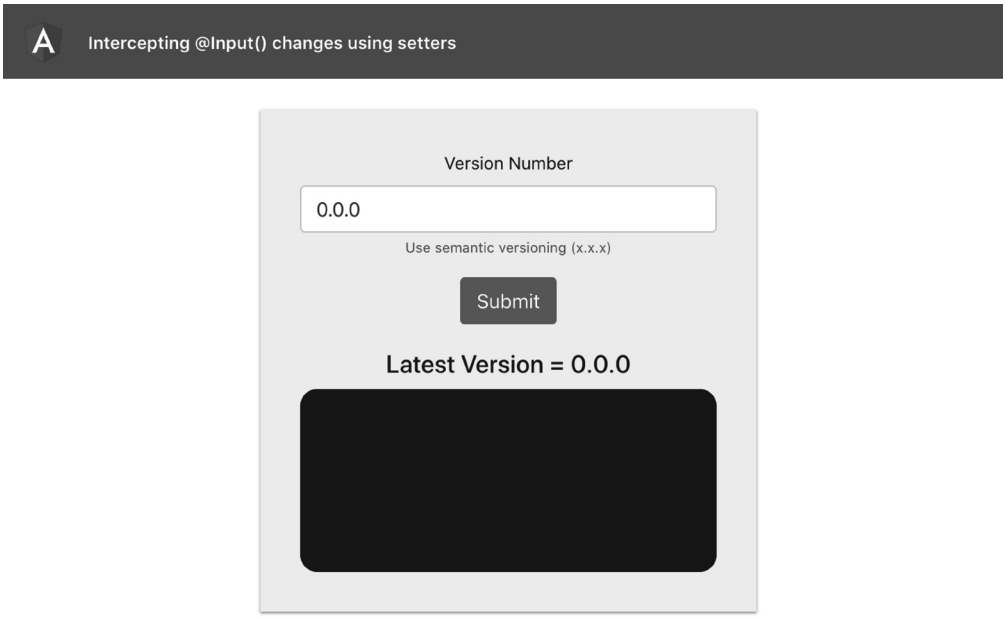
```
<h5>Latest Version = {{vName}}</h5>
<div class="logs">
  <div class="logs__item" *ngFor="let log of logs">
    {{log}}
  </div>
</div>
```

3. Zatim ćemo dodati malo stilizovanja za prikaz kontejnera evidencija i stavki evidencije. Nakon izmena, prikaz bi trebalo da izgleda kao na *slici 1.5*. Ažurirajte datoteku `vc-logs.component.scss` na sledeći način:

```
h5 {
  text-align: center;
}

.logs {
  padding: 1.8rem;
  background-color: #333;
  min-height: 200px;
  border-radius: 14px;
  &__item {
    color: lightgreen;
  }
}
```

Na sledećem snimku ekrana prikazana je aplikacija sa stilovima kontejnera evidencija.



Slika 1.5 Aplikacija cc-setters sa stilovima kontejnera evidencija

4. Sada ćemo konvertovati `@Input()` u `vc-logs.component.ts` da bismo koristili getter i setter, tako da možemo presresti izmene ulaza. Za to ćemo takođe kreirati interno svojstvo koje se zove `_vName`. Kod bi trebalo da izgleda ovako:

```
...
export class VcLogsComponent implements OnInit {
  _vName: string;
  @Input()
  get vName() {
    return this._vName;
  };
  set vName(name: string) {
   this._vName = name;
  }
  logs: string[] = [];
  constructor() { }
```

```
...
}
```

5. Zahvaljujući izmenama u *koraku 4*, aplikacija funkcioniše potpuno isto kao i ranije, tj. savršeno. Sada ćemo izmeniti setter da bismo kreirali te evidencije. Za početnu vrednost imaćemo evidenciju u kojoj piše `'initialversion is x.x.x'`:

```
export class VcLogsComponent implements OnInit {
  ...
  set vName(name: string) {
    if (!name) return;
    if (!this._vName) {
      this.logs.push('initial version is ${name.trim()}')
    }
    this._vName = name;
  }
...
}
```

6. Kao poslednji korak, uvek kada promenimo naziv verzije, moramo prikazati drugu poruku, koja glasi `'versionchanged to x.x.x'`. Na *slici 1.6* prikazan je konačni rezultat. Za potrebne izmene napisaćemo još malo koda u `vName` setteru na sledeći način:

```
export class VcLogsComponent implements OnInit {
  ...
  set vName(name: string) {
    if (!name) return;
    if (!this._vName) {
      this.logs.push('initial version is ${name.trim()}')
    } else {
      this.logs.push('version changed to ${name.trim()}')
    }
    this._vName = name;
  }
```

Na sledećem snimku ekrana prikazan je konačni rezultat.

Slika 1.6 Konačni rezultat dobijen pomoću settera

Kako to funkcioniše...

Getteri i setteri su komponente ugrađene funkcije JavaScripta. Mnogi programeri su ih koristili u svojim projektima dok su koristili osnovni JavaScript, ili čak i TypeScript. Srećom, Angularov `@Input()` takođe može da koristi gettere i settere, jer su oni, u osnovi, svojstvo obezbeđene klase.

Za ovaj recept koristimo getter i, preciznije, setter za naš ulaz, pa kad god se unos promeni, koristimo setter metod za obavljanje dodatnih zadataka. Štaviše, u našem HTML-u koristimo setter istog ulaza, pa prilikom ažuriranja direktno navodimo vrednost u prikazu (view).

Uvek je dobra ideja koristiti privatnu promenljivu/svojstvo u setterima i getterima da biste znali šta komponenta prima kao ulaz i šta posebno skladišti.

Takođe pogledajte

- https://angular.io/guide/component-interaction#interceptinput-property-changes-with-a-setter
- https://www.jackfranklin.co.uk/blog/es5-getters-setters, JackFranklin

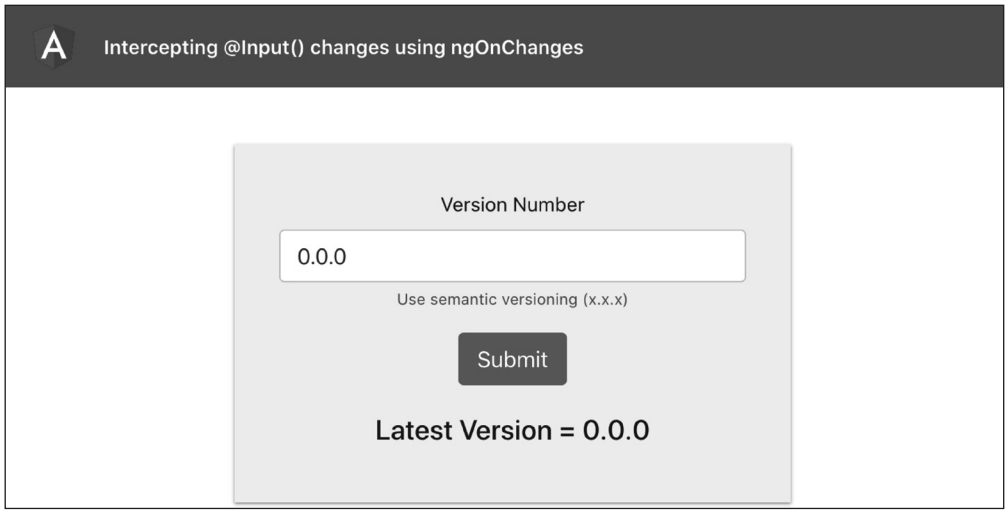
Korišćenje ngOnChangesa za presretanje izmena svojstava unosa

U ovom receptu ćete naučiti kako da koristite `ngOnChanges` za presretanje izmena pomoću API-a `SimpleChanges`. Osluškivaćemo `vName` ulaz koji je prosleđen iz nadređene komponente `VersionControlComponent` u podređenu komponentu `VcLogsComponent`.

Priprema

Projekat za ovaj recept nalazi se u `chapter01/start_here/cc-ng-on-changesu`:

1. Otvorite projekat u Visual Studio Codeu.
2. Otvorite terminal i pokrenite `npminstall` da biste instalirali zavisnosti projekta.
3. Nakon toga, pokrenite `ng serve -o`. Ovo bi trebalo da otvori aplikaciju u novoj kartici pregledača; aplikacija bi trebalo da izgleda kao na sledećoj slici.



Slika 1.7 Aplikacija cc-ng-on-changes je pokrenuta na adresi http://localhost:4200.

Kako to uraditi...

1. Da bismo uskladištili sve evidencije koje ćemo kasnije prikazati pomoću našeg šablona, prvo ćemo niz evidencija u komponenti `VcLogsComponent` kreirati na sledeći način:

```
export class VcLogsComponent implements OnInit {
  @Input() vName;
  logs: string[] = [];
  constructor() { }
...
}
```

2. Sada ćemo kreirati HTML u kojem ćemo prikazivati evidencije. Dodaćemo kontejner evidencija i stavke evidencije u datoteku `vc-logs.component.html` pomoću sledećeg koda:

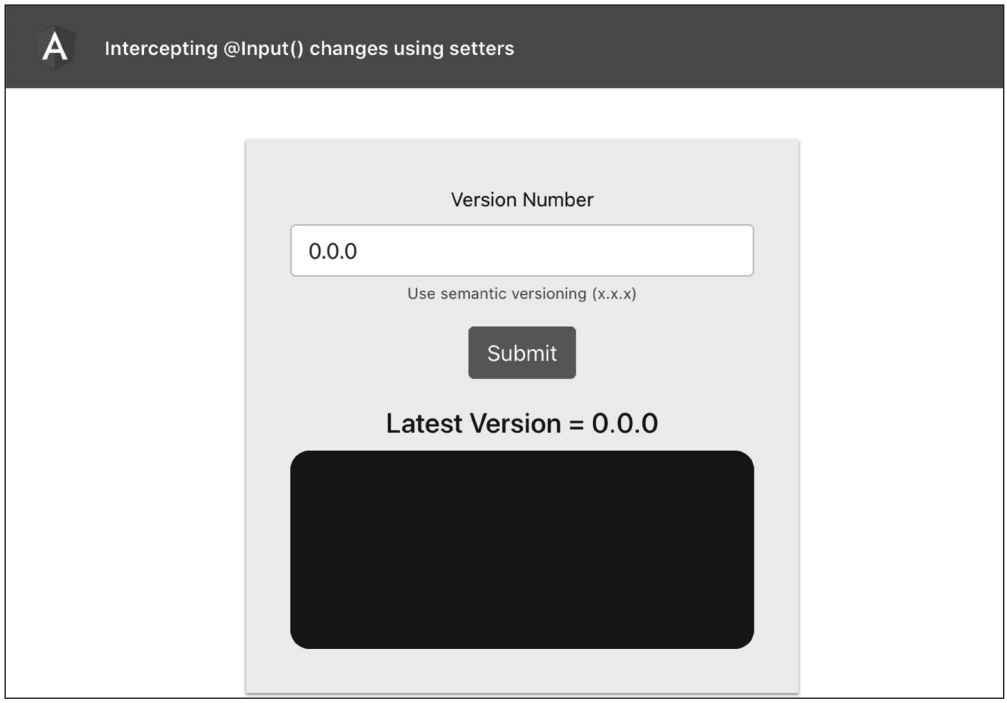
```
<h5>Latest Version = {{vName}}</h5>
<div class="logs">
  <div class="logs__item" *ngFor="let log of logs">
    {{log}}
  </div>
</div>
```

3. Zatim ćemo dodati malo stilizovanja za prikaz kontejnera evidencija i stavki evidencije u datoteku `vc-logs.component.scss` na sledeći način:

```
h5 {
  text-align: center;
}

.logs {
  padding: 1.8rem;
  background-color: #333;
  min-height: 200px;
  border-radius: 14px;
  &__item {
    color: lightgreen;
  }
}
```

Trebalo bi da vidite nešto ovako:



Slika 1.8 Aplikacija cc-ng-on-changes sa stilovima kontejnera evidencija

4. Sada ćemo u datoteku `vc-logs.component.ts` da implementiramo `ngOnChanges`, koristeći jednostavne izmene u komponenti `VcLogsComponent` na sledeći način:

```
import { Component, OnInit, Input, OnChanges,
SimpleChanges } from '@angular/core';
...
export class VcLogsComponent implements OnInit, OnChanges
{
  @Input() vName;
  logs: string[] = [];
  constructor() {}

  ngOnInit(): void {}
  ngOnChanges(changes: SimpleChanges) {
  }
}
```

5. Sada možemo dodati evidenciju za početnu vrednost ulaza `vName` u kojoj piše `'initialversion is x.x.x'`. To radimo tako što proveravamo da li je to početna vrednost pomoću metoda `.isFirstChange()` na sledeći način:

```
...
export class VcLogsComponent implements OnInit, OnChanges
{
  ...
  ngOnChanges(changes: SimpleChanges) {
    const currValue = changes.vName.currentValue;
    if (changes.vName.isFirstChange()) {
      this.logs.push('initial version is
      ${currValue.trim()}')
    }
  }
}
```

6. Sada ćemo da rešimo ovaj slučaj, tako što ćemo da ažuriramo verziju nakon dodeljivanja početne vrednosti. Za to ćemo dodati još jednu evidenciju u kojoj piše `'versionchanged to x.x.x'`, koristeći uslov `else` na sledeći način:

```
...
export class VcLogsComponent implements OnInit, OnChanges
{
  ...
  ngOnChanges(changes: SimpleChanges) {
    const currValue = changes.vName.currentValue;
    if (changes.vName.isFirstChange()) {
      this.logs.push('initial version is
      ${currValue.trim()}')
    } else {
      this.logs.push('version changed to
      ${currValue.trim()}')
    }
  }
}
```

Kako to funkcioniše...

`ngOnChanges` je jedna od mnogih udica (hooks) za „životni ciklus“ koje obezbeđuje Angular. Ova udica se pokreće i pre udice `ngOnInit`. Dakle, *početne vrednosti* dobijate u prvom pozivu, a kasnije dobijate *ažurirane vrednosti*. Kad god se bilo koji od ulaza promeni, povratni poziv `ngOnChanges` se pokreće pomoću `SimpleChangesa` i možete dobiti prethodnu vrednost, trenutnu vrednost i Bulovu vrednost, koje prikazuju da li je ovo prva promena na ulazu (tj. početna vrednost). Kada ažuriramo vrednost ulaza `vName` u nadređenoj komponenti, `ngOnChanges` se poziva pomoću ažurirane vednosti. Zavisno od situacije, dodajemo odgovarajuću evidenciju u naš niz `logs` i prikazujemo ga u korisničkom interfejsu.

Takođe pogledajte

- Angular udice „životnog ciklusa“:
  https://angular.io/guide/lifecycle-hooks
- korišćenje udica za otkrivanje promena u `ngOnChangesu`:
  https://angular.io/guide/lifecycle-hooks#using-change-detection-hooks
- referenca za API`SimpleChanges`:
  https://angular.io/api/core/SimpleChanges

Pristup podređenoj komponenti u nadređenom šablonu pomoću promenljivih šablona

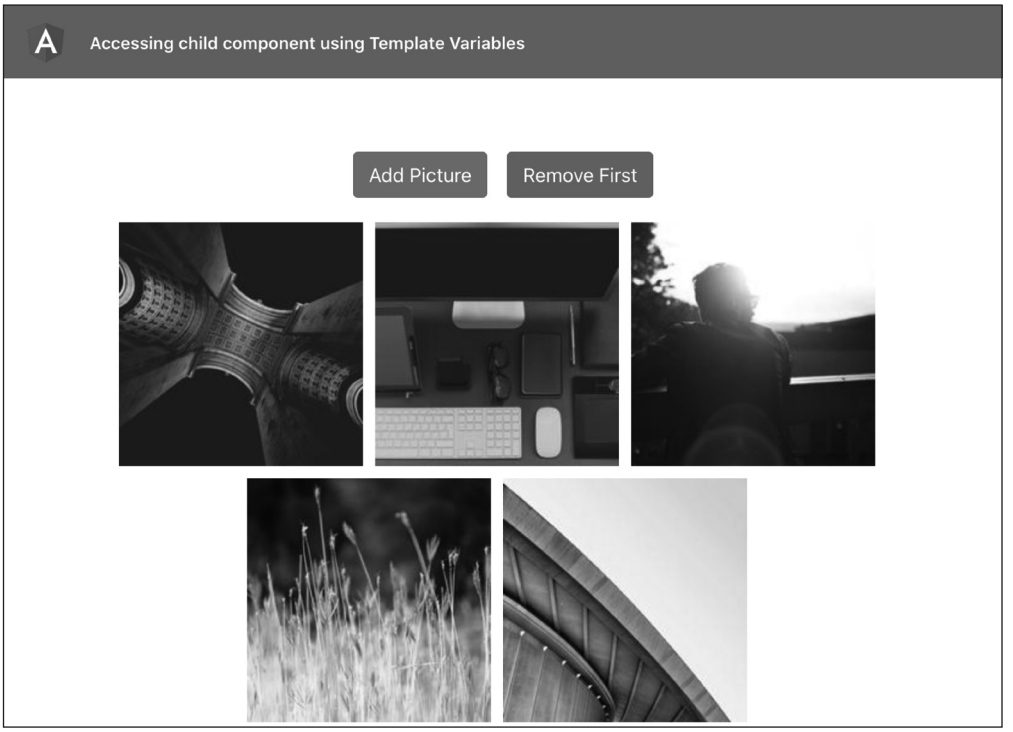
U ovom receptu ćete naučiti kako da koristite **Angular referentne promenljive šablona** za pristup podređenoj komponenti u šablonu nadređene komponente. Prvo ćete koristiti aplikaciju koja ima `AppComponent` kao nadređenu komponentu i `GalleryComponent` kao podređenu komponentu. Zatim ćete kreirati promenljivu šablona za podređenu komponentu u šablonu nadređene komponente da biste pristupili podređenoj komponenti i izvršili neke radnje u klasi komponente.

Priprema

Projekat koji ćemo koristiti nalazi se u `chapter01/start_here/cc-template-varsu` unutar kloniranog spremišta:

1. Otvorite projekat u Visual Studio Codeu.
2. Otvorite terminal i pokrenite `npminstall` da biste instalirali zavisnosti projekta.
3. Nakon toga, pokrenite `ng serve -o`.

Sada bi trebalo da se otvori aplikacija u novoj kartici pregledača i trebalo bi da vidite sledeće:



Slika 1.9 Aplikacija cc-template-vars koja je pokrenuta na adresi http://localhost:4200

4. Kliknite na dugmad pri vrhu da biste videli odgovarajuće evidencije konzole.

Kako to uraditi...

1. Prvo ćemo kreirati promenljivu šablona `#gallery` u komponenti `<app-gallery>` u datoteci `app.component.html`:

```
...
<div class="content" role="main">
  ...
  <app-gallery #gallery></app-gallery>
</div>
```

2. Zatim, modifikujemo metode `addNewPicture()` i `removeFirstPicture()` u datoteci `app.component.ts`, tako da mogu da prihvate parametar `gallery` i promenljivu šablona iz datoteke `app.component.html` kada kliknemo na dugmad. Kod bi trebalo da izgleda ovako:

```
import { Component } from '@angular/core';
import { GalleryComponent } from './components/gallery/
gallery.component';
...
export class AppComponent {
  ...
  addNewPicture(gallery: GalleryComponent) {
    console.log('added new picture');
  }
  removeFirstPicture(gallery: GalleryComponent) {
    console.log('removed first picture');
  }
}
```

3. Sada prosleđujemo promenljivu šablona `#gallery` iz datoteke `app.component.html` u hendlerima klikova za oba dugmeta na sledeći način:

```
…
<div class="content" role="main">
  <div class="gallery-actions">
    <button class="btn btn-primary"
    (click)="addNewPicture(gallery)">Add Picture</button>
    <button class="btn btn-danger"
    (click)="removeFirstPicture(gallery)">Remove
    First</button>
  </div>
  ...
</div>
```

4. Možemo da implementiramo kod za dodavanje nove slike. Za ovo ćemo pristupiti metodu `generatedImage()` klase `GalleryComponent` i dodati novu stavku u niz `pictures` kao prvi element. Kod je sledeći:

```
...
export class AppComponent {
```

```
    ...
    addNewPicture(gallery: GalleryComponent) {
      gallery.pictures.unshift(gallery.generateImage());
    }
    ...
  }
```

5. Za uklanjanje prve stavke iz niza koristićemo metod `shift` u nizu `pictures` u klasi `GalleryComponent` na sledeći način:

```
...
export class AppComponent {
    ...
    removeFirstPicture(gallery: GalleryComponent) {
      gallery.pictures.shift();
    }
}
```

Kako to funkcioniše...

Referentna promenljiva šablona je često referenca za DOM element unutar šablona. Takođe se može odnositi na direktivu (koja sadrži komponentu), element, `TemplateRef` ili veb komponentu (izvor: https://angular.io/guide/templatereference-variables).

U suštini, možemo se pozvati na našu komponentu `<app-gallery>`, koja „iza kulisa" predstavlja direktivu u Angularu. Kada imamo promenljivu u šablonu, prosleđujemo referencu funkcijama u našoj komponenti kao argumente funkcija. Odatle možemo da pristupimo svojstvima i metodima klase `GalleryComponent`. Možemo da dodajemo i uklanjamo stavke iz niza pictures koji se nalazi u GalleryComponentu direktno iz komponente AppComponent koja je nadređena komponenta u celom ovom toku.

Takođe pogledajte

- Angular promenljive šablona: https://angular.io/guide/templatereference-variables
- Angular iskazi šablona: https://angular.io/guide/templatestatements

Pristup podređenoj komponenti u klasi nadređene komponente pomoću dekoratora ViewChild

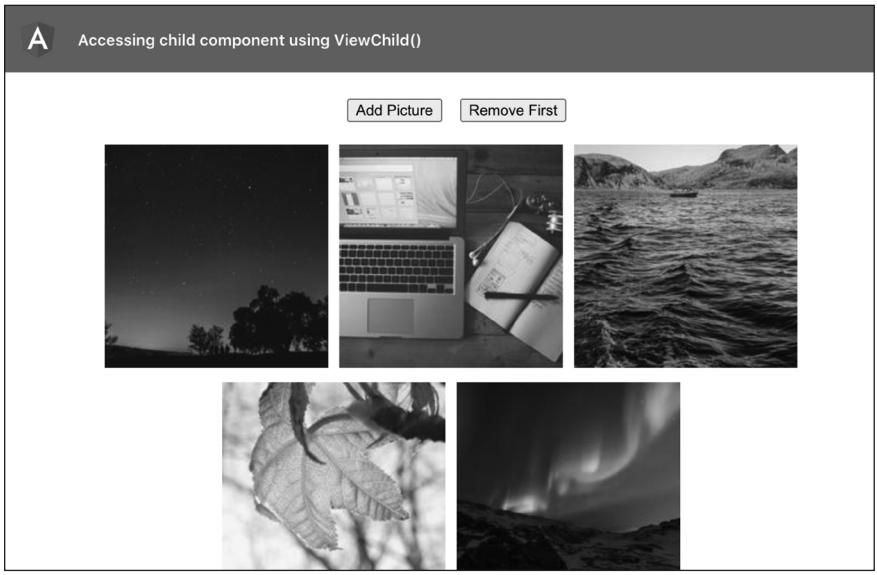
U ovom receptu ćete naučiti kako da koristite dekorator `ViewChild` za pristup podređenoj komponenti u klasi nadređene komponente. Prvo ćete koristiti aplikaciju koja ima `AppComponent` kao nadređenu komponentu i `GalleryComponent` kao podređenu komponentu. Zatim ćete kreirati `ViewChild` za podređenu komponentu u klasi nadređene komponente da biste pristupili podređenoj komponenti i izvršili neke radnje.

Priprema

Projekat koji ćemo koristiti nalazi se u `chapter01/start_here/cc-view-childu` unutar kloniranog spremišta:

1. Otvorite projekat u Visual Studio Codeu.
2. Otvorite terminal i pokrenite `npminstall` da biste instalirali zavisnosti projekta.
3. Nakon toga, pokrenite `ng serve -o`.

   Sada bi trebalo da se otvori aplikacija u novoj kartici pregledača i trebalo bi da vidite sledeće:



Slika 1.10 Aplikacija cc-view-child koja je pokrenuta na adresi http://localhost:4200

4. Kliknite na dugmad pri vrhu da biste videli odgovarajuće evidencije konzole.

Kako to uraditi...

1. Prvo ćemo da uvezemo `GalleryComponent` u našu datoteku `app.component.ts` da bismo mogli da kreiramo `ViewChild`:

```
import { Component } from '@angular/core';
import { GalleryComponent } from './components/gallery/
gallery.component';
...
export class AppComponent {
  ...
}
```

2. Zatim ćemo, koristeći dekorator `ViewChild()`, kreirati `ViewChild` za `GalleryComponent` na sledeći način:

```
import { Component, ViewChild } from '@angular/core';
import { GalleryComponent } from './components/gallery/
gallery.component';

export class AppComponent {
  title = 'cc-view-child';
  @ViewChild(GalleryComponent) gallery;
  ...
}
```

Sada ćemo implementirati logiku za dodavanje nove slike. Za to ćemo u metodu `addNewPicture` unutar komponente `AppComponent` koristiti rekvizit `gallery` koji smo kreirali u *koraku 2*. Možemo pristupiti nizu pictures iz podređene komponente.

3. Nakon toga, koristeći metod `generateImageGalleryComponenta`, dodaćemo novu sliku na vrh tog niza na sledeći način:

```
...
export class AppComponent {
  title = 'cc-view-child';
  @ViewChild(GalleryComponent) gallery: GalleryComponent;

  addNewPicture() {
    this.gallery.pictures.unshift(
    this.gallery.generateImage());
```

```
  }
  ...
}
```

4. Da bismo upravljali uklanjanjem slika, dodaćemo logiku metodu `removeFirstPicture` unutar klase `AppComponent`. Ovo ćemo uraditi i pomoću podređenog prikaza. Jednostavno ćemo koristiti metod `Array.prototipe.shift` u nizu `pictures` da bismo uklonili prvi element na sledeći način:

```
...
export class AppComponent {
...
  removeFirstPicture() {
    this.gallery.pictures.shift();
  }
}
```

Kako to funkcioniše...

`ViewChild()` je, u osnovi, dekorator koji obezbeđuje paket `@angular/core`. On konfiguriše **upit za prikaz** za Angular detektor promena. Detektor promena pokušava da pronađe prvi element koji odgovara upitu i dodeljuje ga svojstvu koje je povezano sa dekoratorom `ViewChild()`. U našem receptu mi kreiramo podređeni prikaz, tako što obezbeđujemo `GalleryComponent` kao parametar upita, tj. `ViewChild (GalleryComponent)`. Ovo omogućava Angular detektoru promena da pronađe element `<pp-gallery>` unutar šablona `app.component.html`, a zatim ga dodeljuje svojstvu `gallery` unutar komponente `AppComponent`. Važno je definisati tip svojstva galerije kao `GalleryComponent` da bismo ga kasnije mogli lako koristiti u komponenti zajedno sa svom TypeScript „magijom".

Važna napomena

Upit za prikaz se izvršava nakon udice „životnog ciklusa" `ngOnInit` i pre udice `ngAfterViewInit`.

Takođe pogledajte

- Angular`ViewChild`: https://angular.io/api/core/ViewChild
- metod niza shift: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/shift

Kreiranje vaše prve dinamičke komponente u Angularu

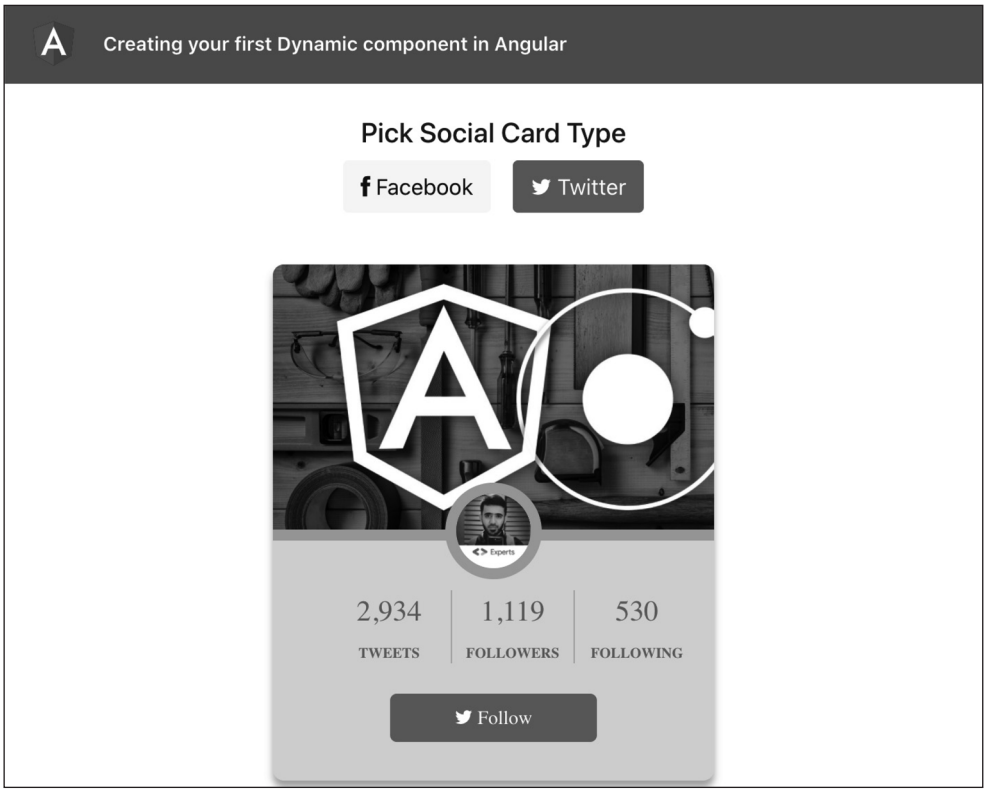
U ovom receptu ćete naučiti kako da kreirate **dinamičke komponente** u Angularu, koje se kreiraju dinamički na zahtev, na osnovu različitih uslova. Zašto? Zato što možda imate nekoliko složenih uslova i želite da učitate određenu komponentu na osnovu tih uslova, umesto da sve moguće komponente stavljate u šablon. Koristićemo servis `ComponentFactoryResolver`, dekorator `ViewChild()` i servis `ViewContainerRef` za postizanje dinamičkog učitavanja. Uzbuđen sam, a i vi verovatno!

Priprema

Projekat koji ćemo koristiti nalazi se u `chapter01/start_here/ng-dynamic-componentsu` unutar kloniranog spremišta:

1. Otvorite projekat u Visual Studio Codeu.
2. Otvorite terminal i pokrenite npm install da biste instalirali zavisnosti projekta.
3. Nakon toga, pokrenite `ng serve -o`.

   Sada bi trebalo da se otvori aplikacija u novoj kartici pregledača i trebalo bi da vidite sledeće:



Slika 1.11 Aplikacija ng-dynamic-components koja je pokrenuta na adresi http://localhost:4200

4. Kliknite na dugmad pri vrhu da biste videli odgovarajuće evidencije konzole.

Kako to uraditi...

1. Pre svega, uklonimo elemente pomoću direktiva `[ngSwitch]` i `*ngSwitchCase` iz naše datoteke `social-card.component.html` i zamenimo ih jednostavnim `divom` pomoću promenljive šablona `#vrf`. Ovaj `div` ćemo koristiti kao kontejner. Kod bi trebalo da izgleda ovako:

```
<div class="card-container" #vrf></div>
```

2. Zatim ćemo u datoteku `social-card.component.ts` dodati servis `ComponentFactoryResolver` na sledeći način:

```
import { Component, OnInit, Input,
ComponentFactoryResolver } from '@angular/core';
...
export class SocialCardComponent implements OnInit {
  @Input() type: SocialCardType;
  cardTypes = SocialCardType;
  constructor(private componentFactoryResolver:
  ComponentFactoryResolver) { }
  ...
}
```

3. Sada ćemo na sledeći način da kreiramo `ViewChild` za `ViewContainerRef` u istoj datoteci, tako da se možemo pozivati na div `#vrf` iz šablona:

```
import { Component, OnInit, Input,
ComponentFactoryResolver, ViewChild, ViewContainerRef }
from '@angular/core';
...
export class SocialCardComponent implements OnInit {
  @Input() type: SocialCardType;
  @ViewChild('vrf', {read: ViewContainerRef}) vrf:
  ViewContainerRef;
  cardTypes = SocialCardType;
  ...
}
```

4. Da bismo dinamički kreirali komponente, moramo osluškivati promene tipa ulaza. Dakle, kad god se tip ulaza promeni, učitavamo odgovarajuću komponentu dinamički. Za ovo ćemo implementirati udicu `ngOnChanges` u `SocialCardComponent` i za sada evidentirati promene u konzoli. Kada je udica implementirana, trebalo bi da vidite evidencije u konzoli nakon dodira na dugmad Facebook ili Twitter:

```
import { Component, OnInit, OnChanges, Input,
ComponentFactoryResolver, ViewChild, ViewContainerRef,
SimpleChanges } from '@angular/core';
...
export class SocialCardComponent implements OnInit,
```

```
OnChanges {
  ...
  ngOnChanges(changes: SimpleChanges) {
    if (changes.type.currentValue !== undefined) {
      console.log('card type changed to:
      ${changes.type.currentValue}')
    }
  }
}
```

5. Sada ćemo kreirati metod u `SocialCardComponent`, koji se zove `loadDynamicComponent` - on prihvata tip društvene kartice, tj. `SocialCardType`, i odlučuje koju komponentu će dinamički učitavati. Takođe ćemo kreirati promenljivu imenovanu komponentu unutar metoda da bismo izabrali koju komponentu treba učitati. Kod bi trebalo da izgleda ovako:

```
import {...} from '@angular/core';
import { SocialCardType } from 'src/app/constants/social-
card-type';
import { FbCardComponent } from '../fb-card/fb-card.
component';
import { TwitterCardComponent } from '../twitter-card/
twitter-card.component';
...
export class SocialCardComponent implements OnInit {
  ...
  ngOnChanges(changes: SimpleChanges) {
    if (changes.type.currentValue !== undefined) {
      this.loadDynamicComponent(
      changes.type.currentValue)
    }
  }
  loadDynamicComponent(type: SocialCardType) {
    let component;
    switch (type) {
      case SocialCardType.Facebook:
        component = FbCardComponent;
        break;
```

```
        case SocialCardType.Twitter:
          component = TwitterCardComponent;
          break;
      }
    }
  }
```

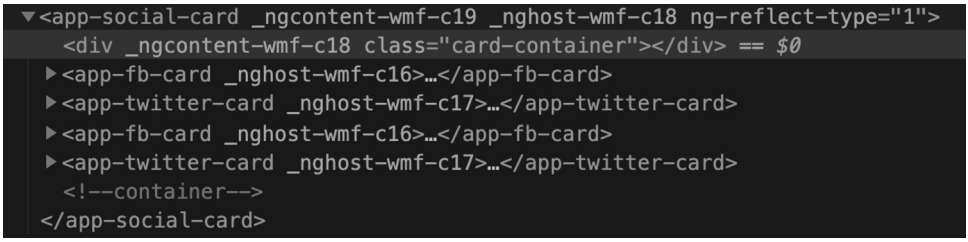
6. Sada, kada znamo koja komponenta treba da se dinamički učita, upotrebićemo `componentFactoryResolver` za razrešavanje komponente, a zatim za kreiranje komponente unutar metoda `VievContainerRef(vrf)` na sledeći način:

```
...
export class SocialCardComponent implements OnInit {
  ...
  loadDynamicComponent(type: SocialCardType) {
    let component;
    switch (type) {
      ...
    }
    const componentFactory = this.componentFactory
    Resolver.resolveComponentFactory(component);
    this.vrf.createComponent(componentFactory);
  }
}
```

Nakon prethodne izmene, skoro smo stigli do kraja. Kada prvi put dodirnete dugme Facebook ili dugme Twitter, trebalo bi da vidite da se odgovarajuća komponenta dinamički kreira.

Međutim, ako ponovo dodirnete bilo koje od ovih dugmadi, videćete da se komponenta dodaje u prikaz kao dodatni element.

Nakon pregleda, to bi moglo izgledati otprilike kao na sledećoj lici.



Slika 1.12 Pregled više elemenata koji se dodaju u ViewContainerRef

Pročitajte u odeljku „Kako to funkcioniše...“ zašto se to dešava. Da bismo to popravili, samo izvršimo metod `clear()` u `ViewContainerRefu` pre nego što kreiramo dinamičku komponentu, na sledeći način:

```
...
export class SocialCardComponent implements OnInit {
  ...
  loadDynamicComponent(type: SocialCardType) {
    ...
    const componentFactory = this.
    componentFactoryResolver.
    resolveComponentFactory(component);
    this.vrf.clear();
    this.vrf.createComponent(componentFactory);
  }
}
```

Kako to funkcioniše...

`ComponentFactoryResolver` je Angular servis koji omogućava dinamičko razrešavanje komponenata tokom izvršavanja. U našem receptu koristimo metod `resolveComponentFactory`, koji prihvata **Component** i vraća `ComponentFactory`. Uvek možemo koristiti metod `create` servisa `ComponentFactory` za kreiranje instanci komponente. Međutim, u ovom receptu koristimo `ViewContainerRefov` metod `createComponent`, koji prihvata `ComponentFactory` kao ulaz. Zatim koristi `ComponentFactory` u pozadini da generiše komponentu, pa je dodaje u priloženi `ViewContainerRef`. Svaki put kada kreirate komponentu i priključite je `ViewContainerRefu`, ona će dodati novu komponentu na postojeću listu elemenata. Za naš recept morali smo prikazati samo jednu po jednu komponentu, to jest ili `FBCardComponent` ili `TwitterCardComponent`. Da bi samo jedan element postojao u `ViewContainerRefu`, koristili smo metod `clear()` pre dodavanja elementa.

Takođe pogledajte

- metod `resolveComponentFactory`: https://angular.io/api/core/ComponentFactoryResolver#resolvecomponentfactory
- dokumentacija za Angular o učitavaču dinamičkih komponenata: https://angular.io/guide/dynamic-component-loader
- dokumentacija za `ViewContainerRef`: https://angular.io/api/core/ViewContainerRef
- dinamičko učitavanje komponenata u Angular 9 pomoću IVY-a: https://labs.thisdot.co/blog/loading-components-dynamically-in-angular-9-with-ivy

Angular

Kuvar

Skenirajte QR kod,
registrujte knjigu
i osvojite nagradu

Angular radni okvir, koji obezbeđuje Google, koristi se u mnogim projektima veb razvoja izrađenim na uređajima različitih veličina. Poznato je da Angular obezbeđuje neophodnu stabilnost i bogat eko-sistem alatki za izradu veb aplikacija i aplikacija za mobilne uređaje koje su spremne za puštanje u rad. Ovaj vodič zasnovan na receptima omogućava da detaljno naučite Angular koncepte, koristeći pristup „korak po korak“. Istražićete širok spektar recepata za ključne zadatke u veb razvoju koji će vam biti korisni u izradi aplikacija visokih performansi.

Na početku knjige ćete upoznati koncepte Angulara, kao što su Angular komponente, direktive i servisi, da biste se pripremili za izradu frontend veb aplikacija. Razvijaćete veb komponente pomoću Angulara i pokrivaćete napredne koncepte, kao što su učitavanje dinamičkih komponenata i upravljanje stanjem pomoću NgRxa, radi postizanja performansi u realnom vremenu. Kasnija poglavlja su fokusirana na recepte za efikasno testiranje vaših Angular aplikacija kako bi bile bezbedne od otkazivanja rada, pre nego što pređete na tehnike za optimizaciju performansi vaše aplikacije. Na kraju ćete kreirati progresivne veb aplikacije (PVA - Progressive Web Apps) pomoću Angulara da biste korisnicima obezbedili intuitivno iskustvo.

Nakon što pročitate ovu knjigu o Angularu u celosti, moći ćete da kreirate prave Angular aplikacije profesionalnog izgleda i steći ćete veštine koje su vam potrebne za razvoj interfejsa, a koje su ključne za svakog Angular programera za preduzeća.

Šta ćete dobiti ovom knjigom:

- Steći ćete bolje razumevanje o načinu kako komponente, servisi i direktive funkcionišu u Angularu.
- Saznaćete kako da kreirate progresivne veb aplikacije korišćenjem Angulara „od nule“.
- Kreiraćete bogate animacije i dodaćete ih u vaše Angular aplikacije.
- Upravljaćete reaktivnošću podataka vaše aplikacije pomoću RxJSa.
- Implementiraćete upravljanje stanjem za vaše Angular aplikacije pomoću NgRxa.
- Optimizovaćete performanse vaših novih i postojećih veb aplikacija.
- Napisaćete bezbednosne jedinične testove i end-to-end testove za vaše veb aplikacije, koristeći Jest i Cypress.
- Upoznaćete Angular CDK komponente za projektovanje efikasnih Angular komponenata.

ISBN: 978-86-7310-569-7