

Seminarski rad iz predmeta: "Informacione tehnologije i društvo"

Tema: Angular

Ana Zrnić (56/20)

23. April, 2021. - 26. April, 2021.

1 Uvod

1.1 Šta je to Angular?

Angular je Google-ov open-source web framework, zasnovan na TypeScript-u. Često još prepoznatljiv pod nazivom Angular 2+, nastao je nakon potpunog redizajna AngularJS-a, koji je nailazio na mnogo problema te je odlučeno da se ova dva framework-a razdvoje. Glavna razlika jesu primarni jezici koji se koriste, AngularJS je u potpunosti zasnovan na JavaScript jeziku, što znači da su učenje i razvijanje bili lakši i jednostavniji. Prelaskom na Typescript (open source jezik izgrađen na JavaScript-u) omogućeno je sprečavanje grešaka u kodu koje bi inače prošle nezapaženo. TypeScript također sadrži klase i dekoratore, pogodne za definisanje komponenti, koji su srž Angular ekosistema, kao i mnoge druge opcije koje će ovaj rad preći.

1.2 Verzije

Tim koji radi na razvoju Angular-a garantuje da će se u prosjeku verzije ovog framework-a ažurirati dva puta godišnje. Nakon objavljivanja 2014. godine, Angular je prošao kroz 9 ažuriranja i od Novembra 2020. je trenutno na verziji 11. Svaka verzija se podržava samo 18 mjeseci i njegovi korisnici moraju biti u toku sa svim promjenama.

VERSION	STATUS	RELEASED	ACTIVE ENDS	LTS ENDS
^11.0.0	Active	Nov 11, 2020	May 11, 2021	May 11, 2022
^10.0.0	LTS	Jun 24, 2020	Dec 24, 2020	Dec 24, 2021
^9.0.0	LTS	Feb 06, 2020	Aug 06, 2020	Aug 06, 2021

Angular versions v4, v5, v6, v7, and v8 are no longer under support.

Slika 1: podaci sa zvaničnog Angular sajta o statusu podržavanja posljednjih verzija

2 Postavljanje okruženja

2.1 Potrebno predznanje

Prije prvih koraka bitno je naglasiti da se pretpostavlja da čitalac poznaje:

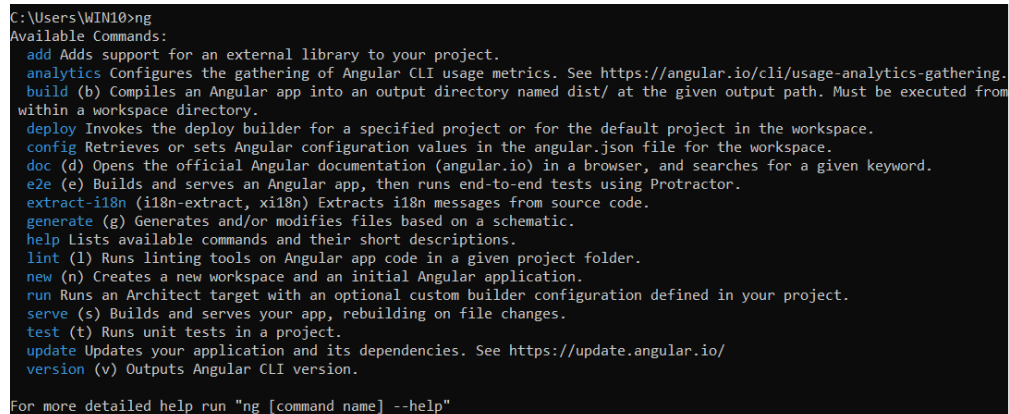
- Html
- CSS
- JavaScript
- klase
- module
- (ne nužno) TypeScript

Ova napomena je potrebna jer Angular nije "beginner-friendly" i njegova kriva učenja je veoma strma (što je autor otkrio na teži način).

2.2 Potrebni alati za rad

Kako bismo napravili prve korake potrebno je prije svega da imamo instalirane posljednje verzije:

- node.js
- npm, odnosno, node package manager (koji će biti instaliran u sklopu node.js instalacije)
- IDE ličnog izbora (npr. VS Code)
- Angular CLI (command line interface)
napomena: instaliranje se vrši u command prompt-u komandom: `npm install -g @angular/cli`



```
C:\Users\WIN10>ng
Available Commands:
  add Adds support for an external library to your project.
  analytics Configures the gathering of Angular CLI usage metrics. See https://angular.io/cli/usage-analytics-gathering.
  build (b) Compiles an Angular app into an output directory named dist/ at the given output path. Must be executed from
within a workspace directory.
  deploy Invokes the deploy builder for a specified project or for the default project in the workspace.
  config Retrieves or sets Angular configuration values in the angular.json file for the workspace.
  doc (d) Opens the official Angular documentation (angular.io) in a browser, and searches for a given keyword.
  e2e (e) Builds and serves an Angular app, then runs end-to-end tests using Protractor.
  extract-i18n (i18n-extract, x18n) Extracts i18n messages from source code.
  generate (g) Generates and/or modifies files based on a schematic.
  help Lists available commands and their short descriptions.
  lint (l) Runs linting tools on Angular app code in a given project folder.
  new (n) Creates a new workspace and an initial Angular application.
  run Runs an Architect target with an optional custom builder configuration defined in your project.
  serve (s) Builds and serves your app, rebuilding on file changes.
  test (t) Runs unit tests in a project.
  update Updates your application and its dependencies. See https://update.angular.io/
  version (v) Outputs Angular CLI version.

For more detailed help run "ng [command name] --help"
```

Slika 2: nakon instalacije kucanjem komande `ng` mozemo da vidimo listu drugih komandi, odnosno alata, koji su nam na raspolaganju za olakšan rad

2.3 Prvi koraci

Nakon što smo utvrdili sve prethodne tačke, vrijeme je za kreiranje samog razvojnog okruženja. Angular CLI nam dozvoljava da na brz način pravimo nove projekte, kucanjem komande:

ng new naziv-projekta

Postavljanje se čeka par minuta i nakon toga u željenom direktorijumu možemo naći novi folder sa imenom projekta. Otvaranjem foldera u našem IDE-u možemo da vidimo sadržaj istog. Kreirano je nekoliko npm paketa i startni projektni fajlovi. Ovdje se nalaze svi dijelovi naše stranice, uključujući i *index.html* fajl koji je onaj krajnji prikazan klijentu. Angular unaprijed instalacijom napravi placeholder sajt. Kako da mu pristupimo?

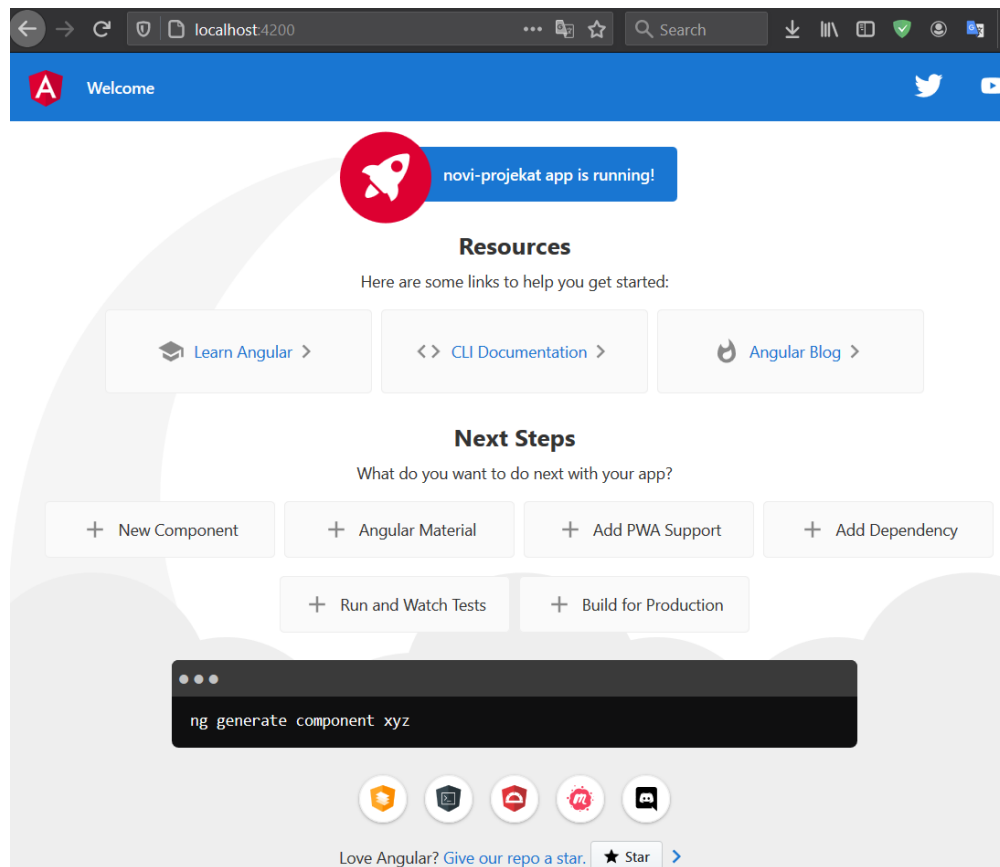
U terminalu provjerimo da li se prvo nalazimo u direktorijumu kreiranog projekta i onda korištenjem sljedeće angular CLI komande startamo lokalni server program:

ng serve

Nakon toga dobijamo poruku koja nas obavještava da je naša stranica aktivna na sljedećem linku:

`http://localhost:4200/`

I zaista kada otvorimo taj link dočeka nas web stranica prikazana na sljedećoj fotografiji.



Slika 3:

Kao što je već spomenuto, upravo ovaj sajt koji vidimo nalazi se u `src/index.html` fajlu, kada ga otvorimo prikaže se sljedeći kod:

Primjer 1 src/index.html

```
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <title>NoviProjekat</title>
6   <base href="/">
7   <meta name="viewport" content="width=device-width, initial-scale=1">
8   <link rel="icon" type="image/x-icon" href="favicon.ico">
9 </head>
10 <body>
11   <app-root></app-root>
12 </body>
13 </html>
```

Vidimo zapravo da je ovaj html fajl veoma oskudan i da njegov body tag sadrži samo custom tag pod nazivom *app-root*, ali ukoliko uđemo u inspect element uočićemo da ovaj custom tag u sebi ipak sadrži neke podatke



Slika 4:

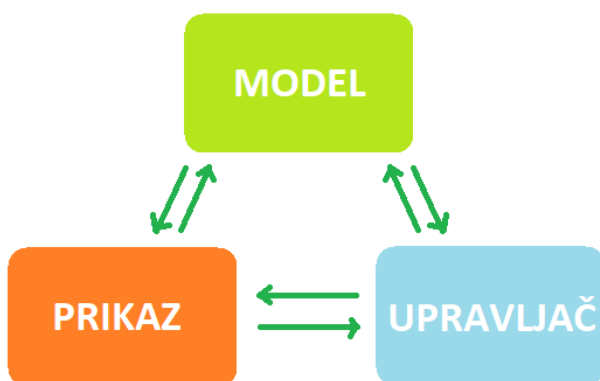
Postavlja se pitanje: "Gdje se nalaze ovi podaci?"
Odgovor na ovo pitanje jeste upravo naš custom tag *app-root* koji označava jednu *komponentu*, naime na ovaj način se uveo potpuno drugi html fajl koji "živi" unutar *src/app* komponente. Otvaranjem *src/app/app.component.html* fajla vidimo upravo ovaj dio html koda koji nam je nedostajao u *index.html* fajlu.

Upravo smo se upoznali sa jednim od ključnih odlika Angular-a, Komponentama. Više o tome ćemo saznati u sljedećem poglavlju.

3 Pregled osnovnih pojmova

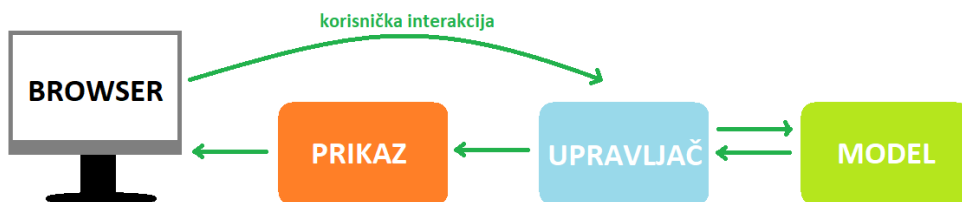
Jedan od razloga zašto je ovaj framework toliko popularan jeste njegova MVC arhitektura. MVC je akronim za Model-View-Controller arhitekturu koja postoji još od 70-ih godina, ono omogućava razdvajanje projekta u više cjelina kao i njihovu interakciju. Samim time, te cjeline se mogu ponovo koristiti (Reusability), kao i mijenjati uredno i pregledno bez velikih zastoja u kodu. Kao što vidimo i iz imena, MVC arhitektura se sastoji iz 3 dijela: model, prikaz i upravljač. Oni su međusobno povezani.

- model je glavni dio aplikacije i ono sadrži podatke sa kojima korisnik vrši interakciju
- prikaz bi predstavljao ono krajnje što korisnik vidi
- upravljač je posrednik između prethodna dva dijela, on prima zahtjeve, odnosno komande, za izvršavanje određenih akcija.



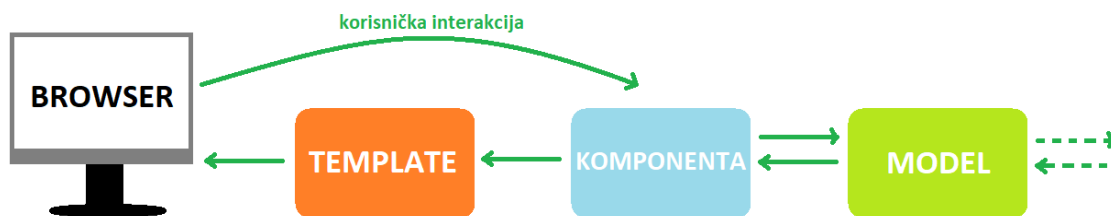
Slika 5: pojednostavljen prikaz MVC arhitekture

Ako ga primijenimo u kontekstu Angular-a uzimajući u obzir da korisnik krajnju interakciju kao i pregled neke stranice obavlja preko browsera, tragovi MVC arhitekture su vidljivi na sljedećoj šemi:



Slika 6: pojednostavljen prikaz Angular MVC arhitekture

Poboljšajmo još ovu šemu tako što ćemo reći da unutar Angular ekosistema upravljač predstavlja zapravo *komponentu* a prikaz *template* i da model još vrši interakciju, npr. sa nekom bazom.



Slika 7: pojednostavljen alternativni prikaz Angular MVC arhitekture

Sve prethodne informacije su bile od velike važnosti prije bilo kakvog stvarnog rada. Zbog toga Angular ima strmu krivu učenja jer se od programera zahtijeva shvatanje prethodnih informacija i terminologije. Sada smo napokon spremni da se bavimo samim stvaranjem web stranice i funkcionalnosti sa kojima se ljubitelji Angular-a susreću prilikom korištenja.

Istražićemo:

1. komponente (Component dekorator, custom tagovima, hijerarhija komponenti, templejti),
2. Binding (interpolacija, prosljeđivanje podataka u template, template references, two-way binding),
3. Component input i output (Input i Output dekoratori, vratiti se na hijerarhiju komponenti),
4. Core direktive (ngIf, ngFor, ngSwitch),
5. Injectable Services (Constructor injection, Dependency injection, nešto malo o radu "get" metode) item Routing (navigacija između komponenti)

3.1 Komponente

Vratimo na odjeljak 2.3 gdje smo postavljali okruženje i prisjetimo se kako je izgledao html file Angularovog "placeholder" sajta. Rekli smo da pored standardnih tagova svakog html dokumenta, u tijelu dokumenta se nalazio samo jedan tag pod nazivom app-root. Svako ko je upoznat sa html-om znao bi da ovo nije dio standardnih tagova html-a.

Primjer 2 src/index.html

```

1  .
2  .
3  <body>
4    <app-root></app-root>
5  </body>
6  .
7  .

```

Naime, prilikom kreiranja našeg okruženja, Angular je napravio unutar src foldera još jedan app folder. Upravo taj app folder čuva zapravo sve naše komponente i predstavlja korjenu komponentu. Već smo rekli da onaj dio koda između custom app-root tag-a sadrži dio koda iz *src/app/app.component.html* fajla. Šta sve app komponenta sadrži pored tog html fajla? Sadržaj komponente čine još i jedan css fajl koji opisuje izgled samo te komponente, kao i nekoliko veoma bitnih TypeScript fajlova.

primijetimo:

Sve datoteke unutar jedne komponente po konvenciji imaju šablon ime-komponente.*component.ekstenzija*

Istražimo app.component.ts datoteku:

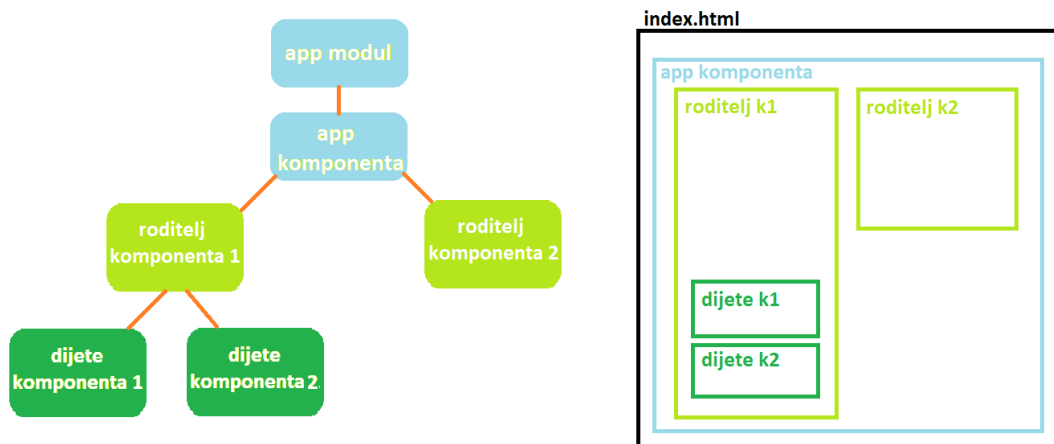
Primjer 3 src/app/app.component.ts

```
1 import { Component } from '@angular/core';
2
3 @Component({
4   selector: 'app-root',
5   templateUrl: './app.component.html',
6   styleUrls: ['./app.component.css']
7 })
8 export class AppComponent {
9   title = 'novi-projekat';
10 }
```

TypeScript nam dozvoljava korištenje dekoratora koje prepoznamo po znaku "@", a unutar component dekoratora vidimo 3 vrijednosti:

- selector koji nosi isti naziv kao i custom tag iz našeg index.html dokumenta, primjećujemo sada da taj custom tag predstavlja upravo jednu komponentu i njegov naziv se ovdje definiše. Korištenje komponente se radi samo preko ovog selektora tako što se njegov naziv napiše unutar html zagrada koje predstavljaju tag. Pisanje tog tag-a u index.html fajlu više puta će napraviti istu komponentu ispod ove prve (vidimo zašto se komponente smatraju "reusable" elementima i zašto jedna komponenta može biti šablon odnosno template)
- templateUrl koji spaja naš template (view) uz komponentu
- styleUrls koji nabroja CSS fajlove za stajling (vidimo da se link nalazi u listi)

Pogledajmo kako izgleda i šta znači hijerarhija komponenata.



Slika 8: Hijerarhija komponenata

Slika 8 sa lijeve strane predstavlja hijerarhiju komponenti, dok sa desne vidimo pojednostavljen prikaz Angular web stranice iza koje stoji ova hijerarhija.

Rekli smo da app komponentu smatramo korijenom. Što znači da u stablu komponenti zauzima sam vrh, iz ovoga slijedi da sve što uvezemo u ovu komponentu konačno možemo vidjeti, a tako i vršiti interakciju sa stranicom. To ne znači da je potrebno uvesti svaku komponentu, kao što vidimo iz slike 8, ona komponenta koja unutar sebe sadrži uvezane druge komponente se smatra roditeljskom i dovoljno je samo nju prikazati kako bismo vidjeli i njene nasljednike.

3.2 Binding

Nakon što smo vidjeli kako se prikazuju i uvoze komponente međusobno, vrijeme je da vidimo šta se dešava unutar njih samih.

Angular, prateći MVC arhitekturu, jasno odvađa model od prikaza, odnosno podatke (TypeScript) od template-a (html). Vratimo se na `app.component.ts` file i primijetimo varijablu pod nazivom `title` kojoj je dodijeljen neki tekstualni podatak.

Primjer 4 `src/app/app.component.ts`

```
1 import { Component } from '@angular/core';
2
3 @Component({
4   selector: 'app-root',
5   templateUrl: './app.component.html',
6   styleUrls: ['./app.component.css']
7 })
8 export class AppComponent {
9   title = 'novi-projekat';
10 }
```

Ako pogledamo `app.component.html` možemo pronaći jedan veoma neobičan izraz koji glasi:

Primjer 5 `src/app/app.component.html`

```
1 .
2 .
3 <span>{{ title }} app is running!</span>
4 .
5 .
```

Ukoliko promijenimo u TS dokumentu sadržaj `title` varijable i sačuvamo dokument možemo da vidimo da umjesto 'novi-projekat' (na slici 3) sada bi pisalo ono čime smo zamijenili prethodni string (Angular CLI server automatski prilikom svakog čuvanja ažurira i sinhronizuje promjene na samoj stranici)

Isto tako, ako promijenimo samu varijablu iz `title` u nešto drugo, vidimo da se taj string uopšte neće prikazati. Izraz `{{ title }}` je nešto što zovemo *interpolacijom*, pisanjem dvostrukih vitičastih zagrada, više se neće posmatrati sadržaj unutra kao html, već kao JS izraz. Unutar ovakvih zagrada može se nalaziti bilo šta što je JS izraz (niz, matematički izraz, funkcije).

Primjeri interpolacije unutar jedne komponente:

Primjer 6 neki ts dokument

```
1 import { Component, OnInit } from '@angular/core';
2
3 @Component({
4   selector: 'app-nova-komponenta',
5   templateUrl: './nova-komponenta.component.html',
6   styleUrls: ['./nova-komponenta.component.css']
7 })
8 export class NovaKomponentaComponent implements OnInit {
9   public name = 'Ana';
10
11   constructor() { }
12
13   ngOnInit(): void {
14   }
15
16 }
```

Primjer 7 neki html dokument

```
1 <h2>{{2+2}}</h2>
2 <h2>{{"Welcome " + name}}</h2>
3 <h2>{{name.length}}</h2>
4 <h2>{{name.toUpperCase()}}</h2>
```

Rezultat bi prikazao:

```
4
Welcome Ana
3
ANA
```

Ukoliko želimo možemo da prosljedimo podatke i na html atribute. Ovo se ostvaruje tako što unutar željenog dokumenta određeni atribut okružimo četvrtastim zagradama. Sada se više ne gleda sadržaj koji mu se dodjeljuje kao običan tekst, već takođe kao JS expression proslijeđen iz TS fajla. Primjer:

Primjer 8 neki ts dokument


```
1 .
2 .
3 public name = 'Ana';
4 public isDisabled = false ;
5 .
6 .
```

Primjer 9 neki html dokument

```
1 <input disabled="false" type="text" [value]="name">
2 <input [disabled]="isDisabled" type="text" [value]="name">
```

Po default-u, postavljanjem atributa disabled="" na input smatra se da je "true", tako da iako napišemo "false" ovaj input će biti onemogućen. Četvrtastim zagradama smo sada od atributa (definisan od strane html-a, ne može se mijenjati) dobili property (definisan od strane DOM-a i može se mijenjati) i smijemo mu

dodijeliti false vrijednost.
Rezultat:



Slika 9: Oba ova inputa treba da onemoguće disabled osobinu i samo je jedan bio uspješan.

Iste zagrade mogu da okružuju "class" atribut (tada dodijelimo u TS fajl nekoj varijabli ime željene klase iz css fajla), kao i inline css atribut, "style". Binding se može uraditi i na evente okružujući event običnim zagradama, u ovom slučaju moramo dodijeliti funkciju, u kojoj smo u TS-u definisali šta treba da se uradi ukoliko je event pokrenut.

Takođe možemo i iz samog html fajla da šaljemo podatke u TS pomoću *template reference* varijabli. bilo gdje unutar nekog taga znakom "#" dodijelimo naziv našem elementu i kasnije preko tog naziva možemo da pristupimo ostalim vrijednostima i da ih npr. proslijedimo nekoj funkciji.

Primjer 10 neki ts dokument

```
1  .
2  .
3  logMessage(value:string){
4      alert(value);
5  }
6  .
7  .
```

Primjer 11 neki html dokument

```
1  <input type="text" value="Ana" #myInput>
2  <button (click)="logMessage(myInput.value)">log</button>
```

Pritiskom na ovo dugme triggerovaće se event i funkcija će vratiti alert sa sadržajem inputa iznad dugmeta, u našem slučaju ispisaće: "Ana"

3.3 Component input i output

Nekada želimo da nešto iz roditeljske komponente proslijedimo nekoj funkciji iz djetete komponente i obrnuto. Ovu komunikaciju nam dozvoljavaju @input i @output dekoratori.

Počnimo sa lakšim slučajem: proslijeđivanje u pravcu djetete-roditelj

Ako samo na nivou djetete komponente napravimo neki event i metodu koja nešto radi ono će isplivati (desiti se) skroz do root elementa čak iako nije u njemu definisana (uzmimo primjer 10 i 11 i pretpostavimo su dio djetete komponente i između te i korjene se nalazi jedan roditelj), ovo je rezultat hijerarhijskog rasporeda komponenti. Output dekorator će služiti ovdje potpuno drugu ulogu od one koju ćemo vidjeti za input dekorator. Iako smo rekli da će event iz djetete komponente isplivati do korjene, dalje se taj isti event može "uhvatiti" na nivou korjene i da joj se dodijeli još jedna funkcija van nasljednika. Pored standardnih event-a sada imamo dozvolu da napravimo custom event-e, upravo preko @output dekoratora, u šta nećemo previše ulaziti.

Komunikacija u smjeru roditelj-dijete je nešto komplikovanija i ona obavezno traži korištenje input dekoratora, dok u prethodnoj situaciji to nije bio slučaj zbog prirode same hijerarhije.

Pripremanje svake komponente:

Kako bi dijete komponenta znala da očekuje neke podatke od svog roditelja potrebno je joj javimo @input dekoratorom šta će dijete da primi ,dovoljno je da navedemo ime varijable i tip podatka kao i da se ta varijabla koristi negdje unutar html-a, interpolacijom npr. Trenutno imamo sljedeću situaciju:

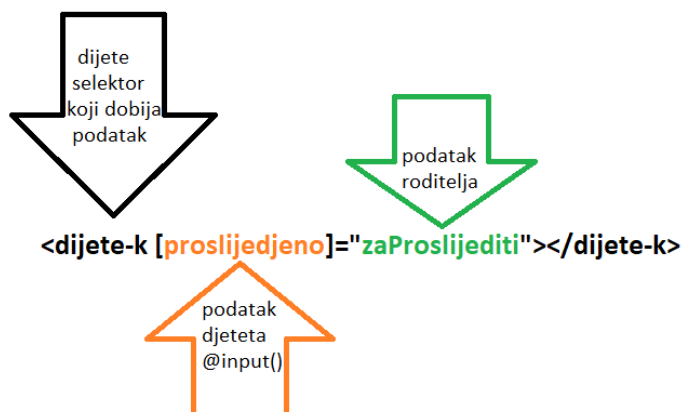
Primjer 12 roditelj.html

```
1 .
2 .
3 <dijete-k></dijete-k>
4 .
5 .
```

Nalazimo se u roditeljskoj komponenti, tagovima smo naveli da koristimo dijete komponentu, i želimo da joj proslijedimo neki string, prethodno definisan. Neka je "prosljedjeno" varijabla tipa string definisana input dekoratorom u dijete komponenti, a "zaProslijediti" varijabla u roditelj komponenti koja sadrzi rečenicu: "Uspjesno proslijedjeno"

Primjer 13 roditelj.html

```
1 .
2 .
3 <dijete-k [prosljedjeno]="zaProslijediti"></dijete-k>
4 .
5 .
```



Slika 10: primjećujemo da ovo nije ništa drugo nego binding gdje se nekom property-u dodjeljuje JS expression

I na ovaj način u dijete komponenti varijabla "prosljedjeno" dobija vrijednost "Uspjesno proslijedjeno".

3.4 Core directives

Angularove core direktive: ngFor, ngIf, ngSwitch, nam omogućavaju lakši i pregledniji kod. Ukoliko znate čemu služe naredbe if/else, switch kao i petlja for, nazire se i njihova uloga.

Jedan čest način gdje možemo vidjeti ngFor u akciji jeste kada koristimo jednu komponentu više puta, recimo

da je jedna komponenta šablon za prikaz jednog oglasa, a oglasna tabla neka je roditeljska komponenta koja sadrži sve oglase u sebi. Kako bismo izbjegli stavljanje oglas selektora u roditeljsku komponentu mnogo puta, to za nas može da uradi jednostavna petlja. Ova strukturalna direktiva (dozvoljava promjenu strukture na sajtu na osnovu js expression-a) može da se pokaže na sljedeći način:

Primjer 14 oglasna-tabla.html

```
1  .
2  .
3  <oglas-k *ngFor="let oglas of oglasi"
4      event za prikazivanje jednog elementa iz kolekcije oglasi
5      kojoj se proslijedi element: "oglas" kao parametar></oglas-k>
6  .
7  .
```

ngFor takođe nudi pristupanje elementima preko indeksa, omogućava da se nešto posebno u petlji uradi samo sa prvim, posljednjim, svakim parnim i svakim neparnim elementom iz kolekcije

*ngIf="" direktiva uzima neki JS expression i kao rezultat prima boolean vrijednosti. U slučaju da dobiye "false" kao rezultat, obrišaće u potpunosti taj element. Kao i svaka if naredba korištenjem ključne riječi "else" možemo napraviti alternativni prikaz ukoliko je ngIf netačne vrijednosti.

*ngSwitch="" ima sličnu ulogu kao i ngIf, s tim da dozvoljava mnogo više slučajeva kao i default case ukoliko nijedan prethodni nije ispunjen. Angular takođe nudi još neke direktive kao što su "ngClass" i "ngStyle" koje služe za uvoz CSS-a, na čemu se nećemo zadržavati.

3.5 Services i Dependency injection

Svi do sad pređeni pojmovi bili su u sklopu view sloja, a sada ulazimo u service sloj koji vrši kontak sa bazom i interakciju sa view slojem.

Service ili "usluga" (može i "servis"), obuhvataju svaki feature, vrijednost, funkciju koja je potrebna nekoj aplikaciji. Najčešće je to klasa koja ima jasno definisan cilj. Angular, kao i do sad, razdvaja servise od komponenti. To im, u pravom duhu Angulara, dozvoljava modularnost, odnosno ponovno i često korišćenje. Komponenta od servisa može tražiti hvatanje podataka iz baze, user input i mnoge druge pozive. Servisi jednom definisani su dostupni svakoj komponenti dokle god je izvršen *Dependency injection*.

Jedan od mnogih usluga, sa kojima Angular unaprijed već dolazi, jeste "get" metod. Kako bi se omogućilo njegovo korištenje potrebno je da ga umetnemo(injecting) u klasu i to radimo pomoću konstruktora u kojem definišemo HttpClient, Angularov http servis. Kako da Angular zna kada treba da se http request desi? Svaka komponenta u sebi sadrži ngOnInit() metod koji je "lifecycle hook" metod. Odnosno, ovaj metod automatski prepoznaje da je neka komponenta (i sva njena djeca) postala dostupna, ili trenutno prikazana korisniku. Njegov poziv ne vrši programer (kod), već Angular u svojoj pozadini. Ovo je upravo zgodan metod za postavljanje neke logike vezane za inicijalizaciju. U njemu se poziva prethodno postavljen konstruktor get metodom.

Primjer 15 pojednostavljen primjer jednog TS fajla koji implementira get metodu

```
1  .
2  .
3  export class AppComponent {
4      oglasi;
5
6      constructor(private http: HttpClient){
7
8      }
9      ngOnInit(){
10         const params=new HttpParams()
11             .set("neki parametar", "vrijednost parametra")
12         }
13         this.http.get('baza', {params})
14             .subscribe(
15                 oglasi => this.oglası = oglasi
16             );
17     }
18     .
19     .
```

Dependency injection ćemo koristiti onda kada je potrebno nekoj komponenti dodati ili "ubrizgati" servis (ne mora nužno uvijek da bude samo servis, moguće je i sa samim funkcijama). Definisanje svih servisa, se radi @Injectable dekoratorom. Nakon što Angular otkrije da nekoj komponenti treba odoređeni servis, dalje se pozivaju konstruktori te komponente, koji preuzimaju posao.

Ova umetanja, ili ubrizgavanja, servisa su mnogo komplikovana za pokriti u jednom radu, te se dalje neće komentarisati detalji implementacije.

3.6 Routing

Angular, takođe nudi opciju "rutiranja", odnosno navigacije između različitih prikaza na nekom sajtu (npr. navigacija u "meni bar-u") jednostavno skrivajući i otkrivajući komponente, bez da server ponovno učitava novu stranicu. Rutiranje se odvija unutar src/app/app-routing.module.ts fajla i zahtijeva postojanje minimalno dvaju komponentata

Primjer 16 src/app/app-routing.module.ts

```
1
2  import { NgModule } from '@angular/core';
3  import { RouterModule, Routes } from '@angular/router';
4
5  const routes: Routes = [];
6
7  @NgModule({
8      imports: [RouterModule.forRoot(routes)],
9      exports: [RouterModule]
10 })
11 export class AppRoutingModule { }
```

Ukoliko ne radimo sa Angular CLI bilo bi potrebno ovaj modul uvesti, ali je to unaprijed za nas već uvezeno. Dalje je potrebno definisati naše rute u "Routes" nizu. Svaki element ovog niza predstavlja jedan JS objekat. Ti objekti u sebi zadrže dvije osobine: path (URL path rute) i component (ime same komponente koju rutiramo). Ovako definisane rute se mogu dalje koristiti. Rute dozvoljavaju i da se navede neki "default" prikaz (npr. error 404) koji će biti omogućen ukoliko korisnik želi da pristupi nekoj nepostojećoj ruti.

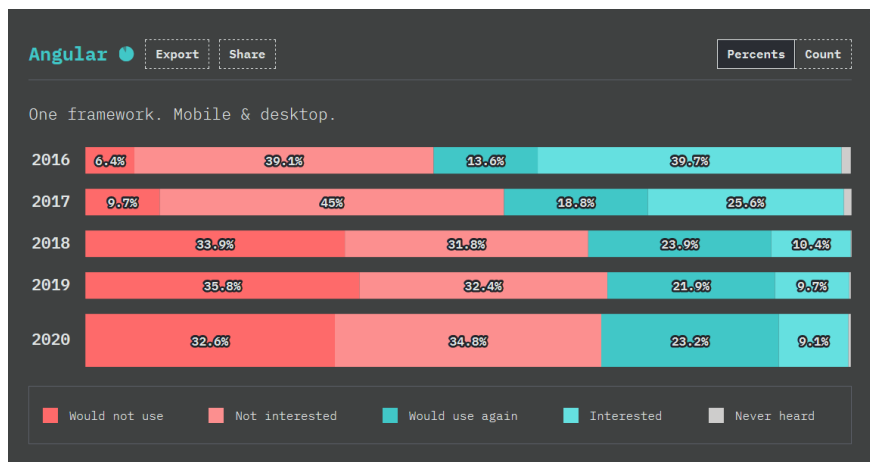
Kao i prethodne, rutiranje je takođe velika tema sa mnogo detalja, koja nije namijenjena opsegu ovog rada. Toplo se preporučuje, ukoliko interesovanje postoji, da se dalje teme istraže samostalno.

4 Zaključak

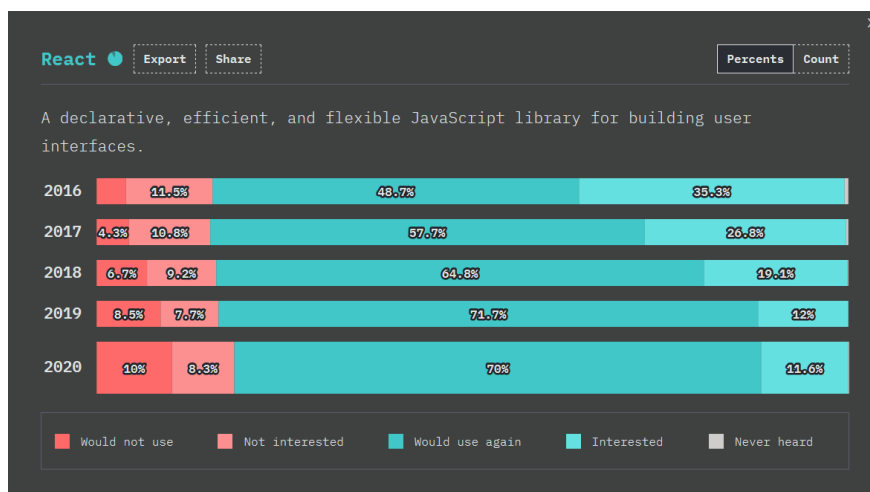
4.1 Da li je Angular za Vas?

Nakon višednevnog istraživanja ali i korištenja ovog frameworka, izdvaja se nekoliko stvari koje se smatraju prednostima Angulara u odnosu na njegovu konkurenciju React: MVC, Dependency injection, rutiranje, two-way binding (može da bude i negativna osobina), detekcija promjena, ne traži dodatne biblioteke za prethodne pojmove. Ali ono što predstavlja nedostatak u odnosu na React je činjenica da je razumijevanje i učenje veoma teško. "Learning curve" ili "kriva učenja" je veoma strma u odnosu na provedeno vrijeme učeći (dok je za React potrebno samo znanje iz JS-a), two-way binding ga nekad zna učiniti komplikovanim ukoliko se previše koristi, kao i česta ažuriranja sa kojima se mora biti u toku.

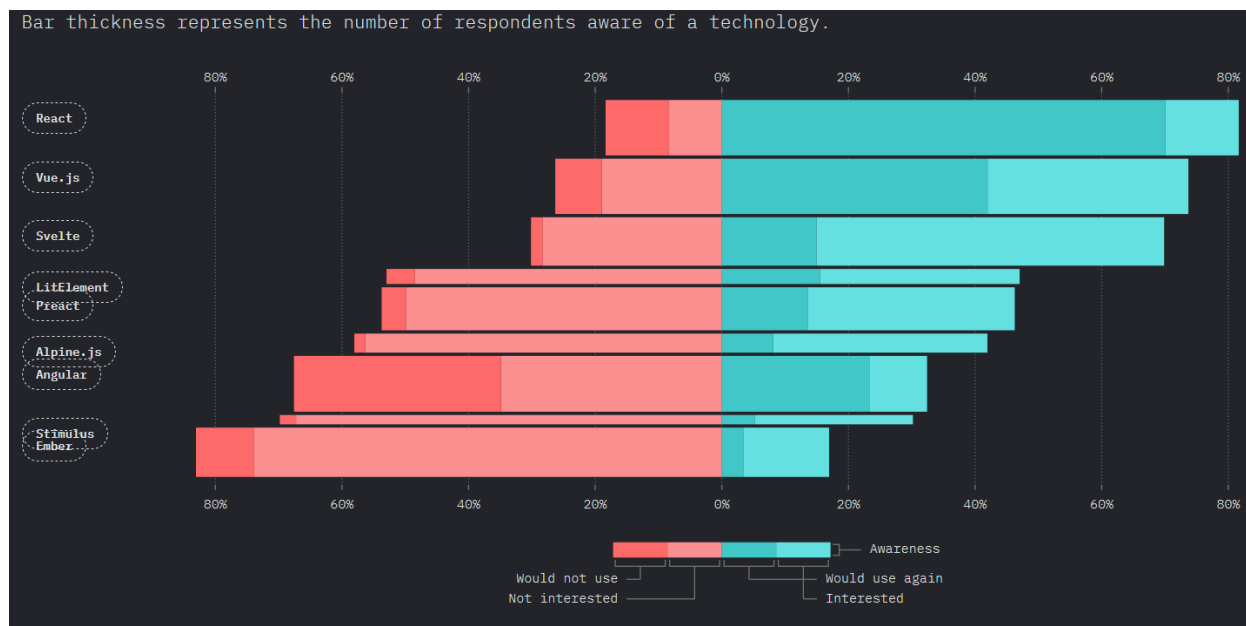
Kakvo mišljenje o ove dvije platforme ima zajednica možemo saznati sa State of JS sajta. Stranica koja par godina unazad već prikuplja mišljenja o različitim tehnologijama, samim tim i front-end tehnologijama. Rezultati su bilježeni na sljedeći način: oni koji su koristili određenu tehnologiju (tamnija nijansa) i oni koji su samo čuli za tehnologiju (svjetlija nijansa), i da li su zainteresovani da je koriste ponovo(plava boja), odnosno ne žele da je koriste uopšte (crvena boja). Od prosječno 20 000 ispitanika godišnje imamo sljedeće rezultate:



Slika 11: izvor fotografije: www.stateofjs.com



Slika 12: izvor fotografije: www.stateofjs.com



Slika 13: izvor fotografije: www.stateofjs.com

Kao što vidimo favorit postoji, ali autoru se ovo predstavlja kao dobar izazov i samoprocjena vlastitih sposobnosti. Nekome bez predznanja navedenih iz poglavlja 2.1 se ne savjetuje uzimanje ove tehnologije, jer bi se još uvijek mogli zvati početnicima. Toplo se preporučuju kursevi sa zvaničnog Angular University sajta, besplatni su i nema boljeg načina da se nešto nauči nego kod ljudi koji posvećuju toliko vremena razvijajući ovaj proizvod.

5 Izvori

<https://www.sitepoint.com/angularjs-vs-angular/>

<https://angular.io/guide/releases>

<http://blog.andolasoft.com/2019/02/why-angular-is-so-popular-in-modern-application-development.html>

<https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-angular-development/>

<https://blog.angular-university.io/why-angular-angular-vs-jquery-a-beginner-friendly-explanation-on-the-advantages-of-angular-and-mvc/>

<https://blog.angular-university.io>

<https://angular.io/guide/hierarchical-dependency-injection>

<https://angular.io/guide/inputs-outputs>

<https://2020.stateofjs.com>