Angular Komponente

Autori:
Milan Segedinac
Goran Savić

1. Komponente

Komponente su osnovni gradivni blokovi Angular aplikacija. One uključuju *prikaz* (definisan HTML i CSS) i *kontroler* koji implementira funkcionalnosti koje se koriste u prikazu. Kontroler ima tri osnovne dužnosti:

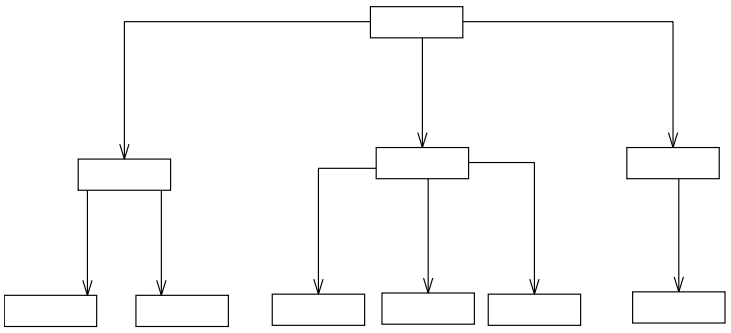
1. Rukovanje modelom odnosno podacima koji će biti korišćeni u prikazu
2. Implementiranje metoda koje se koriste u prikazu kao što je prikazivanje i skrivanje elemenata prikaza, obrada događaja ili slanje podataka na back-end
3. Rukovanje stanjem prikaza, kao što je priprema podataka čiji detalji će biti prikazani

Komponente ne postoje nativno ni u ES3/5/6 ni u TypeScript-u i Angular pruža mogućnost definisanja komponenti kroz povezivanje klasa koje modeluju kontroler i HTML templjeta i CSS stilova koji modeluju prikaz. Ovo povezivanje vrši se pomoću `@Component` dekoratera.

Hiherarhija komponenti

Angular aplikacija je organizovana kao stablo (hijerarhija) komponenti. Aplikacija počinje korenskom komponentom i sve ostale komponente su direktni ili posredni potomci ove komponente. Komponente se pišu tako da sadrže sve potrebne delove u sebi da bismo mogli da ih koristimo u različitim aplikacijama, ali i jer se time postiže efikasniji razvoj i održavanje aplikacije. Na primer, ako su CSS stilovi sadržani u samoj komponenti znamo da druge komponente neće kontaminirati prikaz te komponente. Ukoliko postoji problem u stilovima, makar znamo gde da ga tražimo.

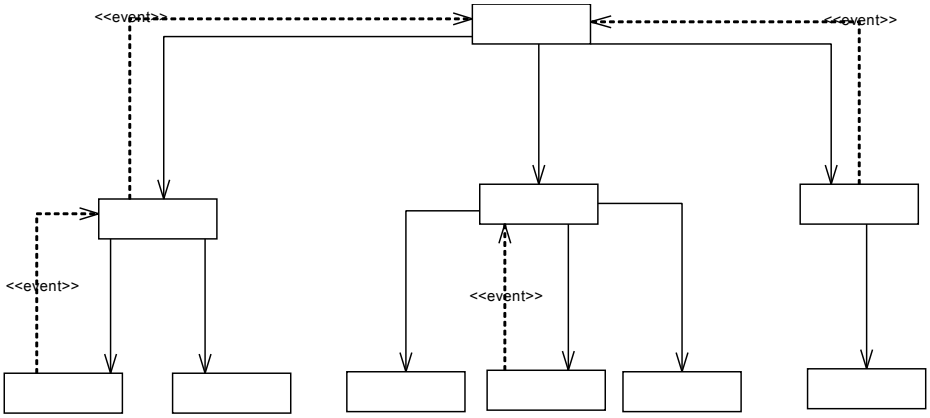
I pored svoje izolovanosti, komponente trebaju međusobno da komuniciraju. U Angularu komponenta može da primi podatke iz svoje roditeljske komponente (input) i da pošalje podatke svojoj roditeljskoj komponenti (output). Pri tome je neophodno specificirati šta će biti input, a šta output komponente. Organizacija komponenti je prikazana slikom ispod.



Slika – organizacija komponenti

Na slici smer strelica u grafu označava smer prenošenja podataka. Vidimo da se podaci uvek prenose iz roditeljske komponente u komponentu direktnog potomka. Takođe, važno je napomenuti da je graf komponenti usmeren i acikličan. To garantuje *jednosmerni tok podataka* (eng. *unidirectional data flow*).

Ovakva struktura ima važna svojstva: predvidiva je, jednostavno je proći kroz čvorove ove strukture, Jendnostavno je identifikovati čvorove na koje će se odraziti promena koja se desi u strukturi (to su samo čvorovi potomci). Ovaj prenos podataka se naziva property binding, realizuje se kroz input atribute i pre svega je namenjen sa prenos podataka iz roditeljske komponente koji će biti prikazani u potomačkim komponentama. Međutim, ovakav pristup nije dovoljan kada postoji potreba za izmenom podataka iz potomačke komponente koja će se odraziti na roditeljsku komponentu. Shodno tome, potomačka komponenta može da izazove događaj na koji roditeljska komponenta reaguje i da kroz taj događaj pošalje podatke koji će se preneti iz potomačke u roditeljsku komponentu. Slika ispod prikazuje hijerahiju komponenti u kojoj su mogući događaji.



Slika – vezivanje događaja u hijerarhiji komponenti

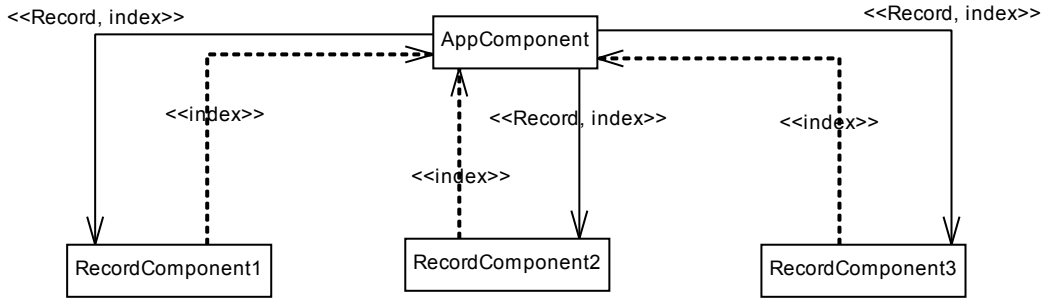
Na slici iznad vidimo da komponenta može da, pomoću događaja obavesti svoju roditeljsku komponentu i prosledi joj događaje. Sa događajima smo se već susreli: na primer događajem klika miša. Angular nam omogućuje da sami definišemo nove događaje u potomačkoj komponenti i da reakcije na njih definišemo u roditeljskoj komponenti. Ovakvo prenošenje podataka naziva se *vezivanje događaja*. Obratite pažnju da ova vrsta vezivanja podataka *ne narušava* jednosmerni tok podataka.

Izdvajanje komponente – primer 1

U primeru prodavnice muzičkih albuma ćemo prikaz jednog albuma izdvojiti u zasebnu komponentu - `RecordComponent` - i komponentu ćemo ponavljati za svaki element iz liste albuma. Izdvojenu komponentu uokvirenu crveno na stranici vidimo na slici ispod.

Slika – komponenta za prikaz albuma

Ta komponenta treba da primi objekat tipa `Record` i prikaže naslov, autora, naslovnu sliku, listu stilova i cenu. Pored toga, komponenta će trebati da primi i indeks prikazanog albuma. To znači da će komponenta imati jedan input preko kog je moguće preneti objekat `Record` za prikaz i još jedan input za prenos indeksa albuma. Takođe, komponenta treba da izmeni listu albuma ukoliko se klikne na dugme obriši, za šta će joj trebati jedan output kroz koji će se slati indeks u nizu albuma za brisanje. Hijerarhija komponenti je prikazana slikom ispod.



Slika – hijerarhija komponenti

Inicijalno kreiranje komponenti pojednostavljeno je uz korišćenje angular-cli paketa i ostvaruje se komandom `ng generate component <naziv-komponente>`. Pokretanje ove komande generiše sve fajlove koji ulaze u sastav komponente u odgovarajućem direktorijumu. Naravno, sama programska logika komponente ne može se izgenerisati tako jednostavno već je zadatak programera da je napiše. Sintaksa zadavanja komponenti ilustrovana je listingom ispod i objašnjena nakon listinga.

```
import { Component, OnInit, Input, Output, EventEmitter } from
'@angular/core';


import { Record } from '../record.model';


@Component({
  selector: 'app-record',
  templateUrl: './record.component.html'
})
export class RecordComponent {

      @Input() record:Record;
      @Input() index:number;
      @Output() deleteRecordIndex: EventEmitter<number> = new
EventEmitter();

  constructor() { }

  deleteRecord(index:number){
      this.deleteRecordIndex.next(index);
  }
}
```

Listing - `RecordComponent`

Komponenta je definisana klasom `RecordComponent`. Ulaze klase defnišemo dekoratorom `@Input()`. Vidmo da su definisana dva ulaza: `record` i `index`. Izlaz se definiše kao emiter događaja, odnosno kao objekat klase `EventEmitter` i anotiran je dekoratorom `@Output()`. Naša komponenta ima jedan izlaz: `deleteRecordIndex`. Prilikom definisanja tipa `deleteRecordIndex` tip `EventEmitter` je parametrizovan tipom number, čime postižemo da smo specificirali da će se događajem slati `indeks` koji je numerička vrednost. Obratite pažnju da smo `Input`, `Output` i `EventEmitter`

importovali iz `'@angular/core'` modula. Takođe, obratite pažnju i na selektor ove komponente - `app-record` - kojim se postiže da će ta komponenta u prikaze drugih komponenti moći da se ugradi kao element `<app-record>`.

Emitovanje događaja ostvaruje se u telu funkcije `deleteRecord`, gde se pozivom `this.deleteRecordIndex.next(index);` emituje izlazni događaj uz slanje indeksa alubma.

Listingom ispod dat je HTML templejt `RecordComponent` komponente.

```
<h2>{{record.title}}</h2>

<div>{{record.author}}</div>
<div><img  width="100"  height="100"  src="{{record.imageUrl}}"></
div>
<div>{{record.price}}</div>
<div *ngIf="record.styles.length!=0">
styles:
     <ul>
          <li *ngFor="let style of record.styles">
               {{style}}
          </li>
     </ul>
</div>
<button (click)="deleteRecord(index)">remove</button>
```

Listing - prikaz `RecordComponent` komponente

Kao što vidimo, klik na dugme remove poziva metodu `deleteRecord` koja će emitovati događaj brisanja albuma.

Ostaje još da vidimo kako `RecordComponent` komponentu možemo da ugradimo u glavnu komponentu, što je prikazano listingom ispod.

```
...

<div>
  <h1>Records</h1>
  <ul>
    <li *ngFor="let record of records; let i = index">
      <app-record [record]="record" [index]="i"
                        (deleteRecordIndex)="delete($event)"></app-
record>
    </li>
  </ul>
</div>
```

Listing - korišćenje `RecordComponent` komponente

Vidimo da je u `*ngFor` petlji za svaki element `record` iz liste `records` dodata nova komponenta `<app-record>`. Prenos ulaznih podataka komponente definiše se pomoću

uglastih zagrada. Tako `[x]="y"` znači da će atribut `x` (koji naravno mora da bude označen kao `@Input`) komponente dobiti vrednost `y` iz roditelja komponente. U našem slučaju, `record` iz roditelja ćemo preneti u `record` u `RecordComponent`, a `i` iz roditelja (indeks `*ngFor` petlje) ćemo preneti u atribut `index` komponente `RecordComponent`.

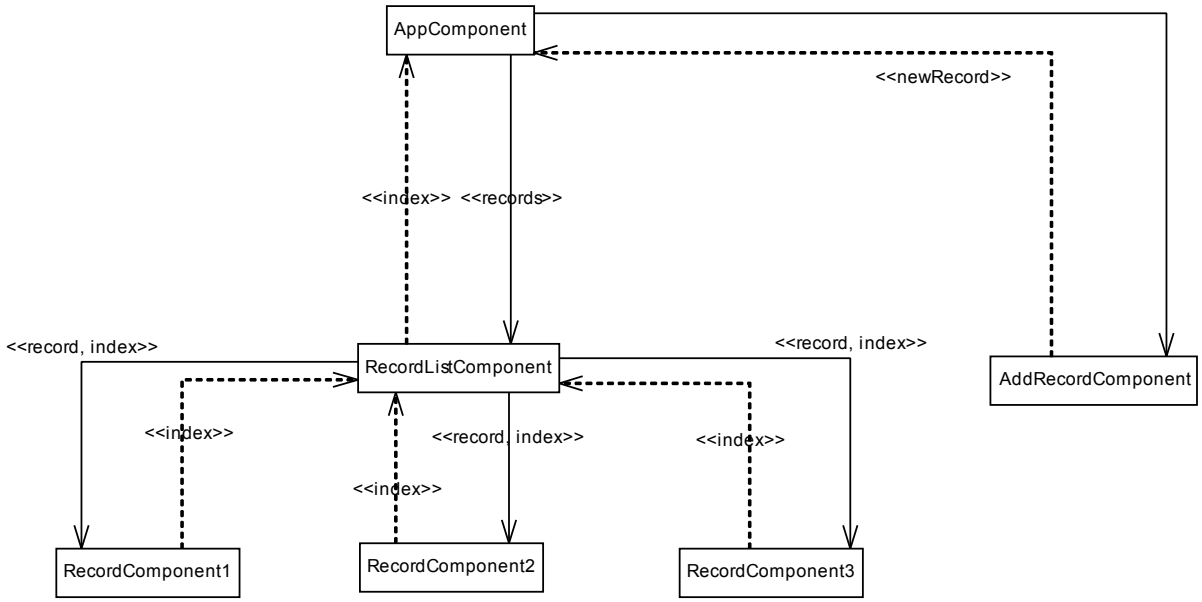
Reakcija na izlazne događaje definiše se pomoću običnih zagrada. Tako `(deleteRecordIndex)="delete($event)"` znači da će se na događaj koji u komponenti `RecordComponent` izaziva `deleteRecordIndex` tipa `EventEmitter` biti pozvana metoda `delete` roditelja. Argument `$event` znači da će se kao prvi parametar roditeljske metode preuzeti vrednost koju emiter emituje.

Izdvajanje komponenti – primer 2

U ovom primeru ćemo nastaviti da razrađujemo aplikaciju razdvajanjem na stablo komponenti. Izdvojićemo još dve komponente: listu albuma (`RecordListComponent`) i formu za unos (`AddRecordComponent`). Slika ispod pokazuje ove komponente u prikazu.

Slika – komponente u prikazu

Na slici ispod možemo videti kako izgled hijerarhija komponenti i kako se prenose podaci između njih.



Slika – hijerarhija kompomenti

`RecordComponent` komponente smo okupili u zasebnu komponentu `RecordListComponent`. Komponenta `RecordListComponent` od `AppComponent` prima kao ulaz listu albuma (`records`), a svaka `RecordComponent` od nje prima jedan objekat `record` i `index` u `*ngFor` petlji. `RecordComponent` na klik na dugme za brisanje reaguje tako što roditelju (`RecordListComponent`) emituje događaj brisanja prosledivši indeks elementa za brisanje. Kada `RecordListComponent` „uhvati“ taj događaj, emituje ga svom roditelju, takođe prosledivši indeks za brisanje. Na ovaj događaj `AppComponent` reaguje pozivom metode delete sa prosleđenim indeksom. U ovom slučaju vidimo da potomačka komponenta može da komunicira samo sa svojim neposredim roditeljem emitovanjem događaja. Korišćenjem reaktivnih ekstenzija RxJS moguće je uspostaviti vezu komunikacije i na proizvoljnoj udaljenosti u stablu.

Komponenta `AddRecordComponent` ne prima ništa od svoje roditeljske komponente, a šalje joj novouneti album putem emisije događaja.

Rekli smo da je i korenska komponenta takođe komponenta. Selektor ove komponente je `app-root`. Ako pogledamo `index.html` iz primera dat listingom ispod u videćemo da je u njemu zadat element `<app-root>`.

```
<html>

<head>
...
</head>
<body>
  <app-root>Loading...</app-root>
</body>
</html>
```

Listing - `AppComponent` u `index.html`

Pokretanje (eng. *bootstraping*) Angular aplikacije počinje upravo od ove komponente i čitava aplikacija se razvija kao stablo komponenti čiji koren je ova komponenta.