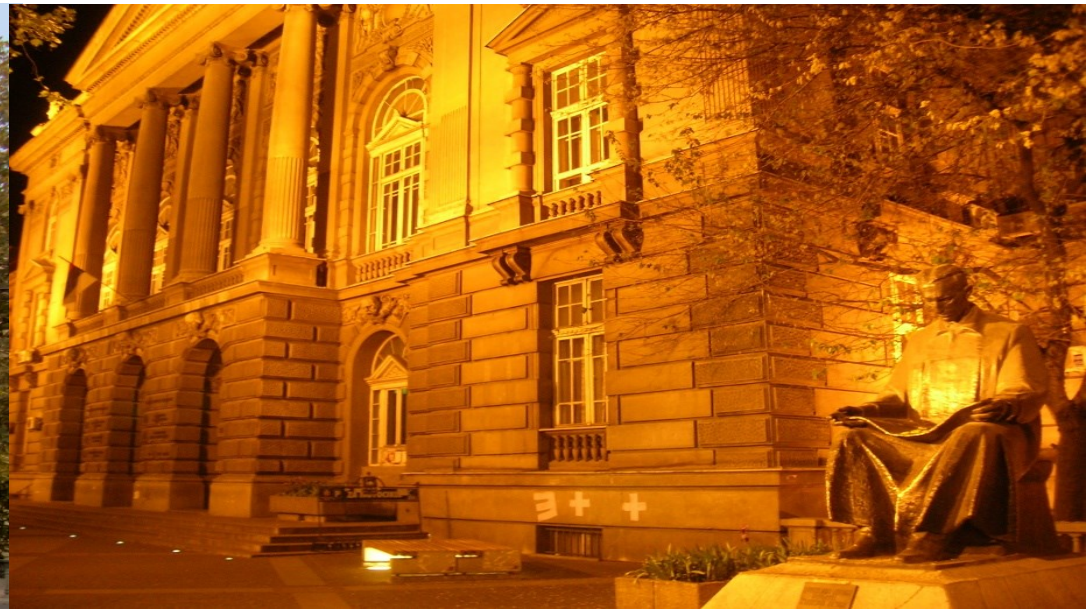
Univerzitet u Beogradu – Elektrotehnički fakultet

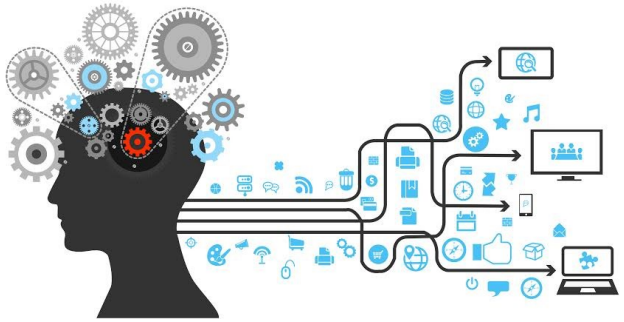
Uvod u Angular i TypeScript



Predavač: Dražen Drašković

Beograd, oktobar 2023. godine

Sadržaj



- Uvod
- Kratak pregled veb aplikacija
- JavaScript programski jezik
- Angular JS (Angular 1.x)
- Angular 2+
- TypeScript programski jezik
- Veb aplikacije sa jednom stranom (*Single Page App*)
- MEAN arhitektura
- JS radni okviri kao alternative

Uvod

- Glavni akteri veb aplikacija: korisnik, veb pregledač, veb server i pozadinski servisi.
- Korisnici: ostvaruju interakciju i imaju očekivanja
- Veb pregledač: mora biti podržan od strane tehnologije, jer on komunicira od veb servera i ka veb serveru.
- Komunikacija:
  - GET zahtev - najčešće za preuzimanje podataka;
  - POST zahtev - za slanje podataka na server;
  - AJAX zahtev – GET ili POST zahtev koji direktno šalje JS zahteve pregledaču.

Pregled programskih jezika kroz godine

1971-1980	1981-1990	1991-2000	2001-2010	2011-данас
1972: C	1981: MS-DOS1	1991: Python	2001: Win XP	2011: Android 4
1972: Prolog	1983: Ada 83	1991: Visual Basic	2003: Scala	2012: Win 8
1968: Algol68	1983: C ++	1991: DOS5, Linux	2003: Red Hat	2012: TypeScript
1974: SQL	1983: Turbo Pascal	1992: Open GL	2004: JSF 1.0	2013: An.JellyBean
1978: Matlab	1983: Word for DOS	1993: Solaris	2004: Gmail Mozilla Firefox	2013: An.KitKat
1979: Atari DOS	1984: Lisp	1995: Delphi	2005: Ubuntu 5	2013: React JS
1980: Ada 80	1985: Windows 1.0	1995: Java / Jscript	2007: MS Vista	2013: JSF 2.2
1980: TCP / IP	1985: NSF NET	1995: PHP, Ruby	2008: Android	2014: HTML 5
	1987: Perl	1995: Windows 95	2008: Goo.Chrome	2014: Hack (FB)
	1987: Windows 2.0	1996: Windows NT	2009: CoffeeScript	2014: Angular 2
	1987: Excel for Win	1997: Mac OS 7 / 8	2009: Win 7	2014: Swift
	1988: MS-DOS 4	1998: C++ stand. JavaServlet stand.	2009: Mongo DB	2016: Kotlin (2011)
	1989: WWW/HTML	1998: Windows 98	2009: Node JS	2017: C++17; JavaServlet 4
	1990: Windows 3.0	2000: C#	2010: Angular JS	2018: Angular 7

JavaScript / ECMAScript

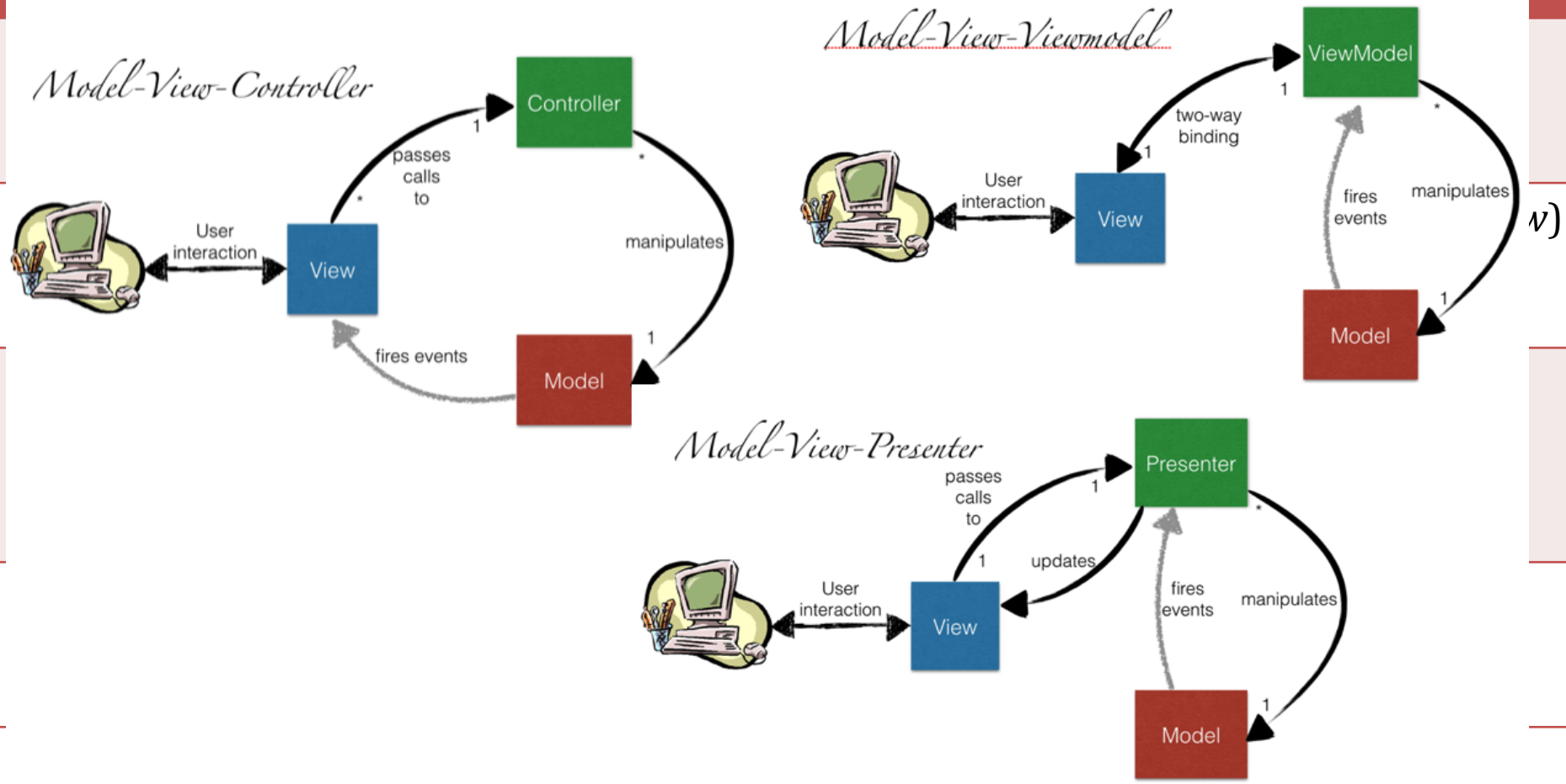
- 1995. razvijen kao klijentski skript jezik
- HTML + CSS + JavaScript = jezgro WWW
- Strukturno programiranje preuzeto iz jezika C, a sličan Javi
- Jezik viših paradigmi, podržava: event-driven, funkcionalne, imperativne stilove (objektno-orijentisan / prototipski-baziran)
- Podržava rad sa: osnovnim tipovima podataka, tekstom i datumom, nizovima, regularnim izrazima, osnovne manipulacije sa DOM (*Document Object Model*)
- Ne podržava: rad sa IO, umrežavanje, ili grafičke objekte
- Danas ugrađen u mnoge softvere i serverski orijentisane tehnologije

Angular tehnologije

- Angular JS, oktobar 2010.
- Razvijen od strane Google kompanije i zajednice
- JavaScript baziran frontend radni okvir
- Podržava razvoj aplikacija na jednoj veb strani (eng. Single-page Web App) i progresivnih veb apl.
- Uzorci: podržava MVC (i MVVM), što je dovelo do MVW
- JavaScript dopunjuje radni okvir Apache Cordova, koji služi za multiplatformi (*cross platform*) razvoj mobilnih aplikacija
- Link: https://angularjs.org/

Projektni uzorci kod današnjih aplikacija

Naziv
MVC
MVP
MVVM
MVW



AngularJS način rada (1)

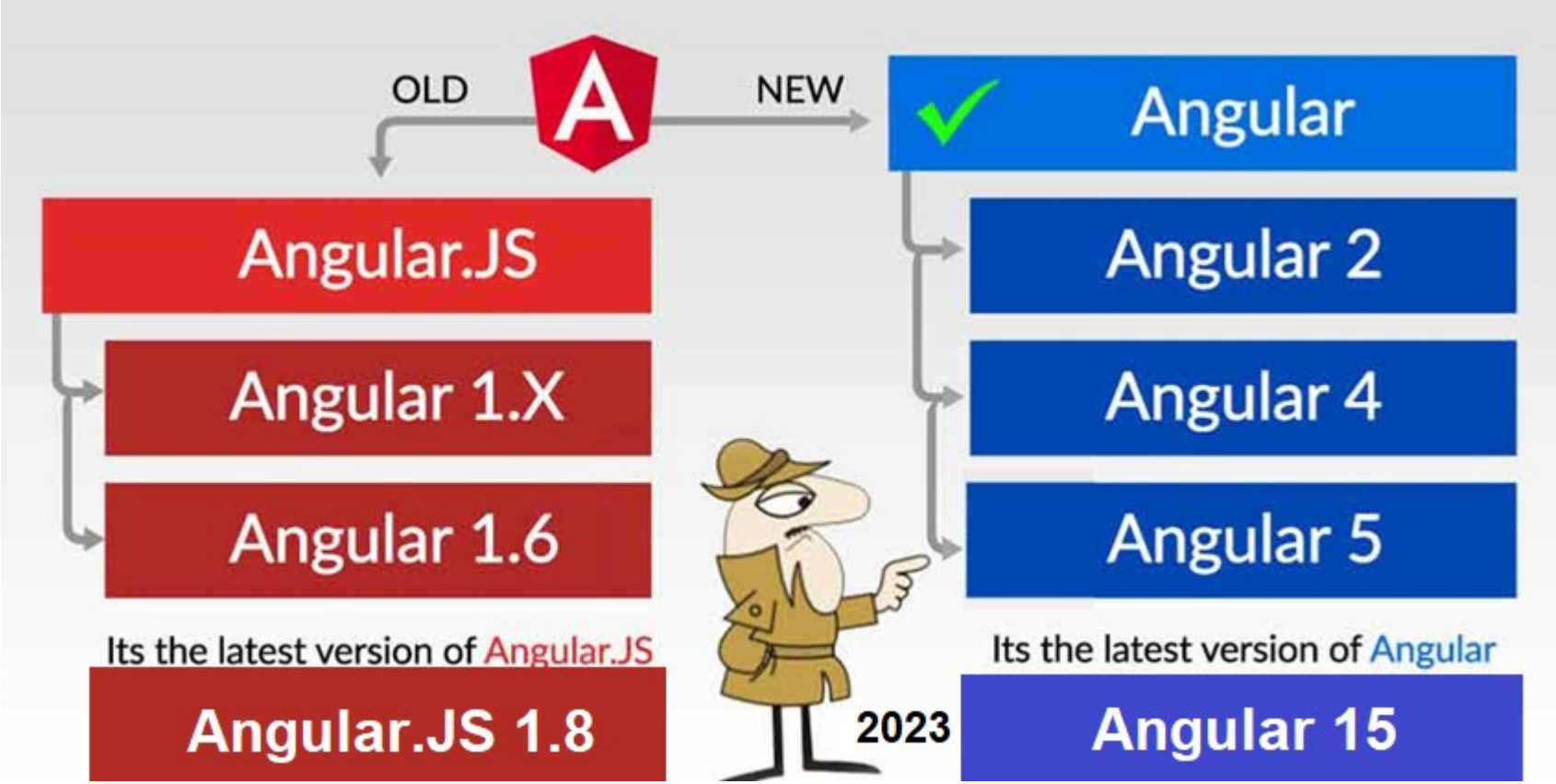
- AngularJS prvo učitava HTML stranu, koja ima ugrađene dodatne prilagođene tag atribute.
- Angular interpretira te atribute kao direktive za povezivanje ulaznih i izlaznih delova strane, sa modelom koji predstavlja standardne JavaScript varijable.
- Vrednost tih JS varijabli: manuelno unutar koda ili preuzeti iz statičkih/dinamičkih JSON resursa.

AngularJS način rada (2)

- Deklarativno programiranje, za korisničke intefejse i povezivane komponenti, pre imperativnog programiranja za definisanje poslovne logike.
- Obuhvata: DOM manipulaciju, validaciju ulaza, serversku komunikaciju, upravljanje URL,...
- Koristi MVC uzorak i dvosmerno povezivanje podataka (automatska sinhronizacija Modela i View)
- Fokus na podršci za programiranje velikih i aplikacija sa jednom veb stranom (poboljšane performanse, testabilnost, reupotreba komponenti, moduli)

AngularJS => Angular 2+

- Od 2014. AngularJS (Angular.JS) postaje Angular aplikaciona platforma (ili kraće Angular).



AngularJS / Angular verzije

- Angular JS 1.0 (2010), Angular JS 1.2 (2012), Angular JS 1.3 (2014), Angular JS 1.6 (2016), Angular JS 1.7 (2018), Angular JS 1.8 (2020), 1.1.2022. - kraj podrške
- 2014: Angular 2.0 (nekompatibilan sa AngularJS)

  // preskočeno: Angular v3.3.0 (zbog neusaglašenosti)
- 2016: Angular 4.0 (kompatibilan sa Angular 2)
- 2017, novembar: Angular 5 (podrška za progresivne veb aplikacije, izgradnja optimizacija,
  i poboljšanja Material Design-a)
- 2018, maj: Angular 6; 2018, oktobar: Angular 7; 2019 maj: Angular 8;
  2020 februar: Angular 9; 2020 jun: Angular 10; ... 2022 novembar: Angular 15;
  2023 septembar: Angular 16.2.5

Šta su ključne razlike u verzijama? (1)

- Arhitektura:
  - AngularJS: MVC / MVW
  - Angular 2: servis/kontroler arhitektura
    (ne postoji mogućnost nadogradnje iz AngularJS)
  - Angular 4: zasnovan na Angular 2, samo je smanjen dati programski kod i brži je razvoj (kompatibilni 2 i 4)
- Programski jezik:
  - AngularJS primenjuje JavaScript u razvoju
  - Angular 2 primenjuje TypeScript (superset programskog jezika JavaScript)
  - Angular 16 kompatibilan sa TypeScript 5.2.2

Šta su ključne razlike u verzijama? (2)

- Mobilni razvoj:
  - Angular 2 omogućio da se ostvari izvorna aplikacija za mobilne platforme (npr. *React Native*)
  - Angular 2 daje aplikacioni sloj i sloj prikazivanja (rendering)
  - Po potrebi, bilo koji pogled (*view*) može da se prikaže u realnom vremenu, za zahtevanu komponentu
- Komponente zasnovane na korisničkom interfejsu:
  - Koncept kontrolera u Angular JS eliminisan u Angular 2
  - Verzije 2+ kor. interfejs zasnivaju na komponenti
  - Verzije 2+ su poboljšale fleksibilnost i ponovnu upotrebu koda

TypeScript

https://www.typescriptlang.org

Tipovi podataka (1)

- String – znakovni tip, sa jednostrukim ili dvostrukim navodnicima

```
var mojTekst: string = 'PIA';
var drugiTekst: string = "PIA Novo";
```

- Broj (*number*) – tip podataka za numeričke vrednosti

```
var broj: number = 10;
var cena: number = 133.55;
```

- Bulove vrednosti (*boolean*) - tip podataka sa vrednostima true/false (kod flegova)

```
var da: boolean = true;
var ne: boolean = false;
```

- niz (*Array*) - indeksirani niz (indeksi od 0 do n-1 za n elemenata)

```
var niz: string[] = ["PIA","MIPS","TS"];
var prvi = niz[0];
var niz2:number[] = [1,2,3];
var niz3: Array<number> = [1,2,3,4,5];
```

Koristi se i ključna reč let umesto var: **let** godine: number = 30;

Tipovi podataka (2)

- torka - ako je poznat ustaljen broj elemenata niza

```
var x: [string, number];
x = ["Drazen", 20];
```

- enum - nabrojivi tip, može dodeljivati i određene vrednosti

```
enum Ljudi {Jelica, Sanja, Bosko, Drazen}
var x = Ljudi.Bosko; //number 2
var y = Ljudi[0]; //string Jelica
enum Boje {Crvena=1, Zelena, Plava}
enum Boje {Crvena=1, Zelena=3, Plava=5}
```

- undefined / null vrednost - nevažeća vrednost (podtipovi svih tipova)

```
var novaGodina = null;
```

- any - promenljivoj možete dodeljivati bilo koji tip podataka

```
var nesto: any = "Tekstualni podatak";
var nesto = 909;
var nesto = true;
```

- void - kada nije potrebno dodeliti tip podatka (često kod funkcija)

```
function prazna(): void { document.write("Zdravo"); }
```

Tipovi podataka (3)

- Objekti - neprimitivni tipovi, to nisu stringovi, numerici, bulove vrednosti,...

```
declare function create(o: object | null): void;
create({ mojobjekat: 0 }); // OK
create(null); // OK
create(42); // Greska
create("moj tekst"); // Greska
create(false); // Greska
create(undefined); // Greska
```

- *Type assertions* - sličan operatoru cast u drugim jezicima
  "Kompajleru, veruj mi, znam šta radim"
  Pomoću zagrada:

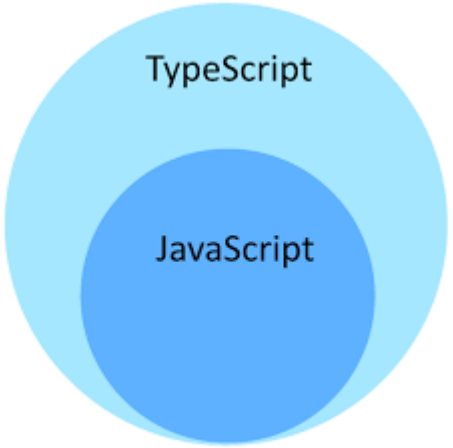
```
let tekst: any = "ETF BEOGRAD";
let duzina: number = (<string>tekst).length;
```

  Pomoću "*as*" sintakse:

```
let tekst: any = "ETF BEOGRAD";
let duzina: number = (tekst as string).length;
```

var, let, const

- *var* - deklaracija u JavaScript-u
- *let* i *const* - dva nova tipa deklaracije promenljivih
- *let* sličan kao *var*, ali omogućava da se izbegnu neke klasične programerske greške
- *const* je proširenje *let*, jer sprečava ponovnu dodelu varijabli



Šta rešava let? (1)

```
function f(uslov: boolean) {
    if (uslov) {
        var x = 10;
    }
    return x;
}
f(true);
// vraca '10'

f(false);
// vraca 'undefined'
```

```
function f(uslov: boolean) {
    let a = 100;

    if (uslov) {
    // mogu da koristim a
        let b = a + 1;
        return b;
    }

    // Greska: b ovde ne postoji
    return b;
}
```

Opseg u blokovima

Šta rešava let? (2)

```
function f(x) {
    var x;
    var x;

    if (true) {
        var x;
    }
}
//ispravno
```

```
let x = 10;
let x = 20; // greska
```

```
function f(x) {
    let x = 100; // greska
}

function g() {
    let x = 100;
    var x = 100; // greska
}
```

Redeklarisanja / redefinisanja

Interfejsi

- Omogućavaju da postavimo strukturu za aplikaciju, ili strukture za objekte, funkcije, nizove, klase
- Mogu da se dodaju i opcione stavke (polja):
  atribut?: tipPodatka

Primer:

```
interface Osoba {
    ime: string;
    bojaKose: string;
    godine: number;
    zaposlen?: boolean;
}
```

Definisanje intefejsa za funkciju

Primer:

```
interface Dodavanje {
     (num1: number, num2: number)
}
var x: number = 5;
var y: number = 10;
var noviBroj: Dodavanje;
noviBroj = function(num1: number, num2: number){
     var rezultat: number = num1 + num2;
     document.write(rezultat);
     return rezultat;
}

var z = noviBroj(x,y);
```

Implementiranje klasa

Primer:

```
class Osoba{
    ime: string;
    godine: number;
    zaposlen: boolean = true;
    constructor(ime: string, godine?: number) {
            this.ime = ime;
            this.godine = godine;
    }
    otpusten(){
            this.zaposlen = false;
            return "Dao otkaz";
    }
}
var ja = new Osoba("Drazen",30);
```

Nasleđivanje klasa

Primer:

```
class Nastavnik extends Osoba {
    brojPredmeta: number = 3;

    dodajPredmet(){
            this.brojPredmeta++;
            return this.brojPredmeta;
    }
    oduzmiPredmet(){
            this.brojPredmeta--;
            return this.brojPredmeta;
    }
}
```

Moduli

- Moduli omogućavaju da se programski kod organizuje u više fajlova.
  Ovo skraćuje fajlove i čini ih održivim.
- Moduli se izvršavaju sa sopstvenim opsegom važenja, ne u globalnom opsegu (ovo znači da klase, varijable, funkcije, definisane u modulu, nisu vidljive van, ukoliko se ne izvezu).
- Odnosi između modula se postižu uvozom (import) i izvozom (export), na nivou fajla.
- Jedan modul uvozi neki drugi modul pomoću modula loader.
  Modul loader je u realnom vremenu odgovoran za lociranje i izvršavanje svih zavisnosti modula, pre nego što ga izvrši.

Export kod modula

- Svaka deklaracija (varijabla, funcija, klasa, interfejs) može biti izvezena (export-ovana).

Primer:

```
//Validation.ts
export interface StringValidator {
    isAcceptable(s: string): boolean;
}
//ZipCodeValidator.ts
export const numberRegexp = /^[0-9]+$/;
export class ZipCodeValidator implements StringValidator {
    isAcceptable(s: string) {
        return s.length === 5 && numberRegexp.test(s);
    }
}
export * from "./ZipCodeValidator"; // eksport iz fajla
```

Import kod modula

- Slično kao i kod izvoza, radi se i uvoz fajla.

Primer:

```
import { ZipCodeValidator } from "./ZipCodeValidator";
let myValidator = new ZipCodeValidator();
```

Sa preimenovanjem:

```
import { ZipCodeValidator as ZCV } from "./ZipCodeValidator";
let myValidator = new ZCV()
```

Funkcije

- Funkcije u TS, slične kao funkcije u JS, ali postoje neke dodatne mogućnosti
- TS funkcije: dodati tipove parametrima koje prosleđujete, dodati tip rezultata funkciji (opciono)
- Primer:

```
function potpis(ime: string, prezime: string): string{
    return ime + ' ' + prezime;
}
```

- Opcioni parametri:
  nazivParametra?: tipPodatka
- Parametri funkcije sa podrazumevanim vrednostima:
  nazivParametra = "PodrazumevanaVred"

tsc - TypeScript kompajler

- npm install -g typescript
- Program: primer.ts
  console.log("Dobrodosli svima");
- tsc primer.ts => primer.js ili greška!

Angular / Angular 2+

https://angular.io

Zašto Angular?

- Radni okvir za razvoj savremenih aplikacija za veb, mobilne i desktop platforme
- Čist i strukturiran pristup pisanju koda
- Dobra proširivost i ponovna upotreba
- Zasnovan na *HTML* i programskom jeziku *TypeScript*
- Veliki broj funkcija za upravljanje korisničkim unosom u pregledaču, za manipulisanje podataka na strani klijenta i za kontrolisanje kako su elementi prikazani u pregledaču
- Dobra kompatibilnost sa drugim tehnologijama (veza sa serverskom stranom ka Node.JS ili MVC .NET ...)
- Obezbeđena dobra struktura veb aplikacija kroz povezivanje podataka, injektiranje zavisnosti, HTTP komunikaciju

Moduli

- Angular app koriste modularni dizajn - moduli nisu obavezni, ali se preporučuju!
- Ključne reči: import i export
- Za razliku od modula u prog. jeziku *TypeScript*, kod NgModules:
  - spoljni moduli se uvoze na vrhu fajla
  - funcije za izvoz se stavljaju na dnu fajla
- Sintaksa:

```
Import {Component} from 'angular2/core';
Export class MojaKlasa{ }
```

- NgModule može povezati komponente sa servisima
- Aplikacija uvek ima koreni modul (AppModule) koji obezbeđuje mehanizam pokretanja aplikacije
- Aplikacija tipično sadrži mnoštvo funkcionalnih modula

Direktive

- Direktive su JS klase sa metapodacima koje definišu strukturu i ponašanje. Obezbeđuju većinu UI funkcija za Angular aplikacije. Tri osnovna tipa direktiva su:
  - **Direktiva komponenata**: sadrži HTML šablon sa JS funkcijama za kreiranje samostalnog UI elementa, koji se može dodati u Angular aplikaciju; komponente su direktive koje se najčešće koriste.
  - **Strukturalne direktive**: direktive koje se koriste kada je potrebno da se manipuliše sa DOM-om; one omogućavaju da kreirate i uklonite element i komponentu iz prikaza.
  - **Atributske direktive**: one menjaju izgled i ponašanje HTML elemenata pomoću HTML atributa.

Komponente

- Svaka Angular aplikacija ima najmanje jednu (root) komponentu, koja povezuje hijerarhiju komponenti sa DOM (document object model) te veb stranice
- Svaka komponenta definiše klasu koja sadrži podatke aplikacije i logiku, i uvezana je sa HTML šablonom (template) tako da definiše pogled koji će biti prikazan
- Sinktaksa:

```
@Component({
    selector:     'app-hero-list',
    templateUrl:  './hero-list-component.html',
    providers:    [ HeroService ]
})
```

Šabloni i povezivanje podataka

- Ugrađeno povezivanje podataka iz komponente sa elementima koji su prikazani na veb stranici (eng. *two-way data binding*)
- Direktive šablona obezbeđuju programsku logiku, a vezivanje povezuje podatke aplikacije i DOM
  - Kada se menjaju podaci na veb stranici, odnosno korisnik nešto unosi, ažuriraju se podaci u modelu (*event binding*)
  - Kada se menjaju podaci u modelu, automatski se ažuriraju sadržaj i novo izračunate vrednosti unutar veb stranice, odnosno podaci se interpoliraju u HTML (*property binding*)

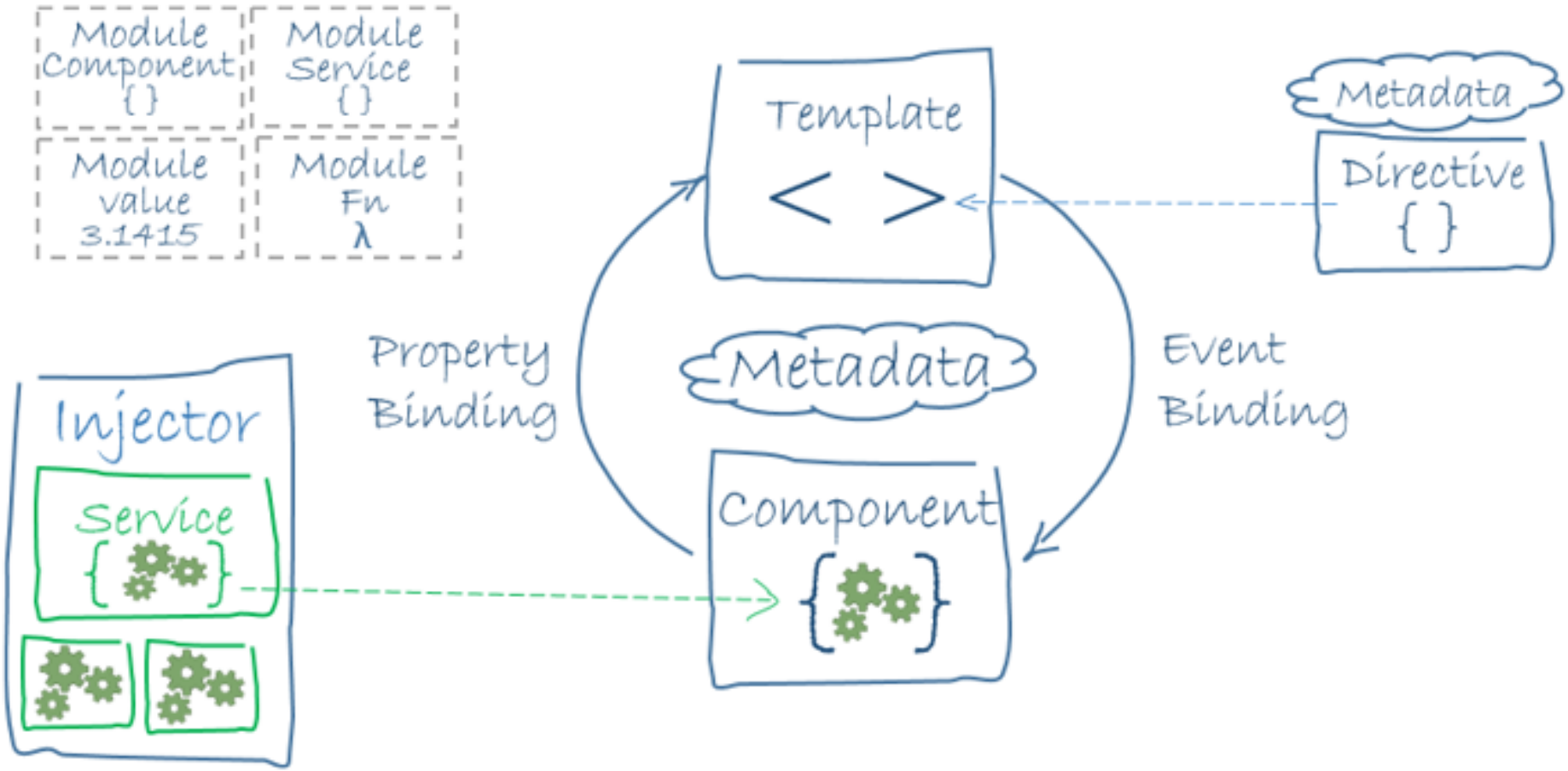
Servisi i *Dependency injection*

- Za podatke ili logiku koja nije povezana sa određenim prikazom, a koju želimo da podelimo sa više različitih komponenti, kreiramo servisnu klasu (servis)
- *Dependency injection* - proces u kome jedna komponenta definiše zavisnost od drugih komponenata
- Definiciji servisne klase uvek prethodi dekorator @Injectable( )
- Dekorator pruža metapodatke koji omogućavaju servisu da ubaci u neku komponentu klijenta zavisnost
  (npr. komponenti koja pristupa veb serveru pomoću HTTP request, ubaciti HTTP servise u komponentu)

Rutiranje

- Ruter NgModule omogućava da definišemo putanju za navigaciju između različitih stanja aplikacije i hijerarhije pogleda u aplikaciji.
- Navigaciona konvencija:
  - Uneta URL adresa u veb pregledaču (adresnom polju) vodi do odgovarajuće veb stranice
  - Linkovi na veb stranicama vode do nove stranice
  - Korak nazad/korak napred navigira nas kroz istorijat veb stranica
- Kada ruter utvrdi da treutno stanje aplikacije zahteva određenu funkcionalnost, a modul koji ga definiše nije učitan, ruter može pokrenuti učitavanje modula na zahtev.
- Pravila navigacije: povezati navigacionu putanju sa vašim realizovanim komponentama

Dijagram Angular steka



Podela odgovornosti

- Pravila koja treba pratiti prilikom implementacije Angulara:
  - Prikaz fukcioniše kao zvanična prezentaciona struktura za aplikaciju. Sve logike aplikacije treba označiti kao direktive u HTML šablonu.
  - Ako želite da izvršite manipulaciju nad DOM, izvršite je samo u JavaScript kodu ugrađene ili prilagođene direktive
  - Implementirajte zadatke koje možete da izvršite više puta kao servise i dodajte ih u module pomoću *Dependency Injection*.
  - Uverite se da metapodaci odražavaju trenutno stanje modela i da predstavljaju jedinstveni izvor za podatke koje se koriste u prikazu.
  - Definišite kontrolere unutar modula imenskog prostora da biste lako mogli da izvršite "pakovanje aplikacije".

Angular CLI

- Dobar alat za proces izrade nove Angular aplikacije
- CLI brzo generiše nove komponente, direktive, procesne tokove, servise
- Jedan scenario:

```
npm install -g @angular/cli
ng new moja-aplikacija
cd moja-aplikacija
ng serve --open
```

- Otvara se http://localhost:4200
- https://angular.io/guide/quickstart

Najvažnije komande u Angular CLI

Komanda	Alt. naziv	Namena
ng new		Kreira novu Angular aplikaciju
ng serve		Gradi aplikaciju i pokreće je za testiranje
ng eject		Omogućava uređivanje fajlova webpack config
ng generate component [ime]	ng g c [ime]	Kreira novu komponentu
ng generate directive [ime]	ng g d [ime]	Kreira novu direktivu
ng generate module [ime]	ng g m [ime]	Kreia modul
ng generate pipe [ime]	ng g p [ime]	Kreira procesni tok
ng generate service [ime]	ng g s [ime]	Kreira servis
ng generate enum [ime]	ng g e [ime]	Kreira enumeraciju
ng generate guard [ime]	ng g g [ime]	Kreira zaštitu
ng generate interface [name]	ng g i [ime]	Kreira interfejs

Prikaz komponente

```
01 import {Component} from '@angular/core';
02 @Component ({
03   selector: 'message',
04   template: `
05       <h1>Zdravo!</h1>
06   `,
07  })
08 export class Chap3Component{
09    title = "Moja prva aplikacija";
10 }
```

Glavni (koreni) modul aplikacije

- Prilikom kreiranja nove angular aplikacije angular automatski generise glavni(koreni, root) modul aplikacije AppModule
- Da bi neka klasa bila modul, mora da ima anotaciju @NgModule
- Prilikom izrade aplikacije, učitavamo različite eksterne module (druge biblioteke) u postojeći modul (aplikaciju)
- Pogledati: **app.module.ts**

Upotreba dekoratora NgModule

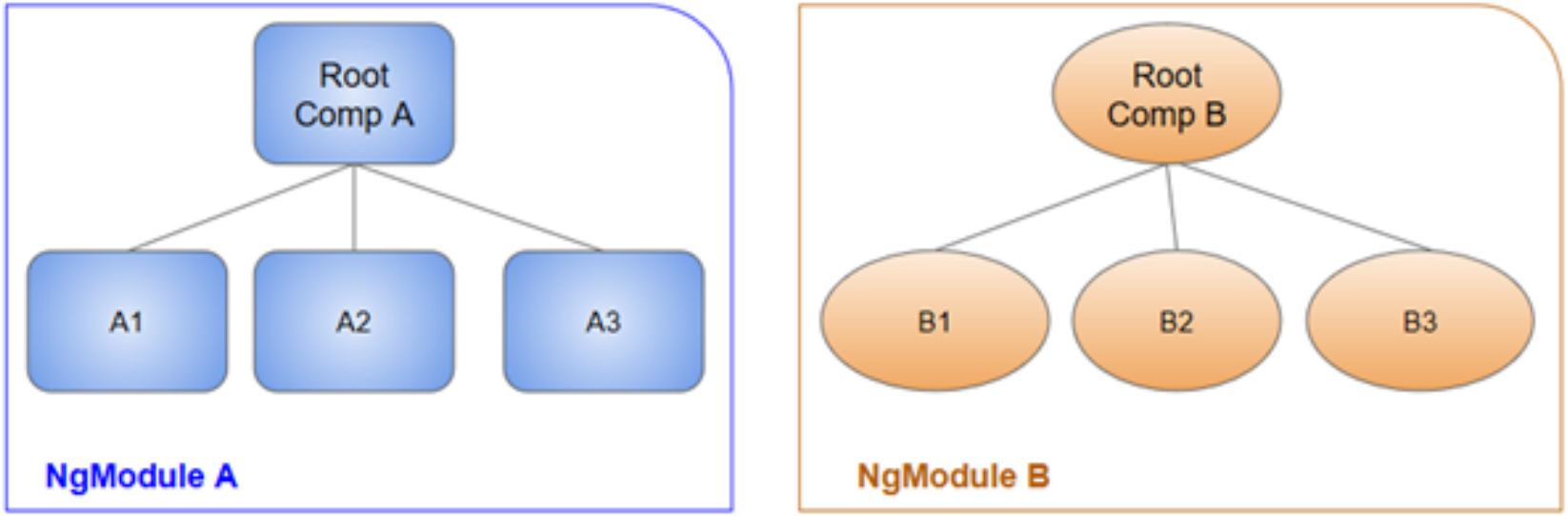
```
//fajl: app.module.ts

import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { MojaKomponentaComponent } from './moja-komponenta/moja-komponenta.component';

@NgModule({
        declarations: [    AppComponent,    MojaKomponentaComponent  ],
        imports: [    BrowserModule,    AppRoutingModule  ],
        providers: [],
        bootstrap: [AppComponent, MojaKomponentaComponent]
})
export class AppModule { }
```

Razumevanje dekoratora NgModule(1)

- `declarations` - niz komponenata, direktiva i procesnih tokova (*pipes*) koji pripada trenutnom modulu (*NgModule*).
- `imports` - niz direktiva, procesnih tokova i/ili komponenata koji će biti izvezeni iz drugih modula, i biće deklarisani u šablonima u okviru trenutnog modula.
- `exports` - podskup deklaracija koje će se koristiti u šablonima komponente drugih modula.
- `providers` - kreatori servisa kojima trenutni *NgModule* doprinosi globalnoj kolekciji servisa; oni postaju dostupni u svim delovima aplikacije.
- `bootstrap` - niz komponenata koje će biti pokrenute kada se pokrene trenutni modul (i te komponente biće ubačene u DOM veb pregledača)

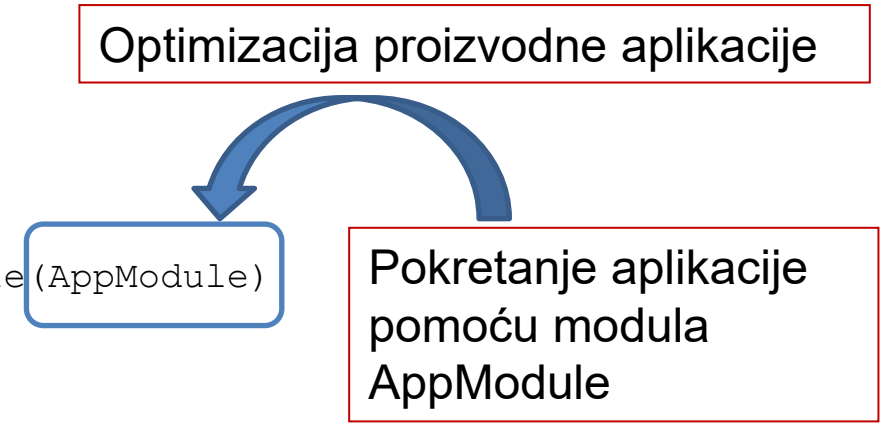
Razumevanje dekoratora NgModule(2)

- *NgModule* obezbeđuje kontekst kompilacije za svoje komponente.
- Osnovni *NgModule* uvek ima korenu (*root*) komponentu, koja se kreira tokom pokretanja, ali svaki *NgModule* može da uključi koliko god želi dodatnih komponenti.
- Cilj *NgModule* je da se prikuplja srodni kod u funkcionalne skupove.
- Komponente se mogu učitati preko rutera, ili kreirati preko šablona.



Kreiranje pokretača Angulara

```
//fajl: main.ts

import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app/app.module';
import { environment } from './environments/environment';

if (environment.production) {
   enableProdMode();
}

platformBrowserDynamic().bootstrapModule(AppModule)
   .catch(err => console.error(err));
```



- Aplikacija uvek ima najmanje osnovni modul (za pokretanje sistema) i obično ima još nekoliko modula.

Angular / Angular 2+

Angular komponente

Šta su komponente?

- Angular komponente - gradivni blokovi koji se koriste za kreiranje aplikacije
- Omogućavaju da se naprave samostalni UI elementi i da se kontroliše izgled i funkcije pomoću *TypeScript* koda i i šablona
- Angular komponenta sadrži 2 dela: dekorator i klasu.
- Dekorator služi za konfigurisanje komponente, uključujući naziv selektora i *HTML* šablon.
- Klasa sadrži podatke i omogućava da komponenti dodelite logiku, ponašanje i hendlere događaja i da je izvezemo u druge *TypeScript* fajlove.
- Klasa je povezana sa *HTML* šablonom koji definiše prikaz koji će biti prikazan u ciljnom okruženju.

Primer i konfiguracija komponente

```
import { Component } from '@angular/core';
@Component({
      selector: 'app-root',
      template: `
         <h1>{{title}}</h1>
         <h2>My favorite hero is: {{myHero}}</h2>
         `
})
export class AppComponent {
      title = 'Tour of Heroes’;
      myHero = 'Windstorm';
}
```

Prikaz opcija kod komponente

- *selector* – definiše naziv HTML taga koji se koristi za dodavanje komponente u aplikaciju pomoću HTML-a
- *template* – dodavanje ugrađene HTML da bismo definisali izgled komponente, korisna kada nemamo mnogo koda i dodatnih fajlova
- *tempateUrl* – služi za uvoz spoljnih šablona, umesto ugrađenog HTML (korisno za izdvajanje velike količine HTML koda)
- *styles* – dodavanje ugrađenog CSS u komponentu
- *stylesUrls* – omogućava uvoz niza spoljnih CSS opisa stilova
- *viewProviders* – niz provajdera za injektiranje zavisnosti, koji služi da koristimo već razvijene Angular servise

Definisanje selektora

- Primena komponente u HTML stranicama.
- Naziv selektora se upotrebljava kao naziv HTML taga.
- Nije dozvoljeno da koristimo razmake u nazivu selektora.
- Primer:

```
@Component({
    selector: 'moja-komponenta'
})
```

Primena u HTML:

```
<moja-komponenta> </moja-komponenta>
```

Izrada šablona

- Služe da definišu izgled *Angular* komponente, pišu se kao *HTML*.
- *Angular* omogućava da se koriste ugrađeni šabloni ili spoljne datoteke šablona.
- Šablon se dodaje u dekorator *@Component*.
- Jednoredni šablon može da koristi jednostruke (apostrofe) ili dvostruke navodnike, a višeredni šablon koristi obrnute navodnike `
- Na ugrađene *CSS* stilove se primenjuju ista pravila kao i na standardne *CSS* šablone

```
styles: [`
    p {
       color: yellow;
       font-size: 25px;
    }
`]
```

Upotreba konstruktora

- Konstruktori – za dodelu podrazumevanih vrednosti komponente.
- Kada se promenljive koriste u komponenti, uvek će biti inicijalizovane.
- Primer sintakse:

```
export class Student {
    name: string;
    constructor(){
       this.name = "Drazen";
    }
}
```

Primer komponente sa konstruktorom

```
import {Component} from '@angular/core';
@Component ({
    selector: 'simple-constructor',
    template: `
            <p>Danas je {{today}}!</p>
    `
})
export class KoriscenjeKonstruktora{
    today: Date;
    constructor(){
            this.today = new Date();
    }
}
```

Upotreba spoljnih šablona

- Upotreba zasebnih datoteka u okviru naše.
- Prednost: lakše se određuje namena datoteke, lakše čitanje komponente, manji broj linija koda.
- Primer:

```
@Component({
        selector: 'my-app',
        templateUrl: "./view.example.html",
        styleUrl: ["./stilovi1.css",
                   "./stilovi2.css"
        ]
})
```

Injektiranje direktiva

- Injektiranje zavisnosti definiše i dinamički injektira objekat zavisnosti u drugi objekat, pa sve funkcije koje obezbeđuje objekat zavisnosti postaju dostupne.
- *Angular* obezbeđuje injektiranje zavisnosti pomoću provajdera i servisa injektora.
- Da bi se koristilo injektiranje zavisnosti na drugoj direktivi ili komponenti, potrebno je da se u okviru modula za aplikaciju doda naziv klase direktive ili komponente u metapodatke *declarations* u dekoratoru *@NgModule*, čime se niz direktiva uvozi u aplikaciju.

Prosleđivanje podataka pomoću injektiranja zavisnosti

- Da bi se u Angularu uneli podaci u drugu direktivu ili komponentu, potrebno je da se uveze dekorator Input iz paketa @angular/core:

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

- Kada se uveze ovaj decorator, možemo da počnemo da definišemo podatke koje ćemo da unesemo u direktivu. Definisanje dekoratora se radi korišćenjem @input(), koji koristi string podatak kao parametar.
- Primer:

```
@Input('name') personName: string;
```

Primer

```
import { Component } from '@angular/core';
import {myInput} from './input.component';
@Component({
   selector: 'app-root',
   template: `
   <myInput name="Brendan" occupation="Student/Author"></myInput>
   <myInput name="Brad" occupation="Analyst/Author"></myInput>
   <myInput name="Caleb" occupation="Student/Author"></myInput>
   <myInput></myInput>
   `
})
   export class AppComponent {
   title = 'Using Inputs in Angular';
}
```

Primer

```
01 import {Component, Input}
          from '@angular/core';
02 @Component ({
03 selector: "myInput",
04 template: `
05 <div>
06    Name: {{personName}}
07    <br/>
08     Job: {{occupation}}
09 </div>
10 `,
11 styles: [`
12 div {
13    margin: 10px;
14    padding: 15px;
15    border: 3px solid grey;
16 }
17 `]
18 })
19 export class myInputs {
20 @Input('name') personName:
                    string;
21 @Input('occupation')
              occupation: string;
22 constructor() {
23  this.personName = "John Doe";
24  this.occupation = "Anonymity"
25  }
26 }
```

Angular / Angular 2+

Rutiranje

Definisanje bazične rute

- Dodavanje *AppRoutingModule* u AppModule i niz *imports*.
- Rute se dodaju u modul:

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

const routes: Routes = [
      { path: 'first-component', component: FirstComponent },
      { path: 'second-component', component: SecondComponent },
]; // sets up routes constant where you define your routes

// configures NgModule imports and exports
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

```
<h1>Primer rutiranja</h1>
<nav>
  <ul>
    <li><a routerLink="/first-component" routerLinkActive="active"
          ariaCurrentWhenActive="page">First Component</a></li>
    <li><a routerLink="/second-component" routerLinkActive="active"
          ariaCurrentWhenActive="page">Second Component</a></li>
  </ul>
</nav>
<!-- The routed views render in the <router-outlet>-->
<router-outlet></router-outlet>
```

Džoker rute - "wildcard"

- Dodavanje dve zvezdice:

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

const routes: Routes = [
      { path: 'first-component', component: FirstComponent },
      { path: 'second-component', component: SecondComponent },
      { path: '**', component: PageNotFoundComponent },
]; // sets up routes constant where you define your routes
```

Dva asteriks znaka (**) označavaju džoker rutu.

Uobičajni izbor uključuje "PageNotFoundComponent" koja definiše prikaz 404 stranice, ili se redirektuje na glavnu komponentu aplikacije.

Ova ruta se navodi kao poslednja, jer odgovara bilo kojoj URL adresi (koja nije navedena među rutama).

Postavljanje preusmeravanja

- Preusmeravanje (redirekcija) ka komponenti:

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

const routes: Routes = [
      { path: 'first-component', component: FirstComponent },
      { path: 'second-component', component: SecondComponent },
      { path: '',   redirectTo: '/first-component', pathMatch: 'full' },
      { path: '**', component: PageNotFoundComponent },
]; // sets up routes constant where you define your routes
```

path - putanja sa koje želimo da se preusmerimo

redirectTo - komponenta na koju se preusmeravamo

pathMatch - vrednost koja govori ruteru u kojoj meri treba da se podudara sa URL putanjom

Ugneždene rute

- Komponente koje su relativne u odnosu na komponentu koja nije vaša osnovna komponenta:

```
const routes: Routes = [
  {
    path: 'first-component',
    title: 'First component',
    component: FirstComponent,  // this is the component with the <router-outlet> in the template
    children: [
      {
        path: 'child-a',  // child route path
        title: resolvedChildATitle,
        component: ChildAComponent,  // child route component that the router renders
      },
      {
        path: 'child-b',
        title: 'child b',
        component: ChildBComponent,  // another child route component that the router renders
      },
    ],
  },
];

const resolvedChildATitle: ResolveFn<string> = () => Promise.resolve('child a');
```

Angular / Angular 2+

Angular izrazi

Upotreba izraza (1)

- Najjednostavniji način za prikaz podataka iz komponente u prozor Angulara je korišćenje izraza. Izrazi su kapsulirani blokovi koda, koji su smešteni unutar zagrada:
  {{expression}}
- Angular kompajler kompajlira izraz u HTML elemente, pa će biti prikazani rezultati izraza
- Primeri:
  {{1+5}}
  {{'ETF' + 'Beograd'}}
- Rezultati izvršavanja ovih izraza:
  6
  ETFBeograd

Upotreba izraza (2)

- Izrazi su povezani sa modelom podataka
- Prva pogodnost:
  možemo koristiti nazive i funkcije koje su definisane u komponenti unutar izraza
- Druga pogodnost:
  izrazi povezani sa komponentom, pa se menjaju kada se menjaju i podaci u komponenti.
- Primer:
  name: string='Drazen';
  score: number=95;
- Direktno ukazivanje na vrednosti name i score:
  Ime: {{name}}
  Ukupno: {{score}}
  Ukupno sa bonusom: {{score+5}}

Sličnost sa TypeScript/JavaScript

- Angular izrazi se razlikuju od TS/JS izraza u sledećem:
  - Izračunavanje atributa - nazivi svojstva u Angularu se izračunavaju prema modelu komponente, umesto prema imenskom prostoru JavaScript-a
  - Angular izrazi su jednostavniji za upotrebu - Ne generišu greške kada se susretnu sa nedefinisanim ili nevažećim tipom promenljivih, već ih tretiraju kao promenljive koje nemaju vrednost.
  - Angular izrazi ne kontrolišu protok - Izrazi onemogućavaju sledeće:
    - dodele (npr. =, +=, -=)
    - operator new
    - uslove
    - petlje
    - inkrementiranje i dekrementiranje operatora (++ i --)
  - Ne mogu se generisati greške unutar izraza.

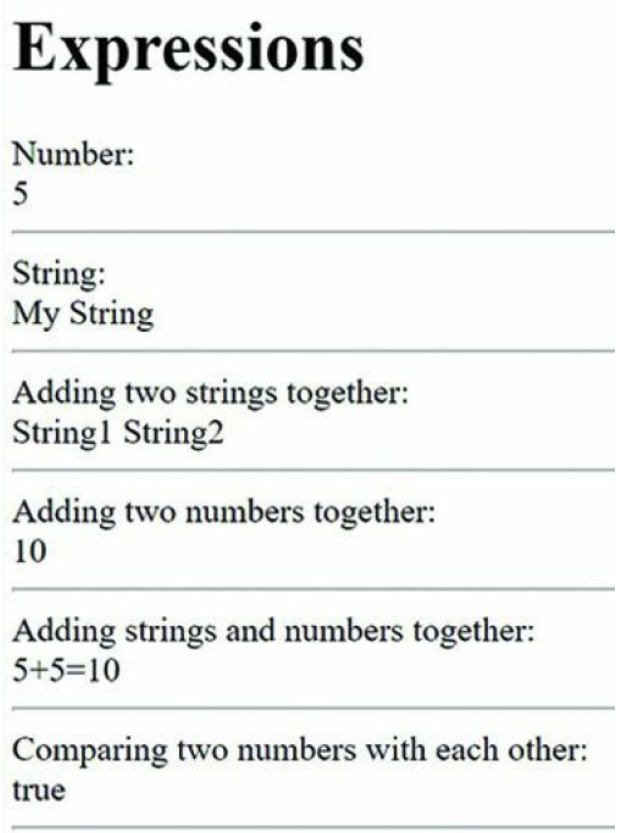
Izrazi kao stringovi

- Angular izračunava izraze kao stringove koji se koriste za definisanje vrednosti direktiva.
- Kada bi se postavila vrednost direktive ng-click u šablon, ona određuje izraz
- Unutar izraza može da se ukaže na promenljivu komponente ili upotrebi neka druga sintaksa izraza, na primer:

- Kako izrazi šablona mogu da pristupe komponenti, znači da je i moguća izmena komponente unutar Angular izraza
- Primer direktive (click) koja menja vrednost msg unutar modela komponente:
  <span (click)="msg='clicked'">

Primer upotrebe izraza

```
import { Component } from '@angular/core';
@Component({
                selector: 'app-root',
                template: `
                  <h1>Expressions</h1>
                  Number:<br>
                  {{5}}<hr>
                  String:<br>
                  {{'My String'}}<hr>
                  Adding two strings together:<br>
                  {{'String1' + ' ' + 'String2'}}<hr>
                  Adding two numbers together:<br>
                  {{5+5}}<hr>
                  Adding strings and numbers together:<br>
                  {{5 + '+' + 5 + '='}}{{5+5}}<hr>
                  Comparing two numbers with each other:<br>
                  {{5===5}}<hr>
                `,
})
export class AppComponent {}
```



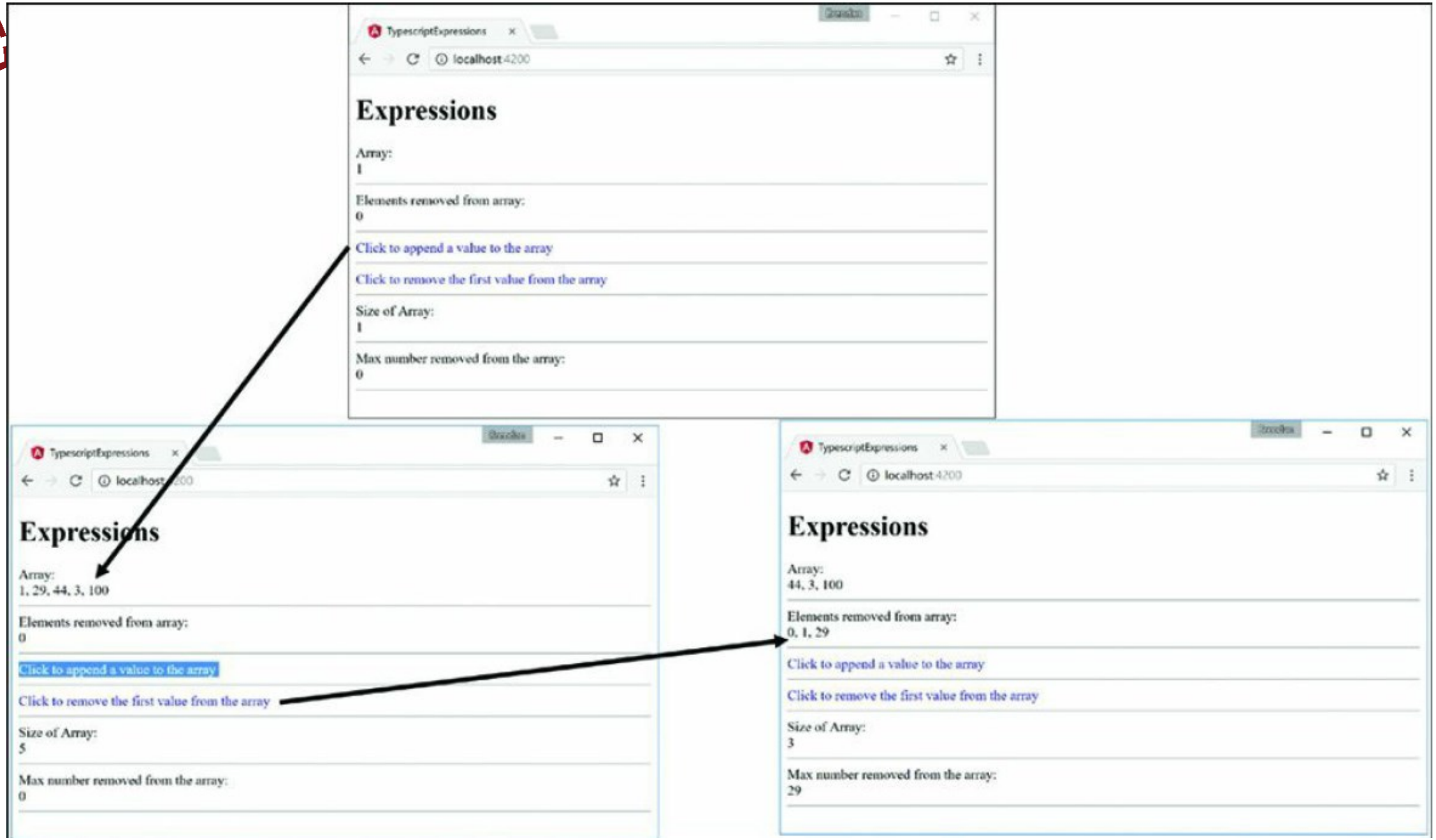
Interakcija sa klasom Component u izrazima

```
01 import { Component } from '@angular/core';
02
03 @Component({
04   selector: 'app-root',
05   template: `
06     Directly accessing variables in the component:<br>
07       {{speed}} {{vehicle}}<hr>
08     Adding variables in the component:<br>
09       {{speed + ' ' + vehicle}}<hr>
10     Calling function in the component:<br>
11       {{lower(speed)}} {{upper('Jeep')}}<hr>
12     <a (click)="setValues('Fast', newVehicle)">
13       Click to change to Fast {{newVehicle}}</a><hr>
14     <a (click)="setValues(newSpeed, 'Rocket')">
15       Click to change to {{newSpeed}} Rocket</a><hr>
16     <a (click)="vehicle='Car'">
17       Click to change the vehicle to a Car</a><hr>
18     <a (click)="vehicle='Enhanced ' + vehicle">
19       Click to Enhance Vehicle</a><hr>
20   `,
21   styles:[`
22     a{color: blue; text-decoration: underline; cursor: pointer}
23   `]
24 })
```

```
25 export class AppComponent {
26   speed = 'Slow';
27   vehicle = 'Train';
28   newSpeed = 'Hypersonic';
29   newVehicle = 'Plane';
30   upper = function(str: any){
31     str = str.toUpperCase();
32     return str;
33   }
34   lower = function(str: any){
35     return str.toLowerCase();
36   }
37   setValues = function(speed: any, vehicle: any){
38     this.speed = speed;
39     this.vehicle = vehicle;

40   }
41 }
```

Upotreba TypeScript u Angular izrazima

```
01 import { Component } from '@angular/core';
02
03 @Component({
04   selector: 'app-root',
05   template: `
06     <h1>Expressions</h1>
07     Array:<br>
08       {{myArr.join(', ')}}<br/>
09       <hr>
10     Elements removed from array:<br>
11       {{removedArr.join(', ')}}<hr>
12     <a (click)="myArr.push(myMath.floor(myMath.random()*100+1))"
13       Click to append a value to the array
14     </a><hr>
15     <a (click)="removedArr.push(myArr.shift())">
16       Click to remove the first value from the array
17     </a><hr>
18     Size of Array:<br>
19       {{myArr.length}}<hr>
20     Max number removed from the array:<br>
21       {{myMath.max.apply(myMath, removedArr)}}<hr>
22   `,
23   styles: [`
24     a {
25       color: blue;
26       cursor: pointer;
27     }
28   `],
29 })
30 export class AppComponent {
31   myMath = Math;
32   myArr: number[] = [1];
33   removedArr: number[] = [0];
34 }
```


TypescriptExpressions
localhost:4200
Expressions
Array:
1
Elements removed from array:
0
Click to append a value to the array
Click to remove the first value from the array
Size of Array:
1
Max number removed from the array:
0
TypescriptExpressions
localhost:4200
Expressions
Array:
1, 29, 44, 3, 100
Elements removed from array:
0
Click to append a value to the array
Click to remove the first value from the array
Size of Array:
5
Max number removed from the array:
0
TypescriptExpressions
localhost:4200
Expressions
Array:
44, 3, 100
Elements removed from array:
0, 1, 29
Click to append a value to the array
Click to remove the first value from the array
Size of Array:
3
Max number removed from the array:
29

Upotreba vertikalnih crta

- Vertikalne crte se implementiraju unutar izraza pomoću sledeće sintakse:
  {{ expression | pipe }}
- Ako spojimo više operatora vertikalne crte, oni se izvršavaju po redosledu po kojem smo ih odredili:
  {{ expression | pipe | pipe }}
- Neki filteri omogućavaju da unosi budu u obliku parametara funkcije:
  {{ expression | pipe:parameter1:parameter2 }}
- Angular omogućava nekoliko tipova operatora vertikalne crte koji omogućavaju da se lako formatiraju stringovi, objekti i nizovi u šablonima komponente.

Filter	Description
`currency[ :currencyCode?[ :symbolDisplay?[ :digits?]]]`	Formats a number as currency, based on the `currencyCode` value provided. If no `currencyCode` value is provided, the default code for the locale is used. Here is an example: `{{123.46 | currency:"USD" }}`
`json`	Formats a TypeScript object into a JSON string. Here is an example: `{{ {'name':'Brad'} | json }}`

`slice:start:end`	Limits the data represented in the expression by the indexed amount. If the expression is a string, it is limited in the number of characters. If the result of the expression is an array, it is limited in the number of elements. Consider these examples:
	`{{ "Fuzzy Wuzzy" | slice:1:9 }}`
	`{{ ['a','b','c','d'] | slice:0:2 }}`

`lowercase`	Outputs the result of the expression as lowercase.
`uppercase`	Outputs the result of the expression as uppercase.

`number[:pre.post-postEnd]`	Formats the number as text. If a `pre` parameter is specified, the number of whole numbers is limited to that size. If `post-postEnd` is specified, the number of decimal places displayed is limited to that range or size. Consider these examples:
	`{{ 123.4567 | number:1.2-3 }}`
	`{{ 123.4567 | number:1.3 }}`

`date[:format]`

Formats a TypeScript date object, a timestamp, or an ISO 8601 date string, using the *`format`* parameter. Here is an example:

```
{{1389323623006 | date:'yyyy-MM-dd HH:mm:ss Z'}}
```

The *`format`* parameter uses the following date formatting characters:

- **`yyyy`**: Four-digit year
- **`yy`**: Two-digit year
- **`MMMM`**: Month in year, `January` through `December`
- **`MMM`**: Month in year, `Jan` through `Dec`
- **`MM`**: Month in year, padded, `01` through `12`
- **`M`**: Month in year, `1` through `12`
- **`dd`**: Day in month, padded, `01` through `31`
- **`d`**: Day in month, `1` through `31`
- **`EEEE`**: Day in week, `Sunday` through `Saturday`
- **`EEE`**: Day in Week, `Sun` through `Sat`
- **`HH`**: Hour in day, padded, `00` through `23`
- **`H`**: Hour in day, `0` through `23`
- **`hh`** or **`jj`**: Hour in a.m./p.m., padded, `01` through `12`
- **`h`** or **`j`**: Hour in a.m./p.m., `1` through `12`
- **`mm`**: Minute in hour, padded, `00` through `59`
- **`m`**: Minute in hour, `0` through `59`
- **`ss`**: Second in minute, padded, `00` through `59`
- **`s`**: Second in minute, `0` through `59`
- **`.sss`** or **`,sss`**: Millisecond in second, padded, `000–999`
- **`a`**: a.m./p.m. marker

- **Z**: Four-digit time zone offset, `-1200` through `+1200`

The *`format`* string for `date` can also be one of the following predefined names:

- **`medium`**: Same as `'yMMMdHms'`
- **`short`**: same as `'yMdhm'`
- **`fullDate`**: same as `'yMMMMEEEEd'`
- **`longDate`**: same as `'yMMMMd'`
- **`mediumDate`**: same as `'yMMMd'`
- **`shortDate`**: same as `'yMd'`
- **`mediumTime`**: same as `'hms'`
- **`shortTime`**: same as `'hm'`

The format shown here is `en_US`, but the format always matches the locale of the Angular application.

`async`	Waits for a promise and returns the most recent value received. It then updates the view.

Angular / Angular 2+

Povezivanje podataka

Povezivanje podataka

- Proces povezivanja podataka iz komponente, sa onim što se prikazuje na veb stranici
- Kada se podaci u komponenti izmene => korisnički interfejs koji se prikazuje korisniku se automatski ažurira
- Model je jedini uvek izvor podataka, prikaz je samo projekcija modela

Tipovi povezivanja

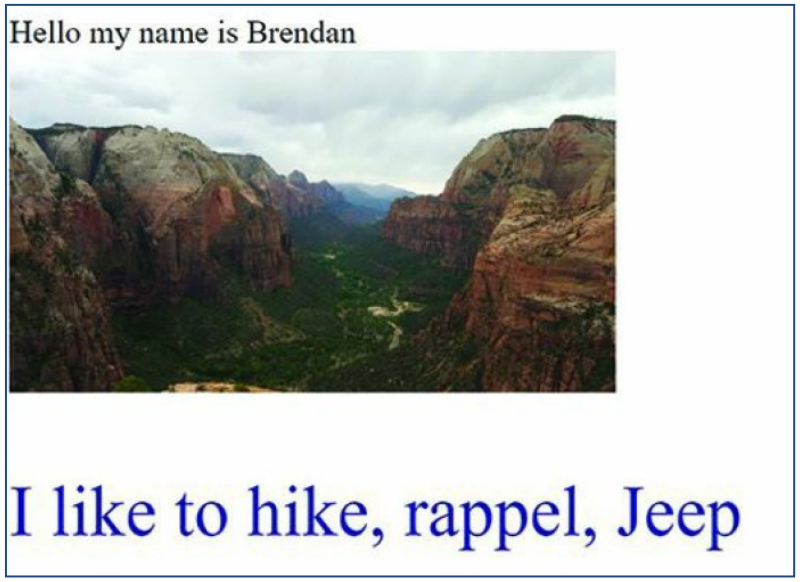
- interpolacija (*interpolation*) - koristimo duple vitičaste zagrade {{ }}
  za dobijanje vrednosti direktno iz klase *Component*
- **povezivanje svojstava (*property binding*) - koristimo za postavljanje HTML elemenata**
- **povezivanje događaja (*event binding*) - koristimo za upravljanje korisničkim unosima**
- povezivanje atributa (*attribute binding*) - omogućava da podesimo atribute HTML elemenata
- povezivanje klasa (*class binding*) - koristimo za postavljanje naziva CSS klase elementa
- povezivanje stilova (*style binding*) - koristimo za kreiranje ugrađenih CSS stilova elementa
- dvosmerno povezivanje (*two way binding*) pomoću direktive ngModel - koristimo u obrascima za unos podataka radi prijema i prikaza podataka

Interpolacija

- Koristi se {{ }} za izračunavanje izraza šablona
- Interpolacija može biti ugrađena u izvorni kod ili može da bude referenca svojstva klase *Component*
- Primer sintakse za zadavanje vrednosti:

- Primer interpolacije u kojoj se koriste stringovi i funkcija

```
import { Component } from '@angular/core';
@Component({
   selector: 'app-root',
   template: `
   {{str1 + ' ' + name}}
   <br>
   <img src="{{imageSrc}}" />
   <br>
   <p>{{str2 + getLikes(likes)}}</p>
   `,
   styles: [`
   img{
    width: 300px;
    height: auto;
   }
   p{
    font-size: 35px;
    color: darkBlue;
   }
   `]
 })
```




```
export class AppComponent {
   str1: string = "Hello my name is"
   name: string = "Drazen"
   str2: string = "I like to"
   likes: string[] = ['hike',
                'swim', 'play piano']
   getLikes = function(arr: any){
     var arrString = arr.join(", ");
     return " " + arrString
   }
   imageSrc: string =
    "../assets/images/mojaSlika.jpg"
 }
```

Povezivanje svojstava

- Povezivanje svojstava se koristi kada treba da postavite svojstvo HTML elementa. Svojstvo ćete postaviti tako što se definiše vrednost koju želite u okviru klase Component.
- Često interpolacija može da postigne isti rezultat kao i prilikom povezivanja svojstava.
- Primer povezivanja svojstava i primene naziva klase

```
import { Component } from '@angular/core';
@Component({
selector: 'app-root',
template: `
   <img [src]="myPic"/>
   <br>
   <button [disabled]="isEnabled">Click me</button><hr>
   <button disabled="{!isEnabled}">Click me</button><br>
   <p [ngClass]="className">This is cool stuff</p>
`,
styles: [`
   img {
     height: 100px;
     width auto;
   }
   .myClass {
     color: red;
     font-size: 24px;
   }
`]
})
```

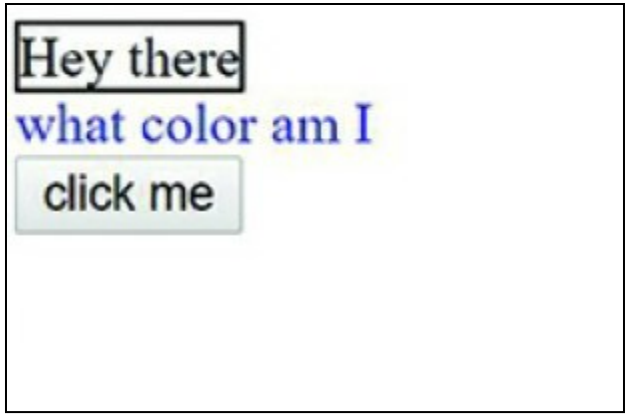

```
export class AppComponent {
   myPic: string =
"../assets/images/sunset.JPG";
   isEnabled: boolean = false;
   className: string =
"myClass";
}
```

Povezivanje stilova

- Definiše se CSS svojstvo stila u zagradama sa izrazom dodele koji se stavlja pod navodike
- Sintaksa je slična za povezivanje klase, ali se koristi prefiks *style*, umesto prefiksa *class*
- *Primer:*

```
<p [style.styleProperty] = "assignment"></p>
<div [style.backgroundColor] = "'green'"></div>
```

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

@Component({
selector: 'app-root',
template: `
    <span [style.border]="myBorder">Hey there</span>
    <div [style.color]="twoColors ? 'blue' : 'forestgreen'">
        what color am I
    </div>
    <button (click)="changeColor()">click me</button>
`
})
export class AppComponent {
   twoColors: boolean = true;
   changeColor = function(){
     this.twoColors = !this.twoColors;
   }
   myBorder = "1px solid black";
}
```



Povezivanje događaja

- Koristi se za upravljanje korisničkim unosima, kao što su: klik na dugme, pritisci na taster tastature i pomeranje miša.
- Povezivanje događaja je kao povezivanje HTML-a i JavaScript-a samo što se uklanja prefiks "on" iz povezivanja i što se događaj potavlja u zagradama, npr: (click) ili (keyup).
- Povezivanje događaja služi za pokretanje funkcija iz komponente.
- Primer:

```
<button (click)="mojaFunkcija()">Dugme</button>
```

Primer (1/4)

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

@Component({
selector: 'app-root',
template: `
   <div (mousemove)="move($event)">
   <img [src]="imageUrl"
   (mouseenter)="mouseGoesIn()"
   (mouseleave)="mouseLeft()"
   (dblclick)="changeImg()" /><br>
   double click the picture to change it<br>
   The Mouse has {{mouse}}<hr>
   <button (click)="changeImg()">Change Picture</button><hr>
   <input (keyup)="onKeyup($event)"
   (keydown)="onKeydown($event)"
   (keypress)="keypress($event)"
   (blur)="underTheScope($event)"
   (focus)="underTheScope($event)">
   {{view}}
```

Primer (2/4)

```
 <p>On key up: {{upValues}}</p>
   <p>on key down:
{{downValues}}</p>
   <p>on key press:
{{keypressValue}}</p>
   <p (mousemove)="move($event)">
       x coordinates: {{x}}
   <br> y coordinates: {{y}}
   </p>
   </div>
`,
styles: [`
   img {
     width: auto;
     height: 300px;
   }
`]
})
```

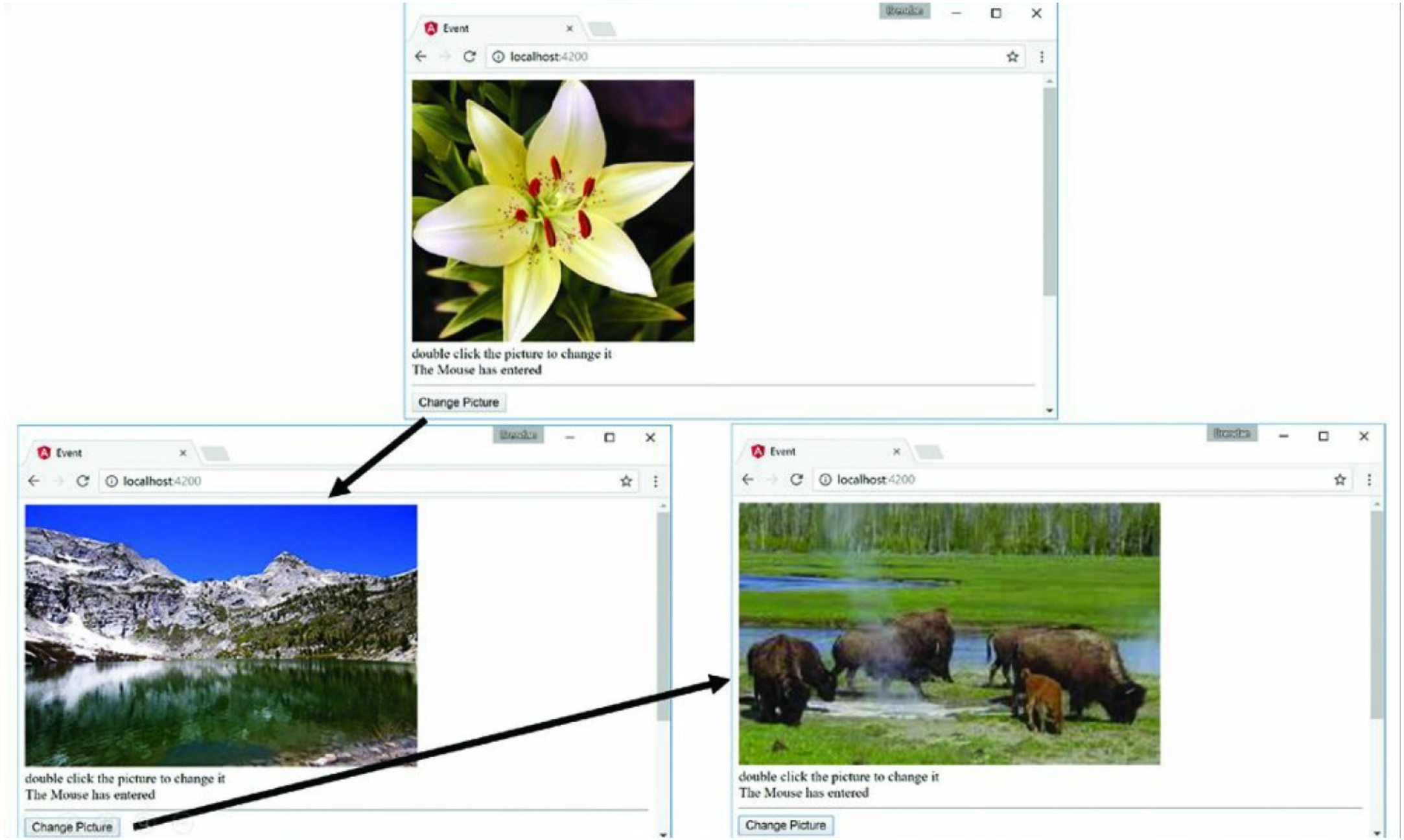
```
export class AppComponent {
   counter = 0;
   mouse: string;
   upValues: string = '';
   downValues: string = '';
   keypressValue: string = "";
   x: string = "";
   y: string = '';
   view: string = '';
mouseGoesIn = function(){
   this.mouse = "entered";
};
mouseLeft = function(){
   this.mouse = "left";
}
imageArray: string[] = [
   "../assets/images/flower.jpg",
   "../assets/images/lake.jpg",
   //extensions are case sensitive
   "../assets/images/bison.jpg",
]
```

Primer (3/4)

```
imageUrl: string = this.imageArray[this.counter];
   changeImg = function(){
     if(this.counter < this.imageArray.length - 1){
        this.counter++;
     }else{
        this.counter = 0;
     }
     this.imageUrl=this.imageArray[this.counter];
   }
onKeyup(event:any){
   this.upValues = event.key;
   //this.upValues += event.target.value + ' | ';
}
onKeydown(event:any){
   this.downValues = event.key;
   //this.downValues += event.target.value + " | ";
}
keypress(event:any){
   this.keypressValue = event.key;
   //this.keypressValue += event.target.value + " | ";
}
```

Primer (4/4)

```
move(event:any){
    this.x = event.clientX;
    this.y = event.clientY;
}
underTheScope(event:any){
    if(event.type == "focus"){
      this.view = "the text box is focused";
    }
    else if(event.type == "blur"){
      this.view = "the input box is blurred";
    }
    console.log(event);
    }
}
```

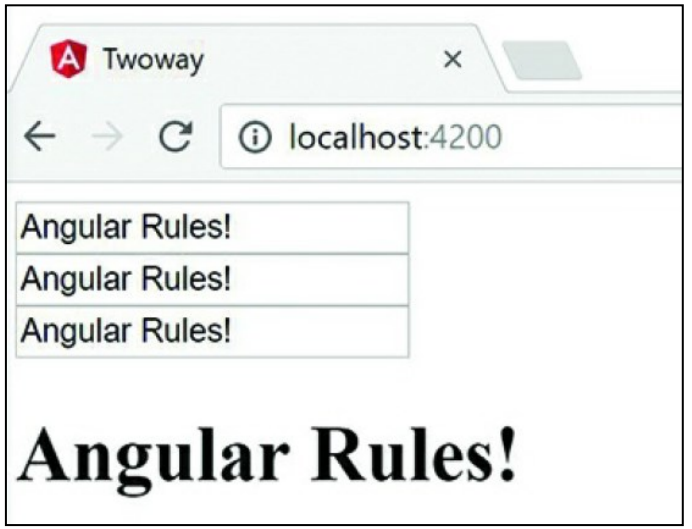

Event
localhost:4200
double click the picture to change it
The Mouse has entered
Change Picture
Event
localhost:4200
double click the picture to change it
The Mouse has entered
Change Picture
Event
localhost:4200
double click the picture to change it
The Mouse has entered
Change Picture

Dvosmerno povezivanje

- Dvosmerno povezivanje - istovremeno se podaci prikazuju i ažuriraju
- Podaci se prikazuju pomoću direktive ngModel
- Sintaksa za dvosmerno povezivanje:
  <input [(ngModel)] = "vrednost">

Primer

```
import { Component } from '@angular/core';
@Component({
selector: 'app-root',
template: `
   <input [(ngModel)]="text"><br>
   <input bindon-ngModel="text"><br>
   <input [value]="text" (input)="text=$event.target.value">
   <h1>{{text}}</h1>
`
})
export class AppComponent {
   text: string = "some text here";
}
```



Angular / Angular 2+

Ugrađene i prilagođene direktive

Razumevanje direktiva

- Direktive predstavljaju kombinaciju Angular oznaka šablona i TypeScript koda
  - Angular direktive su: HTML atributi, nazivi elemenata ili CSS klase
  - TS kod za direktive definiše podatke šablona i ponašanje HTML elemenata
- Angular kompajler pristupa DOM šablonu i kompajlira sve direktive, a zatim se povezuje sa opsegom važenja da bi bio kreiran novi prikaz u realnom vremenu
- Prikaz u realnom vremenu sadrži elemente i funkcije DOM-a koji su definisani u direktivi

Upotreba ugrađenih direktiva

- Tri kategorije Angular direktiva:
  1. Direktive komponenata - direktive sa šablonom
  2. Strukturalne direktive - direktive koje manipulišu elementima u DOM-u
  3. Atributske direktive - direktive koje manipulišu izgledom i ponašanjem elemenata DOM-a
- Direktive komponenata
  - "Srce" Angular tehnologije
  - Komponentu kreira selektor koji se koristi kao HTML tag za dinamičko dodavanje HTML, CSS i Angular logike unutar DOM

Strukturalne direktive (1)

- **ngFor** - koristi se za kreiranje kopije šablona za svaku stravku u okviru objekta kroz koji možemo da iteriramo.
  Primer:
  ```
  <div *ngFor="let stavka of listastudenata"></div>
  ```
- **ngIf** - ovom direktivom u elementu, dodajemo element u DOM ukoliko se vrati vrednost true (ako je false, taj element se uklanja iz DOM-a).
  Primer:
  ```
  <div *ngIf="uslov"></div>
  ```
- **ngSwitch** - ova direktiva prikazuje šablon na osnovu vrednosti koja mu je prosleđena. Funkcioniše slično kao ngIf, odnosno ne kreira element, ako se njegova vrednost ne podudara sa nekom granom (case-om).

Strukturalne direktive (2)

- **ngSwitch** - Primer:

```
<div [ngSwitch]="timeOfDay">
     <span [ngSwitchCase]="'morning'">Jutro</span>
     <span [ngSwitchCase]="'afternoon'">Popodne</span>
     <span [ngSwitchDefault]="'daytime'">Uvece</span>
</div>
```

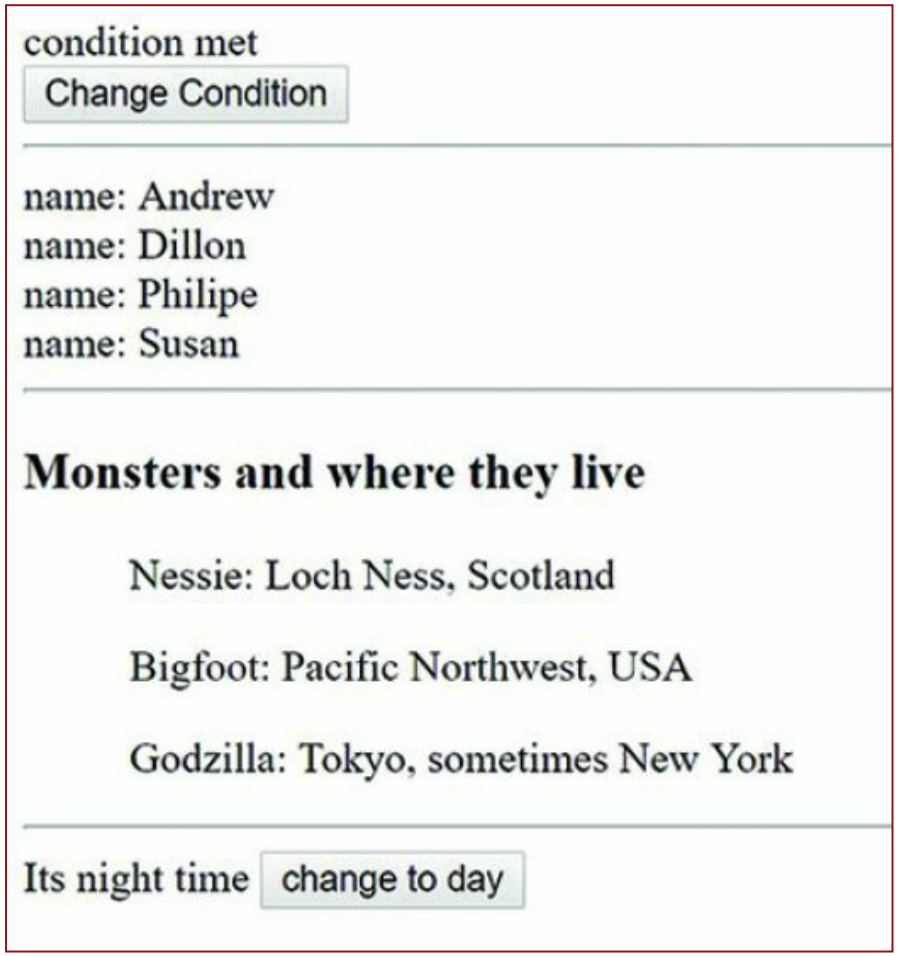
- **ngSwitchCase** - direktiva procenjuje uskladištenu vrednost i vrednost koja je prosleđena direktivi ngSwitch i utvrđuje da li je potrebno kreirati HTML šablon koji je pridružen.
- **ngSwitchDefault** - ako svi ngSwitchCase daje rezultat False

Kod ngFor i ngIf koristi se *, da bi se Angularu pokazala prisutnost directive u elementu.

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  template: `
    <div *ngIf="condition">condition met</div>
    <div *ngIf="!condition">condition not met</div>
    <button (click)="changeCondition()">Change Condition</button> <hr>
    <template ngFor let-person [ngForOf]="people">
       <div>name: {{person}}</div>
    </template>
    <hr> <h3>Monsters and where they live</h3>
    <ul *ngFor="let monster of monsters">
       {{monster.name}}:{{monster.location}}
    </ul> <hr>
    <div [ngSwitch]="time">
     <span *ngSwitchCase="'night'">It's night time
        <button (click)="changeDay()">change to day</button>
     </span>
     <span *ngSwitchDefault>It's day time
     <button (click)="changeNight()">change to night</button></span>
    </div>
  `
})
```

```
export class AppComponent {
   condition: boolean = true;
   changeCondition = function(){
      this.condition = !this.condition;
   }
   changeDay = function(){
      this.time = 'day';
   }
   changeNight = function(){
       this.time = 'night'
   }
   people: string[] = ["Andrew", "Dillon", "Philipe", "Susan"]
   monsters = [
      { name: "Nessie", location: "Loch Ness, Scotland" },
      { name: "Bigfoot", location: "Pacific Northwest, USA" },
      { name: "Godzilla", location: "Tokyo, sometimes New York" }
   ]
   time: string = 'night';
}
```

Primer upotrebe ugrađenih strukturalnih direktiva



Atributske direktive

- Angular atributske direktive modifikuju izgled HTML elemenata i njihovo ponašanje

Direktiva	Opis
ngModel	Ova direktiva prati promene u promenljivoj, a zatim ažurira vrednosti prikaza. Primer: `<input [(ngModel)]="text"> ` `<h1>{{text}}</h1>`

ngForm	Ova direktiva kreira grupu obrazaca i omogućava praćenje vrednosti i validaciju u okviru te grupe obrazaca. Pomoću direktive ngSubmit, možemo da podatke prosledimo događaju slanja kao objekat. Primer:
	`<form #formName="ngForm"`
	`(ngSubmit)="onSubmit(formName)">`
	`</form>`

ngStyle	Ova direktiva ažurira stilove HTML elementa.

Primer (TS fajl)

```
import { Component } from '@angular/core';
@Component({
   selector: 'app-root',
   templateUrl: './attribute.component.html',
   styleUrls: ['./attribute.component.css']
})

export class AppComponent {
   colors: string[] = ["red", "blue", "green", "yellow"];
   name: string;
   color: string = 'color';
   isDisabled: boolean = true;
   classes:string[] = ['bold', 'italic', 'highlight'];
   selectedClass:string[] = [];
   enabler(){
      this.isDisabled = !this.isDisabled;
   }
   addClass(event: any){
      this.selectedClass = [];
      var values = event.target.options;
      var opt: any;
```

```
        for (var i=0, iLen = values.length; i<iLen; i++){
           opt = values[i];
            if (opt.selected){
               this.selectedClass.push(opt.text);
            }
        }
    }
}
```

Primer (html fajl - Angular šablon)

```
<form>
   <span>name: </span>
   <input name="name" [(ngModel)]="name">
   <br> <span>color:</span>
   <input type="checkbox" (click)="enabler()">
   <select #optionColor [(ngModel)]="color" name="color" [disabled]="isDisabled">
      <option *ngFor="let color of colors" [value]="color">{{color}}</option>
   </select><hr>
   <span>Change Class</span><br>
   <select #classOption multiple name="styles" (change)="addClass($event)">
      <option *ngFor="let class of classes" [value]="class" >{{class}}</option>
   </select><br>
   <span>press and hold control/command <br>
   to select multiple options </span>
</form>
<hr>
<span>Name: {{name}}</span><br>
<span [ngClass]="selectedClass" [ngStyle]="{'color': optionColor.value}">
color: {{optionColor.value}}
</span><br>
```

Primer (CSS fajl) i izgled stranice

```
.bold {
   font-weight: bold;
}
.italic {
   font-style: italic;
}
.highlight {
   background-color: lightblue;
}
```

Prilagođene direktive (eng. custom)

- Omogućavaju da proširimo funkcionalnost HTML-a, tako što ćemo implementirati ponašanje elementa. Ako smo napisali kod koji će manipulisati DOM-om, trebalo bi da ga postavimo unutar prilagođene direktive.
- Prilagođena direktiva se implementira pomoću poziva klase @directive, na isti način na koji se definiše komponenta.
- Metapodaci klase @directive treba da sadrže selektor direktive koji se koristi u HTML-u. Direktiva treba da bude smeštena u klasi izvoza Directive.

```
import { Directive } from '@angular/core';
@Directive({
   selector: '[myDirective]'
})
export class myDirective { }
```

Primer - Zumiranje (1)

- Dodavanje prilagođene funkcionalnosti,
  sa ciljem da postignemo interakciju kod nekih elemenata.
- Direktiva zumiranja, kojom će se preko točkića miša uvećavati ili smanjivati slika.
- Osluškujemo točkić miša, i kada se on aktivira, u zavisnosti od smera pomeranja točkića i veličine promene, implementira se izmena elementa na koji primenjujemo opciju zumiranja.
- Uvoz klasa:
  - Directive, ElementRef, HostListener, Input, Renderer
    (sve iz paketa @angular/core)

Primer - Zumiranje (2)

```
//zoom.component.ts

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
   images: string[] = [
   '../assets/images/jump.jpg',
   '../assets/images/flower2.jpg',
   '../assets/images/cliff.jpg'
   ]
}
```

Primer - Zumiranje (3)

```
//zoom.directive.ts
import { Directive, ElementRef, HostListener, Input, Renderer } from '@angular/core';
@Directive({
   selector: '[zoom]'
})

export class ZoomDirective {
   constructor(private el: ElementRef, private renderer: Renderer) { }
   @HostListener('mouseenter') onMouseEnter() {
      this.border('lime', 'solid', '5px');
   }
   @HostListener('mouseleave') onMouseLeave() {
      this.border();
   }
   @HostListener('wheel', ['$event']) onWheel(event: any) {
      event.preventDefault();
      if(event.deltaY > 0){
         this.changeSize(-25);
      }
      if(event.deltaY < 0){
         this.changeSize(25);
      }
   }
```

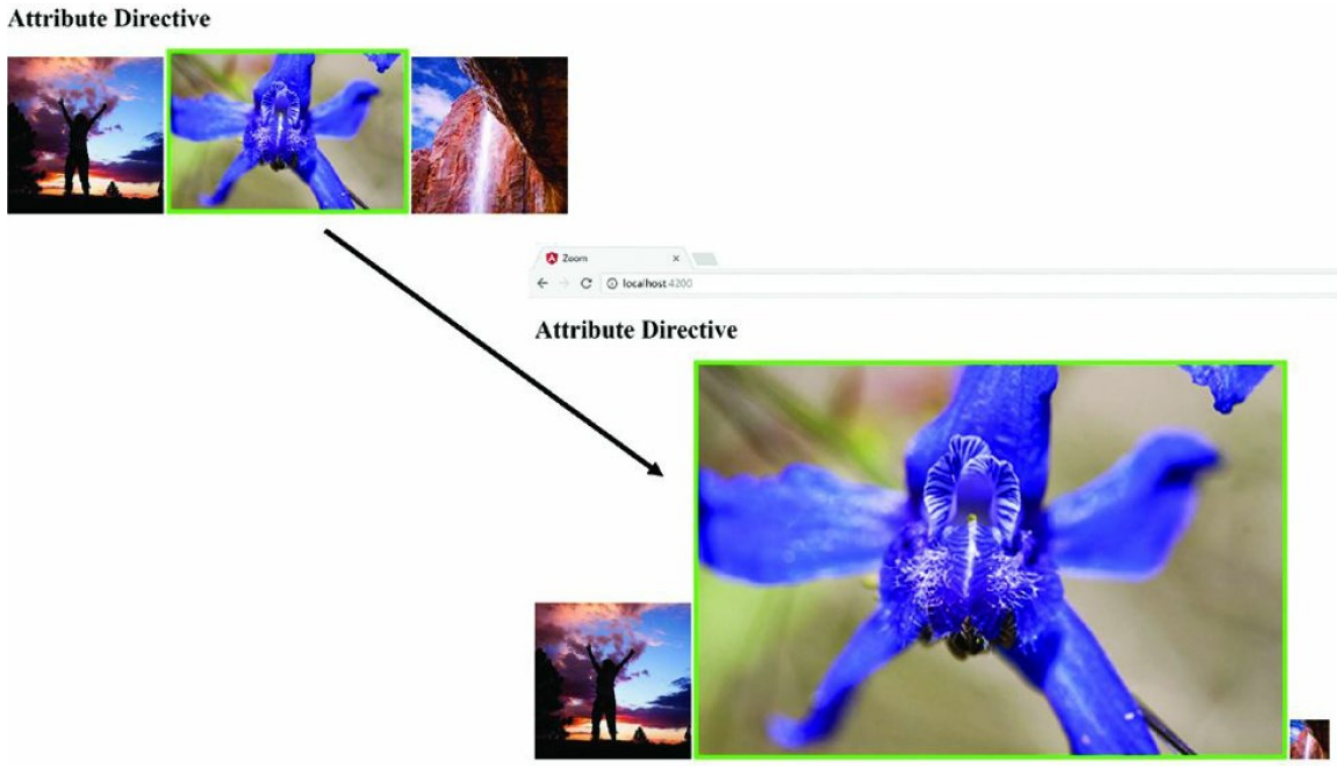
Primer - Zumiranje (4)

```
//zoom.directive.ts nastavak

private border(
      color: string = null,
      type: string = null,
      width: string = null
   ){
      this.renderer.setElementStyle(
      this.el.nativeElement, 'border-color', color);
      this.renderer.setElementStyle(
      this.el.nativeElement, 'border-style', type);
      this.renderer.setElementStyle(
      this.el.nativeElement, 'border-width', width);
   }
   private changeSize(sizechange: any){
      let height: any = this.el.nativeElement.offsetHeight;
      let newHeight: any = height + sizechange;
      this.renderer.setElementStyle(
      this.el.nativeElement, 'height', newHeight + 'px');
   }
}
```

Primer - Zumiranje (5)

```
//app.component.html
<h1>Atributske direktive</h1>
<span *ngFor="let image of images">
   <img zoom src="{{image}}" />
</span>
```

```
//app.component.css
img {
   height: 200px;
}
```



Kreiranje prilagođene direktive pomoću komponente

- Angular komponente su vrsta direktiva
- Glavna razlika:
  komponente koriste HTML šablone za generisanje prikaza!
- Direktiva <ng-content> omogućava da preuzme postojeći HTML između 2 taga elementa u kojima se koristi direktiva i da upotrebi taj HTML u šablonu komponente.
- U sledećem primeru prikazano je korišćenje komponente,
  kao prilagođene direktive za promenu izgleda elementa,
  koji sadrži kontejnerski šablon.
- HTML šablon kontejnera elementa ima ulaze *title* i *description*, koji se koriste da bi se elementu dodali naslov i kratak opis.

Primer - Osnovna komponenta (1)

```
//Koreni fajl: app.component.ts

import { Component } from '@angular/core';
@Component({
   selector: 'app-root',
   templateUrl: './app.component.html',
   styleUrls: ['./app.component.css'],
})
export class AppComponent {
images: any = [
{ src: "../assets/images/angelsLanding.jpg",
  title: "Angels Landing",
  description: "A natural wonder in Zion National Park Utah, USA" },
{ src: "../assets/images/pyramid.JPG",
  title: "Tikal",
  description: "Mayan Ruins, Tikal Guatemala" },
{ src: "../assets/images/sunset.JPG" },
]
}
```

Primer - Osnovna komponenta (2)

```
//HTML kod osnovne komponente: app.component.html
<span *ngFor="let image of images" container title="{{image.title}}"
   description="{{image.description}}">
   <img src="{{image.src}}" />
</span>
<span container>
   <p>Lorem ipsum dolor sit amet</p>
</span>
<span container>
   <div class="diver">
   </div>
</span>

//CSS fajl, app.component.css
img { height: 300px; }
p { color: red }
.diver{
   background-color: forestgreen;
   height: 300px;
   width: 300px;
}
```

Primer - Komponenta kontejnera (3)

```
//TS kod za definisanje kontejnera - container.component.ts
import { Component, Input, Output } from '@angular/core';

@Component({
   selector: '[container]',
   templateUrl: './container.component.html',
   styleUrls: ['./container.component.css']
})
export class ContainerComponent {
   @Input() title: string;
   @Input() description: string;
}

//HTML kod za komponentu kontejnera
<div class="sticky">
   <div class="title"> {{ title }} </div>
   <div class="content">
      <ng-content></ng-content>
   </div>
   <div class="description"> {{ description }} </div>
</div>
```

Primer - Komponenta kontejnera (4)

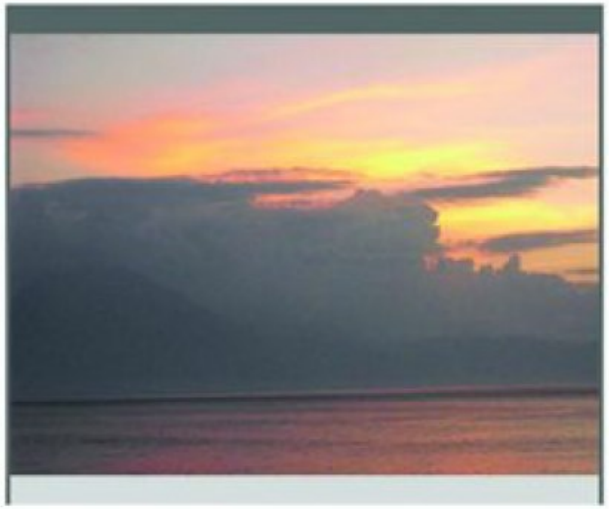
```
//CSS za komponentu kontejnera
.title {
   color: white;
   background-color: dimgrey;
   padding: 10px;
}
.content {
   text-align: center;
   margin: 0px;
}
.description {
   color: red;
   background-color: lightgray;
   margin-top: -4px;
   padding: 10px;
}
.sticky {
   display: inline-block;
   padding: 0px;
   margin: 15px;
   border-left: dimgrey 3px solid;
   border-right: dimgrey 3px solid;
}
```

Angels Landing

A natural wonder in Zion National Park Utah, USA

Tikal

Mayan Ruins, Tikal Guatemala



Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore

Angular / Angular 2+

Događaji i detekcija promena

Upotreba događaja pregledača

- Ugrađeni događaji funkcionišu kao povezivanje podataka
- Naziv događaja naveden u zagradama ( ) ukazuje Angularu na koji događaj treba da se povežemo
- Događaj je najčešće praćen pozivom neke funkcije, npr:

```
<input type="text" (change)="myEventHandler($event)"/>
```

HTML i Angular događaji

HTML sintaksa	Angular sintaksa	Opis događaja
onClick	(click)	Događaj koji se pokreće na klik na HTML element
onChange	(change)	Događaj koji se pokreće promenom vrednosti
onFocus	(focus)	Događaj koji se pokreće kada je izabran neki HTML element (stavljen fokus na njega)
onSubmit	(submit)	Događaj koji se pokreće kada se pošalje forma
onKeyUp, onKeyDown, onKeyPress	(keyup) (keydown) (keypress)	Događaji koji se pokreću kada se pritisnu tasteri na tastaturi
onMouseOver	(mouseover)	Događaji kojise pokreću kada se pokazivač miša nađe iznad nekog od HTML elemenata

Emitovanje prilagođenih događaja

- Događaji omogućavaju da pošaljete obaveštenje različitim nivoima u aplikaciji, kako biste im ukazali da su se pojavili događaji
- Primer: Ukazivanje podređenim komponentama da je promenjena vrednost u nadređenoj komponenti i obrnuto
- Klasa *EventEmitter* i metod *emit( )*, koji šalje događaj na najviše mesto u hijerarhiji nadređene komponente
- Sinktaksa:

```
@Output() name: EventEmitter<any> = new EventEmitter();
myFunction(){
    this.name.emit(args);
}
```

- name - naziv događaja, args - nula ili više argumenata koji se prosleđuju funkcijama hendlera događaja

Upravljanje prilagođenim događajima pomoću "osluškivača"

- Metod hendlera događaja koristi sintaksu u kojoj je name naziv događaja (iz prethodne tabele) koji se osluškuje,
  a *event* je vrednost koja je prosleđena klasi *EventEmitter*:

```
<div (name)="handlerMethod(event)">
```

Primer - Događaji u ugnežđenim komponentama

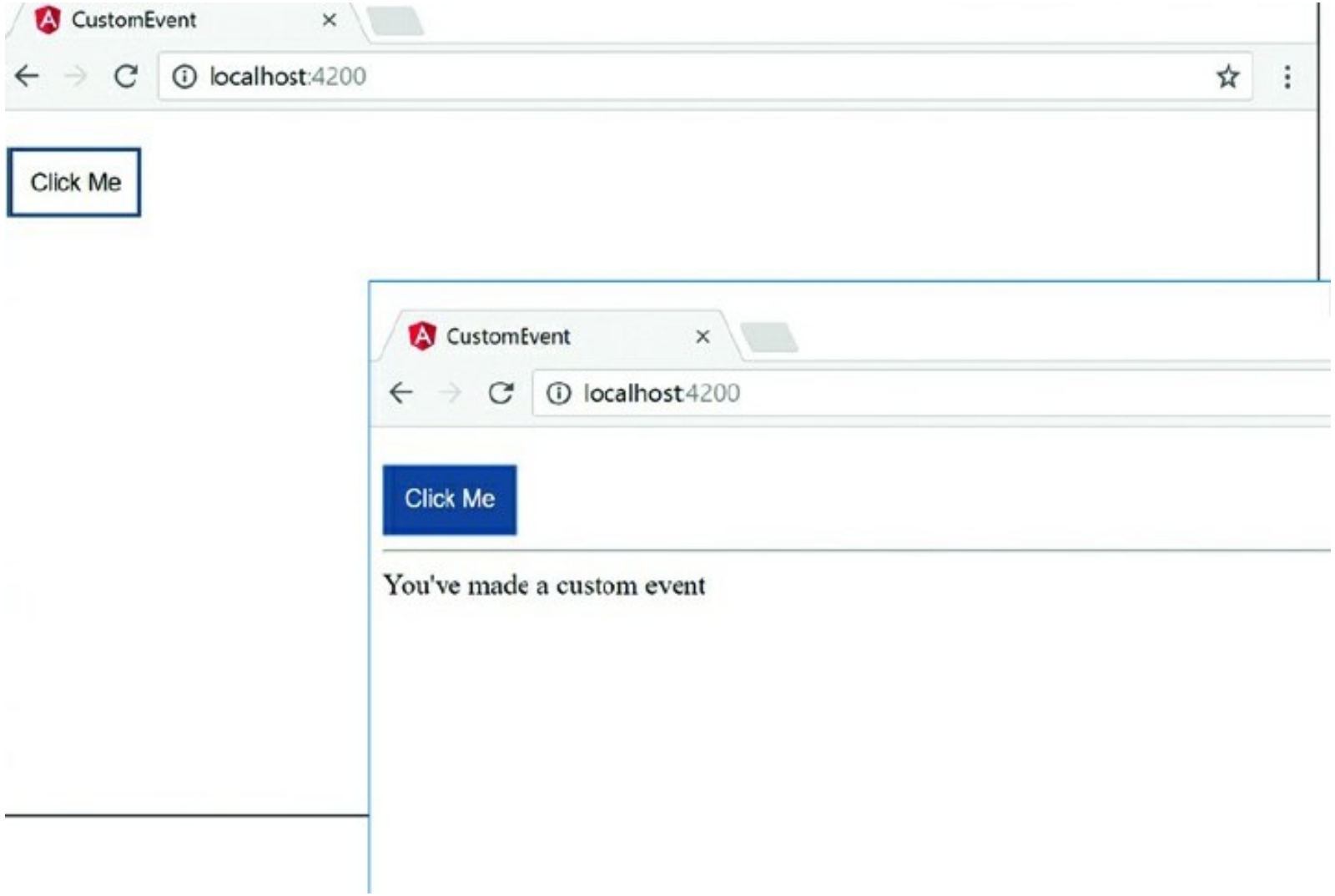
```
//customevent.component.ts - osnovna komponenta sa hendlerom dogadjaja
import { Component } from '@angular/core';
@Component({
   selector: 'app-root',
   templateUrl: 'customevent.component.html'
})
export class AppComponent {
   text: string = '';
   eventHandler(event: any){
      this.text = event;
   }
}
```

```
//customevent.component.html - HTML kod koji implementira prilagodjeni dogadjaj
<child (myCustomEvent)="eventHandler($event)"></child>
<hr *ngIf="text">
{{text}}
```

Primer - Podređena komponenta koja emituje događaj

```
//child.component.ts
import { Component, Output, EventEmitter } from '@angular/core';
@Component({
   selector: 'child',
   template: `
   <button (click)="clicked()" (mouseleave)="mouseleave()">Click Me</button>
   `,
   styleUrls: ['child.component.css']
})
export class ChildComponent {
 private message = "";
 @Output() myCustomEvent: EventEmitter<any> = new EventEmitter();
 clicked() {
   this.message = "You've made a custom event";
   this.myCustomEvent.emit(this.message);
 }
 mouseleave(){
   this.message = "";
   this.myCustomEvent.emit(this.message);
 }
}
```

Primer - Podređena komponenta koja emituje događaj



Primer - Brisanje podataka u nadređenoj komponenti iz podređene

```
//character.component.ts
import { Component } from '@angular/core';
@Component({
   selector: 'app-root',
   templateUrl: './app.component.html',
   styleUrls: ['./app.component.css']
})
export class AppComponent {
   character = null;
   characters = [ {name: 'Frodo', weapon: 'Sting', race: 'Hobbit'},
                  {name: 'Aragorn', weapon: 'Sword', race: 'Man'},
                  {name:'Legolas', weapon: 'Bow', race: 'Elf'},
                  {name: 'Gimli', weapon: 'Axe', race: 'Dwarf'} ]
   selectCharacter(character){ this.character = character; }
   deleteChar(event){
      var index = this.characters.indexOf(event);
      if(index > -1) {
         this.characters.splice(index, 1);
      }
      this.character = null;
   }
}
```

Primer - Brisanje podataka (2)

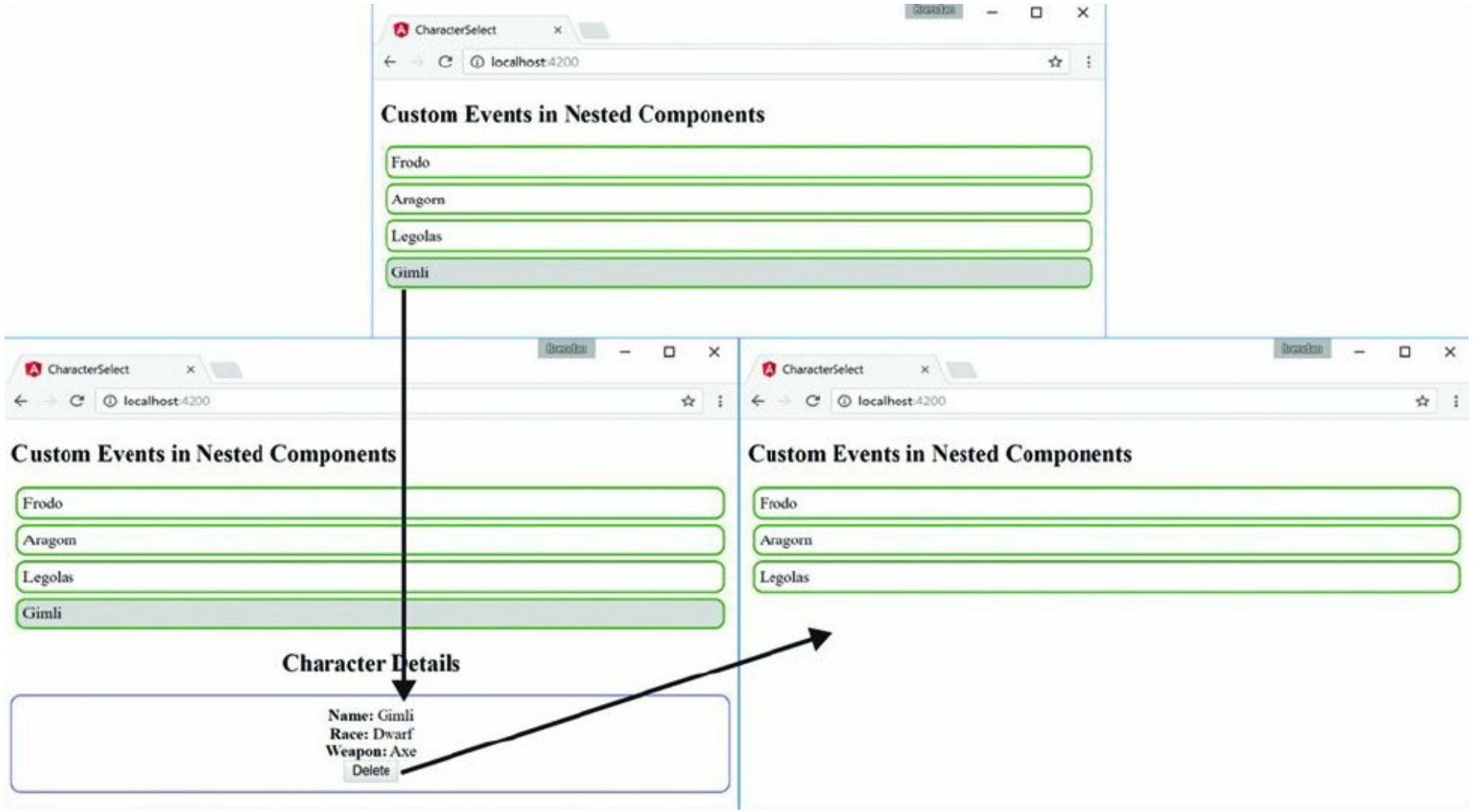
```
//character.component.html
<h2>Custom Events in Nested Components</h2>
<div *ngFor="let character of characters">
   <div class="char" (click)="selectCharacter(character)">
      {{character.name}}
   </div>
</div>
<app-character
   [character]="character"
   (CharacterDeleted)="deleteChar($event)">
</app-character>
```

Primer - Brisanje podataka (3)

```
//details.component.ts
import { Component, Output, Input, EventEmitter } from '@angular/core';
@Component({
   selector: 'app-character',
   templateUrl: './characters.component.html',
   styleUrls: ['./characters.component.css']
})
export class CharacterComponent {
 @Input('character') character: any;
 @Output() CharacterDeleted = new EventEmitter<any>();
 deleteChar(){
   this.CharacterDeleted.emit(this.character);
 }
}
```

Primer - Brisanje podataka (4)

```
//details.component.html
<div>
   <div *ngIf="character">
   <h2>Character Details</h2>
     <div class="cInfo">
      <b>Name: </b>{{character.name}}<br>
      <b>Race: </b>{{character.race}}<br>
      <b>Weapon: </b>{{character.weapon}}<br>
      <button (click)="deleteChar()">Delete</button>
     </div>
   </div>
</div>
```


CharacterSelect
localhost:4200
Custom Events in Nested Components
Frodo
Aragorn
Legolas
Gimli
Character Details
Name: Gimli
Race: Dwarf
Weapon: Axe
Delete

Upotreba opservabli

- **Opservable** obezbeđuju komponente za praćenje podataka, koji se asinhrono menjaju (podaci iz korisničkog unosa ili podaci koji pristižu sa servera).
- Cilj? Da možemo pratiti promene vrednosti.
- Vraćaju niz vrednosti; Niz ne mora biti primljen odjednom.
- Objekat opservable se uvozi iz paketa rxjs/opservable,
  da bi mogao da se koristi u komponenti.
- Nakon što se uveze, kreira se objekat:
  private name: Observable<Array<number>>
- Kada se objekat kreira, opservabla je dostupna za praćenje i ostatak komponente će moći da joj pristupi.
  Nakon implementacije, pratićemo je korišćenjem metode subscribe.

Primer - Objekat opservable

```
private name: Observable<Array<number>>;
ngOnInit(){
   this.name = new Observable(observer => {
     observer.next("my observable")
     observer.complete();
   }
   let subscribe = this.name.subscribe(
      data => { console.log(data) },
      Error => { errorHandler(Error) },
      () => { final() }
   );
  subscribe.unsubscribe();
}
```

=> prosleđivanje podataka opservabli
=> zatvaranje veze opservable

⇒ uspešan prijem podataka
⇒ hendler greške
⇒ završna funkcija (bez obzira na uspeh)

Primer - Opservable za detektovanje (1)

```
import { Component, OnInit } from '@angular/core';
import { Observable } from 'rxjs/observable';
import { Subscription } from 'rxjs/Subscription';
@Component({
   selector: 'app-root',
   templateUrl: "./observable.component.html",
   styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit {
   combinedTotal:number = 0;
   private pass: Observable<any>;
   private run: Observable<any>;
   teams = [];
   ngOnInit(){
         this.teams.push({passing:0, running:0, total:0});
         this.teams.push({passing:0, running:0, total:0});
         //Passing
         this.pass = new Observable(observer => {this.playLoop(observer);});
         this.pass.subscribe(
           data => {
           this.teams[data.team].passing += data.yards;
           this.addTotal(data.team, data.yards);
         });
```

Primer - Opservable za detektovanje (2)

```
        //Running
        this.run = new Observable(observer => {
          this.playLoop(observer);
        });
        this.run.subscribe(
          data => {
          this.teams[data.team].running += data.yards;
          this.addTotal(data.team, data.yards);
        });
        //Combined
        this.pass.subscribe(
          data => { this.combinedTotal += data.yards;
        });
        this.run.subscribe(
          data => { this.combinedTotal += data.yards;
        });
}
```

Primer - Opservable za detektovanje (3)

```
  playLoop(observer){
    var time = this.getRandom(500, 2000);
    setTimeout(() => {
        observer.next(
        { team: this.getRandom(0,2),
          yards: this.getRandom(0,30)});
          if(this.combinedTotal < 1000){
             this.playLoop(observer);
          }
        }, time);
    }
    addTotal(team, yards){
        this.teams[team].total += yards;
    }
    getRandom(min, max) {
       return Math.floor(Math.random() * (max - min)) + min;
    }
}
```

Primer - Opservable za detektovanje (4)

```
<div>
   Team 1 Yards:<br>
   Passing: {{teams[0].passing}}<br>
   Running: {{teams[0].running}}<br>
   Total: {{teams[0].total}}<br>
   <hr>
   Team 2 Yards:<br>
   Passing: {{teams[1].passing}}<br>
   Running: {{teams[1].running}}<br>
   Total: {{teams[1].total}}<hr>
   Combined Total: {{combinedTotal}}
</div>
```

Angular / Angular 2+

Implementacija veb servisa u veb aplikacijama

Angular servisi

- Svrha servisa je da obezbedi koncizan deo koda koji izvršava određene zadatke
- Servis može raditi nešto jednostavno ili nešto složeno
- Servis obezbeđuje kontejner za ponovno upotrebljive funkcije koje su lako dostupne Angular aplikacijama
- Servisi su u Angularu definisani i registrovani pomoću mehanizma za injektiranje zavisnosti (*dependency injection*)
- Servisi se mogu pojaviti u modulima, u komponentama ili u drugim servisima
- Kada se na primer ugrade u modul, servisi se mogu koristiti u čitavoj aplikaciji

Servisi ugrađeni u Angular

Naziv servisa	Opis
animate	Obezbeđuje animirane "kukice" (hooks) za povezivanje sa CSS i JavaScript animacijama
http	Obezbeđuje jednostavne funkcije za slanje HTTP zahteva na veb server ili druge servise
router	Obezbeđuje pristup prikazima i odeljcima u okviru prikaza
forms	Obezbeđuje servis koji omogućava dinamičke i reaktivne forme sa jednostavnim obrascem validacije

Slanje HTTP zahteva GET i PUT pomoću http servisa

- Servis http omogućava da direktno komunicirate sa veb serverom iz Angular koda (servis u osnovi koristi XMLHttpRequest)
- Postoje dva načina upotrebe http servisa. Najjednostavniji je upotreba ugrađenih skraćeih metoda koji odgovaraju standardnim HTTP zahtevima:
  - delete (url, [options])
  - get (url, [options])
  - head (url, [options])
  - post (url, data, [options])
  - put (url, data, [options])
  - patch (url, data, [options])

url = URL veb zahteva; data = podaci;
options = JS objekat koji određuje opcije;

Svojstva (*options*) kod http servisa

Svojstvo	Opis
method	HTTP metod, GET ili POST
url	URL adresa koja se zahteva
params	Parametri koji se šalju (ili u JSON stringu, ili standardni string nekog oblika: ?kljuc1=vred1&kljuc2=vred2&...)
body	Podaci koji se šalju kao telo poruke zahteva
headers	Zaglavlja koja se šalju u zahtevu. Možete da odredite objekat koji sadrži nazive zaglavlja koji se šalju kao svojstva. Ako svojstvo u objektu ima vrednost null, zaglavlje neće biti poslato.
withCredentials	Kada je vrednost flega true, znači da je fleg postavljen na objekat XHR
responseType	Tip odgovora koji se očekuje (JSON ili text)

Konfigurisanje HTTP zahteva

- http.get('/mojaAdresa');
  ili
  http({method: 'GET', url:'/mojaAdresa'})
- Oba pristupa su korektna

Implementiranje funkcija povratnog poziva HTTP odgovora

- Kada pozovete metod zahteva pomoću objekta http,
  dobićete objekat Observable, koji omogućava neprekidno praćenje poslatih ili primljenih podataka na server.
- Neki korisni metodi:
  - *map* - Primenjuje funkciju na svaku vrednost u sekvenci opservable.
    Ovo omogućava da dinamički transformišete izlaz toka opservable
    u prilagođene formate podataka.
  - *toPromise* - Konvertuje opservablu u objekat Promise, koji pristupa metodama u promisu. Objekti Promise obezbeđuju sintaksu za upravljanje asinhronim operacijama.
  - *catch* - Određuje funkciju za lepo upravljanje greškama u sekvenci opservable.
  - *debounce* - Obezbeđuje interval u kome će tok opservable emitovati vrednost. Biće emitovana samo vrednost u intervalu, dok međuvrednosti neće biti emitovane.

Primer zahteva GET koji vraća opservablu

```
get(): Observable<any>{
   http.get(url)
     .map(response => response.JSON())
     .catch(err => Rx.Observable.of('the error was: ${err}'));
}
```

Implementiranje JSON datoteke i upotreba http servisa za pristup

```
[
{
   "userId": 1,
   "userName": "brendan",
   "userEmail": "fake@email.com"
},
{
   "userId": 2,
   "userName": "brad",
   "userEmail": "email@notreal.com"
},
{
   "userId": 3,
   "userName": "caleb",
   "userEmail": "dummy@email.com"
},
{
   "userId": 4,
   "userName": "john",
   "userEmail": "ridiculous@email.com"
},
{
   "userId": 5,
   "userName": "doe",
   "userEmail": "some@email.com"
}
]
```

Komponenta koja implementira servis za GET zahtev

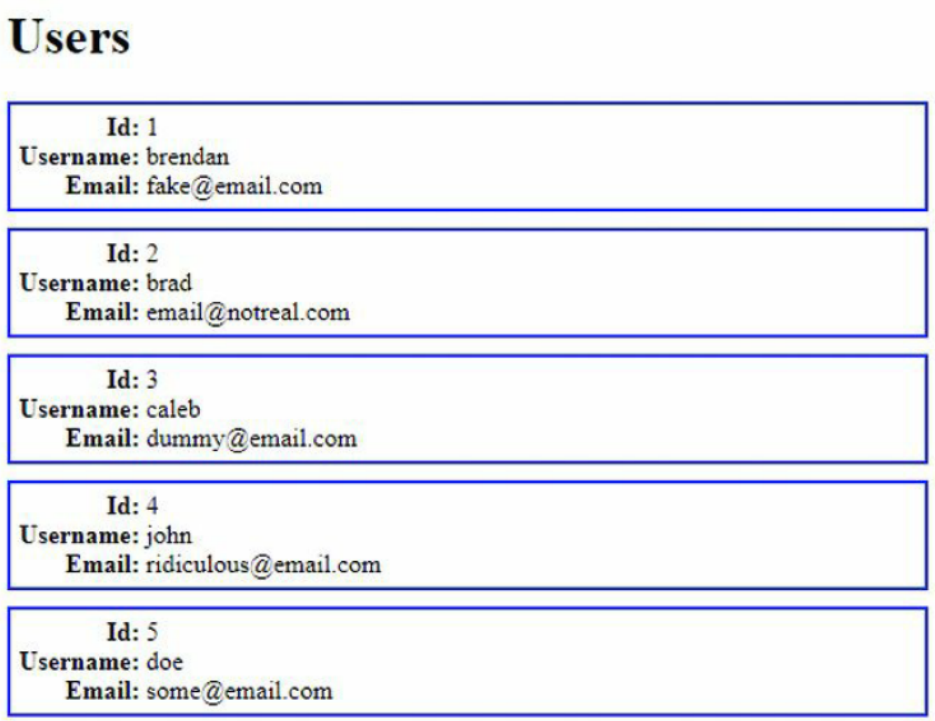
```
import { Component } from '@angular/core';
import { Observable } from 'rxjs/Observable';
import { Http } from '@angular/http';
import 'rxjs/Rx';
@Component({
   selector: 'app-root',
   templateUrl: './app.component.html',
   styleUrls: ['./app.component.CSS']
})
export class AppComponent {
   users = [];
   constructor(private http: Http){
         http.get('../assets/dummyDB.JSON')
            .toPromise()
            .then((data) => {
              this.users = data.JSON()
            })
            .catch((err) =>{ console.log(err);})
   }
}
```

Modul koji uvozi HttpModule

```
//app.module.ts
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { HttpModule } from '@angular/http';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [
     AppComponent
  ],
  imports: [
     BrowserModule,
     FormsModule,
     HttpModule
   ],
   providers: [],
   bootstrap: [AppComponent]
})
export class AppModule { }
```

Lista korisnika isčitana iz baze (JSON)

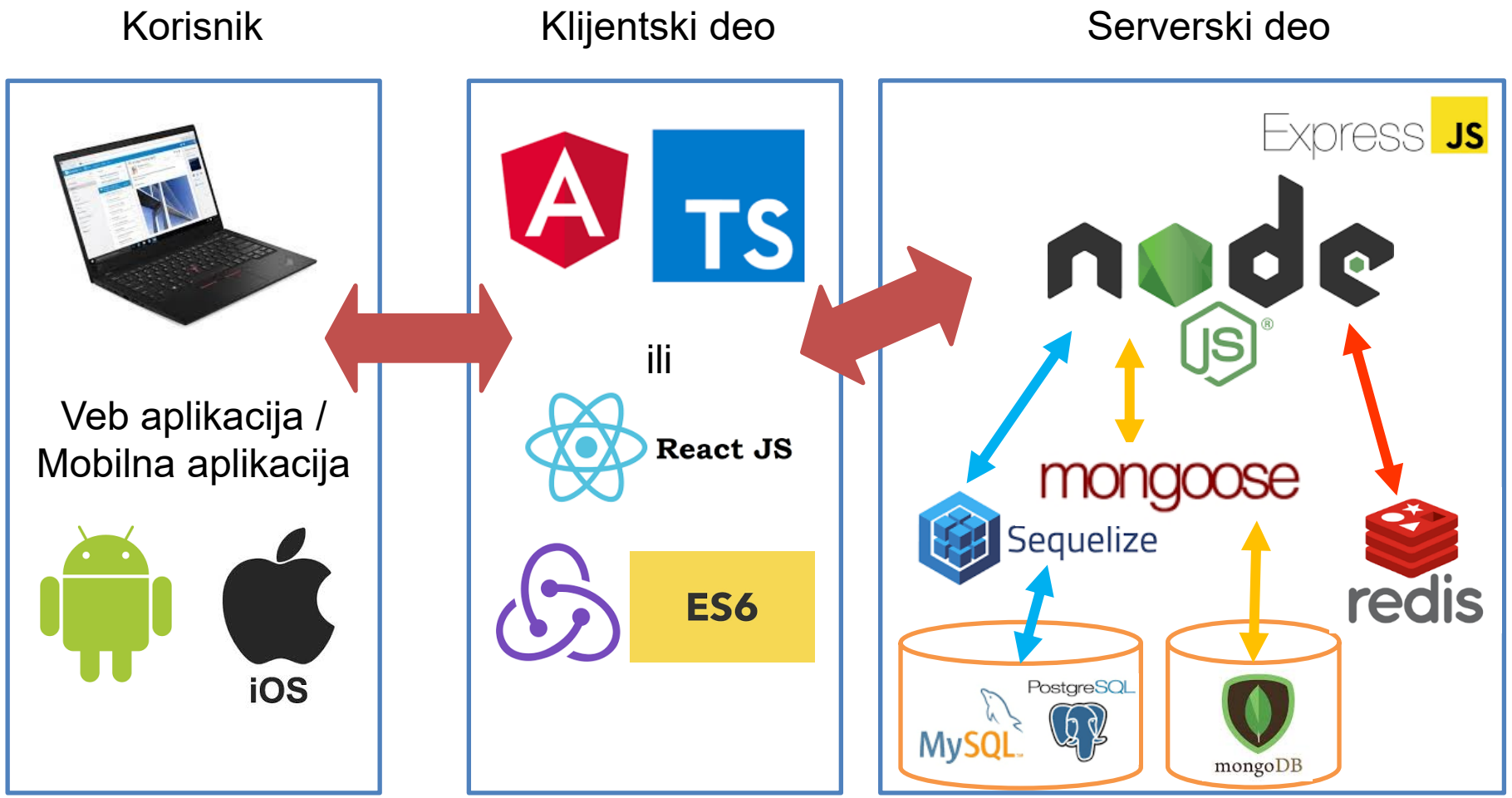
```
//http.component.html
<h1>Korisnici</h1>
<div class="user" *ngFor="let user of users">
   <div><span>Id:</span> {{user.userId}}</div>
   <div><span>Username:</span> {{user.userName}}</div>
   <div><span>Email:</span> {{user.userEmail}}</div>
</div>
```



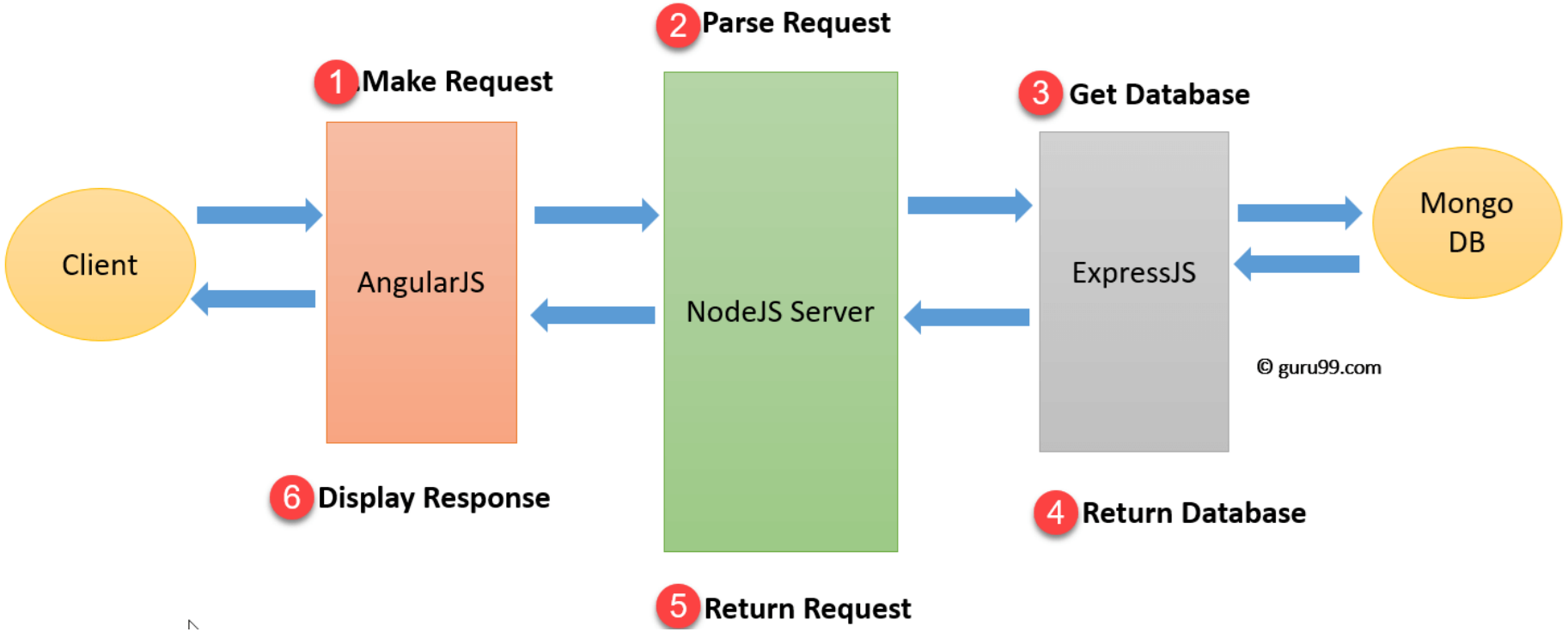
Angular / Angular 2+

Arhitektura aplikacije sa Node.JS

Arhitektura Node.JS aplikacija



MEAN stek


2 Parse Request
1 Make Request
3 Get Database
Client
AngularJS
NodeJS Server
ExpressJS
Mongo DB
© guru99.com
6 Display Response
4 Return Database
5 Return Request

Hvala na pažnji ☺

PITANJA?