Angular Uvod

Autori:
Milan Segedinac
Goran Savić

1. SPA

Jednostranična veb aplikacija (eng. *single-page application*, *SPA*) je veb sajt koji je čitav sadržan u jednoj veb stranici i koji tipično pruža korisničko iskustvo slično radu u desktop aplikacijama. U SPA aplikacijama se ili sav potreban kod (HTML, JavaScript, CSS) dobavlja jednim jedinim učitavanjem ili se, češće, inicijalno učita kostur stranice, a naknadno po potrebi učitavaju dodatni resursi i ugrađuju u učitanu stranicu. I pored toga što se u ovakvim aplikacijama nikada ne učitavaju nove stranice niti se kontrola prepušta drugim stranicama, korisnik može da ima osećaj navigiranja koji se ostvaruje izmenom sadržaja (logičkih stranica) u aplikaciji. Ovakve aplikacije najčešće uključuju intenzivnu komunikaciju sa back-end delom aplikacije.

Shodno tome logika SPA aplikacija uključuje učitavanje podataka sa servera i slanje podataka na server, ugrađivanje dobijenih podataka u strukturu stranice, navigaciju (promenu logičke stranice) uz ažuriranje history objekta, elemente bogatog korisničkog interfejsa koji korisnicima pružaju iskustvu uporedivo sa radom u desktop aplikacijama, ... Iako je sve ovo moguće ostvariti u „vanila“ JavaScript aplikacijama ili aplikacijama koje koriste jednostavne bibliteke poput jQuery, takav pristup razvoju velikih SPA aplikacija je nepraktičan i najčešće rezultuje aplikacijama koje se teško razvijaju, a još teže održavaju. Zamislimo kako bi izgledala SPA aplikacija razvijena samo uz korišćenje jQuery bibliteke za REST back-end koji ima nekoliko stotina resursa!

Jednostranične aplikacije najčešće se razvijaju uz oslonac na neki od specijalizovanih *radnih okvira*. Danas su takvi radni okviri brojni – React, Ember.js, Meteor.js, Aurelia, Vue.js i mnogi drugi. Mi ćemo koristiti Angular radni okvir.

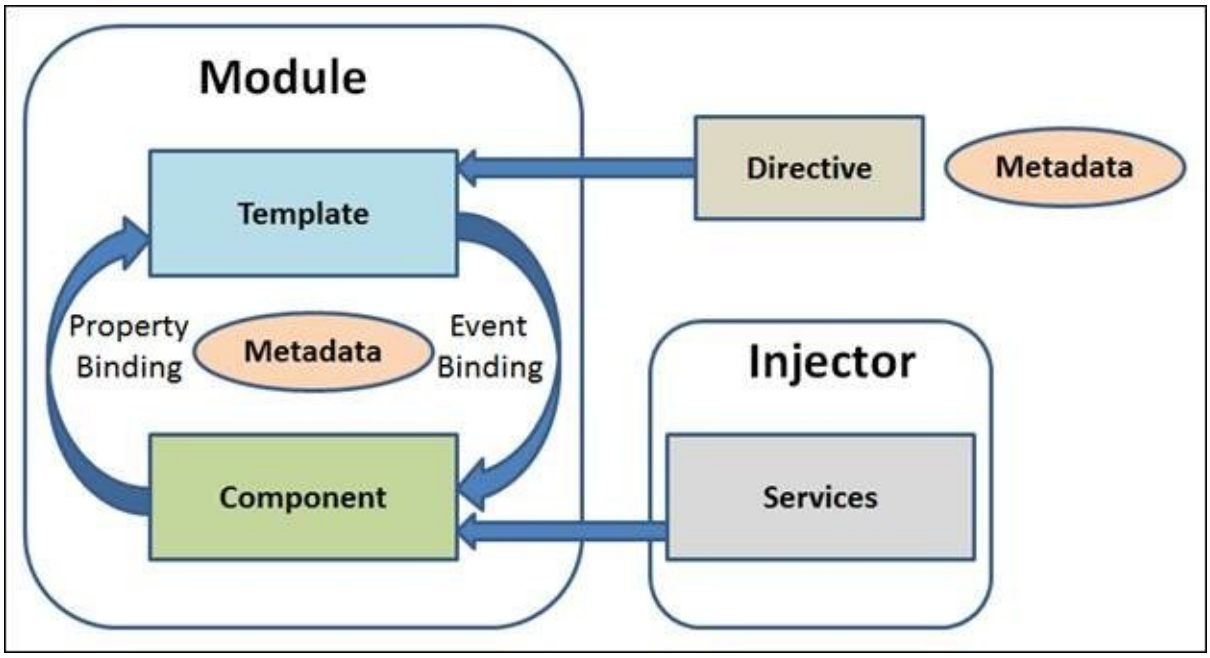
Angular

Angular je radni okvir za razvoj klijentskih aplikacija za programski jezik TypeScript. U nastavku ćemo dati kratak pregled osnovnih gradivnih blokova Angular aplikacije. Svaki od ovih blokova u narednim lekcijama biće detaljno razrađen.

- Modul – Angular aplikacije su modularne. Za modularizaciju aplikacija koriste se ES6 moduli koji su, naravno, dostupni i u TypeScript-u
- Komponenta – Komponente su osnovni gradivni blokovi Angular aplikacije i one objedinjuju HTML templejt i poslovnu logiku. Angular aplikacije prave se kao stabla komponenti, čime ćemo se detaljno baviti kada bude bilo reči o komponentnoj arhitekturi. Za sada spomenimo samo da svaka aplikacija mora da ima makar jednu komponentu – korensku komponentu i da pri pokretanju aplikacije (bootstraping) prikazivanje počinje učitavanjem te komponente. Komponente u Angularu su klase koje definišu programsku logiku dekorisane dekoraterom `@Component`. Ovaj dekorater prima konfiguracioni objekat kroz koji se, između ostalog, prosleđuje i templejt koji je namenjen prikazu komponente.
- Templejt – Prikaz komponente ostvaruje se kroz templejte. Templejti se zadaju u formatu koji je nadskup HTML
- Dekoratori – Konfigurisanje Angular aplikacija bazira se na deokratorima. Na primer, veza klase komponente i templejta za prikaz ostvaruje se dekoratorima.

- Vezivanje podataka – Prenošenje podataka iz programske logike definisane u klasi komponente u prikaz zadat templejtom i u obrnutom smeru automatski se ostvaruje vezivanjem podataka. Angular omogućuje 4 vrste vezivanja podataka: vezivanje svojstava (Property Binding), vezivanje događaja (Event Binding), interpolacija (Interpolation) i dvosmerno vezivanje podataka (Two-Way Binding)
- Direktive – Angular omogućuje proširivanje skupa elemenata HTML, a ova funkcionalnost ostvaruje se pomoću direktiva
- Servisi – Često postoji potreba da se isti programski kod ili isti podaci koriste u različitim delovima aplikacije. Na primer, programski kod za preuzimanje i slanje resursa sa/na server često nam treba u različitim komponentama. U Angularu se servisi koriste za enkapsulaciju takvog koda
- Injekcija zavisnosti – Angular intenzivno koristi ovaj šablon kreiranja pomoću kog komponente pribavljaju potrebne objekte u vremenu izvršavanja. Time se pojednostavljuje održavanje, ponovno korišćenje i jednostavnost testiranja komponenti Angular aplikacija

Ugrubo, gradivni blokovi Angular aplikacije prikazani su slikom ispod.



Slika – gradivni blokovi Angular aplikacije

RecordStore aplikacija – prikaz jednog albuma

Pratićemo razvoj jednostavne Angular aplikacije za rukovanje muzičkim albumima. U vreme pisanja ovih materijala pregledači ne podržavaju TypeScript, pa je Angular aplikacije koje razvijamo u tom programskom jeziku potrebno prevesti u ES5. Takođe, Angular aplikacije zahtevaju veliki broj dodatnih paketa i imaju preporučenu strukturu. Pravljenje i prevođenje Angular aplikacija značajno je jednostavnije uz korišćenje npm paketa `angular-cli`. Za korišćenje ovog paketa prvo nam je potrebno da ga instaliramo globalno (`npm install -g angular-cli`). Nakon što je instaliran, inicijalizovaćemo aplikaciju komandom `ng new <naziv-aplikacije>`. Ta komanda će kreirati inicijalnu

aplikaciju u folderu `<naziv-aplikacije>`. Kada se pozicioniramo u taj folder, aplikaciju pokrećemo komandom `ng serve`. Aplikacija će biti dostupna na portu 4200.

Nakon inicijalizacije aplikacije prvi korak je modelovanje podataka. Napravili smo klasu Record koja modeluje muzički album. Klasa ima naslov (`title`), sliku (`imageUrl`), listu žanrova (`styles`) i autora (`author`). Prilikom kreiranja, konstruktoru ćemo proslediti konfiguracioni objekat koji sadrži sve ove podatke. Da bismo bili sigurni da će konfiguracioni objekat i instance klase imati baš te atribute napravili smo interfejs `RecordInterface` kojim smo te atribute propisali. Taj interfejs smo implementirali klasom `Record` i postavili ga kao tip konfiguracionog objekta koji se prosleđuje konstruktoru ove klase. Programski kod je dat listingom ispod.

```
export class Record implements RecordInterface{


	public title: string;
	public imageUrl: string;
	public style: string[];
	public author: string;

	constructor(recordCfg:RecordInterface)
	{
		this.title = recordCfg.title;
		this.imageUrl = recordCfg.imageUrl;
		this.style = recordCfg.style;
		this.author = recordCfg.author;
	}
}

interface RecordInterface{
	title: string;
	imageUrl: string;
	style: string[];
	author: string
}
```

Listing – model podataka - `Record`

Obratite pažnju da je `record.model` modul iz koje je klasa `Record` eksportovana navođenjem rezervisane reči `export`. To znači da ćemo ovu klasu moći da importujemo u drugim modulima.

Naša aplikacija će, za sada, samo prikazati jedan jedini album. Prikaz albuma realizovaćemo u korenskoj komponenti. Prilikom inicijalizacije aplikacije pomoću paketa angular-cli kreirana je korenska komponenta `app.component.ts`. Ovu komponentu ćemo izmeniti tako da prikazuje jedan album.

Pored importa `import { Component } from '@angular/core';` kojim se iz angular core modula importuje `Component` dekorater, dodali smo i import `Record` modela. Klasa

AppComponent je prerađena tako da ima samo jedan javni atribut: `record` tipa `Record`. U konstruktoru ove klase postavljamo vrednost tog atributa, odnosno `this.record`.

Klasa AppComponent je dekroisana dekoratorom `Component`. Obratite pažnju da ovaj dekorator prima konfiguracioni objekat. Za sada smo postavili samo dva atributa tog objekta: `templateUrl` kojim je zadata relativna putanja do templejta za prikaz ove komponente i atribut `selector` kojim se zadaje naziv elementa za ovu komponentu koji će moći da se koristi u Angula aplikaciji. Vrednost zadatu selektorom možemo posmatrati kao proširenje vokabulara HTML dostupno u Angular aplikaciji. Programski kod `AppComponent` je dat listingom ispod.

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

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  public record: Record;

  constructor(){
      this.record = new Record({
            title:'Highway 61 Revisited',
            imageUrl:  'https://upload.wikimedia.org/wikipedia/en/
9/95/Bob_Dylan_-_Highway_61_Revisited.jpg',
            style: ['rock'],
            author: 'Bob Dylan'
      });
  }
}
```

Listing - `AppComponent` korenska komponenta

Preostaje još jedino da se muzikčki album prikaže u prikazu za šta je zadužen templejt. Svi javni atributi i metode klase komponente dostupni su u templejtu. Najjednostavniji način da se podaci prikažu u templejtu je pomoću interpolacije. Interpolacija je jednosmerno prenošenje podataka iz komponente u templejt koje se ostvaruje obuhvatanjem naziva atributa u dvostruke vitičaste zagrade. Time se postiže da će ukoliko dođe do izmene vrednosti atributa ta izmena biti vidiljiva i u prikazu. U templejtu za prikaz muzičkog albuma prikazaćemo naslov, autora i sliku. Programski kod je dat listingom ispod.

```
<h1>

  RecordStore
</h1>
<div>
  {{record.title}}<br/>
  <img src="{{record.imageUrl}}"/><br/>
  {{record.author}}
</div>
```

Listing – templejt sa interpolacijom

Obratite pažnju da smo interpolacijom prikazali atribut `title` objekta `record`, atribut `author` objekta `record` i atribut `imageUrl` objekta `record`. U stvari, interpolacijom može da bude prikazan bilo koji evaluirani JavaScript izraz. Na primer `{{record.author||'Bob Dylan'}}` bi bio legitiman interpolirani izraz koji, ukoliko nije zadat autor prikazuje da je to baš Bob Dylan. Takođe, obratite pažnju da je interpolirani izraz postavljen i kao vrednost HTML atributa src. Interpolirani izraz može se naći bilo gde u templejtu.

Nakon pokretanja naša aplikacija izgleda kao na slici ispod.

Slika – prikaz jednog albuma

RecordStore aplikacija – prikaz više albuma

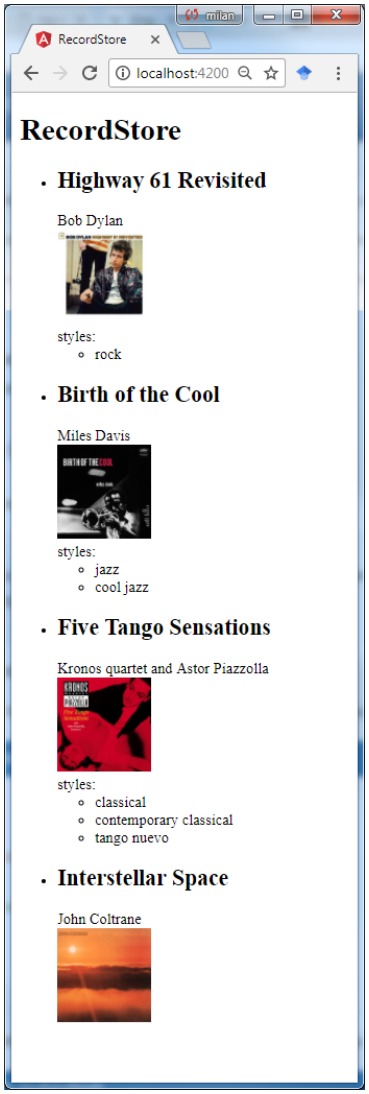
Aplikaciju ćemo proširiti tako da može da se prikazuje više albuma. Pre svega trebamo da izmenimo korensku komponentu tako da, umesto atributa jednog muzičkog albuma imamo listu muzičkih albuma. Izmena je prikazana listingom ispod.

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

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  public records: Record[];

  constructor(){
    this.records = [];
    var record = new Record({...});
    this.records.push(record);
    record = new Record({...});
    this.records.push(record);
    record = new Record({...});
    this.records.push(record);
    record = new Record({...});
    this.records.push(record);
  }
}
```

Listing – korenska komponenta sa listom albuma

Sada je atribut `records` tipa liste objekata tipa `Record`. U konstruktoru smo kreirali 4 objekat tipa `Record` i dodali ih u listu. Zbog prostora u tekstu konfiguracione objekte za kreiranje albuma nismo prikazali, već smo ih zamenuli sa tri tačke. Kompletan kod dostupan je u primerima u primeru 2.

Izgled aplikacije prikazan je slikom ispod.



Slika – stranica sa više albuma

U templejtu nam više nije dovoljno da samo prikazujemo atribute komponente, već nam je potrebno da menjanom strukturu prikaza u zavisnosti od vrednosti tih atributa. Želimo da se ponavlja prikaz svih komponenti. Takođe, ako za album nije zadat ni jedan stil ne želimo da se prikaže naslov liste stilova, kao što je slučaj sa poslednjim albumom na slici. Za uređenje strukture prikaza se koriste *strkuturalne direktive*. U našoj aplikaciji ćemo koristiti dve: `*ngFor` i `*ngIf`.

`*ngFor` je direktiva ponavljanja pomoću koje se prikazuje lista elemenata. Definiše se jedan blok koda kojim se kaže kako će izgledati jedan element iz kolekcije i da se takav prikaz ponovi za svaki element.

`*ngIf` je direktiva pomoću koje se zadaje uslov od čije ispunjenosti zavisi da li će element biti ugrađen u DOM stablo.

Templejt sa korišćenjem ove dve direktive dat je listingom ispod.

```
<h1>

  RecordStore
</h1>
<ul>
  <li *ngFor="let record of records">
    <h2>{{record.title}}</h2>
    <div>{{record.author}}</div>
  <div><img  width="100"  height="100"  src="{{record.imageUrl}}"></
div>
    <div *ngIf="record.styles.length!=0">
    styles:
      <ul>
        <li *ngFor="let style of record.styles">
          {{style}}
        </li>
      </ul>
    </div>
  </li>
</ul>
```

Listing - `*ngFor` i `*ngIf` direktive u templejtu

Napravili smo jednu neuređenu listu. Nad njen element `<li>` primenili smo direktivu `*ngFor`. Za ovu direktivu se zadaje varijabla koja će uzimati jednu po jednu vrednost iz kolekcije - `let record of records` znači da će u svakoj stvci listi biti dostupan po jedan album iz kolekcije vidljiv kroz varijablu `record`. Istu direktivu iskoristili smo i za prikaz stilova.

Ukoliko album ima postavljene stilove želimo da se u zasebnom divu ispiše tekst `"styles:"` i ispod njega lista stilova. Ukoliko nema postavljen ni jedan stil, ne želimo da se ovaj element doda. Tu funkcionalnost postigli smo korišćenjem `*ngIf` direktive. Ovom direktivom zadali smo izraz `record.styles.length!=0` od čije ispunjenosti zavisi da li će element biti dodat u DOM stablo. Ova direktiva može da primi bilo koji JavaScript izraz koji vraća truthy ili falsy vrednost.

RecordStore aplikacija – dodavanje i brisanje albuma

U ovom koraku aplikaciju ćemo proširiti tako da omogućimo dodavanje i brisanje albuma. Prvo nam trebaju funkcija za dodavanje novog objekta tipa `Record` u listu records klase `AppComponent` i za uklanjanje elementa na zadatoj poziciji iz ove kolekcije. Ove dve metode trebaju da budu javne da bi mogle da se koriste u templejtu. Pogledajmu izmenjenu klasu `AppComponent` datu listingom ispod.

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

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  public records: Record[];
  public newRecord: Record;

  constructor(){
    this.records = [];
    this.newRecord = {
      title:'',
      author:'',
      imageUrl:'',
      styles:[]
    };
...
  }

  save(){
    //kloniranje objekta
    var newRecordCopy:Record = Object.assign({},this.newRecord);
    this.records.push(newRecordCopy);
    this.newRecord = {
      title:'',
      author:'',
      imageUrl:'',
      styles:[]
    };
  }

  delete(index: number){
    this.records.splice(index,1);
  }
}
```

Listing – dodavanje i brisanje albuma

U klasu `AppComponent` smo dodali novi atribut: `newRecord` tipa `Record`. U ovaj objekat će se vrednosti iz prikaza postavljati dvosmernim vezivanjem podataka. Metoda `save` će klon objekat `newRecord` dodati u listu `records`. Prva linija ove metoda klonira objekat `newRecord`, druga linija klon dodaje u listu `records`, a treća resetuje sve atrbuta objekta `newRecords`. Šta bi se desilo da smo zaboravili da kloniramo objekat? Nakon reseta `newRecord` objekat i vrednosti u listi bi bile prazne.

Metoda `delete` uklanja element na zadatom indeksu.

Za dodavanje albuma u templejtu koristimo HTML formu, kao što je prikazano listingom ispod.

```
<form (ngSubmit)="save()">

      <label for="input-title">title: </label>
            <input name="input-title" id="input-title"
                  [(ngModel)]="newRecord.title"/><br/>

      <label for="input-author">author: </label>
            <input name="input-author" id="input-author"
                  [(ngModel)]="newRecord.author"/><br/>

      <label for="input-image-url">imageUrl: </label>
            <input name="input-image-url" id="input-image-url"
                  [(ngModel)]="newRecord.imageUrl"/><br/>

      <input type="submit" value="save"/>

</form>
```

Listing – forma za dodavanje albuma

Kako ćemo objekat `newRecord` popuniti vrednostima iz forme? Za to koristimo *dvosmerno vezivanje podataka*. Postavivši na polje za unos `[(ngModel)]="newRecord.title"` zadali smo da se sve vrednosti koje se unose u to polje automatski postave u `newRecord.title`, a i da se svaka izmena vrednosti `newRecord.title` automatski prikazuje u tom polju za unos. Isti mehanizam dvosmernog vezivanja iskoristili smo i za postavljanje autora i urla slike. Obratite pažnju na sintaksu dvosmernog vezivanja: ngModel je prvo obuhvaćen običnim zagradama, a zatim uglastim zagradama.

Ostalo je još da „uhvatimo“ događaj slanja sadržaja forme i da, kao reakciju na njega, postavimo poziv funkcije `save`. Za takve slučajeve koristi se vezivanje događaja. U ovom slučaju događaj je `ngSubmit`, a vezivanje događaja ostvaruje se obuhvatanjem događaja u obične zagrade i postavljanjem funkcije: `(ngSubmit)="save()"`.

Listingom ispod dat je HTML kod brisanja albuma.

```
<li *ngFor="let record of records; let i = index">

      ...
      <button (click)="delete(i)">remove</button>
</li>
```

Za brisanje albuma, pored svakog albuma postavili smo dugme koje ga briše. Na ovo dugme smo postavili reakciju na klik vezivanjem događaja - `(click)="delete(i)"`. Obratite pažnju da smo indeks albuma za brisanje prosledili funkciji delete zahvaljujući

činjenici da indeks trenutne iteracije možemo preuzeti u `*ngFor` direktivi pomoću `let i = index`.