Univerzitet u Istočnom Sarajevu
Elektrotehnički fakultet
Predmet: Internet tehnologije i programiranje

Angular 5

Seminarski rad

Student
Ognjen Kezunović, 1357

Predmetni nastavnik
Doc. dr Danijel Mijić

Istočno Sarajevo, *decembar, 2018.* godine

Sadržaj

1. Uvod

Veb aplikacije predstavljaju programska rješenja koje se izvršavaju na klijentskoj strani, tj. u veb čitaču, uz pristup internet, koje danas sve vise liče na desktop aplikacije. Dostupne su u bilo koje vrijeme, sa bilo kojeg mjesta, sa računara ili mobilnog telefona. Korišćenje istih je danas široko rasprostranjeno, od prostih aplikacija kao sto su veb stranice samo sa tekstom koje imaju opisni sadržaj, oglasa, ponuda, prodaje, do onih mnogo složenijih kao što su društvene mreže, igre, poslovni informacioni sistemi, itd.

Jedana od najpoznatijih platformi za izradu dinamičkih veb aplikacija je Angular, koji kroz koncepte komponenti i modula daje elegantnu tehniku za složene klijentske aplikacije. Napisan je u programskom jezku TypeScript, a prevodi se u JavaScript.

Postoji nekoliko bitnih razlika izmedju radnog okvira i platforme. Radni okvir je obično samo biblioteka kodova koja se koristi za izradu aplikacije, dok je platforma cjelina sastavljena iz vise dijeova i uključuje alate i podršku izvan radnog okvira. AngularJS je bio fokusiran isključivo na izradu veb aplikacija u pretraživaču i jasno je da je on okvir. Angular se isporučuje sa manjom bibliotekom i može uključivati dodatne funkcije dostupne u paketima koji se mogu koristiti po potrebi. Takodje, ima mnogo alata koji ga izdvajaju od ovkira, uključujući sljedeće:

- CLI za razvoj, testiranje i primjenu
- offline mogućnosti za renderovanje na mnogim platformama servera
- okruženje za izradu aplikacija namjenjenih za pretraživač, mobilni ili desktop računare

2. Uopšteno o Angular-u

Slika 1. Logo Angular-a

Angular je moderna platforma za veb aplikacije koja omogućava programerima izradu kompleksnih i robusnih aplikacija uz pomoć sveobuhvatnog skupa alata. Razvijena je od strane Google-a, pisana u jeziku TypeScript. Osnovna ideja je da se omogući izrada aplikacija koje rade na skoro svim platformama: mobilnim, veb ili desktop. Angular je mnogo vise od radnog okvira (eng. *framework*).

Nije lako napraviti veb aplikaciju koja zadovoljava sve zahtjeve korisnika. Postaju sve složenije i zahtjevnije, a korisnici očekuju kvalitetan i pouzdan proizvod. Zato je ova platforma tu da bi pomogla programerima da konačni proizvodi ispunjavaju sve zahtjeve. Ona predstavlja mnogo vise od JavaScript biblioteke koja upravlja nekim od najboljih sajtova na svijetu.

Miško Hevery je 2009. godine najavio AngularJS koji je postao jedan on najpopularnijih okvira za izgradnju veb aplikacija. Projekat je predstavljen u Guglu, a varzija 1.0 je zvanično pokrenuta u oktobru 2010. godine. Bilo je dosta zabune oko verzija. Razvojni tim je odlučio da prvu i sve ostale verzije do 2.0 nazove AngularJS. Verzije 2.0, pa na vise, poznate su kao Angular. 2.0 je kompletna prepisana i sve verzije nakon su inkrementalne u odnosu na nju. Navedene su neke:

- Verzija 2.0 je zvanično najavljena u septembru 2014. godine, da bi se pojavila tek nakon dvije godine, u septembru 2016. godine.
- Verzija 4.0.0 je objavljena 23. marta 2017. godine
- Verzija 5.0.0 je objavljena 1. novembra 2017. godine
- Verzija 6.0.0 je objavljena 3. maja 2018. godine
- Verzija 7.0.0 je objavljena 18. oktobara 2018. godine

TypeScript je besplatan jezik otvorenog koda, razvijen i održavan od strane Microsoft-a. Sintaksa mu je strogi nadskup JavaScript-a. TypeScript podržava pisanje programskog koda u JavaScript-u, korišćenje biblioteka za JavaScript i pozivanje TypeScript koda iz samog JavaScript-a. Drugim riječima, svaki JavaScript program je zapravo i TypeScript program. TypeScript se pomoću specijalnog programa, tzv. *transpliter-a* prevodi u čisti i jednostavan JavaScript programski kod koji se lako pokreće u bilo kojem internet čitaču i na bilo kojem JavaScript pokretaču (eng. *engine*) koji podržava standard ECMAScript 3 ili noviji. TypeScript se koristi za razvoj klijentskih JavaScript aplikacija, ali i serverskih (Node.js) aplikacija. Njegove glavne prednosti su to što je objektno orijetisan i to što su tipovi svih varijabli poznati u vrijeme prevođenja. Na taj način transpliter može otkriti veliki broj trivijalnih grešaka već u najranijim fazama razvoja. Ova svojstva olakšavaju razvoj srednjih i velikih aplikacija. S obzirom da je JavaScript podskup TypeScript-a, programeri dobijaju dodatne mogućnosti kombinovanjem elemenata ta dva programska jezika. TypeScript takođe donosi mnoge mogućnosti standarda ECMAScript, kao što su lambda funkcije, moduli i klase.

U sljedećem kodu (Kod 1) je dat jednostavan primjer koda u TypeScript-u.

```
class Person {
    name: string;
    constructor (message: string) {
        this.name = message;
    }
    printName() {
        return "Hello, " + this.name;
    }
}
```

Kod 1. Primjer koda TypeScript-a

Ovdje je prikazana deklaracija klase *Person*, te eksplicitno navođenje tipa varijabli *name* i *message*. Deklaracije funkcija ne zahtijevaju ključnu riječ *function.*

Odgovarajući JavaScript kod (Kod 2):

```
var Person = (function () {
    function Person(message) {
        this.name = message;
    }
    Person.prototype.printName = function () {
        return "Hello, " + this.name;
    };
    return Person;
})();
```

Kod 2. Odgovarajući JavaScript kod za kod TypeScript-a iz koda 1

Uvođenjem klasa u TypeScript došlo je do potrebe za dopunjavanjem i mijenjanjem postojećih JavaScript klasa (eng. *class*) i članova klasa novim mogućnostima. Upravo to su omogućili dekoratori (eng. *decorator*). Dekorator je poseban oblik deklaracije koji se može pridružiti deklaraciji klase, metode, funkcije za pristup svojstvu (eng. *accessor*), svojstva (eng. *property*) ili parametara. Dekoratori imaju oblik **@izraz**, gdje je **izraz** funkcija koja će se pozvati tokom izvođenja aplikacije. Dekoratori primjenjeni na različite deklaracije unutar klase se primjenjuju sljedećim redoslijedom:

1. dekoratori parametara, metoda, funkcija za pristup svojstvu i svojstava za članove instance (eng. *instance member*);

2. dekoratori parametara, metodam funkcija za pristup svojstvu i svojstava za statičke članove;

3. dekoratori parametara za konstuktor;

4. dekoratori klase su primijenjeni na klase.

Dekorator svojstva deklariše se neposredno prije deklaracije svojstva. Izraz dekoratora pozvaće se kao funkcija, sa dva argumenta:

1. konstruktor klase za statičke članove, ili prototip klase za članove instance;

2. ime člana.

```
function ReadOnly(target: any, key: string) {
  Object.defineProperty(target, key, { writable: false });
}

class Test {
  @ReadOnly
  name: string;
}

let t = new Test();
t.name = 'myName';
console.log(t.name); // 'undefined'
```

Kod 3. Primjer koda za dekorator svojstva

Na prethodnoj slici, varijabli *name* je pridružen decorator *ReadOnly* i na taj način onemogućeno pridruživanje nove vrijednosti toj varijabli, tj. ispisaće se *undefined*.

Dekorator klase deklariše se neposredno prije deklaracije klase. Dekorator klase primjenjuje se na konstruktor klase, a koristi se za proučavanje, modifikaciju ili zamjenu definicije klase. Izraz decorator klase će biti pozvan sa konstruktorom dekorisane klase kao jedinim parametrom. Ako decorator klase vrati vrijednost, ona ce se zamijeniti dekoracijom klase sa pridruženim konstruktorom. Velike su primjene dekoratora klase u praksi.

Dekorator metode deklariše se neposredno prije deklaracije metode. Primjenjuje se na opis svojstva (eng. *property descriptor*) metode. On se koristi za proučavanje, modifikaciju ili zamjenu definicije metode. Izraz dekoratora metode pozvaće se kao funkcija, sa tri parametra:

1. konstruktor klase za statičke članove, ili prototip klase za članove instance;

2. ime metode;

3. novi opis svostva metode.

Dekorator funkcije za pristup svojstvu deklariše se neposredno prije deklaracije funkcije za pristup svojstvu. Primjenjuje se na opis svojstava funkcije za pristup svojstvu, te prati, modifikuje ili mijenja definiciju funkcije za pristup svojstvu. Izraz dekoratora pozvaće se kao funkcija, sa tri argumenta:

1. konstruktor klase za statičke članove, ili prototip klase za članove instance;

2. ime člana;

3. novi opis svojstva člana.

Ne mogu se dekorisati i get i set funkcije za pristup svojstvu istog člana. Umjesto toga, svi dekoratori članove moraju biti primjenjeni na prvu funkciju za pristup svojstvu u dokumentu.

Dekorator parametra deklariše se neposredno prije deklaracije parametra. Primjenjuje se na konstruktor klase ili deklaracije metode. Izraz dekoratora pozvaće se kao funkcija, sa tri argumenta:

1. konstruktor klase za statičke članove, ili prototip klase za članove instance;

2. ime metode;

3. indeks parametra u listi parametara funkcije.

Povratna vrijednost dekoratora parametra se ignoriše, tj. decorator parametra može se koristiti samo za proučavanje parametra.

```
function logPosition(target: any, propertyKey: string, parameterIndex:
    number) {
  console.log(parameterIndex);
}

class TestClass {
testFunction(firstParam: string, @logPosition secondParam: boolean) {
    console.log(firstParam);
  }
}

new TestClass().testFunction('hello', false);
// output: 1 (newline) hello
```

Kod 4. Primjer koda za dekorator parametra

U primjeru iz koda 4, parametru *secondParam* je pridružen decorator *logPosition* koji ispisuje poziciju tog parametra u listi parametara funkcije. Može se viditi da će se decorator pozvati prije tijela funcije.

3. Priprema okruženja

Prvi korak instalacija paketa **Node.js** i **npm (Node Package Manager)**, zatim **Angular CLI** i **TypeCsript**.

Node i npm paketi su esencijalni za razvoj modernih veb aplikacija u Angular-u, ali i na drugim platformama. Node je potreban za izgradnju (eng. *build*) aplikacije, a paket npm je zadužen za instalaciju dodatnih JavaScript biblioteka.

Angular zahtijeva verziju Node-a v4.0.0 i veću, a namjanja potrebna verzija npm-a je 3.0.0. Verzije je moguće provjeriti u terminal ili konzoli, pokretanjem naredbi *node –v* i *npm –v*.

Prvo instalirati Node. Dostupan je na sljedećoj stranici https://docs.npmjs.com/getting-started/installing-node.

Nakon instalacije Node-a, potrebno je instalirati Angular CLI. To je alat koji kreira inicijalni Angular projekt sa svim komponentama postavljenim.

Nakon instalacije Node-a pokrenuti u CMD-u ili Terminalu naredbu: **npm install -g @angular/cli**

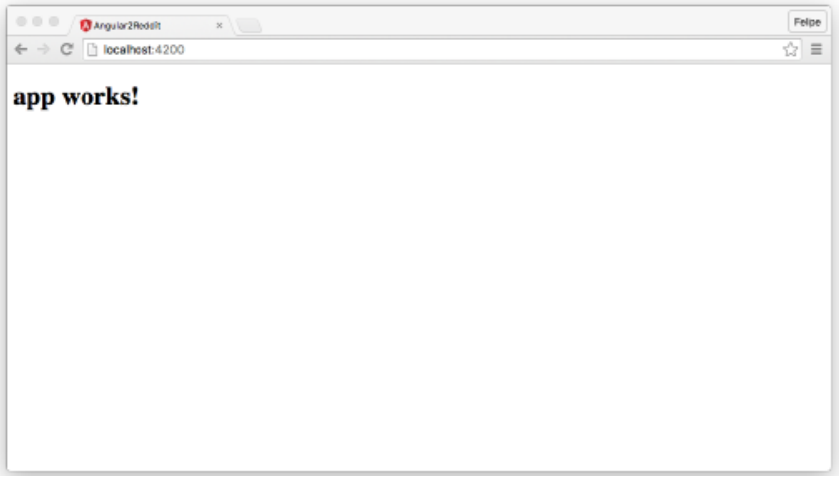
Inače, ng je skraćenica koja se često koristi, a označava zapravo "Angular". Za instaliranje TypeScript-a napisati naredbu: **npm install -g typescript**

Kroz komandnu liniju (CMD) ili Terminal, pozicionirati se željeni folder za kreiranje Angular projekta. Korišćenjem Angular CLI za kreiranje projekta, dovoljna je sljedeća naredba: **ng new ime-projekta**

Prebaciti se u projekat koji je kreiran naredbom: **cd ime-projekta.**

Nakon toga, potrebno je pokrenuti projekt sljedećom naredbom: **ng serve**

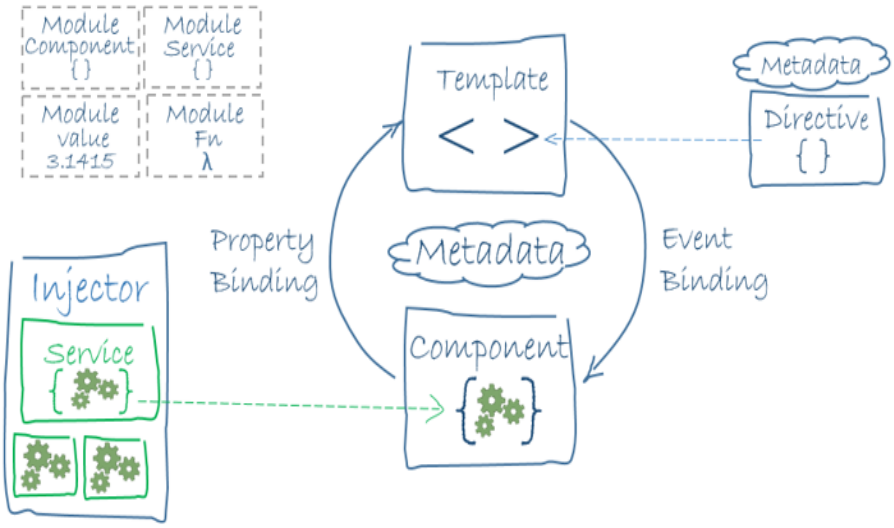
Nakon što se sve odradi, ispisaće se poruka tipa: "**NG Live Development Server is running on http://localhost:4200.**". Dovoljno je otvoriti veb čitač i u URL upisati http://localhost:4200 i otvorice se inicijalna stranica sa sljedeće slike.



Slika 2. Prikaz defaultne stranice kada se pokrene file index.html

4. Arhitektura Angular-a

Na slici 3 je prikazana arhitektura Angular-a.

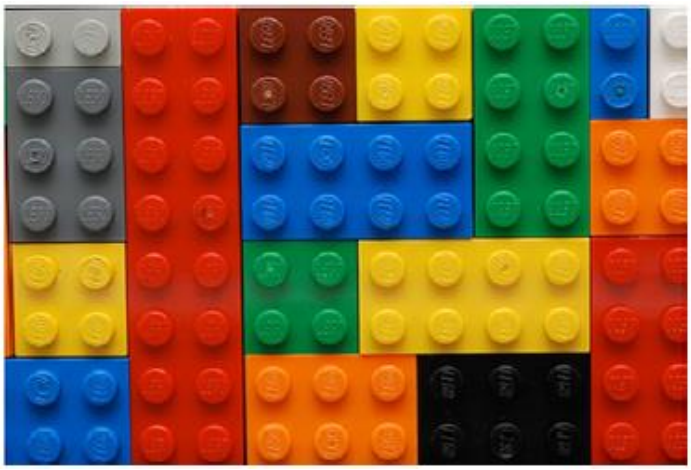


Slika 3. Arhitektura Angular-a

4.1 Moduli

Moduli

Aplikacije u Angular-u su modularne i Angular ima svoj sistem modula (eng. *modularity system*) koji se naziva *NgModules*. Svaka aplikacija sazdrži barem jednu klasu modul (eng. *module class*) koji nazivamo korijenski modul (eng. *the root module*), u kodu obično nazvan *AppModule*. On je dovoljan samo za jako male aplikacije. Veće aplikacije sa više mogućnosti sadrže veći broj modula, a svaki taj modul predstavlja blok koda posvećen jednoj funkciji (karakteristici, opciji) aplikacije. Slika 4 ilustruje izgled cjeline sastavljenje od modula.



Slika 4. Moduli su kao softverski Lego blokovi

Modul je klasa koja ima *@NgModule* decorator. *NgModule* je dekoratorska funkcija koja uzima jedan metapodatak objakat (eng. *metadata object*) čija svojstva opisuju modul.

Neki od najvažnijih svojstava kojima opisujemo module su:

- *declarations* – klase pogleda (eng. *view classes*) koje pripadaju modulu. Angular ima tri tipa takvih klasa pogleda, a to su komponente (eng. *components*), direktive (eng. *directives*) i *pipes*,
- *exports* – podskup deklaracija koje bi trebale biti vidljive i upotrebljive u šablonima komponenata (eng. *component templates*) iz drugih modula,
- *imports* – moduli čije su izvozne klase (eng. *exported classes*) potrebne šablonima komponenata deklarisanih u ovom modulu,
- *providers* – popis usluga (eng. *servoces*) koje ovaj modul pruža. Te usluge postaju dostupne svim dijelovima aplikacije,
- *bootstrap* – glavni pogled aplikacije, koji se naziva korijenska komponenta (eng. *the root component*). On sadrži sve ostale poglede aplikacije. Samo glavni modul, tj. *AppModule* smije postaviti ovo svojstvo.

```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
@NgModule({
  imports:      [ BrowserModule ],
  providers:    [ Logger ],
  declarations: [ AppComponent ],
  exports:      [ AppComponent ],
  bootstrap:    [ AppComponent ]
})
export class AppModule { }
```

Kod 5. Primjer modula

Aplikacija se pokreće podizanjem (eng. *bootstrapping*) modula *AppModule*.

```
import { platformBrowserDynamic } from '@angular/platform-browser-
    dynamic';
import { AppModule } from './app/app.module';

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

Kod 6. Podizanje modula AppModule

Aplikacije su najčešće namijenjene izvođenju u veb čitaču, pa se zato ovdje poziva funkcija *bootstrapModule* koja se nalazi unutar modula *platformBrowserDynamic*, i kojoj se kao parametar prosljeđuje glavni modul aplikacije.

Moduli u JavaScript-u

JavaScript takođe sadrži svoj sistem modula za upravljanje kolekcijama objekata. On se u potpunosti razlikuje od sistema modula Angular-a i nije povezan sa njim. U JavaScript-u, svaka datoteka je modul i svi objekti definisani u toj datoteci pripadaju tom modulu. Takav modul deklariše neke objekte javnim (eng. *public*) pomoću riječi *export*, dok pomoću ključne riječi *import* modul pristupa javnim objektima drugih modula.

```
import { NgModule }      from '@angular/core';
import { AppComponent } from './app.component';

export class AppModule { }
```

Kod 7. Primjer modula u JavaScript-u

Biblioteke u Angular-u

Angular dolazi kao kolekcija JavaScript modula. Mogu se posmatrati kao biblioteke (eng. *library*) u drugim programskim jezicima. Ime svake biblioteke u Angular-u započinje prefiskom *@angular*.

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

4.2 Komponente

Osnovna gradivna jedinica Angular aplikacije je komponenta (eng. *component*). Koncept koponente generalno se pojavljuje u softverskom inženjerstvu kao nezavisna cjelina koda koja samostalno obavlja neku funkcionalnost nudeći i zahtijevajući različite interfejse. Konkretno, u slučaju Angular aplikacije, interfejs neke komponente se sastoji od ulaznih svojstava (eng. *input properties*) i izlaznih događaja (eng. *output events*). Komponenta u Angular aplikaciji se pojavljuje u obliku TypeScript klase. Unutar klase se enkapsulira aplikacijska logika koju treba da posjeduje komponenta, te se deklarišu i definišu članovi koje treba da sadrži. Pošto je definisana kao klasa, više instanci te komponente može da se pojavi u različitim dijelovima aplikacije, čime je postignuta višestruka upotreba iste. Mora biti deklarisana unutar modula u kojem se nalazi, pa prije ključne riječi *class* koristi se ključna riječ *export* kako bi se omogućilo uvođenje komponente u druge dijelove aplikacije.

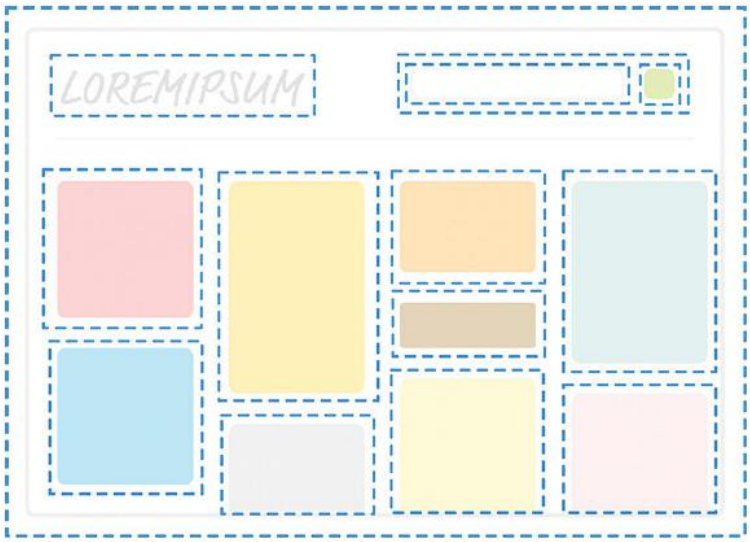
```
@Component({
  selector: 'app-message',
  templateUrl: './app.component.html'
})
export class MessageComp {

  public data: string;

  constructor() {
    this.data = "Hello World";
  }
}
```

Kod 8. Osnovna građa Angular komponente

Angular automatski stvara, osvježava i uništava komponente kako se korisnik kreće kroz aplikaciju. Atribut *selector* sadrži string kojim se određuje kojom će oznakom komponenta biti pozvana unutar neke druge komponente. Sve komponente se uvijek pozivaju iz drugih komponenata, osim ako se radi o korijenskoj komponenti nekog modula. U ovom slučaju, komponentu pozivamo oznakom ***<app-message></app-message>***. Primjer ugnježđavanja komponenti je dat na slici 5.



Slika 5. Ilustracija ugnježđavanja komponenti

4.3 Šabloni

Pogled jedne komponente unutar aplikacije definiše se pomoću njemu pripadnog šablona (eng. *template*). Šablon je kod napisan u HTML-u obogaćen direktivama koje pokazuju Angularu kako da prikaže komponentu.

```
1  <!-- app.component.html -->
2
3  <h2>Title</h2>
4  <p><i>Choose city:</i></p>
5  <ul>
6      <li *ngFor="let city of cities" (click)="selectedCity(city)">
7          {{city.name}}
8      </li>
9  </ul>
10 <city-detail *ngIf="selectedCity" [city]="selectedCity"></city-detail>
```

Kod 9. Primjer šablona

Kao što se vidi u kodu, šablon je u osnovi HTML datoteka, ali ona zapravo proširuje sintaksu HTML-a Angular-ovom sintaksom. U kodu kao primjer Angular-ove sintakse je: **ngFor, {{city.name}}, (click) i <city.detail>*. *selectedCity* je zapravo metoda unutar komponente *AppComponent*, dok je *city* objekat unutar iste komponente. Tag *<city-detail>* je poseban element koji predstavlja Angular-ovu komponentu *CityDetailComponent.*

Angular-ova sintaksa smješta se u HTML elemente. To znači da se unutar jednog HTML fajla miješaju Angular-ova i HTML sintaksa.

4.4 Metapodaci

Unutar Angular-a, pojam metapodaci (eng. *metadata*) označava podatke koji govore Angular-u kako procesirati (obraditi) određenu klasu. Komponente u Angular-u so zapravo obične klase, ali iz definicije te klase nigdje se ne može viditi veza sa Angular-om. Svaka komponenta je samo klasa u programskom jeziku TypeScript, sve dok se Angular-u ne kaže da ona postoji. Kako bi znao da je određena klasa zapravo komponenta, pridružuju joj metapodaci.

```
@Component({
    selector:      'city-detail',
    templateUrl:   './city-detail.component.html',
    providers:   [ MyService ]
})
export class CityDetailComponent  {
  . . .
}
```

Kod 10. Primjer pridruživanja metapodataka

U TypeScript-u, metapodaci pridružuju pomoću dekoratora. Ovdje se uočava decorator *@Component*, koji klasu napisanu neposredno ispod njega označava kao Angular komponentu. Dekoratorska funkcija *@Component* kao parameter prima kofiguracijski objekat (eng. *configuration object*) sa informacijama koje su Angular-u potrebne za stvaranje i prikaz komponente, te njoj pripadajućeg pogleda. Ovo so neke od najvažnijih konfiguracijskih opcija *@Component* dekoratora:

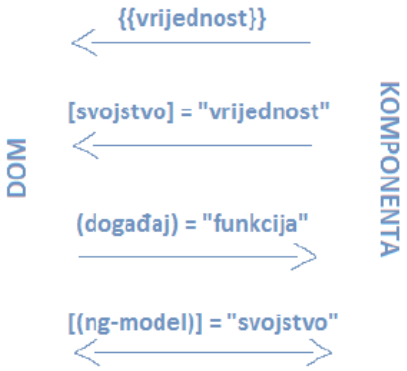
- *selector* – govori Angular-u da stvori i umetne instance ove komponente na svakom mjestu unutar pripadajuće HTML datoteke na kojem pronađe odgovarajući tag (<*city-detail*></*city-detail*>).
- *templateUrl* – relativna adresa šablona ove komponente (*'./city-detail.component.html'*).
- *providers* – niz pružalaca usluga za usluge koje su potrebne ovoj komponenti.

Slika 6. Pogled

Šablon, metapodaci i komponenta zajedno opisuju pogled (slika 6).

4.5 Povezivanje podataka

Jedan od osnovnih principa koji se veže uz samu komponentu unutar Angular-a je vezivanje podataka (eng. *data binding*). Pošto komponenta ima svojstvo višestruke upotrebe, unutar Angular aplikacije može postojati mnoštvo instanci iste komponente.



Slika 7. Povezivanje podataka

```
1 <app-message></app-message>
2 <app-message></app-message>
3 <app-message></app-message>
```

Kod 11. Višestruko korišćenje iste komponente

U ovom slučaju je važno naglasiti da je svaka od tri instance *MessageComp* komeponente dijete instance *AppComp*, a *AppComp* njihov roditelj (komponente čine stablo). Komponenta se može posmatrati kao kontejner koji će na određeni način, unutar svog područja djelovanja (opsega, eng. *scope*), prikazati podatke i omogućiti njihovu manipulaciju. Postoje dvije vrste vezivanja podataka:

- vezivanje ulaznih svojstava,
- vezivanje izlaznih događaja koji čine interfejs komponente.

Da bi se demonstrirao ovaj princip, potrebni su podaci, pa je unutar roditeljske komponente *AppComp* potrebno deklarisati i inicijalizovati polje poruka.

```
1   messages: string[] = [
2     "Hello World",
3     "Have a nice day",
4     "Nice to see you"
5   ];
```

Kod 12. Polje poruka

Ovaj podatak je vidljiv samo unutar opsega *AppComp* komponente. Sa njim se može manipulisati unutar klase, ali i unutar prikazane komponente. Ukoliko se šablon *AppComp* komponente nadogradi nizom ***{{ messages | json }}***, tada će unutar veb čitača biti prikazano to polje.

Ovaj niz šablona sadrži specifični Angular kod – dvostruke vitičaste zagrade označavaju da se radi o umetanju stringa, a vertiklna linija, pomoću mehanizma **filtera**, formatira polje stringova kako bi se to polje pretvorilo u string. Umetanjem stringa omogućen je prikaz podataka koji se nalaze u opsegu komponente. Još uvijek nisu povezani podaci, ali trenutno postoje podaci koji će biti povezani u druge komponente.

```
1 <app-message
2     *ngFor="let message of messages"
3     [innerMessage]="message">
4 </app-message>
```

Kod 13. Povezivanje podataka od roditelja prema djeci

```
1 @Input('innerMessage') message: string;
```

Kod 14. @Input decorator unutar MessageCompkomponente

Pomoću koda *[innerMessage] = "message"* se povezuju podaci, tačnije povezivanje ulaznog svojstva *message* koje je definisano unutar opsega *MessageComp* komponente. Koristeći decorator *@Input('innerMessage')*, pokazuje se da će unutar komponente *MessageComp* postojati podatak nazvan *message* čiji je nadimak *innerMessage*, i atributu *innerMessage* obavijenom uglastim zagradama unutar oznake *MessageComp* komponente koja se nalazi unutar šablona roditeljske komponente, pridružiti varijabla *message* koja je trenutno lokala varijabla šablona[1] komponente *AppComp*, a nastaje iteracijom polja *messages* koristeći direktivu *NgFor* (o čemu poslije biti riječi). Za sada je važno napomenuti da *NgFor* omogućava instanciranje više komponenti istog tipa prolazeći kroz polje. Postupkom iteracije instancirane su tri komponente *MessageComp*, a svakoj od instance prosljeđuje se, koristeći mehanizam povezivanja podataka, string koji sazdrži poruku. Time je postignuto da roditeljska komponenta sluša, odnosno reaguje na događaj koji je poslat iz nekog od djece. Dijete putem *EventEmitter* objekta šalje podatak, a roditelju je vidljiv taj podatak koji se dalje prosljeđuje u njegov opseg.

```
1 @Output('outerMessage') outputEvent = new EventEmitter<string>();
2
3 onClick() {
4     this.outputEvent.emit("Thank you");
5 }
```

Kod 15. @Output decorator unutar MessageComp komponente

[1] **Lokalna varijabla šablona** je svaka varijabla koja nastaje unutar šablona i to tako da unutar oznake nekog HTML elementa dodamo oznaku # i definišemo ime varijable

```
<app-message *ngFor="let message of messages"
             [innerMessage]="message"
             (outerMessage)="print($event)">
</app-message>
```

Kod 16. Povezivanje podataka od djeteta prema roditelju

Oba postupka predstavljaju jednosmjerno povezivanje podataka. Uz to, postoji i mehanizam za dvosjmerno povezivanje podataka (vidjeti sliku 6) koji je jedini način u prvoj verziji Angular okvira. Počevši od druge verzije, odlučeno je da će uobičajeno povezivanje podataka biti jednosmjerno radi efikasnosti. Istovremeno se koriste uglaste i obične zagrade kako bi se postiglo dvosmjerno povezivanje, ali takav način nije uobičajen u današnje vrijeme, upravo radi efikasnosti.

Projekcija sadržaja

Kako bi komponenta postala još skalabilnija i fleksibilnija, uveden je mehanizam projekcije sadržaja. Tim mehanizmom prosljeđuje se dodatni HTML kod sa ukomponovanim Angular kodom unutar komponnente. Unutar šablona komponente u kojoj je potrebno projektovati sadržaj, dodaje se oznaka ***<ng-content></ng-content>***. Oznaci koja instancira komponentu prosljeđuje se HTML kod koji će biti prikazan na mjestu gdje se ta oznaka nalazi. Moguće je smjestiti i vise oznaka koje predstavljaju neki sadržaj, i tada je potrebno koristiti atribut *select* unutar <*ng-content*> oznake.

```
<ng-content select="h1"></ng-content>
<ng-content select="ul"></ng-content>
```

Kod 17. Korišćenje višestruke projekcije sadržaja

Unutar šablona komponente, tj. u oznaci gdje se ta komponenta poziva, nezavisno od redoslijeda HTML elemenata *h1* i *ul*, prosljeđeni elementi se prikazuju redoslijedom unutar šablona komponente.

Životni vijek komponente

Svaka komponenta, od inicijalizacije, pa sve do unuštenja, prolazi kroz različite faze koje opisuju njeno stanje. U zavisnosti od stanja, moguće je izvršiti sljedeće funkcije:

- *ngOnChanges(),*
- *ngOnINit(),*
- *ngDoCheck(),*
- *ngAfterContentInit(),*
- *ngAfterContentChecked(),*
- *ngAfterViewInit(),*
- *ngAfterVeiwChecked(),*
- *ngOnDestroy().*

Angular u modulu *@angular/core* sadrži interfejse koji se mogu implementirati unutar komponente kako bi se pozvala neka od gore navedenih funkcija prilikom nekog trenutka u životnom vijeku komponente. Npr., ukoliko se naveda da komponenta implementira interfejs *OnInit*, tada se unutar klase komponente definiše metoda *ngOnInit()* koja se pokreće svaki put kada je komponenta inicijalizovana. Takav mehanizam je koristan za pripremu podataka koji se nalaze u komponenti, ali i prilikom uništavanja komponente. Tri najvažnije metode koje se pokreću u životnom vijeku komponente su:

- *ngOnChanges()* – pokreće se svaki put prilikom promjene ulaznih svojstava pri povezivanju podataka,
- *ngOnInit()* – pokreće se nakon prvog *ngOnChanges()* poziva,
- *ngOnDestroy()* – pokreće se prilikom unuštavanja komponente, odnosno njenog uklanjanja iz prikaza neke roditeljske komponente.

4.6 Direktive

Direktiva je mehanizam kojim se mijenja sruktura, ponašanje ili izgled DOM-a. Prethodno opisane komponente su takođe directive – očigledno je da komponente utiču na strukturu DOM-a. Preciznije rečeno, komponenta je direktiva sa šablonom definiše prikaz unutar veb čitača. Razlika između obične direktive i komponente je u tome da komponenta sadrži, uz ukomponovanu logiku koja utiče na njena svojstva, i šablon kojim je definisano šta će unutar pretraživača biti prikazano.

Direktiva je TypeScript klasa koja sadrži članove kojim se utiče na prethodno navedena svojstva DOM-a: strukturu, izgled i ponadašanje. Sadrži selector, kao i komponenta. Koristi se tako da se pridruži oznaci HTML elementa kojim se želi manipulisati. Kako je definisana klasom, ona mora imati svoju deklaraciju i definiciju, a kako bi se naznačilo da se radi o direktivi, koristi se *@Directive* decorator iz *@angular/core* biblioteke. Dijele se u tri kategorije:

- Komponentne direktive
- Strukturalne directive (eng. *structural directives*)
- Atriburne directive (eng. *attribute directives*)

Strukturne direktive

Strukturne direktive mijenjaju strukturu DOM-a. One mogu duplirati, ukloniti ili premjestiti neki element unutar DOM-a, a te se radnje manifestuju na sadržaj unutar pretraživača. budući da elementi unutar DOM-a impliciraju sadržaj unutar pregledača. Predefinisane strukturalne direktive unutar Angular aplikacije su *NgIf, NgFor* i *NgSwitch*.

Pomoću *NgFor* direktive prolazi se kroz JavaScript polje (eng. *array*). Kako bi se moglo pristupiti polju, ono treba da se nalazi unutar opsega komponente unutar čijeg je prikaza pozvana ova direktiva. Dodatne lokalne varijable šablona koje se mogu koristiti prilikom prolaska pomoću *NgFor* direktive su: *index* (označava indeks iteracije), *first* (Booleova varijabla za prvi element u polju), *last* (Booleova varijabla za posljednji element u polju), *even* (Booleova varijabla za paran indeks) i *odd* (Booleova varijabla za neparan indeks).

```
<app-message
    *ngFor="let message of messages"
    [innerMessage]="message">
</app-message>
```

Kod 18. Primjer NgFor direktive

Druga bitna strukturalna direktiva je *NgIf*. U zavisnosti o Booleovoj varijabli, ova direktiva uklanja element kojem je pridružena. Ukoliko je Booleova varijabla istinita, element ostaje sadržan unutar DOM-a, u suprotnom biva uklonjen.

```
<p *ngIf="true">I am in DOM</p>
<p *ngIf="false">I am not in DOM</p>
```

Kod 19. Primjer NgIf direktive

Pomoću *NgSwitch* direktive, isto kao i kod *NgIf*, dinamički se uklanja ili dodaje element kojem je direktiva pridružena. Ukoliko je potrebno provjeriti višestruke uslove, zapis *NgSwitch* direktive može biti ekonomičniji u pogledu broja linija koda.

```
<div [ngSwitch]="12">
    <p *ngSwitchCase="12">12</p>
    <p *ngSwitchCase="13">13</p>
    <p *ngSwitchCase="14">14</p>
</div>
```

Kod 20. Primjer NgSwith direktive

Aributske direktive

Pomoću atributskih direktiva se manipuliše atributima elemenata. Te direktive ne mijenjaju strukturu DOM-a već izgled i ponašanje. Tri ugrađene atributske direktive su *NgStyle*, *NgClass* i *NgNonBindable*.

Pomoću direktive *NgStyle* HTML elementu se pridružuje CSS stil na dinamički način. Ukoliko je potrebno da tekst unutar nekog elementa bude crvene boje, direktiva se poziva sljedećim kodom.

[ngStyle] = “color: ‘red’”

NgClass direktivom se dodaju CSS klase elementu kojem je ova direktiva pridružena.

NgNonBindable direktiva se dodaje unutar oznake elementa kojeg ne treba posmatrati kao element Angular sintakse. Sav sadržaj unutar tog elementa će biti prikazan kao obični tekst (eng. *plain text*).

4.7 Filteri

Ukoliko je potrebno dodatno promijeniti string ili podatak koji sadrži string prilikom interpolacije, ali samo u smislu onoga što je vidljivo unutar pretraživača i pritom ne promijeniti podatak, koriste se filteri. Ipak, ukoliko je potrebano, moguće je i promijeniti podatak na koji se filter primjenjuje.

Prilikom interpolacije, naznačava se da se želi primijeniti filter tako što se nakom imena podatka smjesti vertikalna linija i ime filtera. Može se primijeniti više filtera odjednom, a svakome od njih se mogu prosljeđivati dodatni parametri. U ugrađenom *date* filter prosljeđuje se parametar *'shortDate'*, odnosno ovaj filter se primjenjuje nad podatkom tipa *Date* kao:

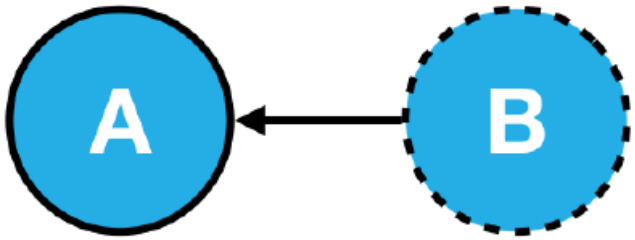
{{ startDate | date: 'shortDate'}},

a za početni string "*Fri Jul 28 2017 14:59:33 GTM+0200 (Central European Daylight time)*", formatirani ispis je *"7/28/2017"*.

Postoji niz korisnih ugrađenih filtera, ko što su: *json* (pretvara JavaScript objekat u string), *async* (void računa o asinhronim podacima), *date* (void računa o format datuma), *currency* (formatira skraćenice nekih valuta), *percent* (za dati broj izmedju 0 i 1 vraća string u obliku procenta) itd.

4.8 Servisi

Servis je instanca neke klase ili vrijednosti nekog tipa podataka dostupna u svim dijelovima aplikacije. Ideja servisa je bazirana nad konceptom ubrizgavanja zavisnosti (eng. *dependency injection*). Kaže seda je dio aplikacije A zavisano o dijelu aplikacije B, odnosno B je zavisan od A, ukoliko je logika koju enkapsulira dio B potrebna za rad dijela A (slika 8).

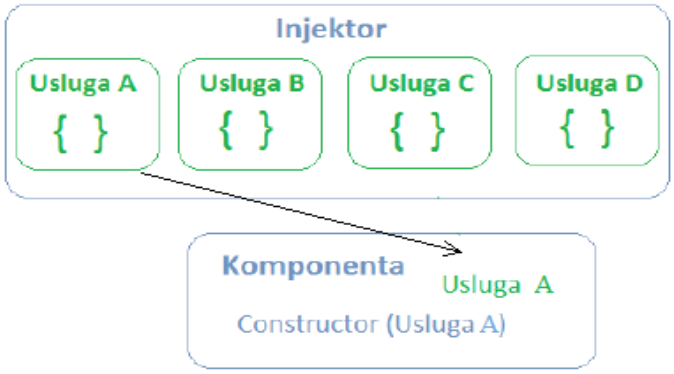


Slika 8. Ilustracija zavisnosti između dijelova aplikacije

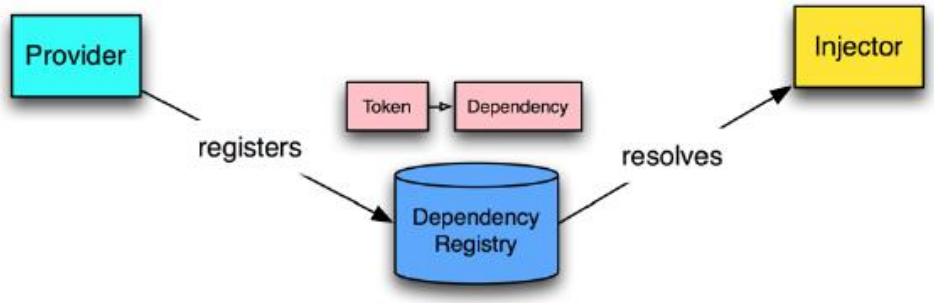
Temeljna ideja u konceptu ubrizgavanja zavisnosti jeste da instancirani objekat koji predstavlja zavisnost proslijediti putem konstruktora klase koja je zavisna. Ubrizgavanje zavisnosti je šablon za oblikovanje (eng. *design pattern*).

4.9 Ubrizgavanje zavisnosti

Angular nudi vrlo fleksibilan način za ubrizgavanje zavisnosti (slika 10). Ova ideja unutar Angulara temelji se na četiri entiteta – token, ubrizgavač (eng. *injector*), snabdjevač (eng. *provider*) i zavisnost.



Slika 9. Injektor



Slika 10. Dijagram ubrizgavanja zavisnosti unutra Angulara

Niz snabdjevača je polje registara svih zavisnosti unutar aplikacije ili dijela aplikacije (u smislu nekog podstabla komponenti), a svaki objekat kojim je definisan snabdjevač sadrži atribut *provide* koji označava jedinstveni identifikator zavisnosti, odnosno token. Drugim riječima, snabdjevač mapira token sa odgovarajućom zavisnošću. Zavisnost se može dohvatiti u obliku instance klase, JavaScript objekta, polja ili varijable. Token može biti string, ime klase (tzv. *type token*) ili instance generičke klase *InjectionToken*. Korišćenje stringa kao tokena se izbjegava jer može doći do kolizije.

Snabdjevač s obzirom na prosljeđeni token definiše rezultat koji će biti ubrizgan. Drugi atribut unutar objekta kojim je definisan snabdjevač može biti *useClass, useExisting, useValue* ili *useFactory*. *useClass* atribut naznačava da će snabdjevač s obzirom na posljedni token kreirati jedinstvenu instance klase koja će biti ubrizgana kao zavisnost ostalim klasama koje je zahtjevaju kao zavisnost. Interno se takva instanca čuva u registru zavisnosti (eng. *dependency registry*) kako bi bila korišćena u raznim dijelovima aplikacije.

Polje snabdjevača je vrijednost atributa **providers** koji se nalazi unutar objekta koji se prosljeđuje *@NgModule, @Component* ili *@Directive* dekoratoru. Drugim riječima, moduli, koponente i direktive mogu imati svoje snabdjevače.

```
providers: [
    {provide: 'api', useValue: api_url},
    {provide: 'value', useValue: 'init'},
    {provide: MainHttpService, useClass: MainHttpService},
    {provide: 'FactoryDependency', useFactory: (value)=>{
        return new ServiceClass(value)
    }, deps: ['value']}
]
```

Kod 21. Polje snabdjevača koje unosimo unutar modula

Mehanizam ubrizgavanja zavisnosti brine o tome da sve zavisnosti sadržane unutar snabdjevača budu registrovane, ali i da zavisnosti mogu biti ubrizgane tamo gdje su potrebne. Za samo ubrizgavanje zavisnosti se brine ubrizgavač koji s obzirom na prosljeđeni parameter koji predstavlja token zavisnosti, unutar registra zavisnosti putem snabdjevača dohvaća instancu ili vrijednost zavisnosti i ubrizgava je. Ubrizgavač se pojavljuje u obliku dekoratora *@Inject*, a smješta se prije parametra koji označava zavisnost unutar konstruktora klase koja sadrži zavisnost. Prethodno je registrovan snabdjevač sa tokenom *'api'* koji snabdijeva vrijednošću *api_url*. Tu vrijednost je potrebno ugrizgati unutar *MainHttpService* klase, pa se deklariše parametar unutar konstruktora i deklariše se sa *@Inject* dekoratorom:

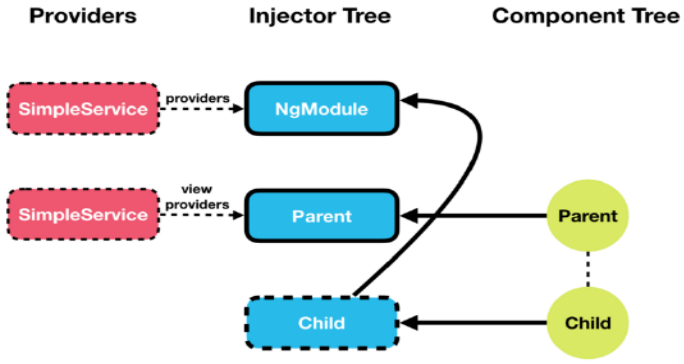
```
constructor(@Inject('api') private api_url: string) { }
```

Unutar neke komponente koja se brine o korisnikovoj interakciji, odnosno o tome da se podaci prilikom korisnikove interakcije dohvate sa servera, ubrizgavamo *MainHttpService* instance:

```
constructor(private mainHttpService: MainHttpService) { }
```

Drugi decorator uz *@Inject* kojim se označava ubrizgavanje zavisnosti je *@Injectable*. Ovim dekoratorom se označava da se želi ubrizgati zavisnost čiji je token jednak imenu klase.

Posebna vrsta snabdjevača je *ViewProvider* koji se odnosi isključivo na komponente. Ukoliko je neki snabdjevač sadržan nad komponentom kao *viewProvider*, zavisnost koja je definisana snabdjevačem neće biti dostupna onoj djeci koja su kreirana projekcijom sadržaja. Ipak, takva djeca se još uvijek nalaze unutar stable komponenti, pa ukoliko postoji snabdjevač sa istim tokenom registrovan u čvoru koji je predak roditeljske komponente, druga instanca iste zavisnosti postaje dostupna.



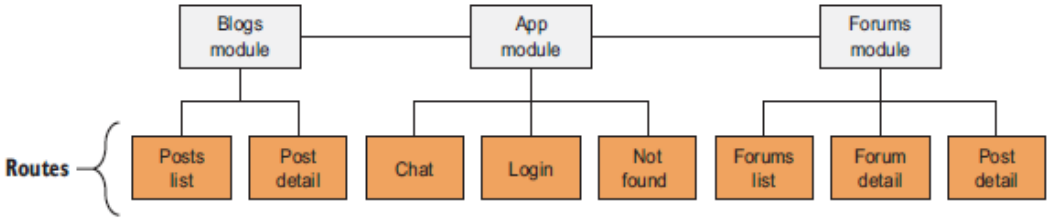
Slika 11. Različite instance zavisnosti iste klase sa različitim načinima snabdijevanja

4.10 Rutiranje

Mnoge aplikacije zahtjevaju mogućnost navigacije između različitih stranica tokom životnog ciklusa aplikacije. Dio Angular-a koji se bavi time je usmjerivač (eng. *router*). Često aplikacija ima nekoliko osnovnih stranica, kao što su stranica za prijavljivanje, početna stranica, stranica korisničkog naloga itd. Ruting je termin koji se koristi za opisivanje mogućnosti aplikacije za promjenu sadržaja stranica u kojoj se korisnik kreće. Posmatrajući navigaciju veb čitača:

- unosom URL adresa veb čitač otvara veb stranicu na toj adresi;
- pritiskom na neki link unutar veb stranice pretraživač otvara odgovarajuću veb stranicu;
- pritiskom na tipku *nazad* ili *naprijed* unutar pretraživača, on vodi kroz istoriju posjećenih veb stranica.

Na slici 12 su ilustrovane rute.



Slika 12. Primjer ruta definisanih za različite module

Na primjer, fiksna komponenta koja služi za promjenu sadržaja unutar pretraživača je navigacija. Unutar navigacije su smješteni linkovi ili dugmići koji na neki događaj generišu novi prikaz. Ova ideja je ugrađena unutar Angular-ovog *RouterModule* modula.

Potrebno je definisati rute kojima će se dinamički stvarati prikaz unutar pretraživača, odnosno, kretati se unutar aplikacije. O kretanjima unutar aplikacije brine se usmjerivač.

```
1 const routes: Routes = [
2     {path: '', redirectTo: 'home', pathMatch:'full'},
3     {path: 'home', component: HomeComponent},
4     {path: 'about', component: AboutComponent},
5     {path: 'products', component: ProductsComponent},
6     {path: '**', redirectTo: 'home'}
7 ];
```

Kod 22. Poljem ruta se navodi kuda je zamišljeno kretanje

U kodu 22 je prikazano polje ruta koje definiše kretanje kroz prikazane komponente *HomeComponent, AboutComponent* i *ProductComponent*. Za praznu rutu (*path: ‘’*) ili bilo koju drugu rutu koja nije definisana unutar polja ruta (*path: ‘**’*) prikazuje se prikaz *HomeComponent* komponente.

Polje rutra tipa *Routes*, gdje je svaka ruta predstavljena objektom, prosljeđuje se statičkoj metodi *RouterModule.forRoot* koja instancira modul koji sadrži sve direktive, rute i servise potrebne za rad sa usmjerivačem. Instanca *RouterModule* klase unosi se unutar glavnog modula.

```
1 <nav>
2     <ul>
3         <li><a [routerLink]="['home']" routerLinkActive="active">Home</a
              ></li>
4         <li><a [routerLink]="['about']" routerLinkActive="active">About</
              a></li>
5         <li><a [routerLink]="['products']" routerLinkActive="active">
              Products</a></li>
6     </ul>
7 </nav>
8 <router-outlet></router-outlet>
```

Kod 23. Dodjeljivanje ruta komponentama

U prethodnom kodu, unutar prikaza u kojem je potrebno da se prikazi komponenata registrovanih unutar ruta prikazuju, ubačena je *RouterOutlet* direktiva, a na istom prikazu se može definisati i navigacija.

Svaki link sadrži odgovarajuću direktivu *RouterLink* čijem selektoru se prosljeđuje polje koje sadrži put do komponente unutar usmjerivača, ali može da sadrži i dodatne parametre. *RouterLinkActive* direktiva odgovarajućem linku koji vodi do prikaza koji je aktivan pridružuje CSS pseudo-klasu *'active'* kako bi se dodatnim kodom moglo naglasiti na kojem prikazu je pozicioniran korisnik.

Sljedeći kod predstavlja izlistavanje svih proizvoda unutar prikaza komponente *ProductsComponent*.

```
1 <ul>
2   <li *ngFor="let product of products">
3     <a [routerLink]="['/product', product.id]">
4       {{product.name}}
5     </a>
6   </li>
7 </ul>
```

Kod 24. Primjer izlistavanja svih proizvoda unutar prikaza komponente

Svaki element liste unutar prikaza sadrži link do proizvoda. U ovom kodu je naznačeno da se putem usmjerivača prikazuje prikaz *ProductComponent* komponente, pritom prosljedivši identifikator proizvoda kako bi on mogao biti dohvaćen.

Ukoliko je potrebno da se definiše usmjerivač unutar usmjerivača, koristi se atribut *children* unutar objekta koji predstavlja rutu, a vrijednost tog atributa je novo polje ruta.

5. Zaključak

Ovaj seminarski rad opisuje platformu za razvijanje aplikacija na klijentskoj strani. Angular je jedan od najpopularnijih platformi za razvijanje aplikacija koje se pokreću u veb čitaču. U radu su navedeni primjeri koda jezika TypeScript i razlike između njega i JavaScript-a, od koga je nastao i u kojeg se prevodi. Opisane su komponente, osnovne gradivne jedinice platforme koja je tema rada. Angular je pogodan, kako za razvijanje jednostavnih, tako i onih komplikovanih aplikacija. Odličan je za one koje su CRUD prirode, dok nije baš najbolji za razvijanje igrica, ali time se bave neke druge platforme i okviri. Alikacije se pokreću na serverima node.js, .NET, PHP i drugim, a iscrtavaju se u veb čitaču koristeći samo HTML i CSS.

Danas je veliki broj veb aplikacija razvijenih u Angular-u koji savršeno funkcionišu u veoma brzo se izvršavaju. Zavisan se od JavaScript-a i TypeScrypta i nije ga teško savladati uz elementarna znanja navedenih uz kombinaciju HTML-a i CSS-a. Ima široku primjenu i u mobilnim tehnologijama. Sastoji se od komponenata koje se ugnježdavaju u stablo, a skup komponenata koje čine logičku cjelinu smješteni su u module. Za manipulaciju prikaza brinu se direktive, dok se ubrizgavanjem zavisnosti snabdjevaju različiti dijelovi aplikacije objektima koji su potrebni za njihov rad.

6. Literatura

[1] W. Jeremy, *Angular in Action*, Manning, Shelter Island Publication, 2018.

[2] C. Mark, *Angular 5 Projects: Learn to Build Single Web applications Using 70+ Projects*, Appress Media LLC, 2018.

[3] M. Nate, C Felipe, L. Ari, T. Carlos, *ng-book: The Complete Guide to Angular*, Fullstack.io, 2018.

[4] A. Hussain, *Angular 4: From Theory to Practice*, Daolrevo Ltd. 2017.

[5] Veb stranica Angular-a, https://angular.io, posjećena 8.12.2018.

[6] Veb stranica TypeScript-a, https://www.typescriptlang.org/, posjećena 8.12.2018.

[7] Veb stranica AngularJS-a, https://angularjs.org/, posjećena 8.12.2018.

7. Dodaci

7.1 Slike

7.2 Kodovi