

MVC (Model-View-Controller)

Angular

arhitektura i gradivni elementi

Dr. Mlađan Jovanović
mjovanovic@singidunum.ac.rs

Sadržaj

- ◆ Moduli
- ◆ Komponente
- ◆ Templejti
- ◆ Data binding
- ◆ Izrazi
- ◆ Direktive
- ◆ Servisi
- ◆ Dependency injection

Razvoj tehnologija klijentskih Web aplikacija

- ◆ Inicijalno HTML stranice sa formama za unos podataka
- ◆ Početni okviri razvoja aplikacija:
 - ◆ Templejti kao mešavina HTML tagova i koda
 - ◆ Tagovi imaju deklarativno značenje: šta prikazati
 - ◆ Kod ima imperativno značenje: kako kreirati prikaz
 - ◆ Primeri: PHP, ASP, JSP
 - ◆ Odvojene biblioteke (paketi) za specifične obrade
 - Rukovanje URL adresiranjem
 - Generisanje HTML sadržaja
 - Upravljanje sesijama
 - Interfejs ka bazi podataka

Savremene tehnologije klijentskih Web aplikacija

- ◆ Okviri zasnovani na JS jeziku čije se aplikacije izvršavaju u pretraživaču
- ◆ Ne zavise od klasičnih serverskih funkcionalnosti:
- ◆ Serverska, tj. poslovna logika premeštena u pretraživač i odvojena od klijentske logike (NodeJS)
- ◆ Ne-relacione baze podataka (MongoDB)
- ◆ Logika aplikacije klijentske aplikacije organizovana po MVC (Model-View-Controller) obrascu
- ◆ Razlog:
 - ◆ Raspodela odgovornosti klijentske strane radi bržeg i kvalitetnijeg razvoja i održavanja aplikacija
 - ◆ Očiglednije sa povećanjem kompleksnosti sistema koji se razvija ili održava

MVC (Model-View-Controller) obrazac

- ◆ **Model** odgovoran za **podatke** aplikacije

- JS objekti sa atributima, tj. podacima

- ◆ **View** odgovoran za **izgled** aplikacije (*look*)

- HTML i CSS

- ◆ **Controller** odgovoran za **ponašanje** aplikacije (*feel*)

- Upravljanje modelima – komunikacija sa serverom (NodeJS)
 - Kontrola izgleda
 - Upravljanje interakcijom sa korisnikom (prihvatanje i obrada ulaza) – DOM osluškivači događaja

Generisanje pogleda (View)

- ◆ Web aplikacije generišu HTML/CSS interfejs
- ◆ Templejti su uobičajena tehnika:
- ◆ Pisanje HTML dokumenta sa delovima koji se ne menjaju (statički delovi)
- ◆ Dodavanje koda koji kreira delove koji se menjaju (dinamički delovi)
- ◆ Rezultati izvršavanja koda predstavljaju promene izgleda dinamičkih delova
- ◆ Prednosti templejta (u odnosu na DOM tehnike):
 - ◆ HTML sadržaj jednostavniji za vizuelizaciju i promenu
 - Dinamički delovi se vide u samom templejtu
- ◆ Lakše je identifikovati dinamičke delove templejta
 - Jasno razdvojeni od ostatka klijentske logike

Controller

- ◆ Povezuje model i templejt (view):
 - ◆ Komunikacija sa serverskom logikom
 - ◆ Obrada podataka aplikacije
 - ◆ Ažuriranje pogleda na osnovu rezultata obrade
- ◆ Kontrola templejta:
 - ◆ Upravlja dinamičkim delovima prikaza templejta
- ◆ Obrada interakcije sa korisnikom:
 - ◆ Registrovanje i obrada događaja nad elementima korisničkog interfejsa (forme, meniji, dugmad, ikone)

Podaci modela

- ◆ Svi promenjivi podaci potrebni za rad kontrolera i generisanje templejta
- ◆ Podaci koji traju za vreme korišćenja aplikacije, tj. korisničke sesije
 - Događaji kao objekti
 - Strukture podataka (objekti, nizovi, liste)
 - Instance klasa sa vrednostima atributa
 - Podaci dobijeni od serverske logike (JSON, komunikacija sa API-jem ili bazom podataka)
- ◆ Perzistentni podaci koji se čuvaju trajnije
- ◆ Baza podataka ili fizičke datoteke
- ◆ Nisu deo modela klijentske strane, već se dobavljaju od serverske strane na zahtev, tj, po potrebi

AngularJS

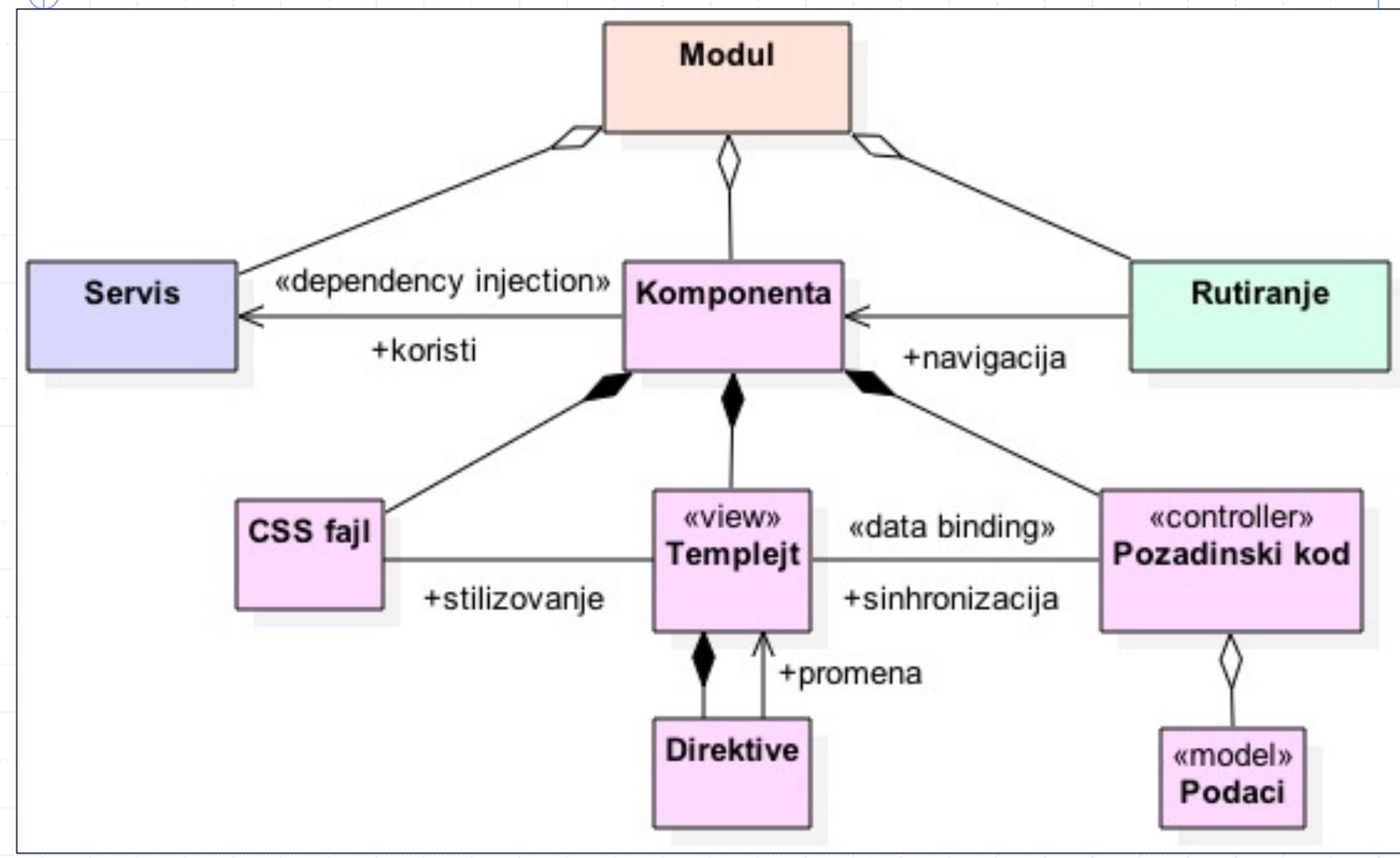
- ◆ MVC okvir za razvoj klijentskih Web aplikacija u HTML i TypeScript jeziku
- ◆ Automatizuje DOM manipulaciju, validaciju ulaza, komunikaciju sa serverom, URL adresiranje (rute)
- ◆ Akcenat na razvoj složenih aplikacija:
- ◆ **Moduli** kao mehanizam grupisanja
- ◆ **Komponente** kao osnovni gradivni elementi
 - Ponovna upotrebljivost (reusability)
- ◆ Okvir je napisan u TypeScript jeziku
- ◆ TypeScript je nadskup JavaScript jezika
- ◆ Nudi tipizirane klase, interfejse i module
- ◆ Prevodi se u JavaScript implicitno
- ◆ Jednostavnija i čitljivija sintaksa

AngularJS

- ◆ **Ideja:**
 - ◆ Aplikacije je kolekcija JS/TS **komponenti** umesto HTML stranica sa pridruženim JS fajlovima
 - ◆ Komponente su organizovane po **modulima**
 - ◆ Akcenat je na unapređenju procesa dizajna i razvoja klijentske Web aplikacije:
 - ◆ Modularan dizajn
 - ◆ Ponovne upotrebljive komponente
 - ◆ Razdvajanje logike klijentske aplikacije
 - MVC obrazac na nivou komponente
 - ◆ Podrška za testiranje aplikacije na nivou komponenti

| Koncept | Opis |
|------------------------|--|
| Modul (module) | Kontejner, tj. mehanizam grupisanja komponenti |
| Komponenta (component) | Osnovna gradivni element aplikacije |
| Templejt (template) | HTML dokument čiji se sadržaj može menjati dinamički (deo komponente) |
| Direktiva (directive) | Naredbe templejta koje omogućavaju dinamičku promenu sadržaja |
| Kontroler | Deo komponente koji čuva njene podatke i definiše ponašanje (TypeScript fajl) |
| Servis (service) | Izdvojena funkcionalnost (TypeScript fajl) |
| Rutiranje (routing) | Mehanizam definisanja putanja do komponenti kako bi bile prikazane i celokupne navigacije u aplikaciji |
| Dependency Injection | Mehanizam kojim funkcije servisa postaju dostupne (vidljive) komponenti |
| Data Binding | Automatizovani mehanizam sinhronizacije između podataka templejta i podataka kontrolera |

Logička šema gradivnih elemenata Angular-a



Modul (*module*)

- ◆ **Modul** – mehanizam grupisanja komponenti
- ◆ Aplikacija je skup povezanih modula kao funkcionalnih jedinica aplikacije
- ◆ Fizički se definiše kao TypeScript klasa
- ◆ Modul logički grapiše i definiše kontekst korišćenja komponenti
- ◆ Svaka aplikacija se sastoji od dva tipa modula:
 - *Korieni (root) modul* koji omogućava pokretanje i inicijalizaciju aplikacije (*AppModule*), tzv. bootstrap mehanizam
 - *Korisnički (feature) moduli* koji definišu izgled i funkcije aplikacije
- ◆ Optimizacija performansi aplikacije – moduli se učitavaju onda kada se zaista i koriste (*lazy loading*)

Komponenta (*component*)

- ◆ Komponente definišu prikaz (*view*) i funkcije (*controller*) elementa aplikacije ili korisničkog interfejsa
- ◆ Sastoji se od tri dela:
 - ◆ Templejt kao view ([HTML dokument](#))
 - ◆ **CSS fajl** kao dodatak za stilizovanje templejta
 - ◆ Pozadinski kod kao controller ([TypeScript klasa](#))
 - Može čuvati i podatke komponente u toku korišćenja aplikacije (model)
- ◆ Prilikom kreiranja komponente, generišu se delovi kao zasebne datoteke koje su logički povezane
- ◆ Svaka aplikacija se sastoji od:
 - ◆ Korene (*root*) komponente koja je vrh DOM hijerarhije komponenti (*AppComponent*)
 - ◆ Korisnički definisanih komponenti koje implementiraju specifične funkcije aplikacije
 - ◆ Angular aplikacija je DOM stablo komponenti u vreme izvršavanja

Servis (*service*)

- ◆ Klasa sa specifičnom i jasno definisom ulogom kao skup funkcionalnosti
 - Rad sa specifičnom strukturom modela, komunikacija sa serverskom logikom, validacija ulaza korisnika, itd.
- ◆ Razdvajanje od komponente radi ponovne upotrebljivosti i modularnosti
- ◆ Izdvojene zajedničke funkcionalnosti koje komponente delegiraju servisima
- ◆ Komponenta je fokusirana na interaktivnu logiku sa korisnikom, dok servisi implementraju funkcionalnosti koje podržavaju ovu logiku
- ◆ Komponente koriste servise putem mehanizma koji se naziva **dependency injection**
 - Registracija servisa u komponenti kako bi bio vidljiv i mogao da se koristi

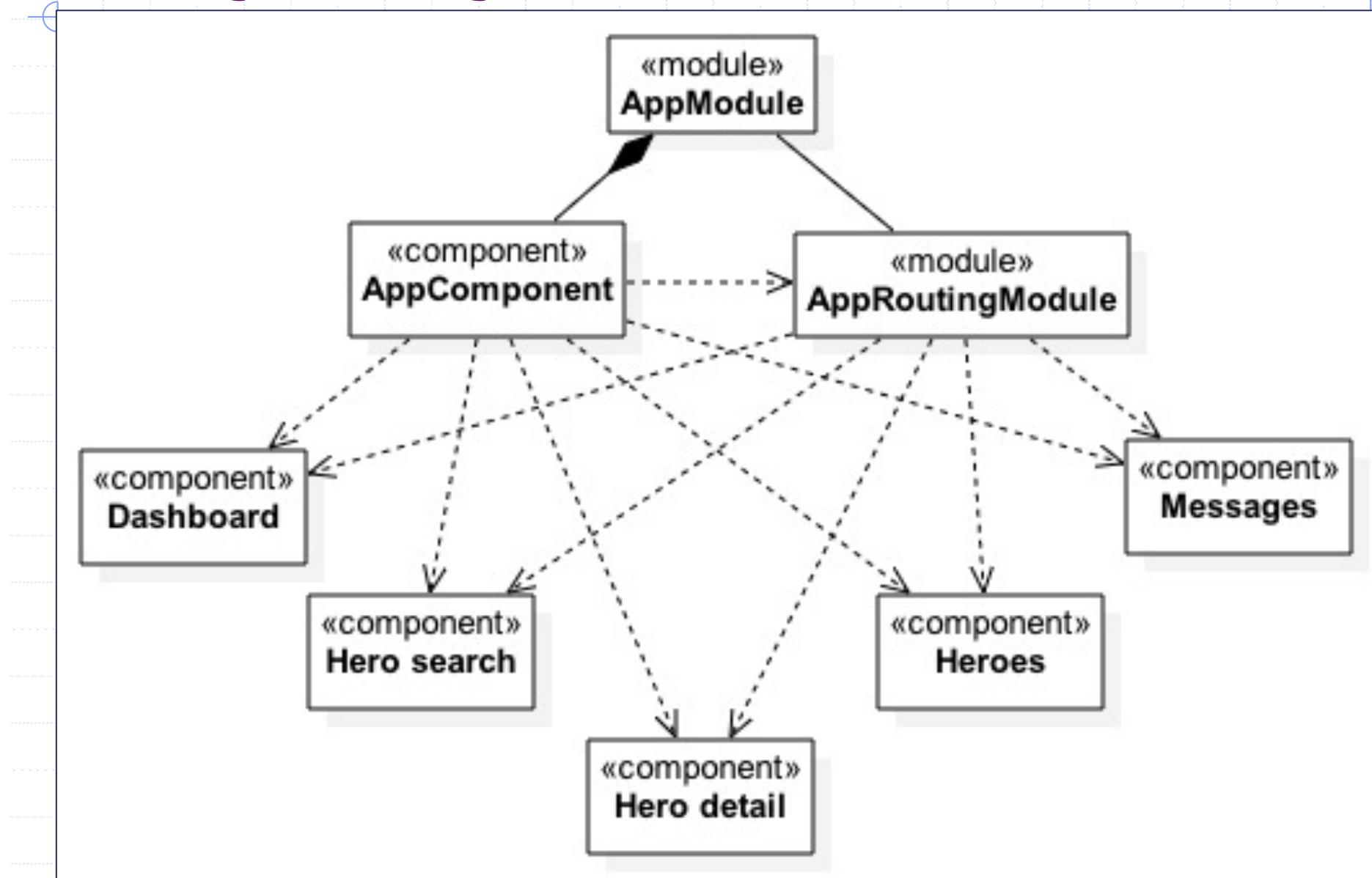
Rutiranje (*routing*)

- ◆ Ugrađeni mehanizam koji omogućava definisanje putanja do komponenti aplikacije i njihovo prikazivanje
- ◆ Definiše se kao modul sa funkcijom rutiranja
 - TypeScript klasa
- ◆ Putanja se definiše do komponente, a ne stranice
 - Na osnovu adrese koju zada korisnik ili akcije korisnika, bira komponentu koju će prikazati (templejt komponente)
 - Rute se prethodno definišu eksplicitno od strane programera kao putanje do komponenti
- ◆ Rezultat: različite komponente se prikazuju u okviru jedne stranice
- ◆ SPA (Single Page Application)
- ◆ Nema ponovnog učitavanja čitave stranice, već samo dela (taga) u okviru kojeg se prikazuje komponenta
 - Veći kvalitet doživljaja korisnika

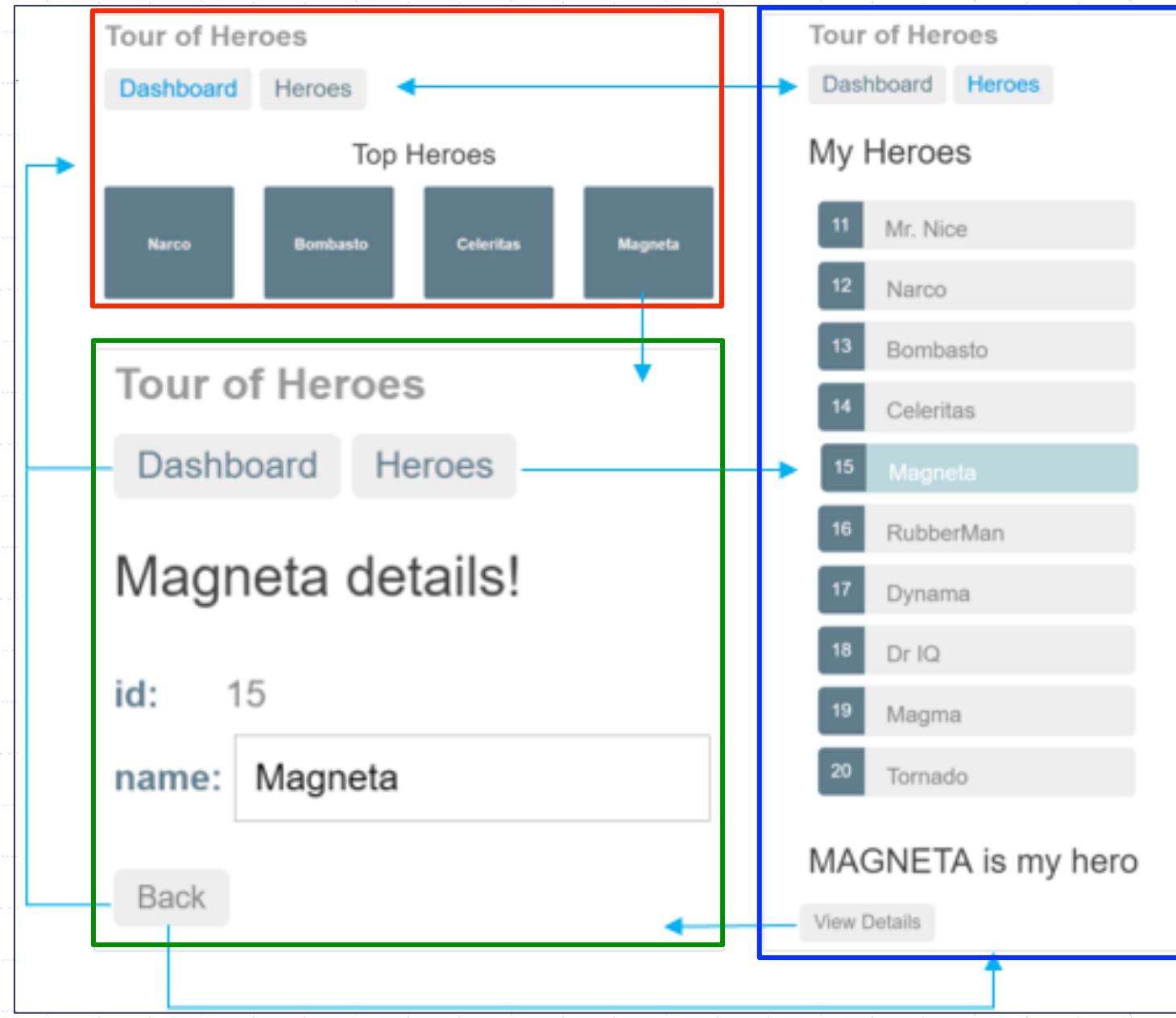
Dekoratori

- ◆ Komponente, servisi i moduli su fizički predstavljeni na identičan način – kao TypeScript klase (datoteke sa ekstenzijom .ts)
- ◆ Kako Angular zna da prepozna i razdvoji ove elemente?
- ◆ Pomoću **dekoratora** koji definišu metapodatke klase, tj. informacije o načinu na koji će ona biti korišćena od strane Angular okruženja
 - Analogno Java anotacijama (@)
- ◆ **@NgModule** govori da je klasa **modul**
- ◆ **@Component** govori da je klasa deo **komponente** (kontroler)
- ◆ **@Injectable** govori da je klasa **servis**
 - Komponente je mogu koristiti kao servis (*dependency injection*)

Studija slučaja – Tour of Heroes šema



Studija slučaja – Tour of Heroes izgled



Dashboard

Hero detail

Heroes

Moduli

- ◆ Definisanje
- ◆ Atributi
- ◆ Primer
- ◆ Često korišćeni moduli

Moduli (@NgModule)

- ◆ Definišu funkcionalni kontekst za skup povezanih komponenti ili servisa
- ◆ Funkcionalni blokovi komponenti ili servisa
- ◆ Fizička implementacija:
- ◆ TypeScript klasa dekorisana kao `@NgModule`
 - U okviru klase se navode komponente i/ili servisi modula
- ◆ Svaka aplikacija ima:
- ◆ Jeden korenji (*root*) modul koji pokreće aplikaciju
 - Root konvencija: `AppModule`, `app.module.ts`
- ◆ Veći broj modula koje definiše programer
- ◆ Mogu koristiti druge module i biti korišćeni od strane tih modula
- ◆ Ponovna upotrebljivost
- ◆ *Lazy loading* – učitivanje modula na zahtev u cilju optimizacije količine koda potrebnog za rad aplikacije

Atributi @NgModule dekoratora

- ◆ Definisani kao nizovi vrednosti:
- ◆ **declarations:** komponente koje pripadaju tekućem modulu
- ◆ **exports:** komponente tekućeg koje su vidljive drugim modulima (mogu ih koristiti)
- ◆ **imports:** moduli čiji će sadržaj biti korišćen u okviru tekućeg modula (od strane njegovih komponenti)
- ◆ **providers:** servisi koje tekući model definiše; vidljivi u okviru čitave aplikacije (analogno exports za komponente)
- ◆ **bootstrap:** korena (root) komponenta aplikacije; samo koren modul navodi vrednost ovog atributa

Koreni modul Angular aplikacije

src/app/app.module.ts

```
import { NgModule }           from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
```

```
@NgModule({
  imports:      [ BrowserModule ],
  providers:    [ Logger ],
  declarations: [ AppComponent ],
  exports:      [ AppComponent ],
  bootstrap:   [ AppComponent ]
})
```

```
export class AppModule { }
```

1. Uvoz klasa iz Angular biblioteka
2. Metapodaci modula
3. Definicija klase modula

Predefinisani tipovi modula često korišćeni u praksi

| NgModule | Import it from | Why you use it |
|---------------------|---------------------------|---|
| BrowserModule | @angular/platform-browser | When you want to run your app in a browser |
| CommonModule | @angular/common | When you want to use NgIf, NgFor |
| FormsModule | @angular/forms | When you want to build template driven forms (includes NgModel) |
| ReactiveFormsModule | @angular/forms | When you want to build reactive forms |
| RouterModule | @angular/router | When you want to use RouterLink, .forRoot(), and .forChild() |
| HttpClientModule | @angular/common/http | When you want to talk to a server |

Primeri modula u praksi

- ◆ Koreni modul koji sadrži glavne komponente aplikacije
- ◆ Modul za rutiranje
 - Definisanje navigacije aplikacije pomoću ruta
- ◆ Moduli servisa
 - Grupisanje servisa specifične namene
- ◆ Moduli kontrola korisničkog interfejsa
 - Dugmad, meniji, ikone, ...
 - Angular Material

Programsko kreiranje modula

`ng generate module ime_modula`

`ng g m ime_modula`

Konvencija imenovanja modula:

ime_modula.module.ts

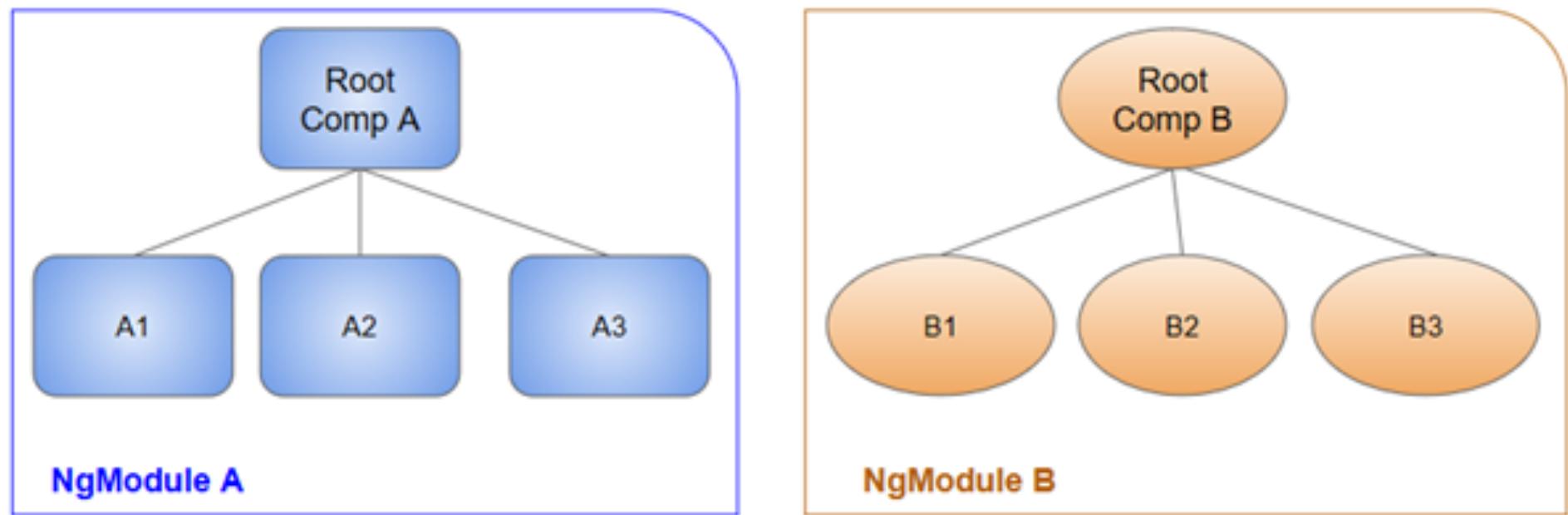
Rezultat:

Modul se kreira kao TS datoteka sa nazivom

ime_modula.module.ts

Klasa ima dekorator `@NgModule`

Modul - struktura

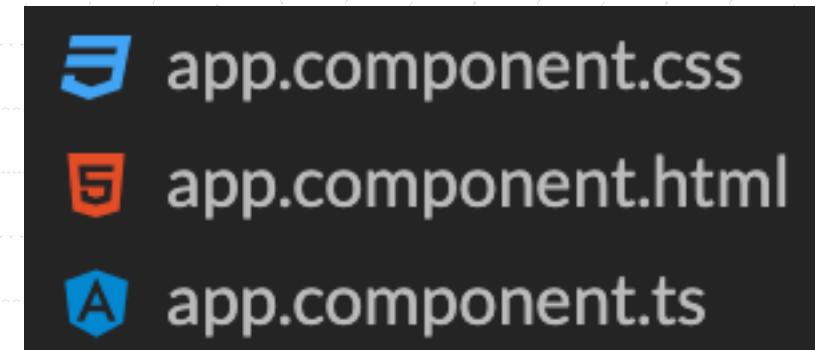


Komponenta (Component)

- ◆ Komponente definišu prikaz (*view*) i funkcije (*controller*) elementa aplikacije ili korisničkog interfejsa
- ◆ View: templejt + css fajl
- ◆ Kontroler: TypeScript klasa
- ◆ Mogu koristiti servise (*service*) koji definišu specifične funkcionalnosti
- ◆ Servisi mogu biti ubačeni ili registrovani (*injected*) u komponente kao zavisnosti (*dependency*)
- ◆ Mehanizam je poznat kao *dependency injection*
- ◆ Modularan, ponovno upotrebljiv i efikasan kod

Komponenta (Component)

- ◆ TypeScript klasa dekorisana kao `@Component` koja implementira aplikativnu logiku i ima pridružen templejt fajl (html) koji definiše izgled komponente
- ◆ Angular aplikacija ima jednu korenú komponentu (*root*) koja predstavlja vrh hijerarhije stabla komponenata
- ◆ Generiše se automatski prilikom kreiranja Angular aplikacije, tj. projekta



Programsko kreiranje komponente

`ng generate component ime_komponente`

`ng g c ime_komponente`

Konvencija imenovanja komponente:

ime_komponente.component.ts

Rezultat:

Folder pod nazivom *ime_komponente* u kojem se nalaze
datoteke komponente:

ime_komponente.component.html (templet)

ime_komponente.component.css (stil)

ime_komponente.component.ts (kontroler)

Komponenta ima dekorator `@Component`

Metapodaci klase tipa @Component

- ◆ Identifikuju gradivne elemente komponente
- ◆ **selector:** CSS selektor taga koji govori Angular okruženju da kreira i insertuje instancu komponente u slučaju da pronađe tag u html templejtu ili stranici
 - Tag identifikator komponente
 - Na primer, ako HTML sadrži <trening-lista></trening-lista>, instanca komponente će postati sadržaj taga
- ◆ **templateUrl:** relativna putanja templejt fajla u odnosu na tekuću komponentu
- ◆ **providers:** servisi potrebni tekućoj komponenti
 - Alternativa je upotreba *import* direktive
- ◆ **styleUrl:** relativna putanja css fajla u odnosu na tekuću komponentu

Angular korena komponenta - kontroler

app.component.ts:

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Tour of Heroes';
}
```

Angular korena komponenta - templejt

app.component.html:

```
<h1>{{title}}</h1>
<nav>
  <a routerLink="/dashboard">Dashboard</a>
  <a routerLink="/heroes">Heroes</a>
</nav>
<router-outlet></router-outlet>
<app-messages></app-messages>
```

Angular korena komponenta - CSS

app.component.css:

```
h1 {  
    font-size: 1.2em;  
    color: #999;  
    margin-bottom: 0;
```

```
nav a {  
    padding: 5px 10px;  
    text-decoration: none;  
    margin-top: 10px;  
    display: inline-block;  
    background-color: #eee;  
    border-radius: 4px;  
}
```

Heroes komponenta - metapodaci

```
import { Component, OnInit } from '@angular/core';
import { Hero } from '../hero';
import { HeroService } from '../hero.service';

@Component({
  selector: 'app-heroes',
  templateUrl: './heroes.component.html',
  styleUrls: ['./heroes.component.css']
})
export class HeroesComponent implements OnInit {
  heroes: Hero[];
```

Životni ciklus komponente : metode

- ◆ Metode omogućavaju dodatnu obradu u specifičnoj fazi životnog ciklusa:
- ◆ ***OnChanges*** – poziva se prva i slučaju promene atributa (podataka) komponente
- ◆ ***OnInit*** – poziva se prilikom inicijalizacije sadržaja (podataka) i izgleda komponente; poziva se jednom i to posle prvog OnChanges poziva
- ◆ ***DoCheck*** – poziva se u slučaju detekcije promene; uvek posle prethodne dve
- ◆ ***OnDestroy*** – poziva se pre uklanjanja komponente; otkazivanje registrovanih osluškivača događaja

Životni ciklus komponente : faze

- ◆ Kreiranje komponente : `onInit`
- ◆ Prikazivanje komponente : `onInit`
- ◆ Kreiranje i prikazivanje komponenti dece : `onInit`
- ◆ Ažuriranje komponente : `onChanges`, `doCheck`
- ◆ Uništavanje komponente : `onDestroy`
 - Uklanjanje iz DOM stabla

Upravljanje životnim ciklusom komponente

◆ Implementacija odgovarajućeg interfejsa

- `onInit`,
- `onChanges`,
- `doCheck`,
- `onDestroy`

◆ Implementacija *handler metode* propisane interfejsom

◆ *Lifecycle hook*

- **`onInit -> ngOnInit()`**
- `onChanges -> ngOnChanges`
- `doCheck -> ngDoCheck()`
- `onDestroy -> ngOnDestroy()`

Životni ciklus komponente - primer

```
import { Component, OnInit } from '@angular/core';
import { Hero } from '../hero';
import { HeroService } from '../hero.service';

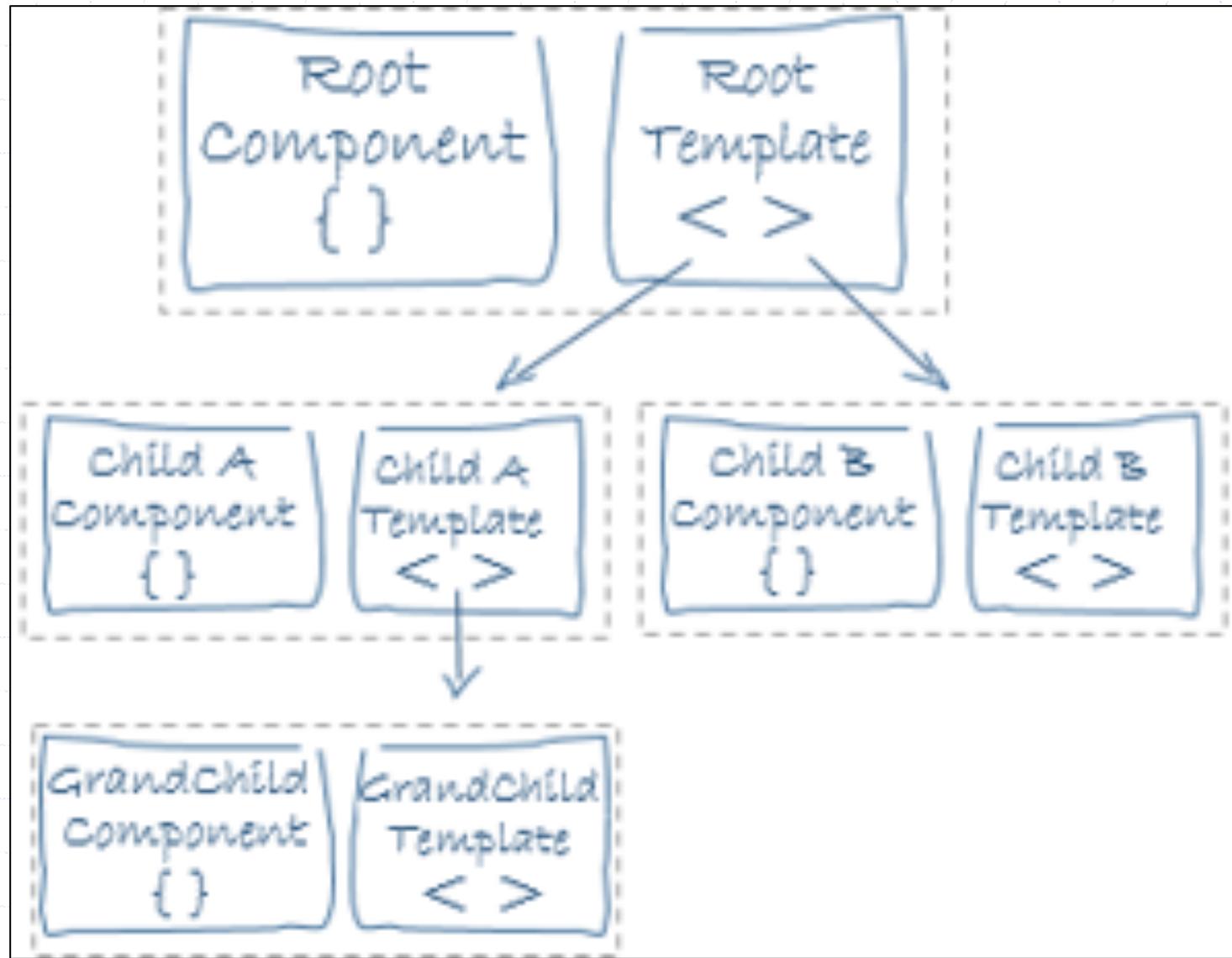
@Component({
  selector: 'app-heroes',
  templateUrl: './heroes.component.html',
  styleUrls: ['./heroes.component.css']
})
export class HeroesComponent implements OnInit {
  heroes: Hero[];

  constructor(private heroService: HeroService) { }

  ngOnInit() {
    this.getHeroes();
  }

  getHeroes(): void {
    this.heroService.getHeroes().subscribe(heroes => this.heroes = heroes);
  }
}
```

Komponente su uređene hijerarhijski



Komponenta – kontroler klasa i metode

```
export class HeroListComponent implements OnInit {  
  heroes: Hero[];  
  selectedHero: Hero;  
  
  constructor(private service: HeroService) { }  
  
  ngOnInit() {  
    this.heroes = this.service.getHeroes();  
  }  
  
  selectHero(hero: Hero) { this.selectedHero = hero; }  
}
```

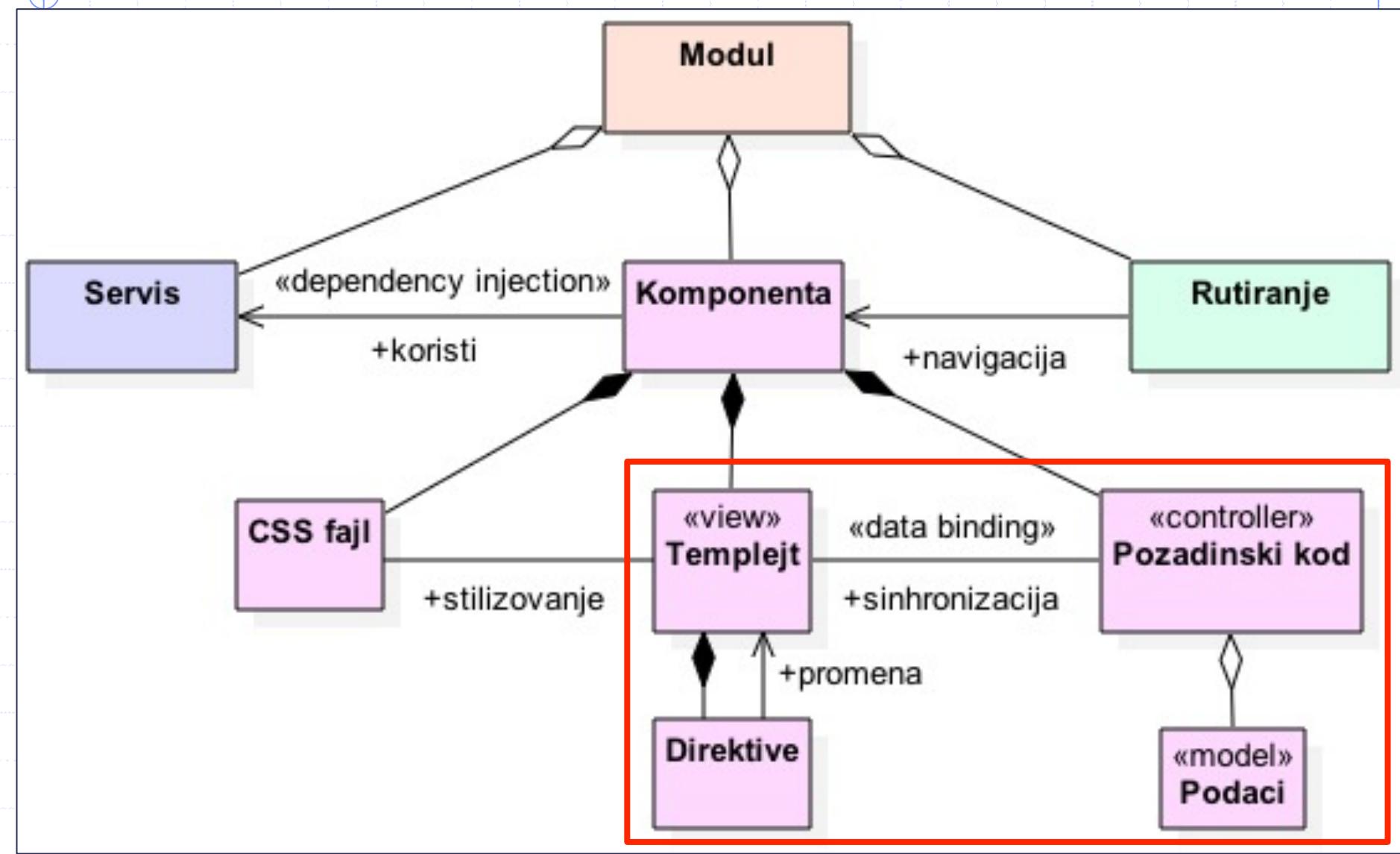
Primeri komponenti u praksi

- ◆ Tema1:
 - ◆ Putnik
 - ◆ Prevoznik
 - ◆ Prijava
 - ◆ Registracija
 - ◆ Vožnja
 - ◆ Istorijat
- ◆ Tema2:
 - ◆ Posetilac
 - ◆ Prijava
 - ◆ Registracija
 - ◆ Događaj
 - ◆ Preporuka
 - ◆ Istorijat

Templejt

- ◆ Korisnički interfejs, tj. izgled komponente
- ◆ HTML datoteka sa dinamičkim sadržajem - kombinuje HTML tagove sa Angular tagovima koji mogu modifikovati html elemente pre nego se prikažu
- ◆ Kako ?
- ◆ Upotreboom **direktiva** koje realizuju programsku logiku u okviru tagova i **binding** mehanizma za razmenu podataka sa kontrolerom
- ◆ Pre nego prikaže templejt, Angular izvršava sve direktive i binding mehanizme kako bi modifikovao templejt u skladu sa logikom i podacima komponente

Logička šema gradivnih elemenata Angular-a



Data binding

- ◆ Mehanizam komunikacije i sinhronizacije između kontrolera i modela komponente, i njenog templejta
- ◆ Proces povezivanja interfejsa i podataka komponente
- ◆ Naredbe koje se definišu u templejtu (*binding markup*)
- ◆ Mehanizam je dvosmeran (*two-way*)
 - Promene templejta se reflektuju na kontroler i model (ulaz korisnika)
 - Promene u kontroleru i modelu se reflektuju na templejt, tj. izgled (izlaz kao rezultat obrade ulaza korisnika)
- ◆ Sa konceptualnog aspekta postoje:

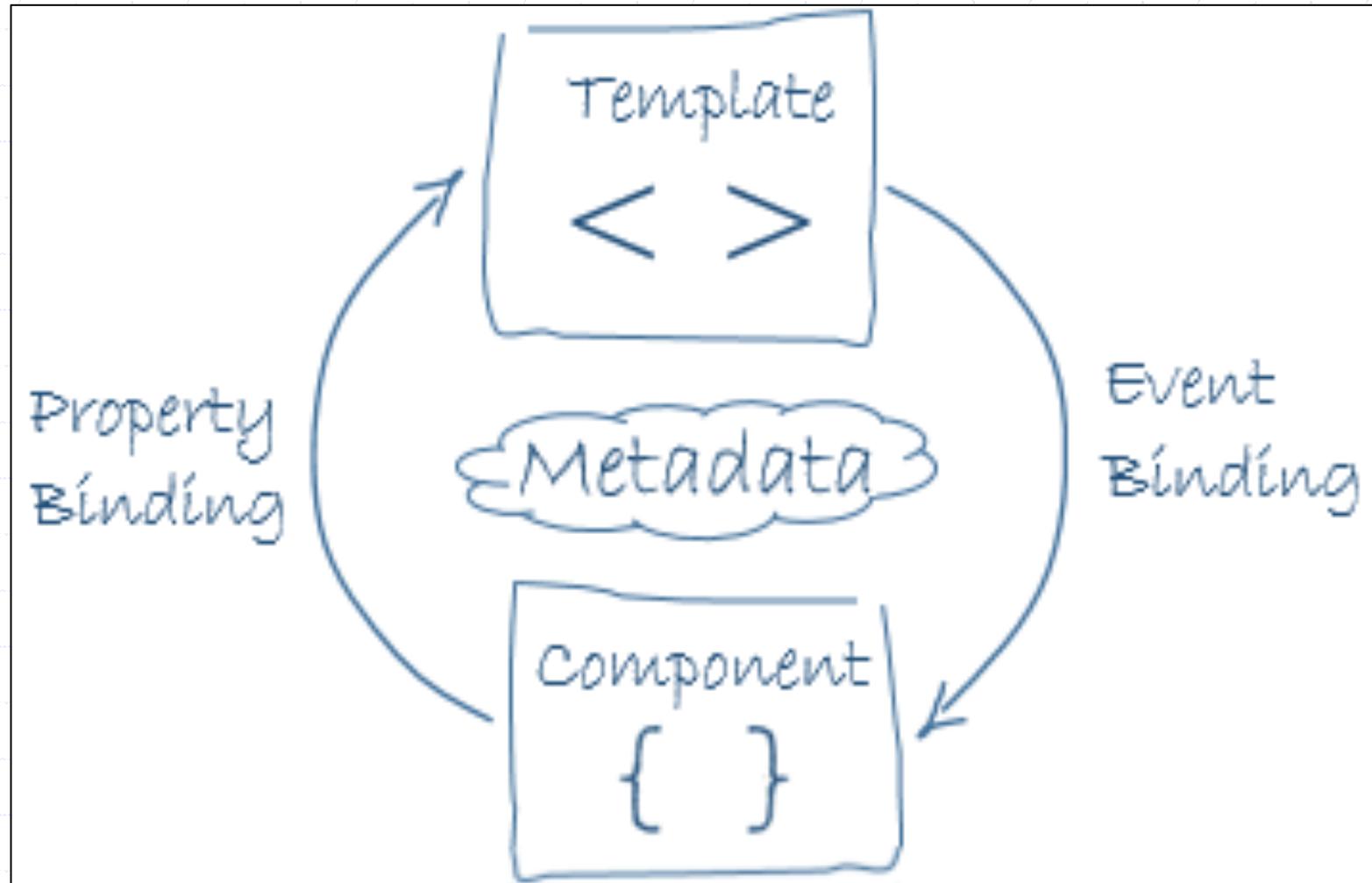
1. Event binding

- Procesiranje i odgovaranje na ulaz korisnika
- Događaji su programski objekti sa podacima

2. Property binding

- Povezivanje i procesiranje podataka iz programske logike komponente i modela

Event- i property- binding šema



Data binding – tipovi i smer razmene podataka

1. Vrednost komponente, tj. kontrolera (lokalna ili pomoćna promenjiva, ili vrednost izraza)
 1. Smer: Kontroler -> Templejt
 2. Sintaksa: {{ vrednost_izraza }}
2. Atribut (*property*) komponente
 1. Smer: Kontroler -> Templejt
 2. Sintaksa: [ime_atributa_templejta] = "ime_atributa_kontrolera"
3. Događaj korisnika (*event*)
 1. Smer: Templejt -> Kontroler
 2. Sintaksa: (ime_događaja) = "ime_handler_metode"
4. Two-way data binding
 1. Smer: Templejt <-> Kontroler
 2. Kombinuje *property* i *event* binding (2 i 3)
 3. Sintaksa: ([ngModel]) = "ime_atributa_kontrolera"

Data binding šema

TEMPLATE

KONTROLER

`{{vrednost}}`

`[atribut] = "vrednost"`

`(dogadjaj) = "ime_osluškivača"`

`[(ng-model)] = "atribut"`

Interpolacija {{}}

`{{ vrednost_izraza }}`

- ◆ **Vrednost izraza** se računa direktno u templejtu (html)
- ◆ Angular koristi podatke u okviru html templejta pomoću izraza (*expression*)
- ◆ Izrazi mogu sadržavati tekst, operatore i promenjive (tipa numerički, string, JS objekti i nizovi)
- ◆ Operator (*uslov* ? *true_izraz* : *false_izraz*) može biti izraz

Interpolacija {{}} - formatiranje

`{{ vrednost_izraza | format_prikaza }}`

- ◆ Pipe
- ◆ **Vrednost izraza** se računa direktno u templejtu (html) i prezentacija formatira na osnovu **specificiranog formata**
 - ◆ Najčešća upotreba: prikaz vremena i datuma
 - ◆ DatePipe
 - ◆ CurrencyPipe
 - ◆ DecimalPipe
 - ◆ PercentPipe

Interpolacija {{}} – formatiranje primer

```
<!-- Default format: output 'Jun 15, 2015'-->
```

```
<p>Today is {{today | date}}</p>
```

templejt

```
<!-- fullDate format: output 'Monday, June 15, 2015'-->
```

```
<p>The date is {{today | date:'fullDate'}}</p>
```

```
<!-- shortTime format: output '9:43 AM'-->
```

```
<p>The time is {{today | date:'shortTime'}}</p>
```

```
export class DatePipeComponent {
```

```
    today: number = Date.now();
```

```
}
```

kontroler

Izrazi - primer

◆ Numerički podaci

<p>Trošak: {{cena * količina}} Rs</p>

◆ Stringovi

<p>Zdravo {{student.ime + " " + student.prezime}}!</p>

◆ TS Objekti i atributi

<p> Broj upisa: {{student.upis}}</p>

◆ Nizovi

<p>Ocene(HCI): {{ocene[3]}}</p>

Property binding (zagrade [])

- ◆ Dodela vrednosti atributa kontrolera **atributu templejta**

[ime_atributa_templejta] = "ime_atributa_kontrolera"

- ◆ Prosleđivanje podataka od kontrolera ka templejtu
- ◆ Kreira se tzv. mapiranje ili parovi atributa
- ◆ Kao posledica, promena vrednosti atributa kontrolera automatski znači promenu vrednosti njegovog para, tj. atributa templejta
- ◆ Obrnuto ne važi, tj. promena se prenosi samo u smjeru od pozadinske logike ka templejtu

Property binding - primeri

```
<img [src]="heroImageUrl">
```

Povezivanje src atributa img taga templejta sa atributom heroImageUrl kontrolera, tj, pozadinske TS klase.
Promena vrednosti heroImageUrl će se automatski preslikati u src atribut.

```
<button [disabled]="isUnchanged">Cancel is disabled</button>
```

Povezivanje disabled atributa dugmeta templejta sa atributom isUnchanged kontrolera, tj, pozadinske TS klase.
Promena vrednosti isUnchanged će se automatski preslikati u disabled atribut.

Event binding (zagrade ())

- ◆ Koriste se za **simboličke nazine događaja** koji se generišu u templejtu

```
<element (event)="method()"></element>
```

- ◆ Prosleđivanje podataka od templejta ka kontroleru
 - Podatak = objekat događaj
- ◆ Uparivanje događaja i metode obrađivača događaja
 - Događaj se generiše u templejtu
 - Metoda obrađivača se definiše i izvršava u kontroleru

Event binding - primer

```
<button (click)="onClickMe()">Click me!</button>
```

```
@Component({  
  selector: 'app-click-me',  
  template: `  
    <button (click)="onClickMe()">Click me!</button>  
    {{clickMessage}}`  
})  
export class ClickMeComponent {  
  clickMessage = '';  
  
  onClickMe() {  
    this.clickMessage = 'You are my hero!';  
  }  
}
```

Obostrani (two-way) data binding

<element [(ngModel)]="ime_atributa_kontrolera">

- ◆ Obostrano mapiranje elementa templejta i atributa kontrolera:
- ◆ Promena sadržaja elementa templejta se automatski reflektuje na promenu vrednosti atributa
- ◆ Promena vrednosti atributa kontrolera se automatski reflektuje na promenu sadržaja elementa templejta
- ◆ Zasnovano na ngModel direktivi koja zapravo kreira par element-atribut za razmenu podataka
- ◆ Gde se najčešće koristi ?
- ◆ Forme, tj. input elementi

Two-way data binding - primer

```
<input [(ngModel)]="hero.name">
```

Automatizovan proces mapiranja i sinhronizacije elementa templejta (*input*) i atributa komponente (*hero.name*) upotrebom direkтиve (*ngModel*).

Kada korisnik unese vrednost u input polju templejta, ista vrednost će automatski biti dodeljena atributu *hero.name* u pozadinskoj klasi (kontroleru).

Analogno event binding-a, sa tim što se prosleđuje vrednost polja.

Važi obrnuto, tj. kada se vrednost atributa *hero.name* promeni u kontroleru (na primer, selekcija korisnika iz baze podataka), ista vrednost će se automatski pojaviti u input polju.

Analogno property binding-u.

Direktive

- ◆ Angular templejti su dinamički
- ◆ Prilikom njihovog prikazivanja, DOM se modificuje na osnovu specifičnih tipova naredbi koje se navode u samom templejtu
- ◆ Naredbe su poznate kao **direktive**
- ◆ Pojavljuju se kao instrukcije zapisane u tagovima templejtu
- ◆ Postoje predefinisane Angular direktive
- ◆ U zavisnosti od rezultata primene mogu biti
 - ***Strukturne** directive
 - Directive **[atributa]**

Strukturne direktive

- ◆ Menjaju izgled komponente tako što dodaju, uklanjuju ili zamenjuju html elemente
- ◆ Promena izgleda je posledica izvršavanja programa kontrolera
- ◆ Odgovaraju upravljačkim naredbama toka
 - Naredba grananja (if/else)
 - Ciklusi (for)
 - Uslovni izrazi (switch)
- ◆ Angular definiše tri osnovne strukturne direktive
- ◆ ***ngIf** – jednostruki uslov
- ◆ ***ngFor** – ciklus (petlja)
- ◆ ***ngSwitch** – višestruki uslov
- ◆ Najčešće korišćenje:
- ◆ Ispisivanje kolekcija elemenata kao html liste
- ◆ Prikaz/sakrivanje elementa u zavisnosti od toka/rezultata izvršavanja aplikacije

*ng-if direktiva za uslovni prikaz elementa

`<tag *ng-if="izraz">sadržaj_taga</tag>`

- ◆ Značenje: ako je izraz=true tag i njegov sadržaj se prikazuju, u suprotnom se ne prikazuju
- ◆ Slučaj korišćenja: deo templejta (stranice) se želi prikazati pod specifičnim uslovima
- ◆ Deo templejta može biti tag koji prikazuje specifičan objekat aplikacije (div i slično)
- ◆ Ukoliko je atribut kontrolera selektovan ili je njegova vrednost različita od null

*ng-if primer

```
<p *ngIf="true">
```

Expression is true and ngIf is true.

This paragraph is in the DOM.

```
</p>
```

```
<p *ngIf="false">
```

Expression is false and ngIf is false.

This paragraph is not in the DOM.

```
</p>
```

*ng-if primer - model

```
export class Hero {  
    id: number;  
    name: string;  
    emotion?: string;  
}  
  
export const heroes: Hero[] = [  
    { id: 1, name: 'Mr. Nice', emotion: 'happy' },  
    { id: 2, name: 'Narco', emotion: 'sad' },  
    { id: 3, name: 'Windstorm', emotion: 'confused' },  
    { id: 4, name: 'Magneta' }  
];
```

*ng-if primer – kontroler i templejt

```
import { Hero, heroes } from './hero';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: [ './app.component.css' ]
})
export class AppComponent {
  heroes = heroes;
  hero = this.heroes[0];
```

Ukoliko je atributu hero dodeljena vrednost ispisće se sadržaj div taga, tj. ime

```
<div *ngIf="hero" class="name">
  {{hero.name}}
</div>
```

*ngFor direktiva

- ◆ Sintaksa: <*tag* *ngFor="let *element* of *kolekcija*">
sadržaja_taga
</*tag*>

Značenje: ponavlja *tag* i *sadržaj_taga* dokle god postoji elementi u kolekciji; najčešće se ispisuje sadržaj elemenata kolekcije

Koristi se za ispisivanje kolekcija preko html lista, tj. *tag* je najčešće </i>

*ng-for primer - model

```
export class Hero {  
    id: number;  
    name: string;  
    emotion?: string;  
}  
  
export const heroes: Hero[] = [  
    { id: 1, name: 'Mr. Nice', emotion: 'happy' },  
    { id: 2, name: 'Narco', emotion: 'sad' },  
    { id: 3, name: 'Windstorm', emotion: 'confused' },  
    { id: 4, name: 'Magneta' }  
];
```

*ng-for primer – kontroler i templejt

```
import { Hero, heroes } from './hero';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: [ './app.component.css' ]
})
export class AppComponent {
  heroes = heroes;
  hero = this.heroes[0];
```

Imena heroja se
ispisuju za svaki
element kolekcije
heroja

```
<ul>
  <li *ngFor="let hero of heroes">{{hero.name}}</li>
</ul>
```

Direktive atributa

- ◆ Menjaju izgled ili ponašanje pojedinačnog elementa templejta ili komponente
- ◆ Izgledaju kao [atributi] html taga
 - ◆ Tri se najčešće koriste:
 - ◆ **[ngModel]** - implementira *two-way data binding*
 - ◆ **[ngClass]** – dodaje/uklanja eksterne css klase za definisanje izgleda tekućeg elementa (taga)
 - ◆ **[ngStyle]** - dodaje/uklanja inline css stilove za definisanje izgleda tekućeg elementa (taga)

ngModel – direktiva atributa

```
<input [(ngModel)]="hero.name">
```

Automatizovan proces mapiranja i sinhronizacije elementa templejta (*input*) i atributa komponente (*hero.name*) upotrebom direktive atrubuta (*ngModel*).

Kada korisnik unese vrednost u input polju templejta, ista vrednost će automatski biti dodeljena atributu *hero.name* u pozadinskoj klasi (kontroleru).

Analogno event binding-a, sa tim što se prosleđuje vrednost polja.

Važi obrnuto, tj. kada se vrednost atributa *hero.name* promeni u kontroleru (na primer, selekcija korisnika iz baze podataka), ista vrednost će se automatski pojaviti u input polju.

Analogno property binding-u.

Servisi (services)

- ◆ Klasa sa uskom, specifičnom i jasno definisom funkcionalnošću
 - Analogno interfejsima u OO programiranju, sa tim što može sadržati i attribute (podatke)
- ◆ Razdvajanje od komponente radi ponovne upotrebljivosti i modularnosti:
- ◆ Ponovna upotrebljivost: Izdvojene zajedničke funkcionalnosti koje različite komponente delegiraju servisima
- ◆ Modularnost: komponenta je fokusirana na interaktivnu logiku sa korisnikom, dok servisi implementiraju funkcionalnosti koje podržavaju ovu logiku
- ◆ Komponente koriste servise putem mehanizma koji se naziva **dependency injection**

Servisi i njihovo korišćenje

- ◆ Komunikacija sa serverskom stranom
- ◆ Prihvati i obrada složenijih struktura podataka modela koji pristižu od serverske strane kako bi ih komponente prikazale
 - Parsiranje JSON datoteka
- ◆ Logovanje akcija korisnika i funkcija interfejsa
 - console.log()
 - console.error()
 - console.warn()
- ◆ Validacija ulaza podataka
- ◆ Operacije nad modelom podataka složenijih elemenata interfejsa
 - Tabela

Servis – primer definicije

```
export class Logger {  
    log(msg: any) { console.log(msg); }  
    error(msg: any) { console.error(msg); }  
    warn(msg: any) { console.warn(msg); }  
}
```

Servis – primer korišćenja

```
constructor(  
    private backend: BackendService,  
    private logger: Logger) { }  
  
getHeroes() {  
    this.backend.getAll(Hero).then( (heroes: Hero[]) => {  
        this.logger.log(`Fetched ${heroes.length} heroes.`);  
        this.heroes.push(...heroes); // fill cache  
    });  
    return this.heroes;  
}
```

Programsko kreiranje servisa

`ng generate service ime_servisa`

`ng g s ime_servisa`

Konvencija imenovanja servisa:

ime_servisa.service.ts

Rezultat:

Servis se kreira kao TS datoteka sa nazivom

ime_servisa.service.ts

Klasa ima dekorator ***@Injectable***

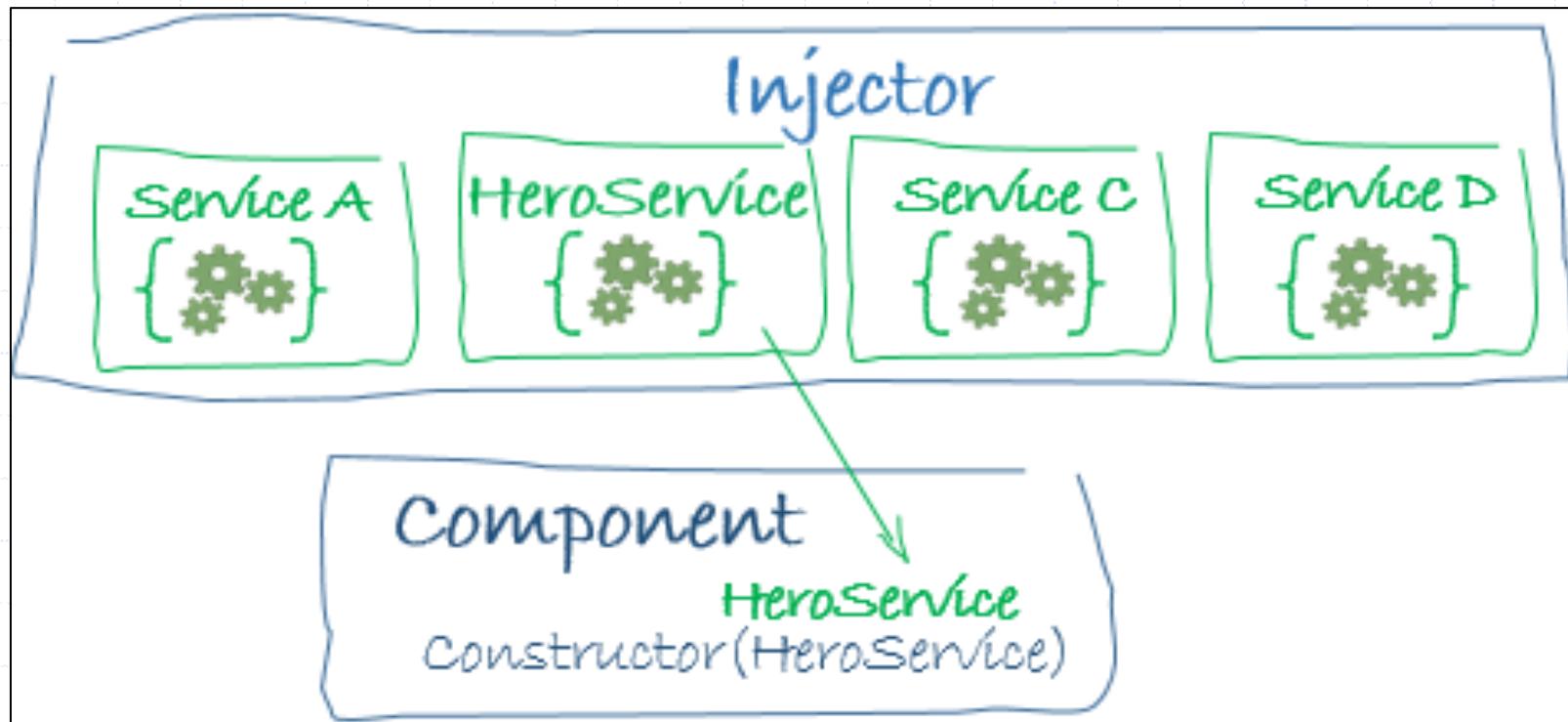
Dependency Injection (DI)

- ◆ Mehanizam koji omogućava korišćenje servisa od strane komponente
- ◆ Poreklo naziva: servis se *inject-uje* u komponentu kako bi joj bio dostupan za korišćenje
- ◆ Sastoji se iz dva koraka:
 - 1. Registracija** servisa na nivou **aplikacije, modula ili komponente**
 - 2. Navođenje** servisa kao parametra **konstruktora** komponente (za komponentu koja koristi servis)
 - ◆ Prilikom kreiranja instance komponente, Angular analizira parametre konstruktora i proverava da li instance pronađenih servisa postoje
 - ◆ Ukoliko ne postoje, one se kreiraju i prosleđuju kao argumenti konstruktora
 - Pozadinski proces - Injector

Injector – pozadinski proces kao registar servisa

Pozadinski proces koji čuva parove komponenta-servis kao skup zavisnosti (reference na instance servise i komponente)

```
constructor(private service: HeroService) { }
```



Načini registracije servisa 1/3

Registracija na nivou **aplikacije**

ng generate service

```
@Injectable({  
  providedIn: 'root',  
})
```

Jedna instanca servisa se kreira za čitavu aplikaciju.

Sve komponente dele jednu instancu servisa !

Načini registracije servisa 2/3

Registracija na nivou modula

```
@NgModule({  
  providers: [  
    BackendService,  
    Logger  
  ],  
  ...  
})
```

Instanca servisa se kreira na nivou modula
Sve komponente datog modula dele jednu instancu servisa !

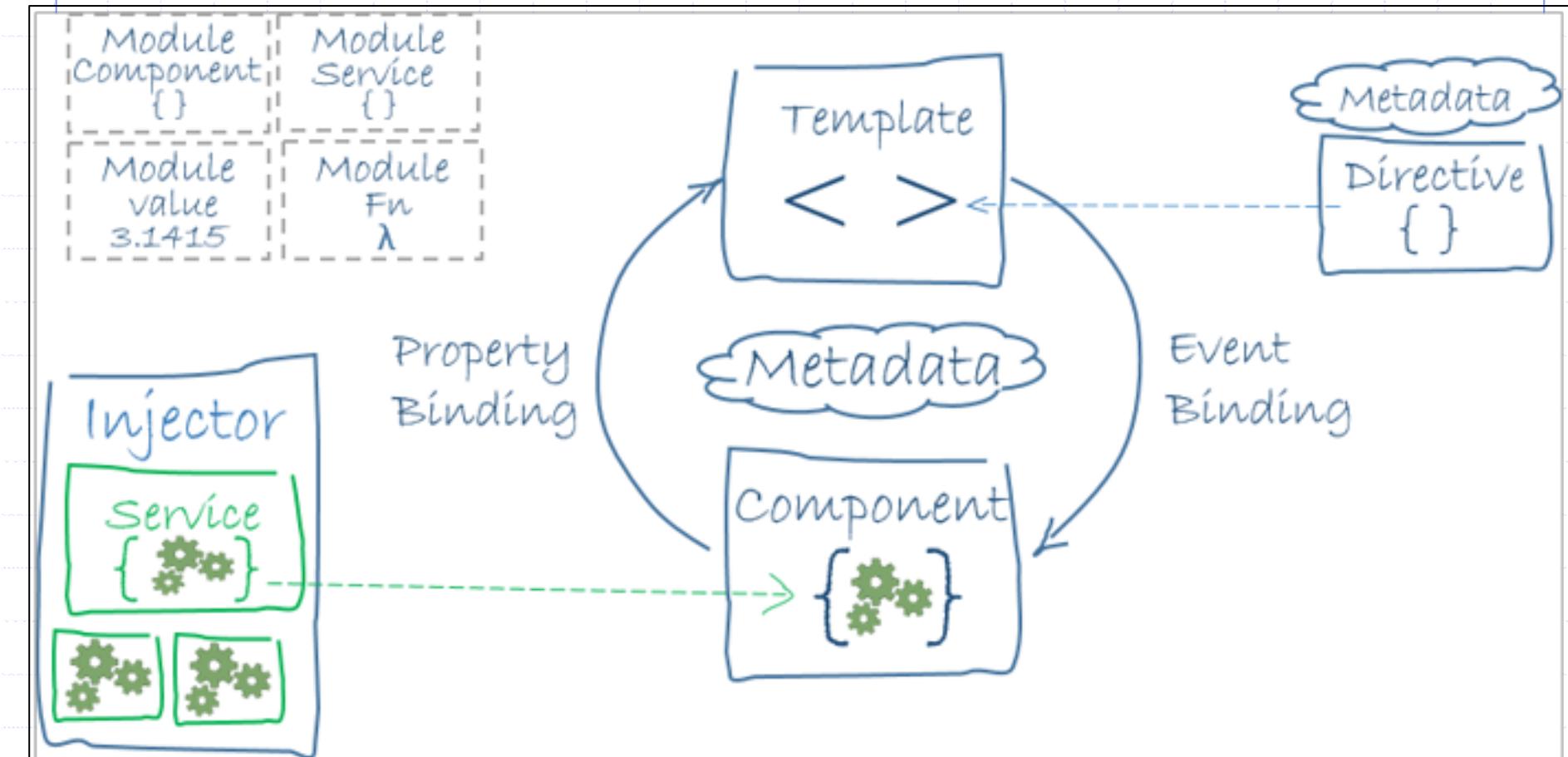
Načini registracije servisa 3/3

Registracija na nivou komponente

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

Instanca servisa se kreira za svaku kreiranu instancu komponente !

Angular – sve komponente



Sažetak

- ◆ Moduli
- ◆ Komponente
- ◆ Templejti
- ◆ Data binding
- ◆ Izrazi
- ◆ Direktive
- ◆ Servisi
- ◆ Dependency injection

Linkovi ka dokumentaciji i primerima

- ◆ Literatura: <https://angular.io/guide/architecture>
- ◆ Podsetnik: <https://angular.io/guide/cheatsheet>
- ◆ Strukturne direktive:
<https://stackblitz.com/angular/jamyalrxmkoa>
- ◆ Servisi:
<https://stackblitz.com/angular/bapgknajrqa>
- ◆ Tour of Heroes aplikacija:
<https://stackblitz.com/angular/bekyekexvog>

MVC (Model-View-Controller) Angular arhitektura i gradivni elementi

Dr. Mlađan Jovanović
mjovanovic@singidunum.ac.rs

Rutiranje

Konfiguracija projekta

Servisi: simulacija pozadinske

logike

Angular Material uvod

Dr. Mlađan Jovanović
mjovanovic@singidunum.ac.rs

Sadržaj

- ◆ Rutiranje
- ◆ Kreiranje i konfiguracija projekta
- ◆ Servisi
 - Slučajevi korišćenja kao simulacija pozadinske logike
- ◆ Angular Material komponente korisničkog interfejsa:
- ◆ Instalacija
- ◆ Kolekcija komponenti
- ◆ Teme
- ◆ Korišćenje (dugme, simboli)

Model navigacije klijentskih Web aplikacija

◆ Klasičan model navigacije:

- ◆ Korisnik unosi URL adresu i pretraživač otvara datu stranicu
- ◆ Birajući linkove na stranici, pretraživač upućuje korisnika na nove stranice
- ◆ Forward i back dugmad se mogu koristiti za kretanje kroz istorijat posećenih stranica

◆ Angular model navigacije:

- ◆ Analogno modelu iznad, sa tim što se navigacija definiše do komponenti
 - ◆ Ono što se prikazuje je templejt komponente (html)
 - ◆ Istorijat se čuva na identičan način (back/forward)

Rutiranje

- ◆ Angular mehanizam koji omogućava definisanje putanja do komponenti aplikacije i njihovo prikazivanje
- ◆ Putanja se definiše do komponente, a ne stranice
- ◆ Različite komponente se prikazuju u okviru jedne stranice
- ◆ SPA (Single Page Application)
- ◆ Nema ponovnog učitavanja čitave stranice, već samo dela (taga) u okviru kojeg se prikazuje komponenta

Definisanje ruta

- 1. Učitavanje potrebnih modula**
- 2. Definisanje ruta**
- 3. Definisanje mesta prikazivanja komponenti
na koje ukazuju rute**
- 4. Definisanje linkova ka rutama**

R1 - Učitavanje potrebnih modula

Angular Router je opcioni deo Angular-a čije su funkcionalnosti implementirane u biblioteci [@angular/router](#)

Na nivou Angular aplikacije se kreira jedna instanca Router servisa (*singleton*)

RouterModule – ugrađeni Angular modul koji implementira mehanizam rutiranja

Routes – Struktura podataka koja definiše niz ruta.

Rute su predstavljene kao parovi

putanja do komponente : naziv komponente

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

R2 – Definisanje ruta

```
const rute:Routes = [
  {path: '', component: WelcomeComponent},
  {path: 'signup', component: SignupComponent},
  {path: 'login', component: LoginComponent},
  {path: 'training', component: TrainingComponent}
]
@NgModule({
  imports: [RouterModule.forRoot(rute)],
  exports: [RouterModule]
})
```

R2 – Definisanje ruta

- ◆ Angular aplikacija ima jednu instancu Router-a
- ◆ Prilikom navođenja URL adrese u pretraživaču, Router traži rutu na osnovu path atributa kako bi identifikovao komponentu koju treba da prikaže
- ◆ Rute se definišu kao niz Routes objekata
 - const rute : Routes
 - Prazna putanja ('') predstavlja home stranicu aplikacije
- ◆ Routes objekat ima dva atributa
 - path – relativna putanja do komponente (bez leading slash '/')
 - componente – naziv komponente do koje vodi putanja
- ◆ Naredba u imports atributu modula konfiguriše Router za datu Angular aplikaciju
 - RouterModule.forRoot(rute)

R3 – Definisanje mesta prikazivanja komponente

router-outlet: tag koji označava mesto u okviru templejta korene komponenta (app.component.html) na kojem se prikazuje sadržaj komponente do koje vodi ruta
Placeholder za komponentu koja će biti prikazana kada se unese odgovarajuća URL adresa
Router na osnovu unete URL adrese pronađe komponentu i prikazuje kao sadržaj router-outlet taga

```
<router-outlet></router-outlet>
```

R4 – Definisanje linkova ka rutama

Elementi korisničkog interfejsa koji korisniku omogućavaju navigaciju

Mogu se navoditi kao linkovi (`<a>`) sa direktivom `routerLink` koja ukazuje na konkretnu rutu

Navodi se sa leading slash '/' simbolom

Može se korisiti i `href`

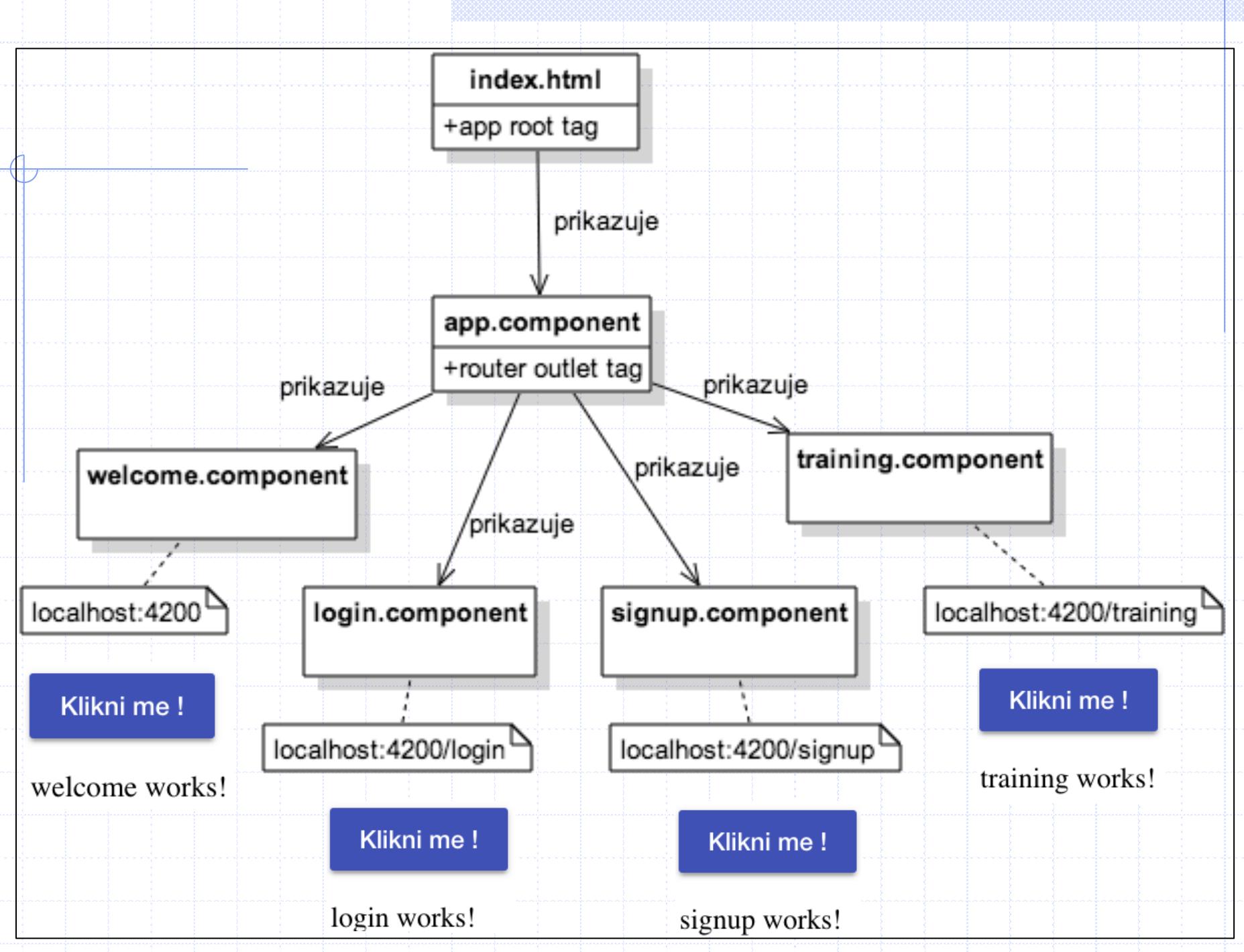
Vrednosti direktiva moraju odgovarati onima specificiranim u definiciji ruta

```
<a routerLink="/signup">Signup</a>
<a routerLink="/login">Login</a>
<a routerLink="/training">Training</a>
<a routerLink="/">Logout</a>
```

routing.module.ts

```
import { NgModule } from '../../../../../node_modules/@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { WelcomeComponent } from './welcome/welcome.component';
import { SignupComponent } from './auth/signup/signup.component';
import { LoginComponent } from './auth/login/login.component';
import { TrainingComponent } from './training/training.component';

const route: Routes = [
{path : '', component: WelcomeComponent},
{path : 'signup', component: SignupComponent },
{path : 'login', component: LoginComponent},
{path : 'training', component: TrainingComponent}
]
@NgModule ({
    imports: [RouterModule.forRoot(route)],
    exports: [RouterModule]
})
export class RoutingModule {}
```



Rutiranje – termini i njihovo značenje

| Koncept | Opis |
|--------------|---|
| RouterModule | Ugrađeni Angular module koji implementira funkcionalnosti rutiranja |
| Routes | Definiše niz ruta, gde svaka ruta definiše URL putanju do komponente (path-component) |
| RouterOutlet | HTML direktiva (<router-outlet>) koja definiše mesto prikazivanja templejta komponente Placeholder element |
| RouterLink | Atribut HTML taga koji sadržu putanju do rute (link ka ruti za korisnika) |

Angular – osnovne instalacije

◆ Visual Studio Code:

<https://code.visualstudio.com/Download>

- Razvojno okruženje za Angular aplikacije

◆ Node.js: nodejs.org/download (preporucena verzija)

- Instalacija **Node Package Manager-a** (npm)

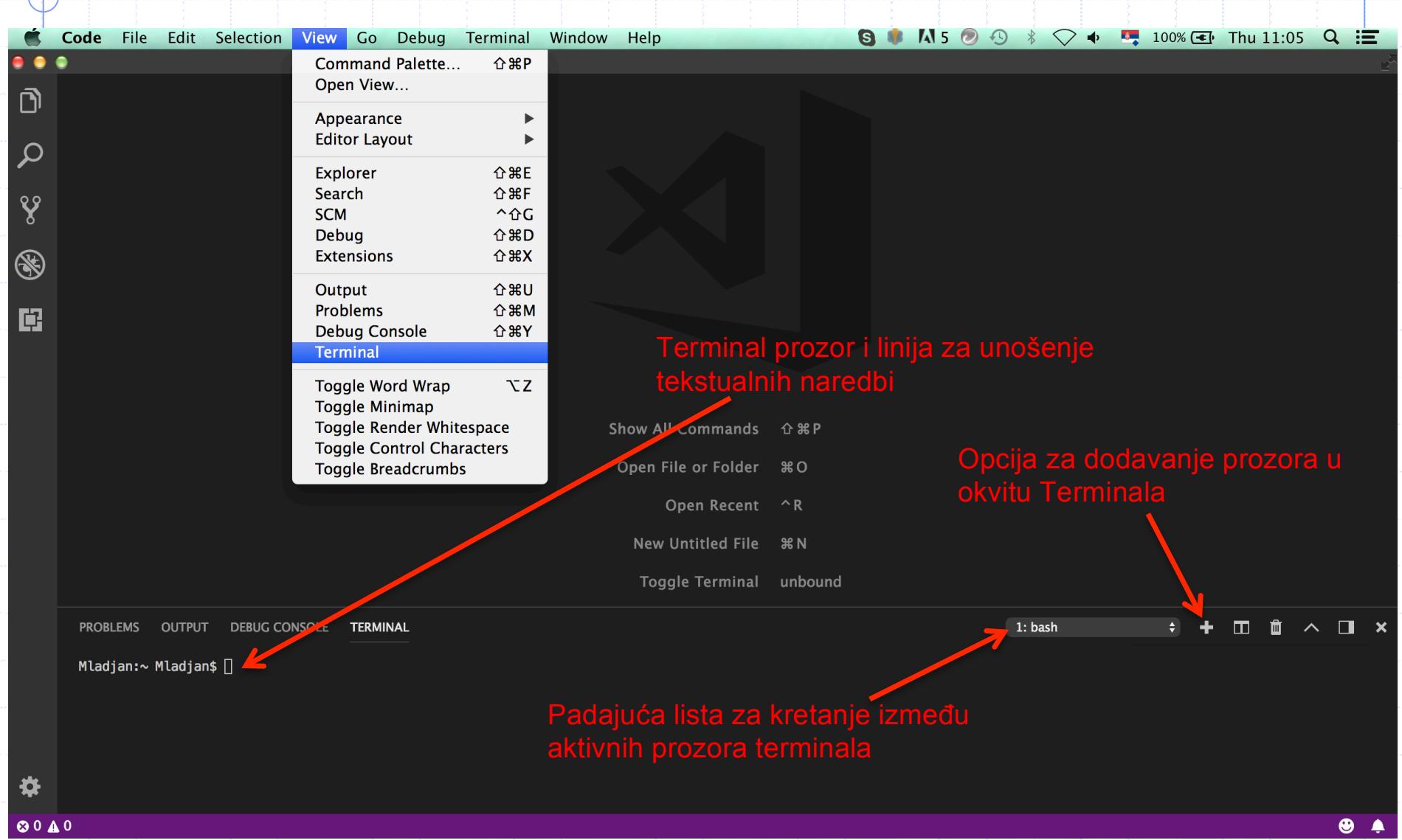
◆ Angular CLI: npm install -g @angular/cli

- Angular interfejs komandne linije (**Command Line Interface**)
- Programsko kreiranje projekta, komponenti, servisa, ...
- Pokretanje potrebnih instalacija

◆ Podrška za Angular Material komponente korisničkog interfejsa

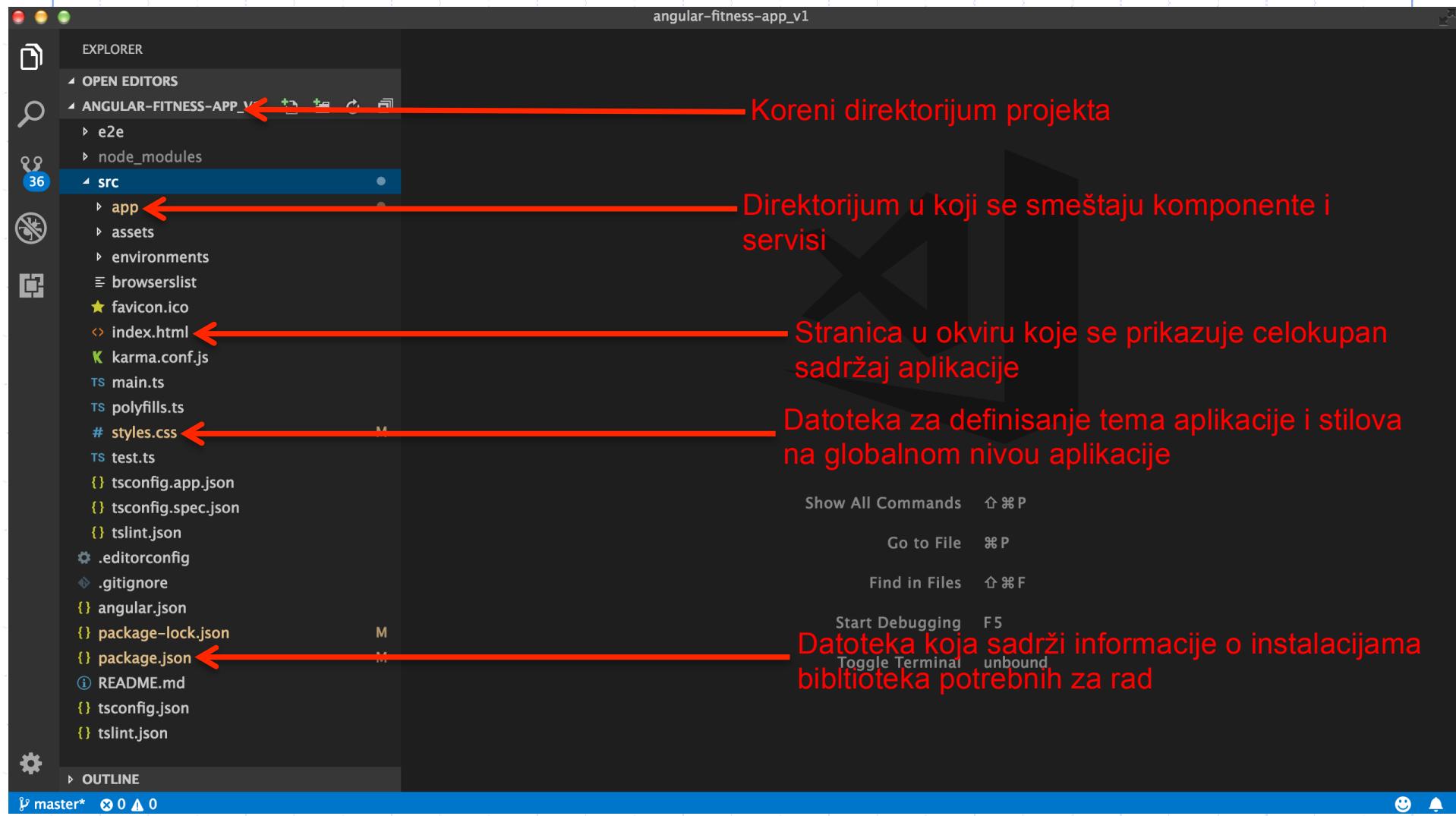
- npm install --save @angular/material @angular/cdk
@angular/animations

Razvojno okruženje



Kreiranje Angular projekta

`ng new ime_projekta`



Struktura Angular projekta

- ◆ **/src/**

- Kod Angular aplikacije

- ◆ **/node_modules/**

- npm paketi instalirani za projekat korišćenjem npm install naredbe

- ◆ **package.json**

- Zavisnosti (dependencies) kao verzije instaliranih biblioteka neophodne za prevođenje i izvršavanje programa

- ◆ **tsconfig.json**

- Konfiguracioni fajl za TypeScript kompajler

/src/ direktorijum Angular projekta

◆ **/app/**

- Elementi aplikacije (komponente, moduli i servisi)

◆ **/environments/**

- Promenjive okruženja (na primer, razvojno ili produpciono okruženje)

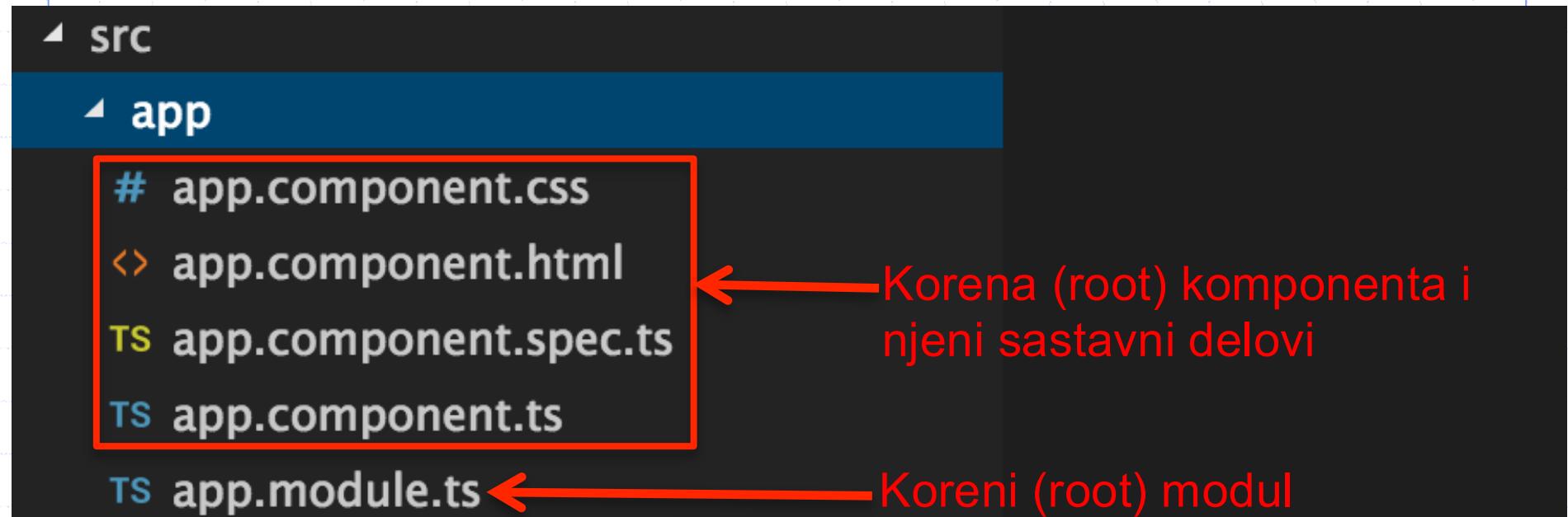
◆ **index.html**

- Home stranica Angular aplikacije
- SPA (Single Page Application), Single Page ≈ index.html

◆ **/assets/**

- Ovde se mogu smestiti objekti koje koristi aplikacija (slike, testni podaci i slično)

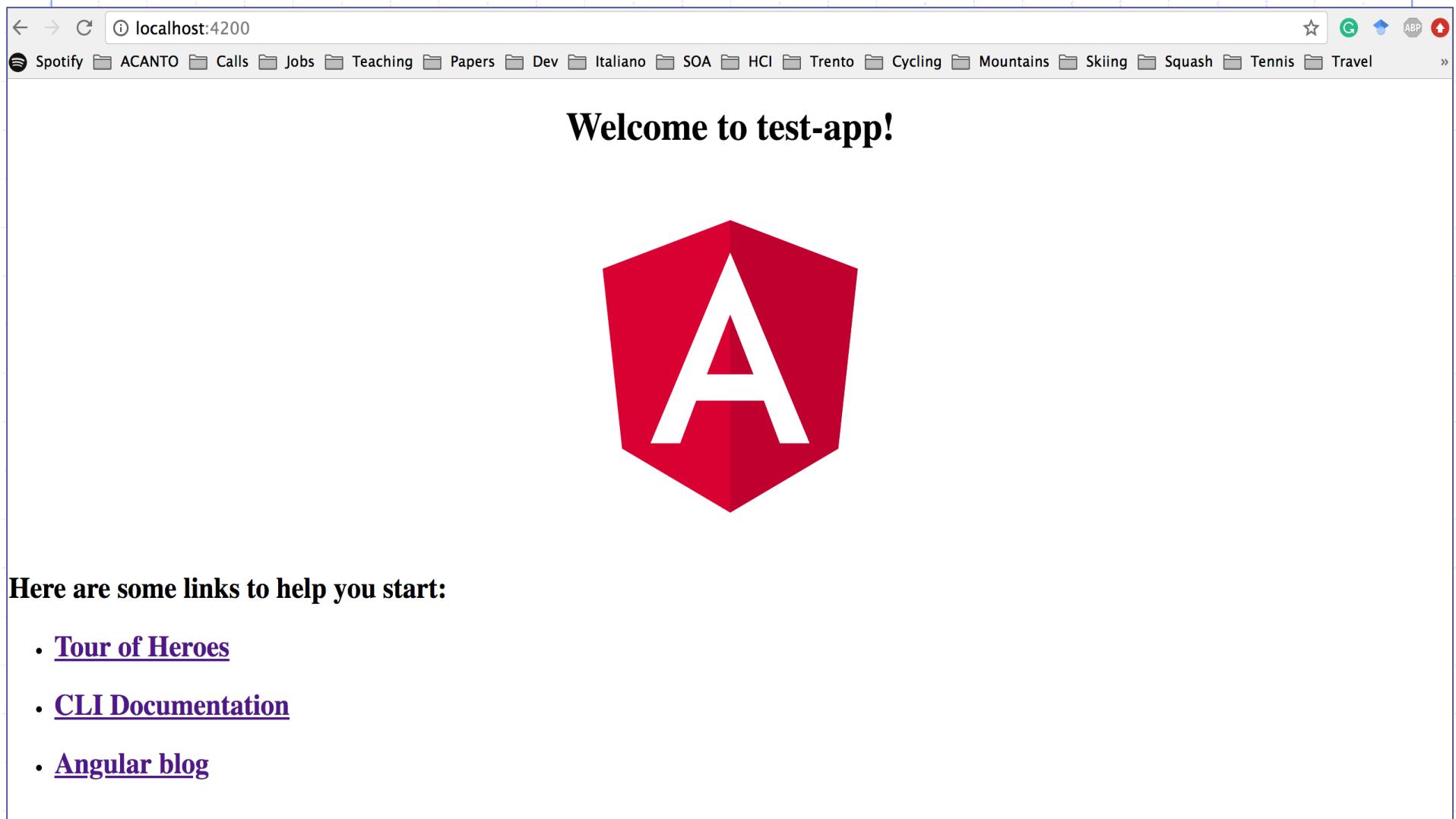
Korena komponenta i modul



Razvojni Web server

- ◆ Testiranje aplikacije iz razvojnog okruženja
- ◆ Terminal prozor CLI (interfejs komandne linije):
 - ng serve**
 - ng s**
- ◆ Testiranje aplikacije iz razvojnog okruženja
- ◆ Server automatski registruje i učitava promene prilikom svakog pamćenja promena (save)
- ◆ U nekim slučajevima se mora restartovati (ctrl+c)
 - Na primer, definisanje ruta
- ◆ Adresa razvojnog Web servera: localhost:4200

Razvojni server – nov projekat



A screenshot of a web browser window titled "localhost:4200". The address bar shows the URL. The browser interface includes standard navigation buttons (back, forward, search) and a toolbar with icons for Spotify, ACANTO, Calls, Jobs, Teaching, Papers, Dev, Italiano, SOA, HCI, Trento, Cycling, Mountains, Skiing, Squash, Tennis, and Travel. Below the toolbar, the main content area displays the text "Welcome to test-app!" in a large, bold, black font. In the center of the page is the large red Angular logo, which is a red hexagon containing a white stylized letter 'A'. At the bottom left of the page, there is a section of text that reads: "Here are some links to help you start:" followed by a bulleted list of three items: "Tour of Heroes", "CLI Documentation", and "Angular blog", each with a blue underline.

- [Tour of Heroes](#)
- [CLI Documentation](#)
- [Angular blog](#)

Programsko kreiranje komponenti

◆ Terminal prozor razvojnog okruženja:

ng g c auth/signup

ng g c auth/login

ng g c welcome

ng g c training

ng g c training/previous-training

ng g c training/current-training

ng g c training/new-training

Kreirane komponente

The image shows a screenshot of a code editor with a dark theme. On the left is the Explorer sidebar, which lists the project structure under 'ANGULAR-FITNESS-APP_V1'. The 'src' folder contains 'app', 'auth', 'login', 'signup', 'training', and 'welcome' components, each with their respective CSS, HTML, and TS files. The 'welcome' component is currently selected in the Explorer. On the right is the main editor area, showing the content of 'welcome.component.html'. The code is as follows:

```
1 <p>
2   welcome works!
3 </p>
4
```

Servisi – primeri korišćenja kao simulacija pozadinske logike

- ◆ Interaktivni prototip korisničkog interfejsa bez pozadinske logike, tj. implementacije backend-a
- ◆ Pozadinska logika se simulira u pozadinskom kodu aplikacije
- ◆ Koraci:
 - ◆ Kreiranje interfejsa koji definiše strukturu objekta aplikacije
 - ◆ Definisanje servisa koji implementira funkcije čitanja, izmene i brisanja objekata
 - Testni objekti se kreiraju kao kolekcija objekata interfejsa (na primer JS niz tipa object ili TS Array)



Simulacija pozadinske logike: Slučaj rada sa nalozima korisnika

1 - Kreiranje interfejsa kao strukture objekta aplikacije sa testnim podacima

```
export interface User {  
    id: number;  
    email: string;  
    password: string;  
    firstName: string;  
    lastName: string;  
    phone: string;  
    address: string;  
    type: 'driver' | 'passenger';  
    rating: number;  
}
```

2.1 – Definisanje servisa za rad sa testnim podacima interfejsa – rad sa korisničkim nalozima

```
@Injectable()
export class UserService {
    currentUser: User = UserService.dummyUserList[0];
    static dummyUserList: Array<User> = [
        {
            id: 1,
            email: "test@test.com",
            password: "test123",
            firstName: "Test",
            lastName: "Test",
            phone: "+3816711223344",
            address: "Test Adresa 1",
            type: "driver",
            rating: 2.5,
        },
    ],
}
```

2.2 – Definisanje servisa za rad sa testnim podacima interfesa – prijava korisnika

```
import { User, UserService } from './user.service';

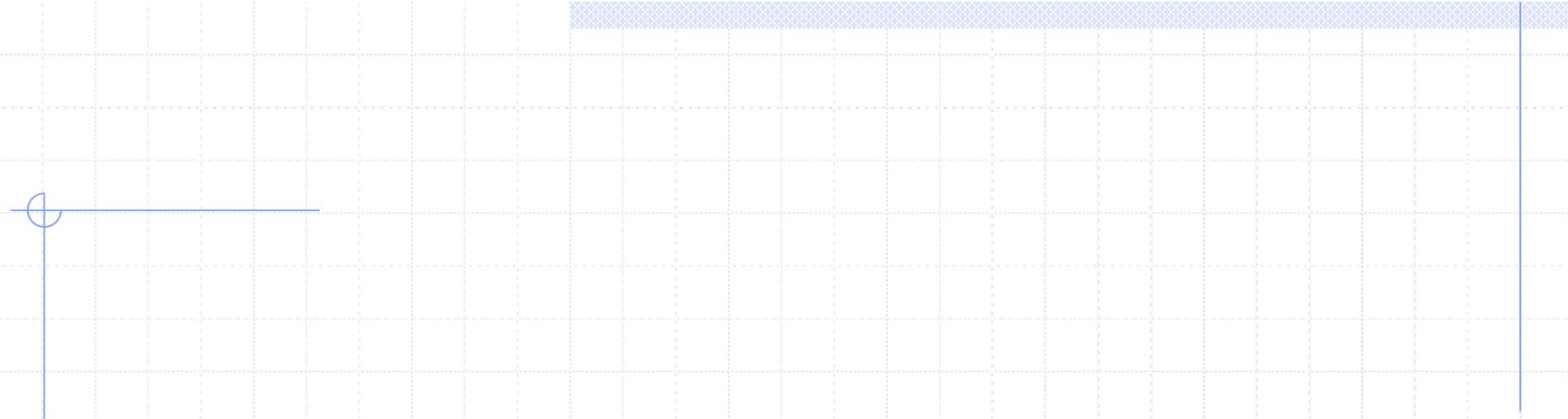
export class AuthService {
    registeredUsers: Array<User> = UserService.dummyUserList;

    getUser(userEmail: string) {
        return this.registeredUsers.find(userToFind =>
            userToFind.email == userEmail);
    }

    isPasswordCorrect(userEmail: string, password: string) {
        return this.registeredUsers.find(userToFind => (
            userToFind.email == userEmail &&
            userToFind.password == password)) != undefined;
    }
}
```

2.3 – Definisanje servisa za rad sa testnim podacima interfesa – registracija korisnika

```
registerUser(email: string, password: string, firstName: string,  
lastName: string, phone: string, address: string, type) {  
    var maxId: number = 0;  
    this.registeredUsers.forEach(user => {  
        if (maxId < user.id) {  
            maxId = user.id;  
        }  
    });  
    var id = ++maxId;  
    var user: User = {id, email, password, firstName,  
                    lastName, phone, address, type, rating: 0};  
    this.registeredUsers.push(user);  
    UserService.dummyUserList.push(user);  
  
    return user;  
}
```



Simulacija pozadinske logike: Slučaj rada sa vožnjama

1 - Kreiranje interfejsa kao strukture objekta aplikacije sa testnim podacima

```
export interface Ride {  
    id: number;  
    title: string;  
    driverId: number;  
    passengerIds: Array<number>;  
    startTime: Date;  
    startHours: number;  
    startMinutes: number;  
    startLocation: string;  
    destinationLocation: string;  
    openSlots: number;  
    status: 'open' | 'in progress' | 'finished' | 'cancelled';  
}
```

2 – Definisanje servisa za rad sa testnim podacima interfejsa – rad sa voznjama

```
@Injectable()  
export class RideService {  
  
    public dummyRideList: Array<Ride> = [  
        {  
            id: 1,  
            title: "We are going to stop at the post office on the way",  
            driverId: 1,  
            passengerIds: [],  
            startTime: new Date("2019-04-17 17:00"),  
            startHours: 17,  
            startMinutes: 0,  
            startLocation: "Danijelova 32",  
            destinationLocation: "Kumodraska 261a",  
            openSlots: 3,  
            status: 'cancelled'  
        },  
    ]};
```

2 – Definisanje servisa za rad sa testnim podacima interfejsa – čitanje vožnji

```
getAllRides() {
    return this.dummyRideList;
}

getDriverById(id: number) {
    var foundUser: User;
    UserService.dummyUserList.forEach(user => {
        if (user.id == id) {
            foundUser = user;
        }
    });
    return foundUser;
}

getRideById(id: number) {
    var foundRide: Ride;
    this.dummyRideList.forEach(ride => {
        if (ride.id == id) {
            foundRide = ride;
        }
    });
    return foundRide;
}
```

2 – Definisanje servisa za rad sa testnim podacima interfejsa – dodavanje i uklanjanje vožnji

```
createNewRide(ride: Ride) {  
    this.dummyRideList.push(ride);  
}  
  
removeRide(rideToDelete: Ride) {  
    this.dummyRideList = this.dummyRideList.filter(ride =>  
        ride.id != rideToDelete.id);  
}
```

Korišćenje servisa iz kontrolera komponente

Testni podaci kao model za tabelu

```
export class RidesComponent implements OnInit {  
    displayedColumns = ["title", "driver", "startLocation",  
    | | | | | | | | "destination", "date", "status"];  
    ridesSource = new MatTableDataSource<Ride>()  
  
    @ViewChild(MatSort) sort: MatSort;  
    @ViewChild(MatPaginator) paginator: MatPaginator;  
  
    constructor(private rideService: RideService,  
    | | | | | private userService: UserService) { }  
  
    ngOnInit() {  
        this.ridesSource.data = this.rideService.getAllRides();  
    }  
}
```

Angular Material (AM)

- ◆ Angular komponente korisničkog interfejsa koje implementariju Google Material Design specifikaciju
- ◆ **Google Material Design:** standard za kreiranje upotrebljivih Web korisničkih interfejsa
 - Komponente
 - Boje, oblici, tipografija, simboli
 - Vizuelni dizajn i raspoređivanje
- ◆ Angular Material: implementacija standarda
 - Fokus na komponente i raspoređivanje
- ◆ Kreiranje korisnih i upotrebljivih Web korisničkih interfejsa
- ◆ Olakšava i ubrzava razvoj Web korisničkih interfejsa
 - Efikasnost (brzina) – predefinisane komponente koje razvoj čine bržim
 - Efektivnost (kvalitet) – implementacija GMD kao prakse dobrog dizajna

Angular Material (AM)

◆ Sastoji se iz dva dela:

◆ Components

- Elementi korisničkog interfejsa
- Dugme
- Meni
- Tabela ...

◆ Component Dev Kit (CDK)

- Zajedničke funkcionalnosti komponenti
- Drag&Drop
- Pagination
- Layout
- Scroll ...

AM Components

<https://material.angular.io/components/categories>

(izborom kategorije dobijaju se konkretni elementi)

Form Controls

Controls that collect and validate user input.

Navigation

Menus, sidenavs and toolbars that organise your content.

Layout

Essential building blocks for presenting your content.

Buttons & Indicators

Buttons, toggles, status and progress indicators.

Popups & Modals

Floating components that can be dynamically shown or hidden.

Data table

Tools for displaying and interacting with tabular data.

Komponente formi (Form Controls)

(izborom komponente dobija se opis izabrane komponente)

Autocomplete



Suggests relevant options as the user types.

Checkbox



Captures boolean input with an optional indeterminate mode.

Datepicker



Captures dates, agnostic about their internal representation.

Form field

Wraps input fields so they are displayed consistently.

Input



Enables native inputs to be used within a Form field.

Radio button



Allows the user to select one option from a group.

Select



Allows the user to select one or more options using a dropdown.

Slider



Allows the user to input a value by dragging along a slider.

Slide toggle



Captures boolean values as a clickable and draggable switch.

Komponente navigacije (Navigation)

Menu



A floating panel of nestable options.

Sidenav



A container for content that is fixed to one side of the screen.

Toolbar



A container for top-level titles and controls.

Komponente rasporedivanja (Layout)

Card



A styled container for pieces of itemized content.

Grid list



A flexible structure for presenting content items in a grid.

Tabs



Only presents one view at a time from a provided set of views.

Divider

A vertical or horizontal visual divider.

Expansion Panel

A container which can be expanded to reveal more content.

List



Presents conventional lists of items.

Stepper

Presents content as steps through which to progress.

Tree

Presents hierarchical content as an expandable tree.

Komponente dugmadi i indikatora (Buttons & Indicators)

Button



An interactive button with a range of presentation options.

Chips



Presents a list of items as a set of small, tactile entities.

Progress bar



A linear progress indicator.

Button toggle



A groupable on/off toggle for enabling and disabling options.

Icon



Renders a specified icon.

Badge

A small value indicator that can be overlaid on another object.

Progress spinner



A circular progress indicator.

Ripples

Directive for adding Material Design ripple effects

Komponente dijaloga (Popups & Modals)

1

Bottom Sheet

A large interactive panel primarily for mobile devices.

Dialog



Snackbar



Tooltip



Displays floating content when an object is hovered.

Komponente table (Popups & Modals)

Paginator

Controls for displaying paged data.

Sort header

Allows the user to configure how tabular data is sorted.

Table

A configurable component for displaying tabular data.

Dokumentacija AM komponente

◆ Opis svake komponente je dat u tri sekcije:

◆ **Pregled (overview)**

- Opis komponente sa primerima korišćenja

◆ **API**

- Definicija komponente
- Opisi atributa (properties)

◆ **Primeri (examples)**

- Primeri korišćenja komponente
- HTML, TS i CSS
- Mogućnost editovanja u StackBlitz online razvojnom okruženju

Primer dokumentacije AM komponente

OVERVIEW API EXAMPLES

<mat-radio-button> provides the same functionality as a native <input type="radio"> enhanced with Material Design styling and animations.

Basic radios

Option 1 Option 2

All radio-buttons with the same name comprise a set from which only one may be selected at a time.

Radio-button label

Contents

- Radio-button label
- Radio groups
- Use with @angular/forms
- Accessibility

AM - instalacija

- ◆ Podrška za AM se instalira na nivou konkretnog projekta

```
npm install --save @angular/material @angular/cdk  
@angular/animations
```

--save ažurira sadržaj package.json sa zavisnostima (verzije instaliranih biblioteka)

@angular/material – biblioteka AM komponenti

@angular/cdk – biblioteka zajedničkih funkcija komponenti (component dev kit)

@angular/animations – predefinisane animacije komponenti (na primer, efekat talasa na klik dugmeta)

Instalirane biblioteke – package.json

```
"dependencies": {  
    "@angular/animations": "^7.2.12",  
    "@angular/cdk": "^7.3.7",  
    "@angular/common": "~7.2.0",  
    "@angular/compiler": "~7.2.0",  
    "@angular/core": "~7.2.0",  
    "@angular/flex-layout": "^7.0.0-beta.24",  
    "@angular/forms": "~7.2.0",  
    "@angular/material": "^7.3.7",  
    "@angular/platform-browser": "~7.2.0",  
    "@angular/platform-browser-dynamic": "~7.2.0",  
    "@angular/router": "~7.2.0",  
    "core-js": "^2.5.4",  
    "hammerjs": "^2.0.8",  
    "rxjs": "~6.3.3",  
    "tslib": "^1.9.0",  
    "zone.js": "~0.8.26"  
},
```

Korišćenje AM komponenti

- ◆ Svaka AM komponenta ima svoju implementaciju
 - MatButtonModule, MatIconModule, MatTableModule
 - Nalaze su u @angular/material biblioteci
- ◆ Da bi se komponenta koristila mora se eksplisitno importovati u aplikaciju
- ◆ Da bi AM komponenta bila vidljiva u čitavoj aplikaciji **preporuka** je sledeća:
 1. Kreira se modul za registraciju AM komponenti u src/app direktotrijumu projekta (material.module.ts)
 2. Modul se zatim registruje u korenom modulu aplikacije (app.module.ts)
 3. Prilikom prvog korišćenja AM komponente, potrebno je registrovati komponentu u material.module.ts
- ◆ Na ovaj način AM komponenta je vidljiva u okviru čitave aplikacije, tj. ne mora se importovati u svakoj komponenti aplikacije koja koristi dati element interfejsa !

material.module.ts – modul za registraciju AM komponenti

Kreira se programski u src/app folderu
AM komponente se registruju kao ispod:

- ◆ import naredba
- ◆ imports i exports atributi @NgModule dekoratora

```
import { NgModule } from '....../node_modules/@angular/core';
import { MatButtonModule } from '@angular/material';

@NgModule ({
  imports: [MatButtonModule],
  exports: [MatButtonModule]
})

export class MaterialModule {}
```

app.module.ts (korenji modul)



Nako toga se kreirani modul registruje u korenom modulu aplikacije:

- ◆ **import** naredba u zaglavlju
- ◆ **imports** atribut **@NgModule** dekoratora

```
@NgModule({  
  declarations: [  
    AppComponent,  
    SignupComponent,  
    LoginComponent,  
    WelcomeComponent,  
    TrainingComponent,  
    PreviousTrainingComponent,  
    CurrentTrainingComponent,  
    NewTrainingComponent  
  ],  
  imports: [  
    BrowserModule,  
    BrowserAnimationsModule,  
    MaterialModule,  

```

Teme

- ◆ AM tema je skup boja koje će biti primenjene na komponente korisničkog interfejsa Angular aplikacije
- ◆ Generišu se statički, za vreme prevodenja aplikacije
- ◆ Tema se sastoji od pet paleta boja za vizuelno označavanje različitih stanja u kojima može biti komponenta korisničkog interfejsa:
 - ◆ **primarna** paleta (*primary*) - najčešće korišćene boje
 - ◆ **naglašena** (*accent*) paleta - boje za interaktivne elemente,
 - ◆ paleta **upozorenja** (*warn*) - boje za prikazivanje grešaka,
 - ◆ **frontalna** paleta (*foreground*) - boje za tekst i ikone (elementi u prvom planu),
 - ◆ **pozadinska** paleta (*background*) - boje za pozadine elementa

Tema – šematski prikaz sastavnih paleta

Primary Palette

Most widely used color across all screens

Accent Palette

Used on interactive elements

Warn Palette

Convey warnings

Foreground Palette

For text and icons

Background Palette

For backgrounds

Predefinisane teme

- ◆ Četiri predefinisane teme (naziv odgovara primarnoj boji ili paleti):
 - ◆ `deeppurple-amber.css`,
 - ◆ `indigo-pink.css`,
 - ◆ `pink-bluegrey.css`,
 - ◆ `purple-green.css`
- ◆ Nalaze se u `@angular/material/prebuilt-themes` biblioteci
- ◆ Da bi se koristile, tj, bile primenjene na elemente korisničkog interfejsa moraju se navesti eksplicitno u `styles.css`:
`@import '~@angular/material/prebuilt-themes/indigo-pink.css';`

Nazivi predefinisanih tema
odgovaraju nazivima
paleta boja
Google Material Design

deeppurple-amber.css

| | | | |
|----------------|---------|----------|---------|
| Deep purple 50 | #EDE7F6 | Amber 50 | #FFF8E1 |
| 100 | #D1C4E9 | 100 | #FFECB3 |
| 200 | #B39DDB | 200 | #FFE082 |
| 300 | #9575CD | 300 | #FFD54F |
| 400 | #7E57C2 | 400 | #FFCA28 |
| 500 | #673AB7 | 500 | #FFC107 |
| 600 | #5E35B1 | 600 | #FFB300 |
| 700 | #512DA8 | 700 | #FFA000 |
| 800 | #4527A0 | 800 | #FF8F00 |
| 900 | #311B92 | 900 | #FF6F00 |
| A100 | #B388FF | A100 | #FFE57F |
| A200 | #7C4dff | A200 | #ffd740 |
| A400 | #651FFF | A400 | #ffc400 |
| A700 | #6200EA | A700 | #ffab00 |

Nazivi predefinisanih tema
odgovaraju nazivima
paleta boja
Google Material Design

indigo-pink.css

| | | | |
|-----------|---------|---------|---------|
| Indigo 50 | #E8EAF6 | Pink 50 | #FCE4EC |
| 100 | #C5CAE9 | 100 | #F8BBDO |
| 200 | #9FA8DA | 200 | #F48FB1 |
| 300 | #7986CB | 300 | #F06292 |
| 400 | #5C6BC0 | 400 | #EC407A |
| 500 | #3F51B5 | 500 | #E91E63 |
| 600 | #3949AB | 600 | #D81B60 |
| 700 | #303F9F | 700 | #C2185B |
| 800 | #283593 | 800 | #AD1457 |
| 900 | #1A237E | 900 | #880E4F |
| A100 | #8C9EFF | A100 | #FF80AB |
| A200 | #536DFE | A200 | #FF4081 |
| A400 | #3D5AFE | A400 | #F50057 |
| A700 | #304FFE | A700 | #C51162 |

Nazivi predefinisanih tema
odgovaraju nazivima
paleta boja
Google Material Design

pink-bluegrey.css

| | | | |
|---------|---------|--------------|---------|
| Pink 50 | #FCE4EC | Blue Gray 50 | #ECEFF1 |
| 100 | #F8BBD0 | 100 | #CFD8DC |
| 200 | #F48FB1 | 200 | #B0BEC5 |
| 300 | #F06292 | 300 | #90A4AE |
| 400 | #EC407A | 400 | #78909C |
| 500 | #E91E63 | 500 | #607D8B |
| 600 | #D81B60 | 600 | #546E7A |
| 700 | #C2185B | 700 | #455A64 |
| 800 | #AD1457 | 800 | #37474F |
| 900 | #880E4F | 900 | #263238 |
| A100 | #FF80AB | | |
| A200 | #FF4081 | | |
| A400 | #F50057 | | |
| A700 | #C51162 | | |

Nazivi predefinisanih tema
odgovaraju nazivima
paleta boja
Google Material Design

purple-green.css

| | | | |
|-----------|---------|----------|---------|
| Purple 50 | #F3E5F5 | Green 50 | #E8F5E9 |
| 100 | #E1BEE7 | 100 | #C8E6C9 |
| 200 | #CE93D8 | 200 | #A5D6A7 |
| 300 | #BA68C8 | 300 | #81C784 |
| 400 | #AB47BC | 400 | #66BB6A |
| 500 | #9C27B0 | 500 | #4CAF50 |
| 600 | #8E24AA | 600 | #43A047 |
| 700 | #7B1FA2 | 700 | #388E3C |
| 800 | #6A1B9A | 800 | #2E7D32 |
| 900 | #4A148C | 900 | #1B5E20 |
| A100 | #EA80FC | A100 | #B9F6CA |
| A200 | #E040FB | A200 | #69F0AE |
| A400 | #D500F9 | A400 | #00E676 |
| A700 | #AA00FF | A700 | #00C853 |

Predefinisane teme

- ◆ Implementacije predefinisanih tema se mogu videti u projektu
- ◆ Generišu se sa AM instalacijom

node_modules/@angular/material/prebuilt-themes

```
◀ prebuilt-themes
# deeppurple-amber.css
# indigo-pink.css
# pink-bluegrey.css
# purple-green.css
```

Teme – primer korišćenja

- ◆ Jednom kada izaberemo temu, ona će se automatski primenjivati na komponente korisničkog interfejsa
- ◆ AM omogućava bojenje komponente tako što se *color* atributu komponente dodeli jedna od tri vrednosti - *primary*, *accent* ili *warn*
- ◆ Primer teme deeppurple-amber.css

```
<button mat-raised-button color="primary">Primary</button>
<button mat-raised-button color="accent">Accent</button>
<button mat-raised-button color="warn">Warn</button>
```

Primary

Accent

Warn

Animacije AM komponenti

- ◆ Angular omogućava korišćenje predefisanih animacija kao različitih oblika tranzicija (pomeraji i promene boja, oblika i pozicija)
- ◆ One su definisane kao modul (BrowserAnimationsModule) koji je potrebno registrovati u koreni modul (app.module.ts):

```
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
```

```
imports: [
  BrowserModule,
  MaterialModule,
  RoutingModule,
  BrowserAnimationsModule],
```

Dugme (button) 1/4

Korišćenje:

```
import {MatButtonModule} from '@angular/material';
imports i exports atributi @NgModul dekoratora
(material.module.ts)
```

Varijante:

Istaknuto dugme (*raised button*): <mat-raised-button>

```
<button mat-raised-button>Basic</button>
<button mat-raised-button color="primary">Primary</button>
<button mat-raised-button color="accent">Accent</button>
<button mat-raised-button color="warn">Warn</button>
<button mat-raised-button disabled>Disabled</button>
```

Basic

Primary

Accent

Warn

Disabled

Dugme (button) 2/4

Ovičeno dugme (*stroked button*): <mat-stroked-button>

```
<button mat-stroked-button>Basic</button>
<button mat-stroked-button color="primary">Primary</button>
<button mat-stroked-button color="accent">Accent</button>
<button mat-stroked-button color="warn">Warn</button>
<button mat-stroked-button disabled>Disabled</button>
```

Basic

Primary

Accent

Warn

Disabled

Ravno dugme (*flat button*): <mat-flat-button>

```
<button mat-flat-button>Basic</button>
<button mat-flat-button color="primary">Primary</button>
<button mat-flat-button color="accent">Accent</button>
<button mat-flat-button color="warn">Warn</button>
<button mat-flat-button disabled>Disabled</button>
```

Basic

Primary

Accent

Warn

Disabled

Dugme (button) 3/4

Dugme ikona (*icon button*): <mat-icon-button> i <mat-icon>

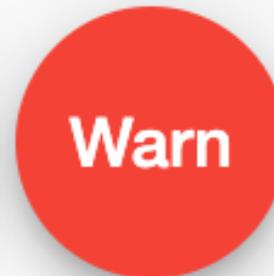
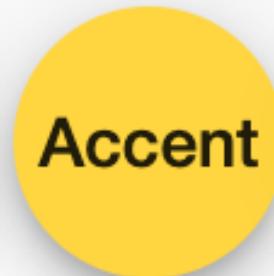
```
<button mat-icon-button>
  <mat-icon>favorite</mat-icon>
</button>
<button mat-icon-button color="primary">
  <mat-icon>favorite</mat-icon>
</button>
<button mat-icon-button color="accent">
  <mat-icon>favorite</mat-icon>
</button>
<button mat-icon-button color="warn">
  <mat-icon>favorite</mat-icon>
</button>
<button mat-icon-button disabled>
  <mat-icon>favorite</mat-icon>
</button>
```



Dugme (button) 4/4

Kružno dugme (*fab button*): <mat-fab> <mat-mini-fab>

```
<button mat-fab>Basic</button>  
<button mat-fab color="primary">Primary</button>  
<button mat-fab color="accent">Accent</button>  
<button mat-fab color="warn">Warn</button>  
<button mat-fab disabled>Disabled</button>  
  
<button mat-fab>  
  <mat-icon>favorite</mat-icon>  
</button>
```



Dugme alternativa

- ◆ Za svaki od podtipova umesto <button> se može koristiti <a>
- ◆ Na primer, za linkove ruta

```
<a mat-raised-button color="primary" routerLink=".">Link</a>
```



```
<a mat-fab routerLink=".">Link</a>
```



Ikone (icons)

- ◆ Predstavljaju grafičke simbole kao vektorske objekte (svg datoteke)
`<mat-icon>ime_simbola</mat icon>`
- ◆ Simboli se mogu instalirati lokalno, ili se može direktno referencirati Google Material Design kolekcija ikona
- ◆ Mogu se bojiti kao i dugmad (color = {primary | warn | accent})
- ◆ Upotrebe:
 - ◆ Dekorativna: informacija bez interaktivnosti (legenda)
 - ◆ Interaktivna: dugme ikona koja pokreće akciju
 - ◆ Indikator: poruka ili status izvršenja akcije (obično uz tekst)

Ikone (icons)

Korišćenje:

```
import {MatIconModule} from '@angular/material';
imports i exports atributi @NgModul dekoratora
(material.module.ts)
```

Interaktivna upotreba: <mat-icon> kao dete <button> ili
<a> elementa

Indikator: kombinacija sa inline kontejnerom za
ispisivanje poruke (kao deo bloka teksta)

Korišćenje Google Material kolekcije ikona

U index.html stranicu dodati referencu na kolekciju:

```
<link href="https://fonts.googleapis.com/icon?family=Material+Icons"
rel="stylesheet">
```

Google Material Design kolekcija ikona

<https://material.io/tools/icons/>

Simboli se referenciraju po nazivu

Mogu se koristiti i lokalno (download)

The screenshot shows a grid of Google Material Design icons. On the left, there is a sidebar with filtering options:

- Filter by name:** A search bar.
- THEMES:**
 - Filled
 - Outlined
 - Rounded
 - Two-Tone
 - Sharp
- CATEGORIES:** A dropdown menu set to "All".

The main area displays a 5x10 grid of icons, each with its name below it. The icons are categorized by theme:

- Filled:** 3d_rotation, accessibility, accessibility..., accessible, accessible_for..., account_balance, account_box, account_circle, add_shopping...
- Outlined:** alarm, alarm_add, alarm_off, alarm_on, all_inbox, all_out, android, announcement, arrow_right, aspect_ratio
- Rounded:** assessment, assignment, assignment_in..., assignment_la..., assignment_re..., assignment_re..., assignment_tu..., autorenew, backup, book
- Two-Tone:** bookmark, bookmark_bord..., bookmarks, bug_report, build, cached, calendar_today, calendar_view..., camera_enhanc..., card_gif...
- Sharp:** card_membership, card_travel, change_history, check_circle, check_circle..., chrome_reader..., class, code, commute, compare_arrow...

Ikone (icons)

```
<button mat-icon-button color="accent">  
    <mat-icon>home</mat-icon>  
</button>
```



Ako se online kolekcija ikona pretraži po ključnoj reči 'home' dobiće se simbol iznad



home

AM i teme podsetnik za rad

◆ AM komponente:

- ◆ Kreirati modul za registraciju AM komponenti (material.module.ts)
- ◆ Registrovati modul u korenom modulu aplikacije (app.module.ts)
- ◆ Svaku AM komponentu koja se želi koristiti je potrebno registrovati u material.module.ts

◆ Teme:

- ◆ Definisati globalnu temu na nivou aplikacije u styles.css
 - Izabrati postojeću ili kreirati svoju

Sažetak

- ◆ Rutiranje
- ◆ Kreiranje i konfiguracija projekta
- ◆ Servisi
 - Slučajevi korišćenja kao simulacija pozadinske logike
- ◆ Angular Material komponente korisničkog interfejsa:
- ◆ Instalacija
- ◆ Kolekcija komponenti
- ◆ Teme
- ◆ Korišćenje (dugme, simboli)

Materijali

- ◆ **Upustvo za kreiranje projekta:**

Angular – razvojno okruženje i kreiranje projekta.pdf

- ◆ **Instalacije:**

Instalacije.pdf

- ◆ **Servisi:**

auth.service.ts

ride.service.ts

user.service.ts

- ◆ **Projekat:**

projekat_primer.zip (kreiran po uputsvu)

Linkovi ka dokumentaciji i primerima

- ◆ Google Material Design: <https://material.io/design/>
- ◆ Angular Material: <https://material.angular.io/>
- ◆ Angular Material komponente:
<https://material.angular.io/components/categories>
- ◆ Angular Material teme:
<https://material.angular.io/guide/theming>
- ◆ Google Material Design paleta boja:
<https://material.io/design/color/the-color-system.html#tools-for-picking-colors>
- ◆ Primeri dugmeta za promenu tema i boja:
<https://stackblitz.com/angular/dnbnrjyaxaeqy?file=app%2Fbutton-types-example.html>
- ◆ Primer za simbole ikona:
<https://stackblitz.com/angular/yonlxjxpgjd?file=index.html>
- ◆ Google Material Design kolekcija ikona:
<https://material.io/tools/icons/>

Rutiranje

Konfiguracija projekta

Servisi: simulacija pozadinske

logike

Angular Material uvod

Dr. Mlađan Jovanović
mjovanovic@singidunum.ac.rs

Angular Material komponente

Dr. Mlađan Jovanović
mjovanovic@singidunum.ac.rs

Sadržaj

- ◆ Input
- ◆ Select
- ◆ Checkbox
- ◆ Forma (form)
 - Pristup podacima, korišćenje podataka, validacija
- ◆ Kalendar (DatePicker)
- ◆ Sidebar
- ◆ Toolbar
- ◆ Lista (list)
- ◆ Tab
- ◆ Kartica (card)
- ◆ Tabela
 - Prikaz, straničenje, sortiranje, filtriranje
- ◆ Dijalog (dialog)
 - Prosleđivanje, prihvata, prikaz podataka
- ◆ Napomena: Kreiranje i korišćenje modula za registraciju AM komponenti (material.module.ts) je opisano u prethodnoj lekciji

Input komponenta 1/3

Korišćenje:

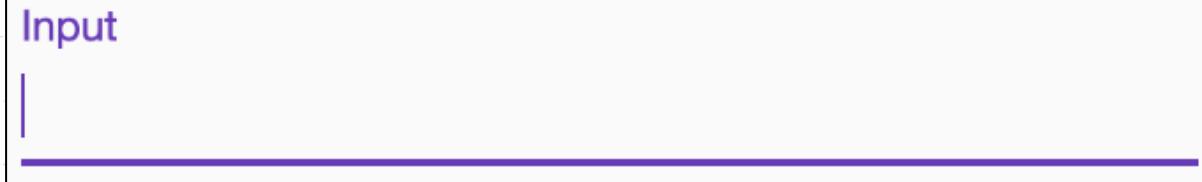
```
import {MatInputModule} from '@angular/material';
imports i exports atributi @NgModul dekoratora
(material.module.ts)
```

AM varijanta standardnih HTML tagova formi za unos podataka korisnika (<input> i <text area>)

Koriste se isključivo kao deo elementa forme !

input – jednolinijsko polje za unos teksta

```
<mat-form-field>
|   <input matInput placeholder="Input">
</mat-form-field>
```



Input

Input komponenta 2/3

textarea – višelinijsko polje za unos teksta

```
<mat-form-field>
|   <textarea matInput placeholder="Textarea"></textarea>
</mat-form-field>
```



Jednolinijsko polje može biti specifičnog formata sadržaja:

date
email
password
text
number

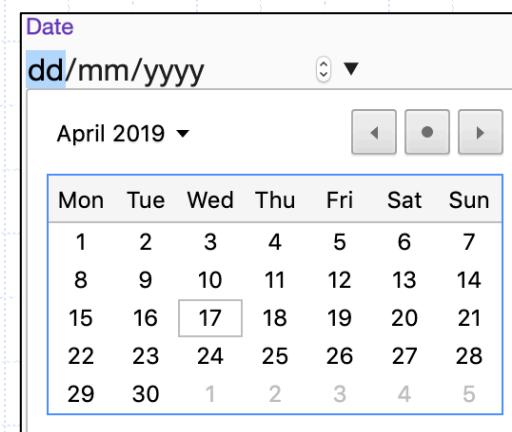
Input komponenta 3/3

```
<mat-form-field>
|   <input matInput type="email" placeholder="Email">
</mat-form-field>
```

Email

mladja@acm.org|

```
<mat-form-field>
|   <input matInput type="date" placeholder="Date">
</mat-form-field>
```

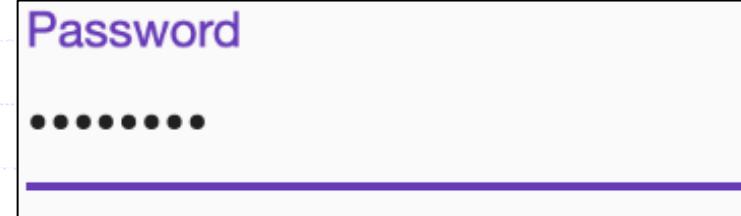


Input komponenta

```
<mat-form-field>
| <input matInput type="password" placeholder="Password">
</mat-form-field>
```

Password

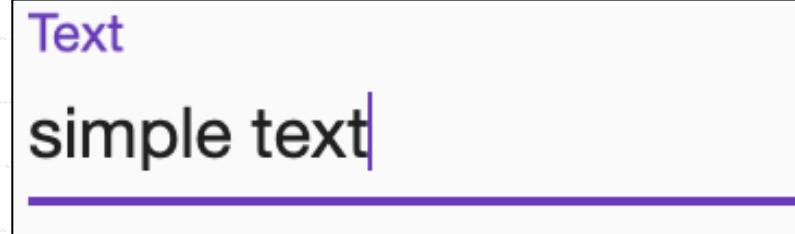
••••••



```
<mat-form-field>
| <input matInput type="text" placeholder="Text">
</mat-form-field>
```

Text

simple text



Select

Korišćenje:

```
import {MatSelectModule} from '@angular/material';  
imports i exports atributi @NgModule dekoratora  
(material.module.ts)
```

Element koji omogućava izbor vrednosti iz skupa opcija

Koriste se isključivo kao deo forme !

AM implementacija HTML select elementa

```
<mat-select placeHolder="placeholder_text">  
<mat-option value="value1" [selected]>Value1</mat-option >  
<mat-option value="value1" [disabled]>Value1</mat-option >
```

...

```
<mat-select>
```

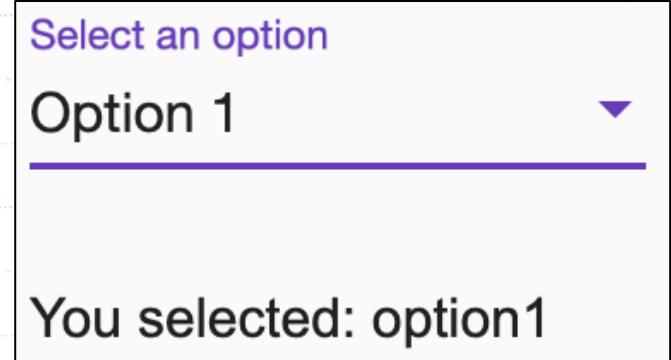
atribut value predstavlja vrednost izabrane opcije (može se koristiti is kontrolera, tj. pozadinskog TS koda komponente)

Opcije se mogu unapred izabrati (select) ili onemogućiti (disabled)

Select

Obostrani (two-way) data binding
Templejt (html) korisničke komponente:

```
<mat-form-field>
  <mat-select [(value)]="selected" placeholder="Select an option">
    <mat-option>None</mat-option>
    <mat-option value="option1">Option 1</mat-option>
    <mat-option value="option2">Option 2</mat-option>
  </mat-select>
</mat-form-field>
<p>You selected: {{selected}}</p>
```



Kontroler korisničke komponente (TS):

```
export class SelectValueBindingExample {
  selected : string = 'option2';
}
```

Select

Dinamičko kreiranje opcija iz kontrolera
Templejt (html) korisničke komponente:

```
<mat-form-field>
  <mat-select [(ngModel)]="selectedCar" placeholder="Select car:>
    <mat-option *ngFor="let car of cars" [value]="car.value">
      {{car.viewValue}}
    </mat-option>
  </mat-select>
</mat-form-field>
<p> Selected car: {{selectedCar}}</p>
```

Select car:

Saab

Selected car: saab

Kontroler korisničke komponente (pozadinski kod):

```
export interface Car {
  value: string;
  viewValue: string;
}
```

```
export class SelectFormExample {
  selectedCar: string;
  cars: Car[] = [
    {value: 'volvo', viewValue: 'Volvo'},
    {value: 'saab', viewValue: 'Saab'},
    {value: 'mercedes', viewValue: 'Mercedes'}];
}
```

CheckBox

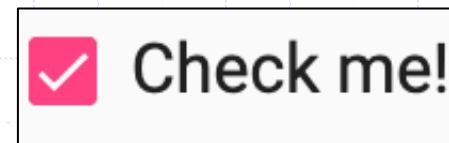
Korišćenje:

import {MatCheckboxModule} from '@angular/material';

imports i exports atributi @NgModul dekoratora
(material.module.ts)

AM alternativa HTML elementu (<input type="checkbox">) sa stilizovanim izgledom i animacijama

```
<mat-checkbox>Check me!</mat-checkbox>
```



Stanja: **checked, indeterminate, disabled**

CheckBox – prve tri kontrole daju primer za obostrani data binding, dok poslednja kontrola predstavlja primer property binding-a



Templejt (html) korisničke komponente:

```
<mat-checkbox [(ngModel)]="checked">Checked</mat-checkbox>
<mat-checkbox [(ngModel)]="indeterminate">Indeterminate</mat-checkbox>
<mat-checkbox [(ngModel)]="disabled">Disabled</mat-checkbox>
<br>
<mat-checkbox
  [checked]="checked"
  [indeterminate]="indeterminate"
  [disabled]="disabled">
  I'm a checkbox
</mat-checkbox>
```

Kontroler korisničke komponente (pozadinski kod):

```
export class CheckboxConfigurableExample {
  checked = false;
  indeterminate = false;
  disabled = false;
}
```

CheckBox – rezultat primera

Checked Indeterminate Disabled

I'm a checkbox

Checked Indeterminate Disabled

I'm a checkbox

Checked Indeterminate Disabled

I'm a checkbox

Forma (form) i polje forme (mat-form-field)

Korišćenje:

```
import {FormsModule} from '@angular/forms';
import {MatFormFieldModule} from '@angular/material';
imports i exports atributa @NgModule dekoratora
(material.module.ts)
```

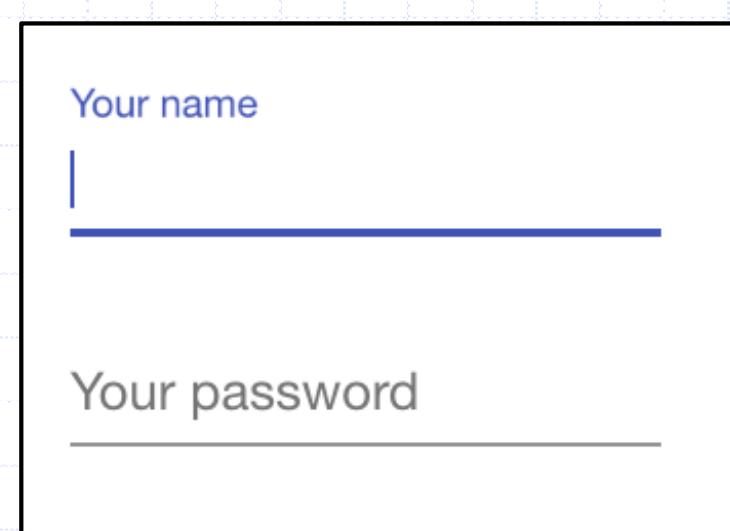
Složena kontrola korisničkog interfejsa za prihvati i obradu podataka korisnika, tzv. *data-entry* zadaci:

1. Prihvati i validacija ulaza korisnika u view delu (html templejt)
2. Kreiranje modela podataka forme (form model) u pozadini (kontroler)
3. Manipulacija modelom i praćenje promena podataka (kontroler)

Forma (form) i polje forme (mat-form-field) - struktura

```
<form fxLayout="column" fxLayoutAlign="center center">
<mat-form-field>
  <input type="email" matInput placeholder="Your name">
</mat-form-field>
<mat-form-field>
  <input type="password" matInput placeholder="Your password">
</mat-form-field>
</form>
```

form je kontejnerski tag u okviru kog se input elementi smeštaju u mat-form-field tag omotač (wrapper)



The image shows a screenshot of a web browser displaying a simple form. The form consists of two input fields. The first input field is a text input with the placeholder "Your name". The second input field is a password input with the placeholder "Your password". Both fields have a blue outline and a blue placeholder text. The entire form is contained within a white rectangular box.

Forma (form) i polje forme (mat-form-field) – pristup podacima forme

```
<form fxLayout="column" fxLayoutGap="10px" fxLayoutAlign="center center"  
#f="ngForm" (ngSubmit)="onSubmit(f)">  
  <mat-form-field>  
    <input type="email" matInput placeholder="Your name"  
          | | | | ngModel name="email">  
  </mat-form-field>  
  <mat-form-field>  
    <input type="password" matInput placeholder="Your password"  
          | | | | ngModel name="password">  
  </mat-form-field>  
  <button type="submit" mat-raised-button color="primary">Submit</button>  
</form>
```

Forma se kao promenjiva tipa objekta forme (`#f = ngForm`) prosleđuje metodi osluškivača na događaj slanja podataka forme (`ngSubmit`)
Pojedinačnim poljima forme se može pristupiti tako što se ona najpre proglase delom objekta forme koji se kreira u pozadini, tzv. model forme (`ngModel`) i zatim dodeli naziv polju (`name` atribut)

Forma (form) i polje forme (mat-form-field) – korišćenje podataka forme

```
onSubmit(form: NgForm) {  
    var email = form.value.email;  
    var password = form.value.password;  
    console.log(form);  
}
```

Poljima modela se pristupa na osnovu naziva, tj. vrednosti name atributa templejta

```
name: "email"  
path: (...)  
pending: (...)  
pristine: (...)  
status: (...)  
statusChanges: (...)  
touched: (...)  
untouched: (...)  
update: EventEmitter {_isScalar: false}  
valid: (...)  
validator: (...)  
value: "mladjan.jovanovic@gmail.com"
```

Forme - validacija

- ◆ Validacija AM formi se vrši u templejtu komponente, tj. pre nego se podaci proslede kontroleru komponente
- ◆ Definiše se lokalna promenjiva templejta (#emailInput) u kojoj se čuva sadržaj polja za unos email adrese
- ◆ Poruke o grešci se prikazuju u <mat-error> tagu pomoći ngIf

```
<mat-form-field>
  <input type="email" matInput placeholder="Your name"
    ngModel name="email" email required #emailInput="ngModel">
    <mat-error *ngIf="emailInput.hasError('required')">
      | Field should not be left empty.
    </mat-error>
    <mat-error *ngIf="!emailInput.hasError('required')">
      | Please provide correct email format.
    </mat-error>
</mat-form-field>
```

Forme - validacija

```
<mat-form-field>
  <input type="email" matInput placeholder="Your name"
    ngModel name="email" email required #emailInput="ngModel">
  <mat-error *ngIf="emailInput.hasError('required')">
    | Field should not be left empty.
  </mat-error>
  <mat-error *ngIf="!emailInput.hasError('required')">
    | Please provide correct email format.
  </mat-error>
</mat-form-field>
```

Your name *

mladjan

Please use the correct email format.

Your password *

.....

Please enter minimum 6 characters.

11 / 6

Submit

Kalendar (DatePicker)

Korišćenje:

```
import {MatDatepickerModule} from '@angular/material';  
imports i exports atributa @NgModule dekoratora  
(material.module.ts)
```

Često se koristi kao deo forme (na primer, registracija korisnika)

Vrednost se može uneti ručno ili izabrati iz kalendarja

Kontrola se sastoji iz tri dela:

1. tekstualnog polja za unos datuma ([input](#))
2. kalendarja ([mat-datepicker](#)) i
3. ikone koja prikazuje kalendar ([mat-datepicker-toggle](#))

Kalendar (DatePicker)

```
<mat-form-field>
  <input matInput placeholder="Your date of birth."
         [matDatepicker]="picker" ngModel name="birthDate" required>
  <mat-datepicker-toggle matSuffix [for]="picker">
  </mat-datepicker-toggle>
  <mat-datepicker #picker></mat-datepicker>
</mat-form-field>
```

Tekstualno polje i kalendar se povezuju pomoći **matDatePicker** property-ja.

Definiše se lokalna promenjiva (**picker**) koja predstavlja kalendar i prosleđuje kao parametar polju za prikaz datuma i ikoni.

Korišćenjem **matSuffix** direktive ikona se prikazuje posle tekstualnog datuma (kao sufiks).

Kalendar (DatePicker)

```
<mat-form-field>
  <input matInput placeholder="Your date of birth."
         [matDatepicker]="picker" ngModel name="birthDate" required>
  <mat-datepicker-toggle matSuffix [for]="picker">
  </mat-datepicker-toggle>
  <mat-datepicker #picker></mat-datepicker>
</mat-form-field>
```

Your name *

mladjan.jovanovic@gmail.com

Your password *

.....

Please enter minimum 6 characters.

8 / 6

Your date of birth. *

10/11/2018



Submit

Sidebar (vertikalna linija alata)

Korišćenje:

import {MatSidenavModule} from '@angular/material';

imports i exports atributa @NgModule dekoratora
(material.module.ts)

Vertikalni element kontejner za zaglavja, nazine ili akcije

Akcije mogu biti predstavljene tekstualno (dugmad ili linkovi) ili kao simboli (ikone)

Može se koristiti i kao *collapsible* kontrola koja se nalazi uz levu ivicu prikaza i otvara/zatvara po potrebi (mobilni interfejsi)

U praksi se kombinuje sa horizontalnom linijom (toolbar) tako da se izgled dinamički prilagođava dimenzijama:

veće dimenziye (tj. širina) => prikazuje se toolbar

manje dimenziye (tj. širina) => prikazuje se sidenav (hamburger meni kod mobilnih telefona)

Opisani mehanizam je poznat pod nazivom *responsive layout*

Sidebar (vertikalna linija alata)

Kontrola se sastoji iz tri dela:

1. koren i tj. kontejnerski element (`mat-sidenav-container`)
2. sidenav deo koji kontroliše prikaz, tj. vidljivost sadržaja (`mat-sidenav`)
3. sadržaj koji se prikazuje (`mat-sidenav-content`)

Često se koristi za definisanje globalne navigacije aplikacije

```
<mat-sidenav-container>
  <mat-sidenav opened="true">
    <p>Sidenav</p>
  </mat-sidenav>
  <mat-sidenav-content>
    <router-outlet></router-outlet>
  </mat-sidenav-content>
</mat-sidenav-container>
```

Toolbar (horizontalna linija alata)

Korišćenje:

```
import {MatToolbarModule} from '@angular/material';
```

imports i exports atributa @NgModule dekoratora
(material.module.ts)

Horizontalni element kontejner za zaglavla, nazine ili akcije
Akcije mogu biti predstavljene tekstualno (dugmad ili linkovi) ili kao simboli (ikone)

Može se koristiti samostalno kao horizontalna linija alatki ili linkova koja se prikazuje u zaglavlu stranice

U praksi se često kombinuje sa horizontalnom linijom za alate (*sidebar*)

Toolbar (horizontalna linija alata)

Koristi se kao kontejner (`mat-toolbar`) u okviru kojeg se dodaju AM ili HTML elementi

Kombinacija za vertikalnom linijom alata je prikazana ispod
Toolbar kao element se smešta u sadržaj vertikalne linije alata
(`mat-sidenav-content`)

```
<mat-sidenav-content>
  <mat-toolbar color="primary">
    <div><button (click)="sidenav.toggle()">Toggle sidenav</button></div>
    <div>Logo</div>
    <div>
      <a routerLink="/signup">Signup</a>
      <a routerLink="/login">Login</a>
      <a routerLink="/training">Training</a>
    </div>
  </mat-toolbar>
  <router-outlet></router-outlet>
</mat-sidenav-content>
```

```
<mat-sidenav-container>
  <mat-sidenav #sidenav opened="true">
    <p>Sidenav</p>
  </mat-sidenav>
  <mat-sidenav-content>
    <mat-toolbar color="primary">
      <div>
        <button (click)="sidenav.toggle()">
          <mat-icon>menu</mat-icon>
        </button>
      </div>
      <div>
        <a routerLink="/">Logo</a>
      </div>
      <div>
        <a routerLink="/signup">Signup</a>
        <a routerLink="/login">Login</a>
        <a routerLink="/training">Training</a>
      </div>
    </mat-toolbar>
  </mat-sidenav-content>
</mat-sidenav-container>
```

List (lista)

Korišćenje:

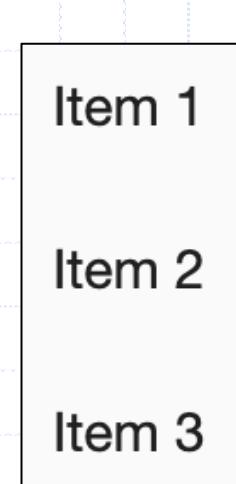
```
import {MatListModule} from '@angular/material';
imports i exports atributa @NgModule dekoratora
(material.module.ts)
```

AM element nabranjanja (analogno HTML list elementu)

Varijante:

1. Osnovna lista

```
<mat-list>
  <mat-list-item>Item 1</mat-list-item>
  <mat-list-item>Item 2</mat-list-item>
  <mat-list-item>Item 3</mat-list-item>
</mat-list>
```



List (lista)

2. Navigaciona lista (*navigation list*)

```
<mat-sidenav #sidenav opened="true">  
  <mat-nav-list>  
    <a mat-list-item routerLink="/signup">Signup</a>  
    <a mat-list-item routerLink="/login">Login</a>  
    <a mat-list-item routerLink="/training">Training</a>  
    <a mat-list-item routerLink="/">Logout</a>  
  </mat-nav-list>  
</mat-sidenav>
```

3. Akciona lista (*action list*)

```
<mat-action-list>  
  <button mat-list-item (click)="save()"> Save </button>  
  <button mat-list-item (click)="undo()"> Undo </button>  
</mat-action-list>
```

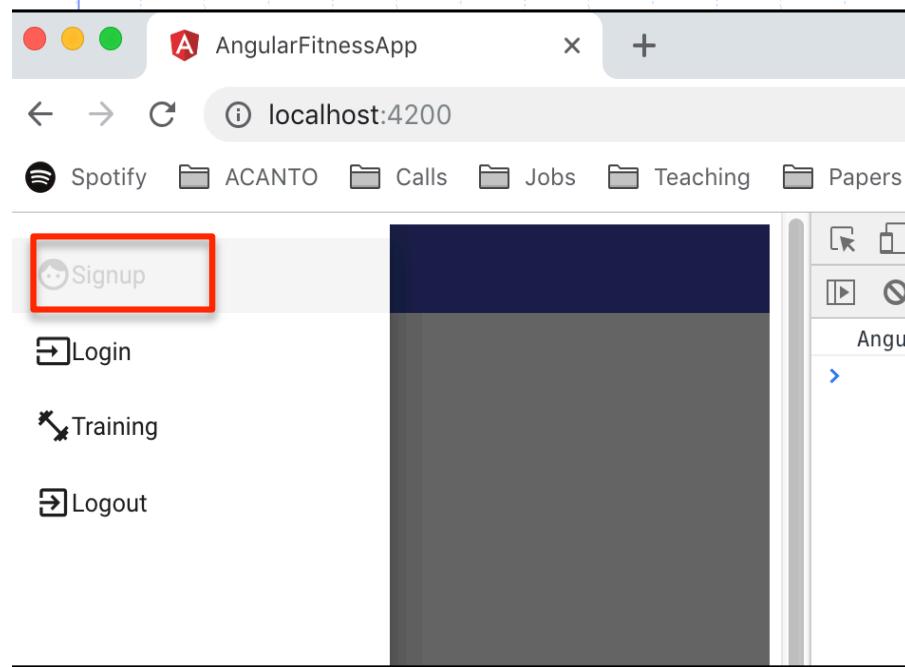
Navigaciona lista kao ikone

```
<mat-nav-list>
  <a mat-list-item routerLink="/signup">
    <mat-icon>face</mat-icon>
    Signup
  </a>
  <a mat-list-item routerLink="/login">
    <mat-icon>input</mat-icon>
    Login
  </a>
  <a mat-list-item routerLink="/training">
    <mat-icon>fitness_center</mat-icon>
    Training
  </a>
  <a mat-list-item routerLink="/">
    <mat-icon>exit_to_app</mat-icon>
    Logout
  </a>
</mat-nav-list>
```

Navigaciona lista kao ikone

```
<mat-nav-list>
    <a mat-list-item routerLink="/signup" (click)="sidenav.close()">
        <mat-icon>face</mat-icon>
        Signup
    </a>
    <a mat-list-item routerLink="/login" (click)="sidenav.close()">
        <mat-icon>input</mat-icon>
        Login
    </a>
    <a mat-list-item routerLink="/training" (click)="sidenav.close()">
        <mat-icon>fitness_center</mat-icon>
        Training
    </a>
    <a mat-list-item routerLink="/" (click)="sidenav.close()">
        <mat-icon>exit_to_app</mat-icon>
        Logout
    </a>
</mat-nav-list>
```

Navigaciona lista



AngularFitnessApp

localhost:4200/signup

Spotify ACANTO Calls Jobs Teaching Papers

Logo

Your name *

Your password *

Please enter minimum 6 characters. 0 / 6

Your date of birth. *

Submit

I accept the terms of conditions of use.

Tab (tabovi)

Korišćenje:

```
import {MatTabsModule} from '@angular/material';
```

imports i exports atributa @NgModule dekoratora
(material.module.ts)

Organizacija sadržaja kao odvojenih pogleda, gde je u datom trenutku vidljiv samo jedan pogled

Aktivan tab se može postaviti korišćenjem atributa `selectedIndex`

Sadržaj mat-tab taga može biti bilo koji element ili komponenta

```
<mat-tab-group>
  <mat-tab label="First"> Content 1 </mat-tab>
  <mat-tab label="Second"> Content 2 </mat-tab>
  <mat-tab label="Third"> Content 3 </mat-tab>
</mat-tab-group>
```

Tab – navigacija i sadržaj komponenti aplikacije

```
< <mat-tab-group>
    <mat-tab label="Current training">
        <app-current-training></app-current-training>
    </mat-tab>
    <mat-tab label="New training">
        <app-new-training></app-new-training>
    </mat-tab>
    <mat-tab label="Past training">
        <app-previous-training></app-previous-training>
    </mat-tab>
</mat-tab-group>
```

Card (kartica)

Korišćenje:

```
import {MatCardModule} from '@angular/material';
```

```
imports i exports atributa @NgModul dekoratora  
(material.module.ts)
```

Organizacija specifičnog tipa sadržaja po sekcijama kao zaglavlje, sadržaj, akcije i podnožje (footer)

<mat-card> je kontejnerski tag koji sadrži elemente:

<mat-card-title> Naslov kartice

<mat-card-content> Primarni sadržaj kartice (tekst, simboli)

<mat-card-actions> Akcije kao dugmad ispod sadržaja

<mat-card-footer> Podnožje kartice (footer) za eventualne dodatne informacije

Card (kartica)

```
<mat-card fxFlex="400px">
  <mat-card-title fxLayoutAlign="center">
    Start training
  </mat-card-title>
  <mat-card-content fxLayoutAlign="center">
    Choose exercise:
  </mat-card-content>
  <mat-card-actions fxLayoutAlign="center">
    <button type="submit" mat-button>Start</button>
  </mat-card-actions>
</mat-card>
```

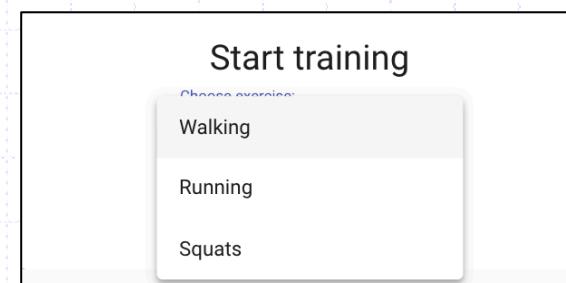
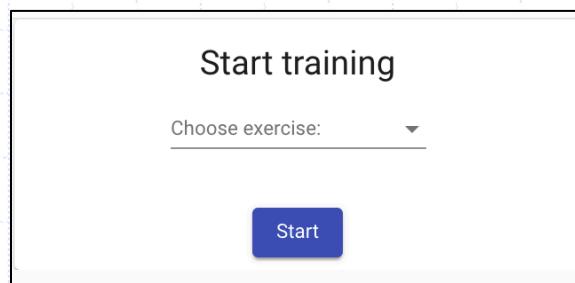
Naslov kartice

Sadržaj kartice

Akcije kartice

Card (kartica) – sadržaj može biti složeniji (select)

```
<mat-card-content fxLayoutAlign="center" fxLayoutGap="20x">
  <mat-form-field>
    <mat-select placeholder="Choose exercise:">
      <mat-option value="walking">
        Walking
      </mat-option>
      <mat-option value="running">
        Running
      </mat-option>
      <mat-option value="squats">
        Squats
      </mat-option>
    </mat-select>
  </mat-form-field>
</mat-card-content>
```



Tabele

Korišćenje:

```
import {MatTableModule} from '@angular/material';
imports i exports atributa @NgModule dekoratora
(material.module.ts)
```

AM funkcionalnost tabele sa dodatnim funkcijama sortiranja, filtriranja i straničenja kao zasebnim modulima

Obrazac za definisanje tabele je sledeći (po redosledu navođenja):

- 1. Glavni tag `<mat-table>`** čiji je glavni atribut *dataSource* (izvor podataka koji se prikazuju)
- 2. Kolone** su definisane u okviru `<ng-container>` taga koji sadrži tagove zaglavlja ćelija kolone (`<mat-header-cell>`) i vrednosti ćelija kolone (`<mat-cell>`)
- 3. Redovi:** `<mat-header-row>` koji definiše prvi red sa zaglavljima kolona i `<mat-row>` koji ispisuje vrednosti ćelija kolona.

Tabele – osnovni slučaj (templejt komponente)

```
<mat-table [dataSource]="dataSource">  
  <ng-container matColumnDef="position">  
    <mat-header-cell *matHeaderCellDef>No.</mat-header-cell>  
    <mat-cell *matCellDef="let element"> {{element.position}} </mat-cell>  
  </ng-container>  
  <ng-container matColumnDef="name">  
    <mat-header-cell *matHeaderCellDef>Name</mat-header-cell>  
    <mat-cell *matCellDef="let element"> {{element.name}} </mat-cell>  
  </ng-container>  
  <ng-container matColumnDef="weight">  
    <mat-header-cell *matHeaderCellDef>Weight</mat-header-cell>  
    <mat-cell *matCellDef="let element"> {{element.weight}} </mat-cell>  
  </ng-container>  
  <ng-container matColumnDef="symbol">  
    <mat-header-cell *matHeaderCellDef> Symbol </mat-header-cell>  
    <mat-cell *matCellDef="let element"> {{element.symbol}}</mat-cell>  
  </ng-container>  
  <mat-header-row *matHeaderRowDef="displayedColumns"></mat-header-row>  
  <mat-row *matRowDef="let row; columns: displayedColumns;"></mat-row>  
</mat-table>
```

Tabele – osnovni slučaj (kontroler komponente)

```
export interface PeriodicElement {  
    name: string;  
    position: number;  
    weight: number;  
    symbol: string;  
}  
  
const ELEMENT_DATA: PeriodicElement[] = [  
    {position: 1, name: 'Hydrogen', weight: 1.0079, symbol: 'H'},  
    {position: 2, name: 'Helium', weight: 4.0026, symbol: 'He'},  
    {position: 3, name: 'Lithium', weight: 6.941, symbol: 'Li'},  
    {position: 4, name: 'Beryllium', weight: 9.0122, symbol: 'Be'},  
    {position: 5, name: 'Boron', weight: 10.811, symbol: 'B'},  
    {position: 6, name: 'Carbon', weight: 12.0107, symbol: 'C'},
```

```
export class TableBasicExample {  
    displayedColumns: string[] = ['position', 'name', 'weight', 'symbol'];  
    dataSource = ELEMENT_DATA;  
}
```

Tabele – osnovni slučaj (izgled tabele)

| No. | Name | Weight | Symbol |
|-----|-----------|---------|--------|
| 1 | Hydrogen | 1.0079 | H |
| 2 | Helium | 4.0026 | He |
| 3 | Lithium | 6.941 | Li |
| 4 | Beryllium | 9.0122 | Be |
| 5 | Boron | 10.811 | B |
| 6 | Carbon | 12.0107 | C |
| 7 | Nitrogen | 14.0067 | N |

Tabele – složeniji slučaj (templejt komponente)

```
<mat-table [dataSource]="exerciseSource">
  <ng-container matColumnDef="date">
    <mat-header-cell *matHeaderCellDef>Date</mat-header-cell>
    <mat-cell *matCellDef="let element">{{ element.date | date }}</mat-cell>
  </ng-container>
  <ng-container matColumnDef="name">
    <mat-header-cell *matHeaderCellDef>Name</mat-header-cell>
    <mat-cell *matCellDef="let element">{{ element.name }}</mat-cell>
  </ng-container>
  <ng-container matColumnDef="duration">
    <mat-header-cell *matHeaderCellDef>Duration</mat-header-cell>
    <mat-cell *matCellDef="let element">{{ element.duration | number }}</mat-cell>
  </ng-container>
  <ng-container matColumnDef="calories">
    <mat-header-cell *matHeaderCellDef>Calories</mat-header-cell>
    <mat-cell *matCellDef="let element">{{ element.calories | number }}</mat-cell>
  </ng-container>
  <ng-container matColumnDef="status">
    <mat-header-cell *matHeaderCellDef>Status</mat-header-cell>
    <mat-cell *matCellDef="let element">{{ element.status }}</mat-cell>
  </ng-container>
  <mat-header-row *matHeaderRowDef="displayedColumns"></mat-header-row>
  <mat-row *matRowDef="let rows; columns: displayedColumns;">[]</mat-row>
</mat-table>
```

Tabele – složeniji slučaj (kontroler komponente)

```
import { Component, OnInit } from '@angular/core';
import { MatTableDataSource } from '@angular/material';
import { Exercise } from '../exercise.model'
import { TrainingService } from '../training.service';

@Component({
  selector: 'app-previous-training',
  templateUrl: './previous-training.component.html',
  styleUrls: ['./previous-training.component.css']
})
export class PreviousTrainingComponent implements OnInit {

  displayedColumns = ["date", "name", "duration", "calories", "status"];
  exerciseSource = new MatTableDataSource<Exercise>();

  constructor(private trainingService : TrainingService) { }

  ngOnInit() {
    this.exerciseSource.data = this.trainingService.getExercises();
  }
}
```

Tabele – složeniji slučaj (servis i struktura podataka)

```
export interface Exercise {  
    id : string;  
    name : string;  
    duration : number;  
    calories : number;  
    date? : Date;  
    status? : 'done' | 'cancelled' | null;  
}
```

```
import { Exercise } from "./exercise.model";  
  
export class TrainingService {  
    private exercises : Exercise [] = [  
        {id: 'running', name: 'Morning running', duration: 15, calories: 100},  
        {id: 'squats', name: 'Midday squats', duration: 5, calories: 70, date: new Date('2023-09-15T12:00:00Z')},  
        {id: 'walking', name: 'Afternoon walking', duration: 20, calories: 120},  
        {id: 'running', name: 'Evening running', duration: 15, calories: 100}  
    ]  
    getExercises() {  
        return this.exercises;  
    }  
}
```

Tabele – složeniji slučaj (izgled tabele)

| New training | Past training | |
|--------------|-------------------|----------|
| Date | Name | Duration |
| Dec 2, 2018 | Morning running | 15 |
| Dec 2, 2018 | Midday squats | 5 |
| Dec 2, 2018 | Afternoon walking | 20 |
| Dec 2, 2018 | Evening running | 15 |

Sortiranje tabele

Korišćenje:

```
import {MatSortModule} from '@angular/material';
```

```
imports i exports atributa @NgModule dekoratora (material.module.ts)
```

Dodavanje sortiranja kao funkcije se svodi na ubacivanje specifičnih direktiva u postojeću tabelu:

matSort direktiva u koreni element tabele (**mat-table**) – govori da se tabela kao celina može sortirati

mat-sort-header direktiva u zaglavljue kolone tabele (**mat-header-cell**)
– govori da se specifična kolona može sortirati

```
<ng-container matColumnDef="position">
  <th mat-header-cell *matHeaderCellDef mat-sort-header> Name </th>
  <td mat-cell *matCellDef="let element"> {{element.position}} </td>
</ng-container>
```

Sortiranje tabele – templejt komponente

```
<mat-table [dataSource]="exerciseSource" matSort>
  <ng-container matColumnDef="date">
    <mat-header-cell *matHeaderCellDef mat-sort-header>Date</mat-header-cell>
    <mat-cell *matCellDef="let element">{{ element.date | date }}</mat-cell>
  </ng-container>
  <ng-container matColumnDef="name">
    <mat-header-cell *matHeaderCellDef mat-sort-header>Name</mat-header-cell>
    <mat-cell *matCellDef="let element">{{ element.name }}</mat-cell>
  </ng-container>
  <ng-container matColumnDef="duration">
    <mat-header-cell *matHeaderCellDef mat-sort-header>Duration</mat-header-cell>
    <mat-cell *matCellDef="let element">{{ element.duration | number }}</mat-cell>
  </ng-container>
  <ng-container matColumnDef="calories">
    <mat-header-cell *matHeaderCellDef mat-sort-header>Calories</mat-header-cell>
    <mat-cell *matCellDef="let element">{{ element.calories | number }}</mat-cell>
  </ng-container>
  <ng-container matColumnDef="status">
    <mat-header-cell *matHeaderCellDef mat-sort-header>Status</mat-header-cell>
    <mat-cell *matCellDef="let element">{{ element.status }}</mat-cell>
  </ng-container>
  <mat-header-row *matHeaderRowDef="displayedColumns"></mat-header-row>
  <mat-row *matRowDef="let rows; columns: displayedColumns;"></mat-row>
</mat-table>
```

Sortiranje tabele – pozadinski kod

- ◆ Sortiranje predstavlja vrstu funkcije
- ◆ Da bi se pravilno implementirala potrebno je:
 1. Definisati objekat u pozadinskom kodu koji referencira elemente templejta, tj. podatke tabele
`sort: MatSort`
 2. Definisati samu funkciju sortiranja nad referenciranim podacima
`matTableDataSource.sort = sort`
 3. Potrebno je još obezbediti da se tabela može sortirati tek kada se podaci zaista učitaju i prikažu
implementacija interfejsa `AfterViewInit`

Sortiranje tabele – pozadinski kod

```
import { Component, OnInit, ViewChild, AfterViewInit } from '@angular/core';
import { MatTableDataSource, MatSort } from '@angular/material';
import { Exercise } from '../exercise.model'
import { TrainingService } from '../training.service';

@Component({
  selector: 'app-previous-training',
  templateUrl: './previous-training.component.html',
  styleUrls: ['./previous-training.component.css']
})
export class PreviousTrainingComponent implements OnInit, AfterViewInit {

  displayedColumns = ["date", "name", "duration", "calories", "status"];
  exerciseSource = new MatTableDataSource<Exercise>();

  @ViewChild(MatSort) sort : MatSort;

  constructor(private trainingService : TrainingService) { }

  ngOnInit() {
    this.exerciseSource.data = this.trainingService.getExercises();
  }

  ngAfterViewInit() {
    this.exerciseSource.sort = this.sort;
  }
}
```

Sortiranje tabele - izgled

Past training

| Name | Duration | Calories ↓ |
|-------------------|----------|------------|
| Morning running | 15 | 100 |
| Evening running | 15 | 100 |
| Afternoon walking | 20 | 80 |
| Midday squats | 5 | 70 |

Straničenje tabele

Korišćenje:

```
import {MatPaginatorModule} from '@angular/material';  
imports i exports atributa @NgModule dekoratora  
(material.module.ts)
```

Dodavanje straničenja kao funkcije se svodi na ubacivanje specifičnih direktiva u postojeću tabelu:

<mat-paginator> tag se dodaje u nastavku tabele sa atributima [pageSize] za veličinu stranice kao broj redova i [pageSizeOptions] kao opcije za promenu veličine stranice

Dodavanje elemenata u pozadinsku logiku (slično sortiranju)

MatPaginator

Straničenje tabele – templejt komponente

```
<mat-table [dataSource]="exerciseSource" matSort>
  <ng-container matColumnDef="date">
    <mat-header-cell *matHeaderCellDef mat-sort-header>Date</mat-header-cell>
    <mat-cell *matCellDef="let element">{{ element.date | date }}</mat-cell>
  </ng-container>
  <ng-container matColumnDef="name">
    <mat-header-cell *matHeaderCellDef mat-sort-header>Name</mat-header-cell>
    <mat-cell *matCellDef="let element">{{ element.name }}</mat-cell>
  </ng-container>
  <ng-container matColumnDef="duration">
    <mat-header-cell *matHeaderCellDef mat-sort-header>Duration</mat-header-cell>
    <mat-cell *matCellDef="let element">{{ element.duration | number }}</mat-cell>
  </ng-container>
  <ng-container matColumnDef="calories">
    <mat-header-cell *matHeaderCellDef mat-sort-header>Calories</mat-header-cell>
    <mat-cell *matCellDef="let element">{{ element.calories | number }}</mat-cell>
  </ng-container>
  <ng-container matColumnDef="status">
    <mat-header-cell *matHeaderCellDef mat-sort-header>Status</mat-header-cell>
    <mat-cell *matCellDef="let element">{{ element.status }}</mat-cell>
  </ng-container>
  <mat-header-row *matHeaderRowDef="displayedColumns"></mat-header-row>
  <mat-row *matRowDef="let rows; columns: displayedColumns;"></mat-row>
</mat-table>
<mat-paginator [pageSize]="1" [pageSizeOptions]="[2,4,6,8,10]">/mat-paginator>
```

Straničenje tabele – pozadinski kod

- ◆ Straničenje predstavlja vrstu funkcije
- ◆ Da bi se pravilno implementiralo potrebno je:
 1. Definisati objekat u pozadinskom kodu koji referencira elemente templejta, tj. podatke tabele paginator: `MatPaginator`
 2. Definisati samu funkciju straničenja nad referenciranim podacima
`matTableDataSource.paginator = paginator`
 3. Potrebno je još obezrediti da se tabela može straničiti tek kada se podaci zaista učitaju i prikažu implementacija interfejsa `AfterViewInit`

Straničenje tabele – pozadinski kod

```
import { Component, OnInit, ViewChild, AfterViewInit } from '@angular/core';
import { MatTableDataSource, MatSort, MatPaginator } from '@angular/material';
import { Exercise } from '../exercise.model'
import { TrainingService } from '../training.service';

@Component({
  selector: 'app-previous-training',
  templateUrl: './previous-training.component.html',
  styleUrls: ['./previous-training.component.css']
})
export class PreviousTrainingComponent implements OnInit, AfterViewInit {

  displayedColumns = ["date", "name", "duration", "calories", "status"];
  exerciseSource = new MatTableDataSource<Exercise>();

  @ViewChild(MatSort) sort : MatSort;
  @ViewChild(MatPaginator) paginator : MatPaginator;

  constructor(private trainingService : TrainingService) { }

  ngOnInit() {
    this.exerciseSource.data = this.trainingService.getExercises();
  }

  ngAfterViewInit() {
    this.exerciseSource.sort = this.sort;
    this.exerciseSource.paginator = this.paginator;
  }
}
```

Straničenje tabele - izgled

Duration

15

Calories

100

Status

done

5

70

done

Items per page:

2 ▾

1 - 2 of 4



Filtriranje tabele

- ◆ Filter ne postoji kao zasebna AM komponenta već se dodaje kao *input* polje forme (*mat-form-field*)
- ◆ Obično se dodaje pre, tj. iznad tabele templejt komponente:

```
<mat-form-field>
  <input matInput type="text" (keyup)="doFilter($event.target.value)" placeholder="Filter">
</mat-form-field>
```

pozadinski kod:

```
doFilter(filterValue : string) {
  this.exerciseSource.filter = filterValue.trim().toLowerCase();
}
```

AM datasource filter podrazumevano vrši konkatenaciju (spajanje) svih vrednosti reda tabele kao stringova, i potom traži vrednost filtera u rezultujućem stringu.

Filtriranje tablele

The screenshot shows a mobile application interface for managing training sessions. At the top, there is a blue header bar with the word "Logo" on the left and "Signup Login Training" on the right. Below the header, there are two tabs: "New training" and "Past training", with "Past training" being the active tab, indicated by a blue underline. A "Filter" input field is positioned below the tabs. The main content area displays a table with columns: Date, Name, Duration, Calories, and Status. One row is visible, showing a session from Dec 9, 2018, named "Morning running 15" with a duration of 100 and status "done". At the bottom, there is a pagination control with "Items per page: 1" and arrows for navigating through the results. The overall design is clean and modern.

| Date | Name | Duration | Calories | Status |
|-------------|--------------------|----------|----------|--------|
| Dec 9, 2018 | Morning running 15 | 100 | | done |

Items per page: 1 ▾ 1 - 1 of 4 < >

Dijalog (dialog)

Korišćenje:

```
import {MatDialogModule} from '@angular/material';  
imports i exports atributa @NgModule dekoratora  
(material.module.ts)
```

Kreiranje AM stilizovanih modalnih dijaloga

Nekoliko direktiva stoji na raspolaganju za organizaciju sadržaja dijaloga:

mat-dialog-title atribut za naslov dijaloga, u heading elementu (`<h1>`, `<h2>`)

<mat-dialog-content> Glavni sadržaj dijaloga

<mat-dialog-actions> Kontejner za akcije, tj. dugmad

mat-dialog-close atribut koji se dodaje `<button>` elementu, čini da dugme zatvara dijalog sa opcionom vrednošću koja se može koristiti

Dijalog (dialog) – templejt

```
<section>
  <h1 mat-dialog-title>Stop training</h1>
  <mat-dialog-content>Are you sure?</mat-dialog-content>
  <mat-dialog-actions>
    <button mat-button [mat-dialog-close]="true">Yes</button>
    <button mat-button [mat-dialog-close]="false">No</button>
  </mat-dialog-actions>
</section>
```

Dijalog- pozadinski kod

- ◆ Dijalog se otvara i zatvara iz kontrolera
- ◆ Da bi se pravilno implementiralo potrebno je:
 1. Definisati servis u pozadinskom kodu koji upravlja dijalogom (otvaranje/zatvaranje)
`dialog: MatDialog`
 2. Kreirati servis kao parametar konstruktora
 3. Otvaranje i zatvaranje dijaloga pomoću servisa
`dialogRef = this.dialog.open()`
`dialogRef.afterClosed()`

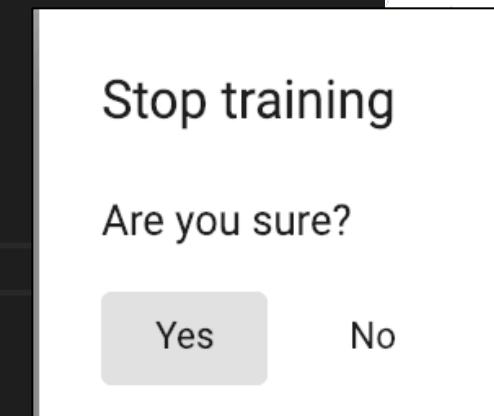
Registracija MatDialog servisa

- ◆ U cilju optimizacije performansi aplikacije, Angular izvršno okruženje zahteva od programera da komponente koje će se generisati dinamički (MatDialog) eksplicitno navede u vreme pisanja koda programa.
- ◆ Konkretno, komponente koje će se koristiti na ovaj način se navode kao element *entryComponents* niza *@Component* dekoratora korenog modula aplikacije (app.module.ts) kao ispod:

```
imports: [
  BrowserModule,
  MaterialModule,
  RoutingModule,
  BrowserAnimationsModule,
  FlexLayoutModule,
  FormsModule],
providers: [TrainingService],
bootstrap: [AppComponent],
entryComponents: [StopTrainingComponent])
```

Otvaranje dijaloga iz pozadinskog koda pozivajuće komponente

```
export class CurrentTrainingComponent implements OnInit {  
    progress = 0;  
    timer : any;  
    constructor(private dialog : MatDialog) { }  
    ngOnInit() {  
        this.timer = setInterval(() => {  
            this.progress += 5;  
            if (this.progress >= 100) {  
                clearInterval(this.timer);  
            }  
        }, 1000);  
    }  
    onStop() {  
        clearInterval(this.timer);  
        this.dialog.open(StopTrainingComponent);  
    }  
    onResume(){  
        this.timer = setInterval(() => {  
            this.progress += 5;  
            if (this.progress >= 100) {  
                clearInterval(this.timer);  
            }  
        }, 1000);  
    }  
}
```



Prosleđivanje podataka ka dijalogu

- ◆ Prosleđivanje podataka dijalogu se vrši prilikom otvaranja, tj. u pozivu metode *open*
- ◆ *Open* metoda može imati i drugi parametar tipa *any*
- ◆ Podaci koje prosleđujemo se kreiraju kao objekti programa predstavljeni u formatu *ime:vrednost*

```
onStop() {  
    clearInterval(this.timer);  
    this.dialog.open(StopTrainingComponent, {  
        data: {  
            progress: this.progress  
        }  
    });  
}
```

Prihvatanje prosleđenih podataka u dijalogu

- ◆ `MAT_DIALOG_DATA` je predefinisana simbolička konstanta koja nam omogućuje pristup podacima koje AM interno (u formi objekata u pozadini) čuva kada pozovemo *open* metodu (otvaranje dijaloga)
- ◆ `@Inject` direktiva se koristi za "ubacivanje" podataka referenciranih sa `MAT_DIALOG_DATA` u promenjivu koju mi definišemo (`receivedData`). Promenjivu definišemo da bismo ispisali preuzete podatke u templejtu

Prihvati prosleđenih podataka

```
import { Component, OnInit, Inject } from '@angular/core';
import { MAT_DIALOG_DATA} from "@angular/material";

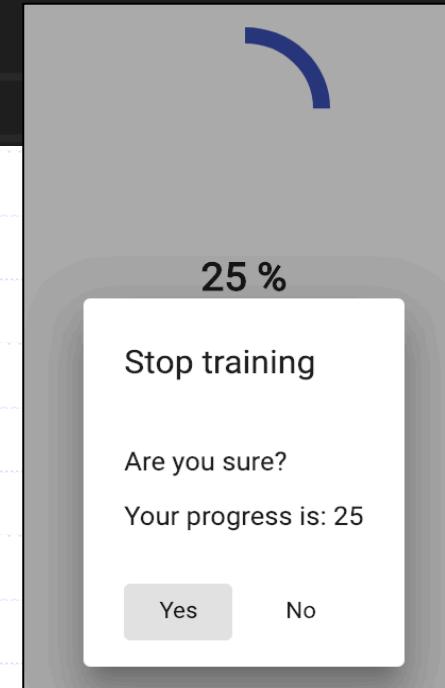
@Component({
  selector: 'app-stop-training',
  templateUrl: './stop-training.component.html',
  styleUrls: ['./stop-training.component.css']
})
export class StopTrainingComponent implements OnInit {

  constructor(@Inject(MAT_DIALOG_DATA) public receivedData: any) { }

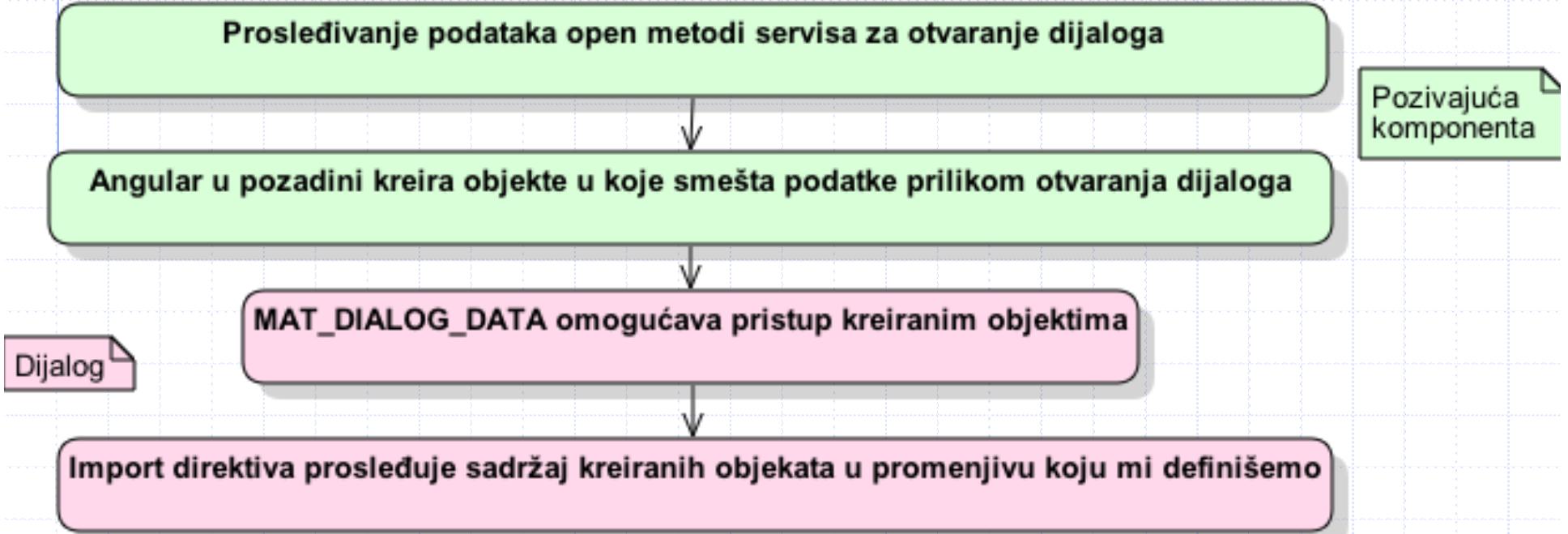
  ngOnInit() {
  }
}
```

Prikaz prosleđenih podataka

```
<section>
  <h1 mat-dialog-title>Stop training</h1>
  <mat-dialog-content>
    <p>Are you sure?</p>
    <p>Your progress is: {{ receivedData.progress }}</p>
  </mat-dialog-content>
  <mat-dialog-actions>
    <button mat-button [mat-dialog-close]="true">Yes</button>
    <button mat-button [mat-dialog-close]="false">No</button>
  </mat-dialog-actions>
</section>
```



Šema prosleđivanja podataka ka dijalogu



Sažetak

- ◆ Input
- ◆ Select
- ◆ Checkbox
- ◆ Forma (form)
 - Pristup podacima, korišćenje podataka, validacija
- ◆ Kalendar (DatePicker)
- ◆ Sidebar
- ◆ Toolbar
- ◆ Lista (list)
- ◆ Tab
- ◆ Kartica (card)
- ◆ Tabela
 - Prikaz, straničenje, sortiranje, filtriranje
- ◆ Dijalog (dialog)
 - Prosleđivanje, prihvatanje, prikaz podataka

Materijali

◆ Angular Material primer.pdf

- Korak-po-korak kreiranje horizontalnog prototipa aplikacije
- Horizontalni prototip = frontend

◆ angular-material-primer.zip

- Implementirani projekat

Linkovi ka dokumentaciji i primerima

- ◆ Input komponenta primeri:

<https://material.angular.io/components/input/examples>

- ◆ Select – obostrani data binding:

<https://stackblitz.com/angular/aeeyokplndq?file=app%2Fselect-value-binding-example.ts>

- ◆ Select – dinamičko kreiranje opcija:

<https://stackblitz.com/angular/mnamdxoxvay?file=app%2Fselect-overview-example.ts>

- ◆ Checkbox binding primeri:

<https://stackblitz.com/angular/kganmelael?file=app%2Fcheckbox-configurable-example.html>

- ◆ Tabela:

<https://material.angular.io/components/table/overview>

Angular Material komponente

Dr. Mlađan Jovanović
mjovanovic@singidunum.ac.rs

TypeScript Flexbox Flexlayout

Dr. Mlađan Jovanović
mjovanovic@singidunum.ac.rs

Sadržaj

- ◆ **TypeScript:**
 - ◆ Instalacija i korišćenje
 - ◆ Tipovi podataka
 - ◆ Deklaracija i definicija varijabli i funkcija
 - ◆ Interfejsi
 - ◆ Klase
 - ◆ Prevođenje u JS (primeri)
- ◆ **Flexbox:**
 - ◆ Direktive za definisanje rasporeda u CSS fajlu
- ◆ **Flexlayout:**
 - ◆ Direktive za definisanje rasporeda u templejtu (HTML)

Uvod u TypeScript

- ◆ Nadskup JavaScript jezika
 - JS kod predstavlja validan TS kod
- ◆ TS dodaje elemente JS jeziku radi jednostavnije upotrebe
 - Objektno-orientisani (OO) elementi
- ◆ TS kompajler prevodi TS kod u JS kod koji se zatim izvršava
- ◆ Instalacija TS kompajlera:
- ◆ Najpre je potrebno instalirati Node.js kao JS kompajler i izvršno okruženje
- ◆ Lokalna
 - Za svaki Angular projekat kreiran u Visual Studio Code razvojnom okruženju (**ng new**) automatski se dodaje podrška za TS
- ◆ Globalna
 - `npm install -g typescript`
 - `tsc --version`

Lokalna TS instalacija u Angular projektu

tsconfig.json

```
{  
  "compileOnSave": false,  
  "compilerOptions": {  
    "baseUrl": "./",  
    "outDir": "./dist/out-tsc",  
    "sourceMap": true,  
    "declaration": false,  
    "module": "es2015",  
    "moduleResolution": "node",  
    "emitDecoratorMetadata": true,  
    "experimentalDecorators": true,  
    "importHelpers": true,  
    "target": "es5",  
    "typeRoots": [  
      "node_modules/@types"  
    ],  
    "lib": [  
      "es2018",  
      "dom"  
    ]  
  }  
}
```

Build Angular aplikacije

ng serve – lokalno pokretanje Angular aplikacije

ng build --prod – kreiranje produkcionе verzije aplikacije

kreira **dist** direktorijum (distribution) sa izvšnom verzijom
koja se može pokrenuti pomoću Web servera (Apache)

prod opcija kreira izvršni kod što je moguće manje veličine

package.json

```
"scripts": {  
  "ng": "ng",  
  "start": "ng serve",  
  "build": "ng build",  
  "test": "ng test",  
  "lint": "ng lint",  
  "e2e": "ng e2e"  
},
```

TS tipovi podataka

number

string

boolean

null

undefined

object

function

any

Primitivni tipovi:

“osnovni tipovi” na osnovu kojih se mogu izvesti složeniji tipovi

Složeni tipovi: izvedeni iz osnovnih tipova

any:

Specijalan tip koji može predstavljati bilo koju vrednost

TypeScript tipovi podataka 1/2

- ◆ Osnovni tipovi:

- ◆ **Boolean**

```
let isDone: boolean = false;
```

- ◆ **Number** (floating point broj)

```
let decimal: number = 6;
```

- ◆ **String** (upotreba i korišćenje identični kao u JS)

```
let fullName: string ='Mladjan Jovanovic';
```

```
let sentence: string ='Hello, my name is ${ fullName }';
```

- ◆ **Nizovi** (upotreba i korišćenje identični kao u JS):

- ◆ Primitivan tip

```
let list: number[] = [1, 2, 3];
```

- ◆ Generički tip

```
let list: Array<number> = [1, 2, 3];
```

TypeScript tipovi podataka 2/2

- ◆ **Enum** element nabranja

```
enum Color {Red, Green, Blue}
```

```
let c: Color = Color.Green;
```

- ◆ **Any** (korisno ukoliko nismo unapred sigurni u tip vrednosti promenjive u vreme pisanja programa)

```
let notSure: any = 4;
```

```
notSure = "maybe a string instead";
```

```
notSure = false;
```

TS je strogo tipiziran jezik, ukoliko se tip ne navede on se prvi put implicitno izvodi iz tipa vrednosti koja se dodeljuje, ali se ne može menjati !

```
let name = "Hello";
```



```
let name: string = "Hello";
```

```
let someValue = 1;
```



```
let someValue: string = 1;
```

TS deklaracija varijabli i funkcija

```
let variableName: TypeName;
```

```
const CONSTANT_NAME: TypeName;
```

```
function functionName(param1: Type1, param2: Type2):  
ReturnType {}
```

```
let age: number = 20;
```

```
const ALPHABET: string =  
"ABCDEFGHIJKLMNOPQRSTUVWXYZ";
```

```
function divides(divisor: number, dividend: number):  
boolean { }
```

object tip je identičan kao u JS

```
let dog = {  
  type: 'mammal',  
  name: 'dog',  
  sounds: ['woof', 'bark', 'yip', 'ruff'] };
```

```
let enigma = {  
  rotors: [],  
  lamps: [],  
  keys: [] };
```

```
let profile = {  
  name: "Mladjan Jovanovic",  
  imageUrl: "http://image.url/img.png",  
  language: "English" };
```

Interfejsi (interface)

- ◆ TS interfejsi definišu **strukturu** objekata
- ◆ Interfejsi nisu objekti, tj. object tip
- ◆ Interfejsi nemaju funkcionalnosti već opisuju isključivo strukturu
- ◆ Korišćenje – definisanje složenijih struktura aplikacije koje se sastoje od atributa (property)

```
interface InterfaceName {  
    property1: Type1  
    property2: Type2  
}
```

- ◆ Opcioni atributi se navode korišćenjem ? na kraju naziva atributa

```
interface Point {  
    x?: number  
    y?: number  
}
```

Nasleđivanje interfejsa

- ◆ Interfejsi se mogu nasleđivati (**extend**)

```
interface Shape {  
    color: string;  
}  
  
interface Square extends Shape {  
    sideLength: number;  
}  
  
let square = <Square>{};  
square.color = "blue";  
square.sideLength = 10;
```

Interfejs – nasleđivanje i instanciranje

```
interface SquareConfiguration {
    width: number
    height: number
    color: string
}

interface Square extends SquareConfiguration {
    area: number
    perimeter: number
}

function createSquare(config: SquareConfiguration): Square {
    return {
        width: config.width,
        height: config.height,
        color: config.color,
        area: config.width * config.height,
        perimeter: config.width * 2 + config.height * 2
    }
}

let square = createSquare({width: 2, height: 2, color: "Green"});
console.log("The new square's color is: " + square.color);
```

Klase (class)

- ◆ Predstavljaju složen tip, identično kao u OO programiranju
- ◆ Mogu se posmatrati i kao interfejsi sa funkcijama
- ◆ Korišćenje identično kao u OO programiranju:
- ◆ Nasleđivanje (**extends**)
 - Izvedene klase moraju pozivati konstruktor nadređene klase (**super**)
- ◆ Kreiranje objekta klase (**new**)
- ◆ Pristup atributima klase (**this**)
 - This unutar tela klase ukazuje na tekući objekat
 - readonly – atribut čija se vrednost ne može menjati (navodi se konstruktoru)
- ◆ Modifikatori vidljivosti
 - public – podrazumevano za sve metode i attribute
 - private – privatan atribut (nije vidljiv van objekta klase)
 - protected – private + vidljivi u izvedenim klasama
- ◆ Accessor metode (**get/set**)
 - Novije verzije TS omogućavaju pristup atributima bez `_` simbola !
- ◆ Statički atributi (**static**)
 - Zajednički za sve instance klase

Klase – kreiranje i instanciranje

```
class Profile {  
    readonly name: string;  
    private imageUrl: string;  
    statuses: string[];  
    constructor(initialName: string, initialImage: string) {  
        this.name = initialName;  
        this.imageUrl = initialImage;  
        this.statuses = [];  
    }  
    getImage(): string {  
        return this.imageUrl;  
    }  
    setImage(newImage: string): void {  
        this.imageUrl = newImage;  
    }  
    addStatus(newStatus: string): void {  
        this.statuses.push(newStatus);  
    }  
}  
let mladjan = new Profile("Mladjan Jovanovic", "https://image.url/");
```

Vidljivost TS klase i interfejsa

- ◆ Prilikom definisanja klase ili interfejsa može se korisiti ključna reč **export**
- ◆ Značenje ključne reči je isključivo da će klasa ili interfejs biti vidljivi u čitavoj Angular aplikaciji
- ◆ Korišćenje ključne reči nije obavezno prilikom definisanja klase ili interfejsa !

TS funkcije

- ◆ Imaju identično značenje i korišćenje kao u JS

- ◆ Imenovana funkcija:

```
function add(x: number, y: number): number {  
    return x + y; }
```

- ◆ Anonimna funkcija:

```
let myAdd = function(x: number, y: number): number  
{ return x + y; };
```

- ◆ Void funkcija – funkcija koja ne vraća vrednost:

```
function warnUser(): void {  
    console.log("This is my warning message"); }
```

TS funkcije

- ◆ Opcioni parametri:

```
function buildName(firstName: string, lastName?: string) :  
string {  
    if (lastName)  
        return firstName + " " + lastName;  
    else  
        return firstName;  
}
```

- ◆ Podrazumevani parametri:

```
function buildName(firstName: string, lastName =  
"Smith") : string {  
    return firstName + " " + lastName;  
}
```

TS funkcije - primer

```
function getMonth(month: number): string {  
    switch (month) {  
        case 1: return "January";  
        case 2: return "February";  
        case 3: return "March";  
        case 4: return "April";  
        case 5: return "May";  
        case 6: return "June";  
        case 7: return "July";  
        case 8: return "August";  
        case 9: return "September";  
        case 10: return "October";  
        case 11: return "November";  
        case 12: return "December";  
        default: return "Not a month!";  
    }  
}  
  
function logMonth(month: string): void {  
    console.log(getMonth(parseInt(month)));  
}
```

TS iteratori (iterators)

◆ **for ... in**

- Iterator koji vraća **ključeve** objekta nad kojim iterira

◆ **for ... of**

- Iterator koji vraća **vrednosti** objekta nad kojim iterira

```
let list = [4, 5, 6]
for (let i in list) {
    console.log(i); // '0','1','2'
}
for (let i of list) {
    console.log(i); // '4','5','6'
}
```

TS nabrojivi tip (enum)

- ◆ Omogućava definisanje skupa imenovanih konstanti
 - Analogno Java enum

```
const enum Direction { Up, Down, Left, Right, }
```

U primeru iznad Up dobija vrednost 0, Down vrednost 1, itd.

Nabrojivi tip radi po principu auto-inkrementa

```
let directions = [Directions.Up,  
                 Directions.Down,  
                 Directions.Left,  
                 Directions.Right]
```

Ručno prevođenje TS

- ◆ U Visual Studio Code razvojnom okruženju otvoriti neki od primera (na primer, Interface.ts)
- ◆ U Terminal prozoru se pozicionirati u direktorijum primera i kompajlirati TS klasu naredbom
tsc Interface.ts
- ◆ Rezultat prevođenja će biti JS fajl sa istim nazivom (Interface.js)
- ◆ JS fajl se može izvršiti iz Terminala naredbom
node Interface.js
- ◆ Uporediti izgled TS i JS fajla !
 - Za klasu (Class.ts), interfejs (Interface.ts) i funkciju (Function.ts)
 - **klasa (class) -> prototip (Prototype)**

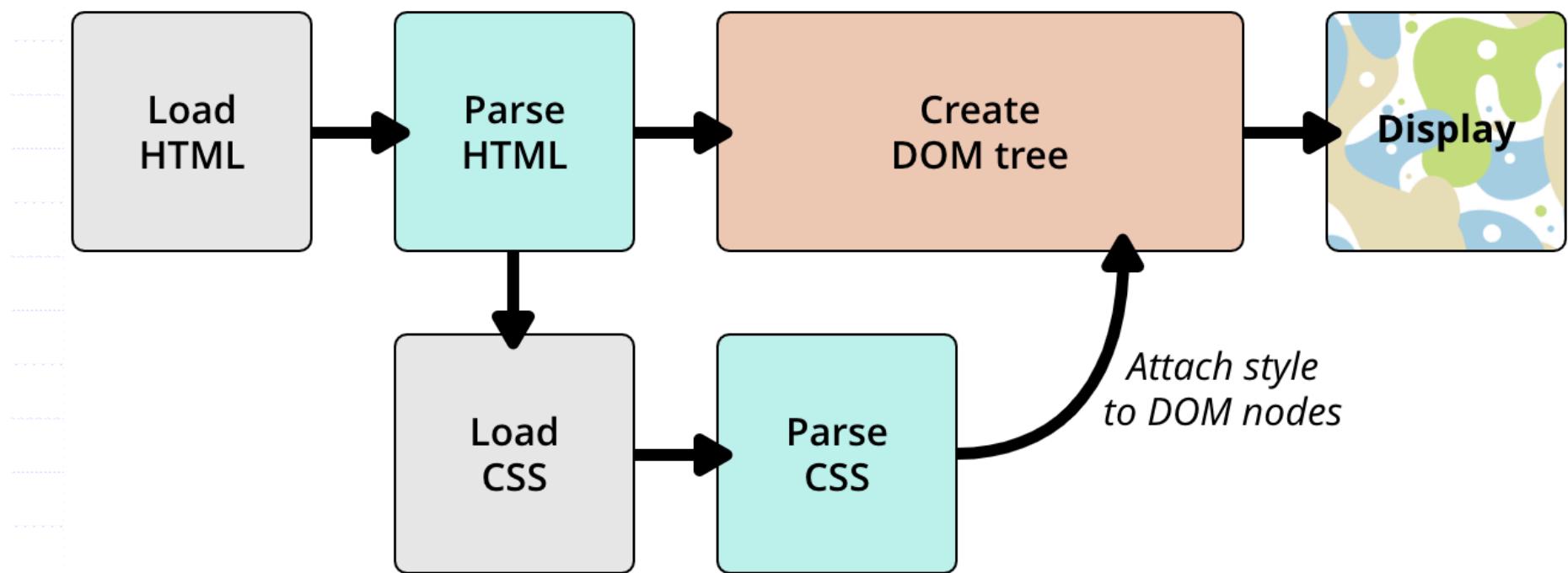
TypeScript sažetak i resursi

- ◆ Instalacija i korišćenje
- ◆ Tipovi podataka
- ◆ Deklaracija i definicija varijabli i funkcija
- ◆ Interfejsi
- ◆ Klase
- ◆ Prevođenje u JS (primeri)
 - TS Klasa -> JS Prototip
- ◆ Zvanična dokumentacija:
<https://www.typescriptlang.org/docs/handbook/basic-types.html> (handbook sekcije)

TypeScript materijali

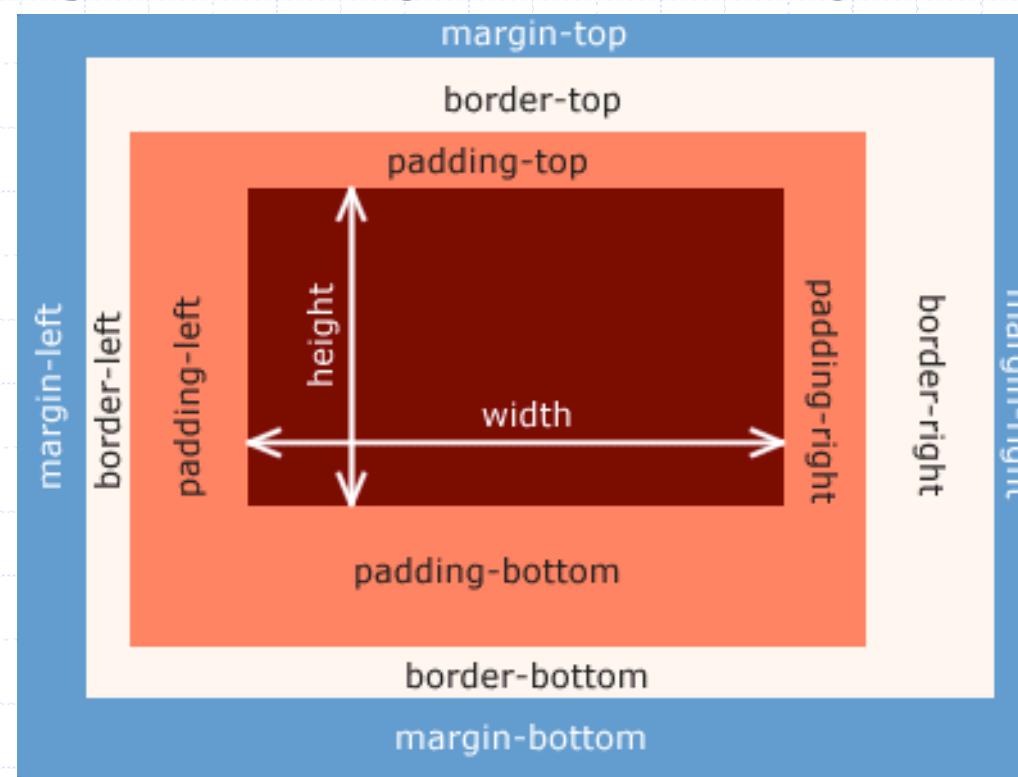
- ◆ Primer klase:
- ◆ Class.ts i Class.js (rezultat prevodenja TS klase u JS prototip)
- ◆ Primer interfejsa:
- ◆ Interface.ts i Interface.js (rezultat prevodenja TS interfejsa u JS funkciju)
- ◆ Primer funkcije:
- ◆ Function.ts i Function.js (rezultat prevodenja TS funkcije u JS funkciju)

Kako funkcioniše CSS ?



CSS box model

- ◆ **Margine** (margin) određuju prostor sa **spoljne strane** granica kontejnera
- ◆ **Pading** (padding) određuje prostor sa **unutrašnje strane** granica kontejnera



CSS veličine i boje

- ◆ height: visina elementa
- ◆ width: širina elementa
- ◆ max-height: maksimalna visina elementa
- ◆ max-width: maksimalna širina elementa
- ◆ min-height: minimalna visina elementa
- ◆ min-width: minimalna širina elementa

- ◆ color: frontalna boja elementa (tekst)
- ◆ background-color: pozadinska boja elementa

CSS fontovi

- ◆ font-family: "Helvetica Neue", "Arial", sans-serif;
- ◆ font-style: italic;
- ◆ font-weight: 100;
- ◆ font-size: 2em;
- ◆ letter-spacing: 1px

CSS jedinice mere

◆ **em:** skalabilna jedinica mere

- Na primer, 1em odgovara tekućoj veličini fonta. Ako je veličina fonta 12pt, $1\text{em}=12\text{pt}$, $2\text{em}=24\text{pt}$, $.5\text{em}=6\text{pt}$ i slično
- Popularna zbog skalabilnosti i responsive dizajna

◆ **px:** pikseli displeja računara kao fiksne jedinice

- Nije skalabilna jedinica – ne prilagođava se dimenzijama displeja

◆ **pt:** fiksna jedinica mere, $\text{pt}=1/72 \text{ inch}$

- Koristi se u štampi dokumenata

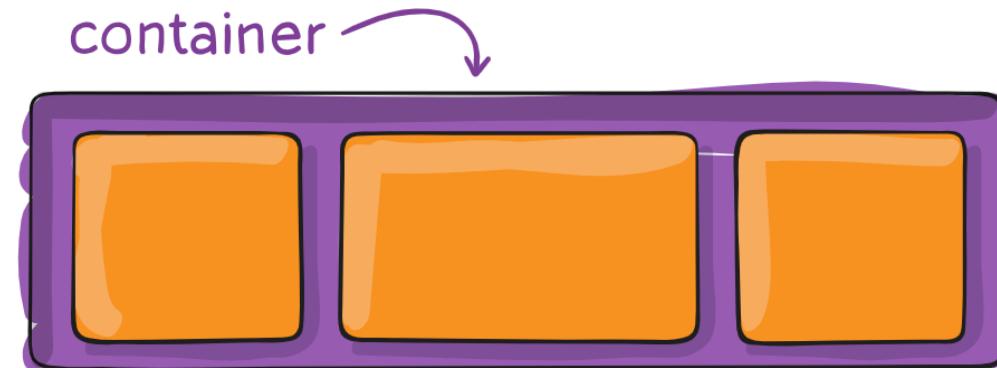
◆ **%:** skalabilna jedinica mere, sa tim što se modifikacije tekuće veličine izražavaju kao procenti

Šta je Flexbox ?

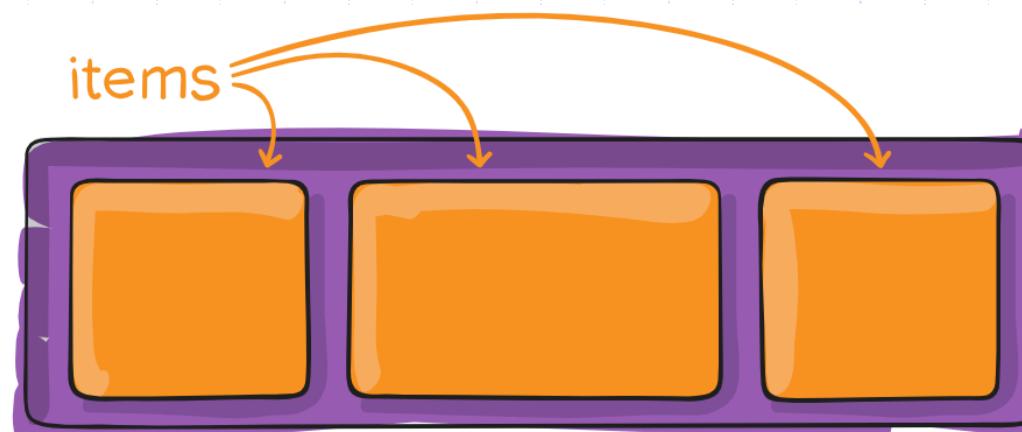
- ◆ Flexbox predstavlja W3C preporuku od Oktobra 2017 koja pruža efikasan načina za raspoređivanje, ravnanje i podelu prostora komponenti Web korisničkih interfejsa
- ◆ Čak i kada njihova veličina nije unapred poznata ili je dinamička (*flex*)
- ◆ Osnovne ideje:
- ◆ Mogućnost kontejnera da menja veličinu i raspored elemenata dece u cilju optimalnog iskorišćenja raspoloživog prostora ekrana
- ◆ Fleksibilnost u pogledu definisanja vizuelnog koordinatnog sistema za raspoređivanje komponenti
 - Block-layout je baziran na vertikalnom rasporedu
 - Inline-layout je baziran na horizontalnom rasporedu

Flexbox - osnovne komponente

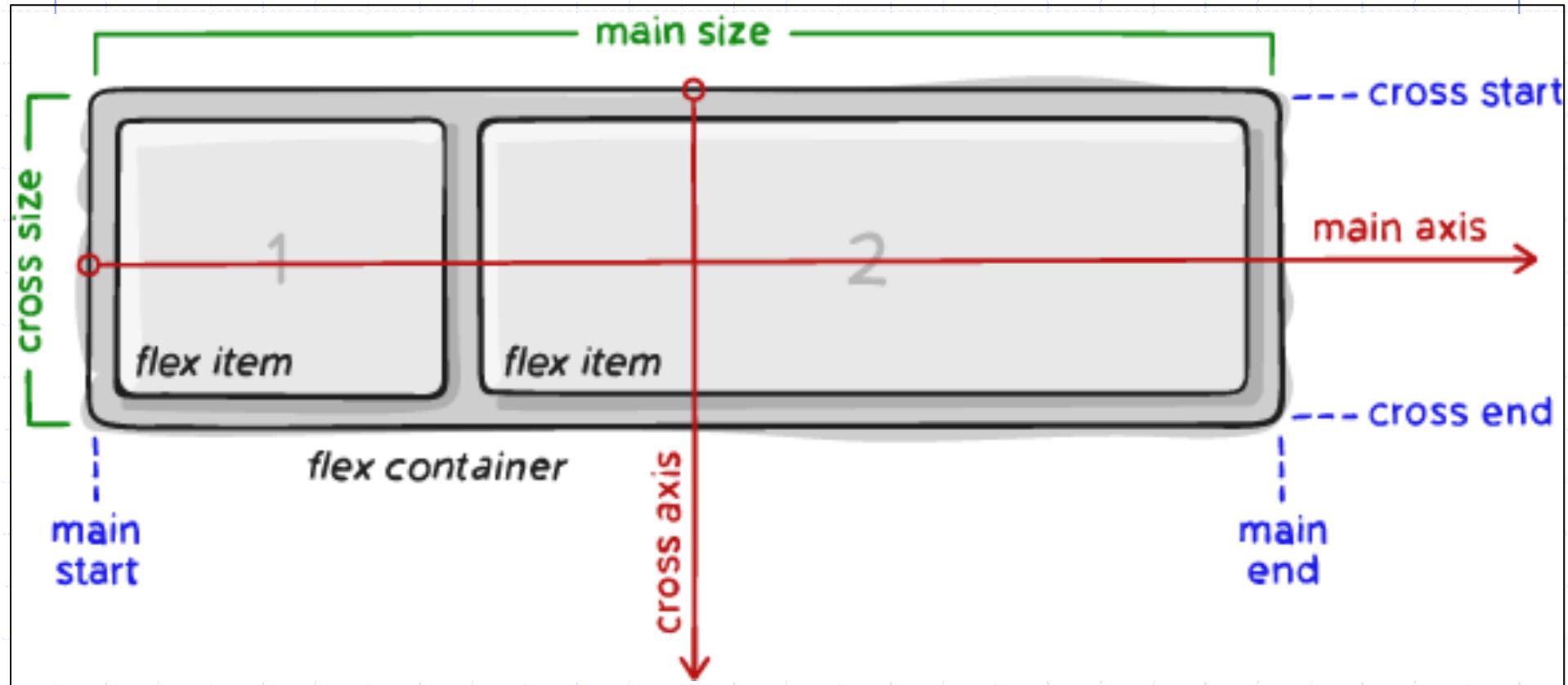
- ◆ Roditelj (*container*)



- ◆ Dete (*item*)



Flexbox – elementi rasperda



Flexbox – elementi rasporeda

- ◆ Deca (items) se raspoređuju po glavnoj osi (**main-axis**) ili upravnoj osi (**cross-axis**)
- ◆ Glavna osa je **podrazumevano horizontalna** i deca se raspoređuju sa leva (**main-start**) na desno (**main-end**)
 - Orijentacija se može menjati programski (**flex-direction**)
- ◆ Upravna osa je uvek normalna na glavnu osu. Njena orijentacija zavisi od orijentacije glavne ose
- ◆ Deca se (podrazumevano) raspoređuju odozgo (**cross-start**) na dole (**cross-end**)
- ◆ Raspored se definiše pomoću CSS naredbi, tj. direktiva



Direktive rasporeda roditeljskih elemenata

Roditeljski element - **display**

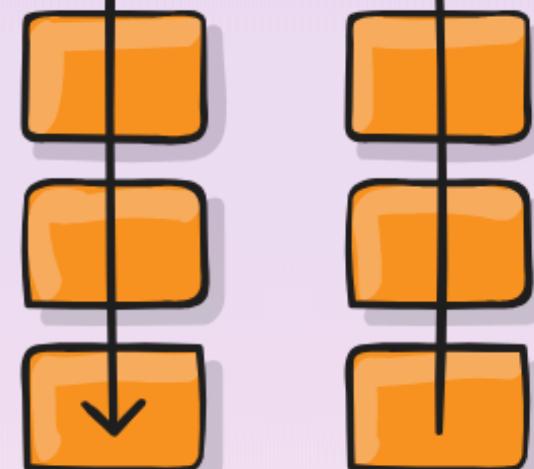
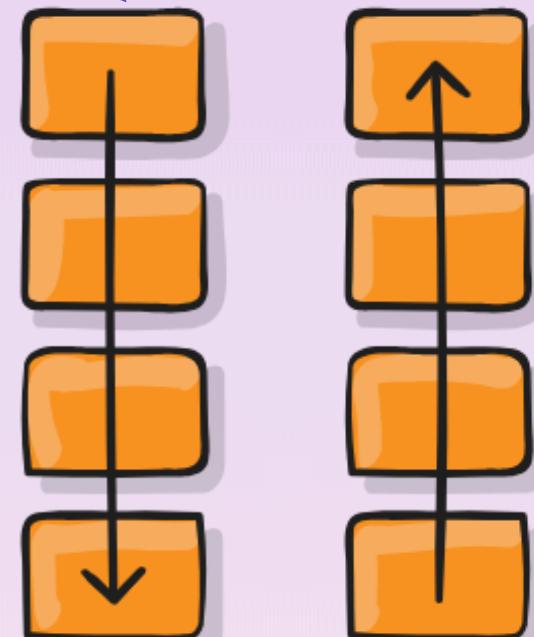
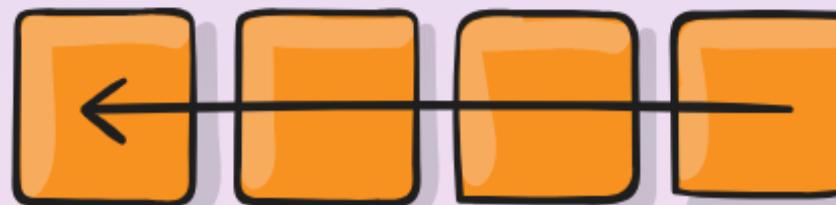
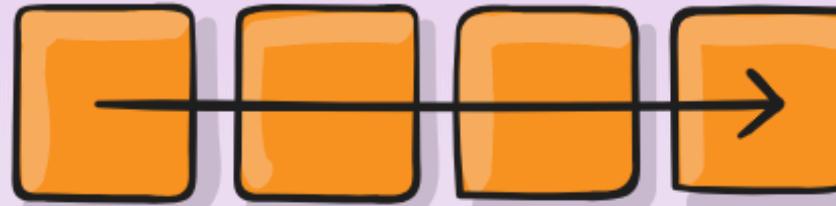
- ◆ Proglašava roditelja Flexbox kontejnerom
- ◆ Podrazumevan *raspored* dece:
- ◆ Jeden red, sa leva na desno, sa mogućim preklapanjem ukoliko nema prostora po glavnoj osi
- ◆ Deca ne povećavaju veličinu (tj. širinu) po glavnoj osi
- ◆ Deca mogu povećati veličinu (tj. visinu) po upravnoj osi u slučaju raspoloživog prostora

```
.container {  
    display: flex; /* or inline-flex */  
}
```

Roditeljski element – **flex-direction**

- ◆ Definiše orijentaciju i smer glavne ose (**main-axis**) na osnovu kojih se raspoređuju deca

```
.container {  
    flex-direction: row | row-reverse | column | column-reverse;  
}
```



Roditeljski element – **flex-wrap**

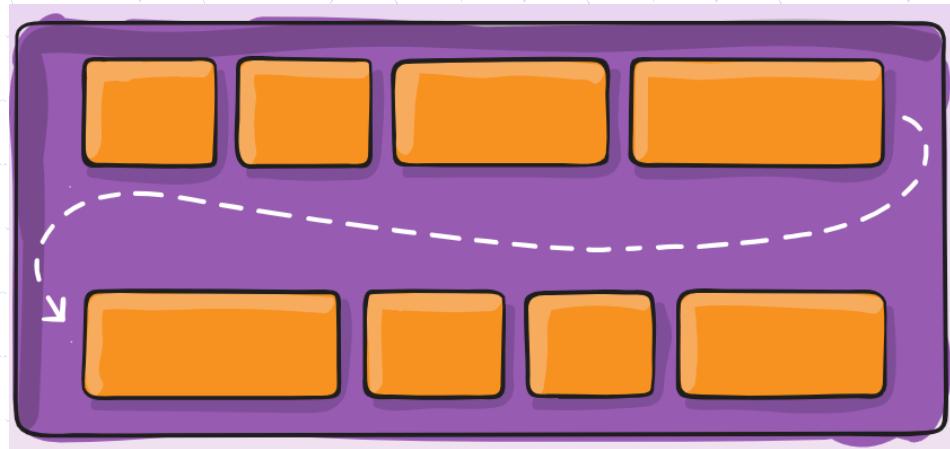
- ◆ Elementi deca se pokušavaju smestiti u jedan red
- ◆ Direktiva omogućava dodatnu fleksibilnost
- ◆ Definiše redosled dece po horizontali, tj. da li se smeštaju u jedan red ili se mogu nastavljati u novim redovima

.container{ flex-wrap: nowrap | wrap | wrap-reverse; }

nowrap: podrazumevano; deca se smeštaju u jedan red

wrap: deca se smeštaju u više redova.

wrap-reverse: deca se smeštaju u više redova, ali u obrnutom redosledu (od dna ka vrhu)



Roditeljski element – justify-content

- ◆ Definiše ravnjanje po glavnoj osi (main-axis) i raspodelu prostora između dece

```
.container {  
justify-content: flex-start |  
           flex-end |  
           center |  
          baseline |  
           stretch;  
}
```

Ravnjanje u odnosu na početak i kraj ose sa ravnomernim međuprostorom

Svako dete dobija jednaku porciju slobodnog prostora oko sebe

Udaljenost od početka/kraja ose i međuprostori između dece su jednaki

flex-start



flex-end



center



space-between



space-around



space-evenly



Roditeljski element – justify-content

- ◆ Primer:

https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flexible_Box_Layout/Basic_Concepts_of_Flexbox#justify-content

Roditeljski element – align-items

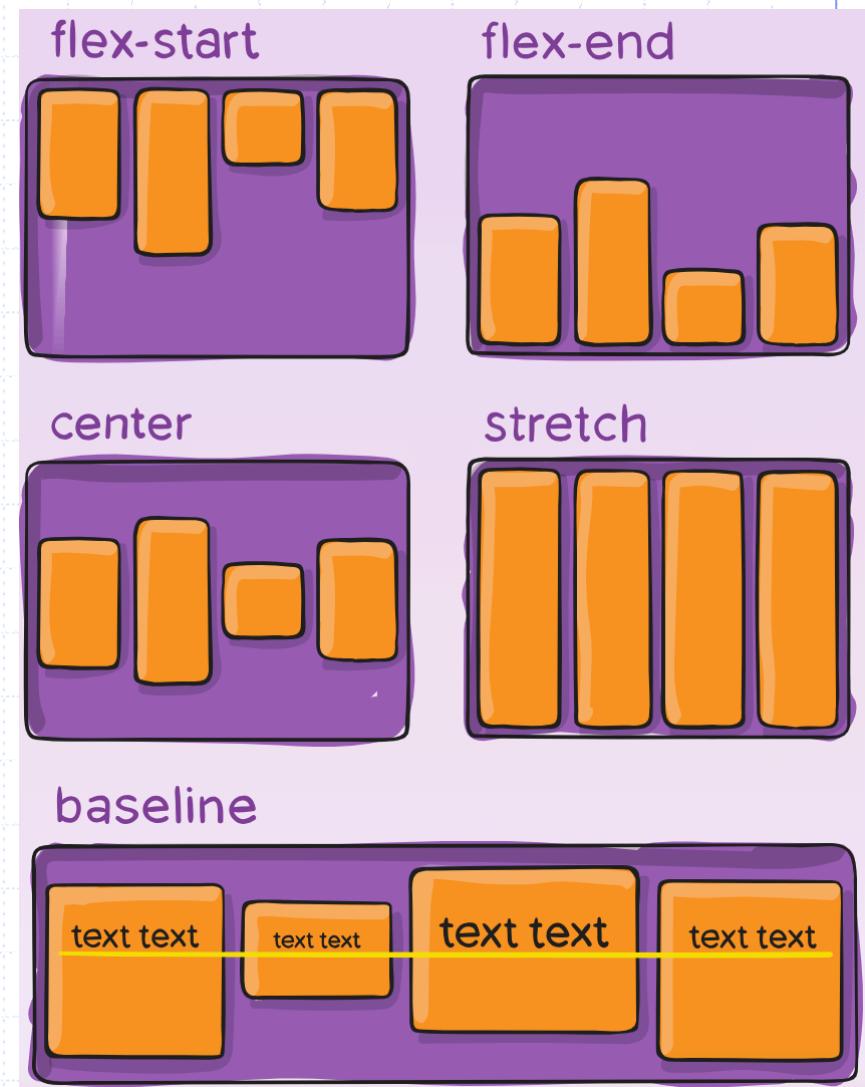
- ◆ Definiše ravnanje po upravnoj osi (cross-axis) i raspodelu slobodnog prostora između dece

```
.container {  
    align-items: flex-start |  
                flex-end |  
                center |  
                baseline |  
                stretch;  
}
```

stretch: podrazumevano
ponašanje; deca mogu redefinisati
svoju max veličinu

center: centriranje po upravnoj osi

baseline: ravnanje u odnosu na osu
koja predstavlja osnovnu liniju
teksta elemenata dece



Roditeljski element – align-items

◆ Primer:

https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flexible_Box_Layout/Basic_Concepts_of_Flexbox#align-items

Roditeljski element – align-content

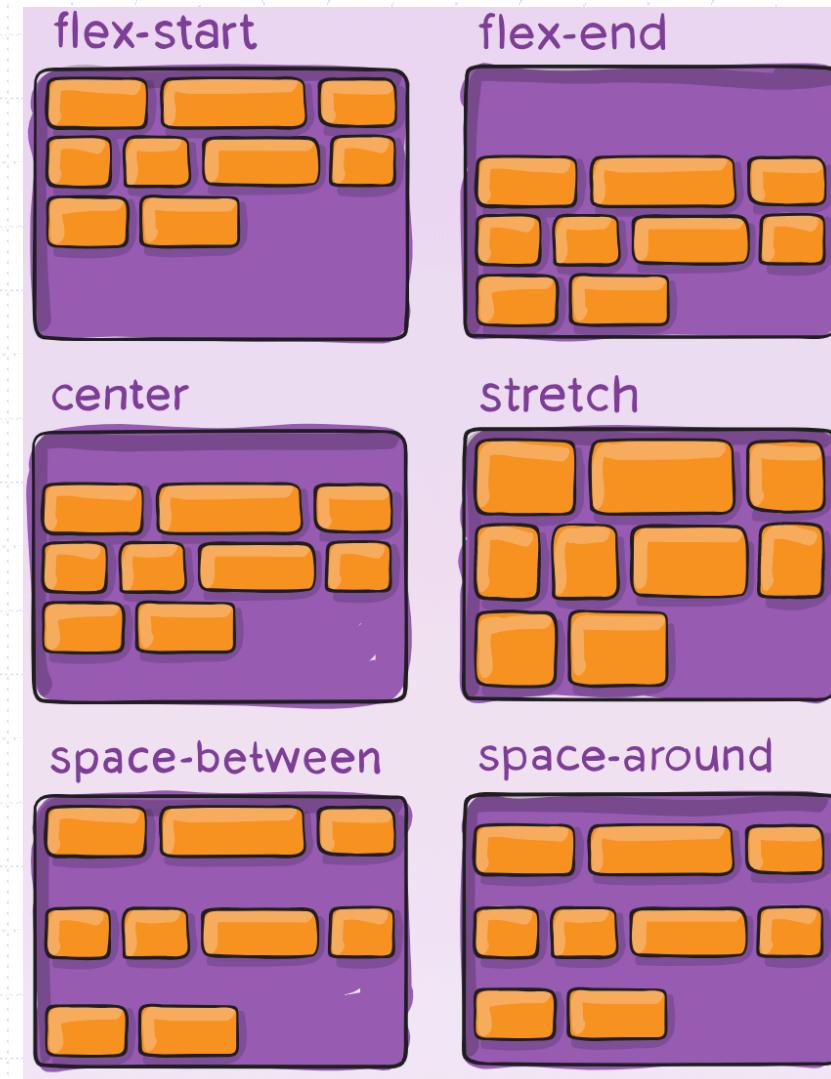
- ◆ Definiše ravnanje grupa, tj. linija dece po upravnoj osi (cross-axis) - samo u slučaju **više redova** dece

```
.container {
```

```
  align-content: flex-start |  
    flex-end |  
    center |  
    space-between |  
    space-around |  
    stretch;
```

```
}
```

stretch: podrazumevano ponašanje; dimenzije dece-redova su identične; deca mogu redefinisati svoju max veličinu
space-between: ravnomeran međuprostor sa ravnanjem uz granice
space-around: ravnomeran međuprostor uključujući i granice



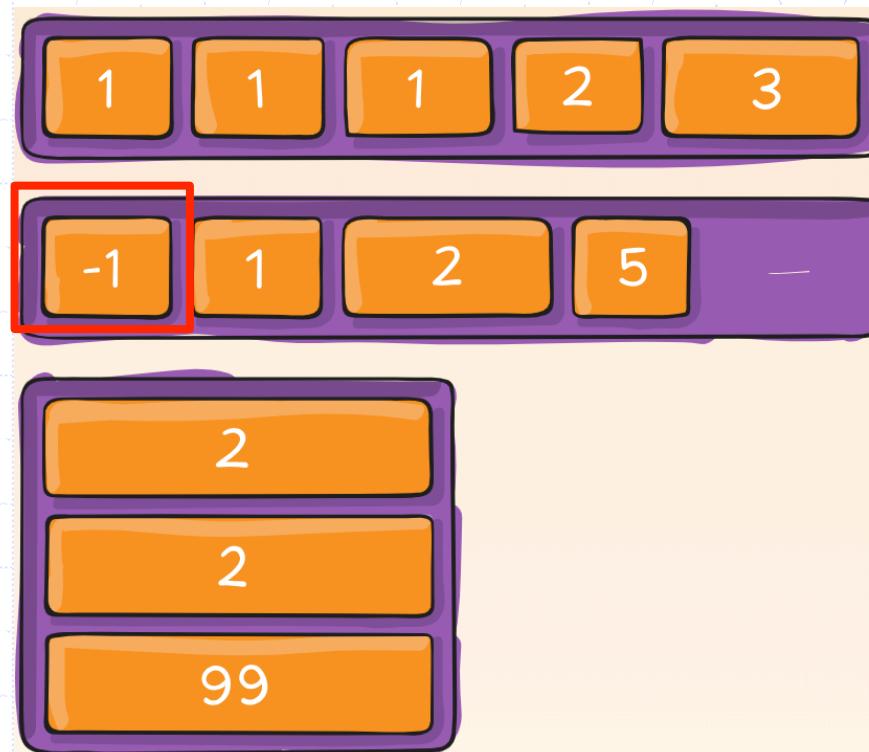


**Direktive rasporeda
elemenata dece**

Element dete – order

- ◆ Deca se podrazumevano prikazuju redosledom kojim su definisana
- ◆ Definiše redosled prikaza deteta u roditelju

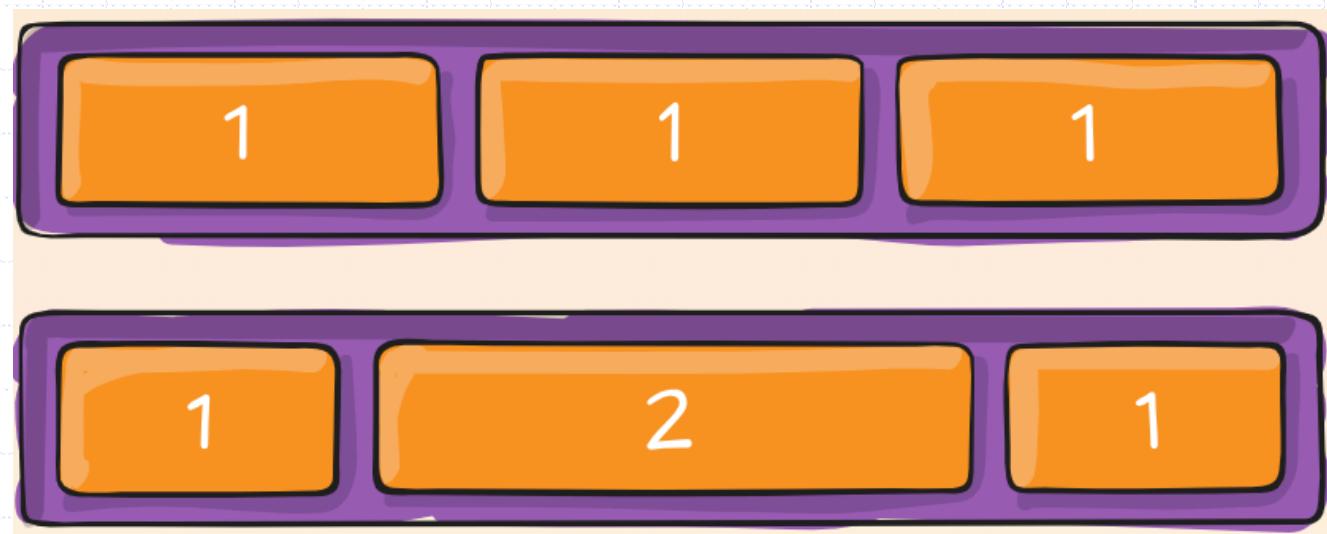
```
.item { order: <integer>; /* default = 0 */ }
```



Element dete – **flex-grow**

- ◆ Mogućnost deteta da povećava svoju veličinu u odnosu na glavnu osu (**main-axis**) u zavisnosti od slobodnog prostora
- ◆ Prihvata broj koji služi kao proporcija – udeo slobodnog prostora kontejnera koji dete može zauzeti u odnosu na drugu decu
- ◆ Podrazumevano je 0 (nema povećana veličine)

```
.item { flex-grow: <pozitivan broj>; /* default = 0 */ }
```



Element dete – **flex-shrink**

- ◆ Mogućnost deteta da smanjuje svoju veličinu u odnosu na glavnu osu (**main-axis**) u slučaju smanjenja prostora
- ◆ Prihvata broj koji služi kao proporcija – veća vrednost broja znači da će se element više smanjivati u odnosu na susedne elemente
- ◆ Minimalna veličina deteta se uzima u obzir ukoliko je zadata
- ◆ Podrazumevano je 0 (nema smanjenja veličine)
- ◆ U praksi se obično zadaje ista vrednosti kako bi se veličine susednih elemenata smanjivale ujednačeno

```
.item { flex-shrink: <pozitivan broj>; /* default = 0 */ }
```

Element dete – **flex-basis**

- ◆ Definiše podrazumevanu veličinu deteta po glavnoj osi (**main-axis**) tj. širini
- ◆ Vrednost može biti broj ili ključna reč
- ◆ Preduslov za primenu dve prethodne direktive, tj, da li će se veličina elementa povećavati (**flex-grow**) ili smanjivati (**flex-shrink**)
- ◆ Podrazumevena vrednost je **auto** što znači da će biti konsultovana veličina deteta ako je zadata
- ◆ Ako veličina nije zadata, veličina sadržaja deteta će definisati prostor koji ono zauzima

```
.item { flex-basis: <širina> | auto; }
```

Skraćeni oblik za više direktiva - flex

- ◆ Kombinuje povećanje veličine (**flex-grow**), smanjenje veličine (**flex-shrink**) i podrazumevanu veličinu (**flex-basis**)
- ◆ Vrednosti se navode po redosledu iznad
- ◆ Mogu se koristiti i simboličke konstante kao ispod

flex: initial = **flex: 0 1 auto** (podrazumevano)

flex: auto = **flex: 1 1 auto**

flex: none = **flex: 0 0 auto**

Skraćeni oblik za više direktiva - flex

- ◆ Primer:

https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flexible_Box_Layout/Basic_Concepts_of_Flexbox#Shorthand_values_for_the_flex_properties

Flexbox - sažetak

- ◆ Flexbox naredbe se navode u stilu Angular komponente (css datoteka)
- ◆ Princip korišćenja:
- ◆ Definisati klase stila u css datoteci
 - Naredbe koje se odnose na roditelja (.container)
 - Naredbe koje se odnose na decu (.item)
- ◆ Klase dodeliti kontejnerskim elementima templejta komponente (div, section)
 - `div class="container"` ili `div class="item"`

Materijal pripremljen na osnovu:

<https://css-tricks.com/snippets/css/a-guide-to-flexbox/>

Flexbox resursi

- ◆ Flexbox tutorijal:

<https://css-tricks.com/snippets/css/a-guide-to-flexbox/>

- ◆ Flexbox MDN tutorijal:

https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flexible_Box_Layout/Basic_Concepts_of_Flexbox

- ◆ Flexbox sa definisanje globalnog layout-a Web sajta:

<https://www.sitepoint.com/flexbox-css-flexible-box-layout/>

Flexlayout - sadržaj

- ◆ Flexlayout:
- ◆ Definicija
- ◆ Terminologija
- ◆ Roditeljski elementi stila
- ◆ Elementi stila dece

Šta je Flexlayout ?

- ◆ Angular biblioteka koja sadrži HTML direktive (ili atribute) za definisanje fleksibilnih **rasporeda** elemenata interfejsa
- ◆ Predstavlja nadogradnju Flexbox-a koje je intuitivnija za korišćenje:
- ◆ Umesto upotrebe CSS stilova i sintakse, programer definiše raspored direktno u templejtu Angular komponente
- ◆ Slično Flexbox-u, postoje direktive templejta za roditeljske elemente (kontejneri) i direktive za elemente decu

Zbog čega Flexlayout ?

- ◆ Nastao kao potreba zajednice programera Web korisničkih interfejsa da definišu **fleksibilan** i **dinamički** raspored elemenata direktno u HTML dokumentu
- ◆ Fleksibilan: prilagodljiv displejima različitih dimenzija tako da obezbedi podjednako kvalitetan osećaj korišćenja za različite korisnike
- ◆ Dinamički: fleksibilan u vreme korišćenja, tj. izvršavanja aplikacije

Flexlayout korišćenje i paketi

- ◆ Flexlayout direktive se instaliraju kao dodatna biblioteka na nivou Angular projekta

```
npm install @angular/flex-layout --save
```

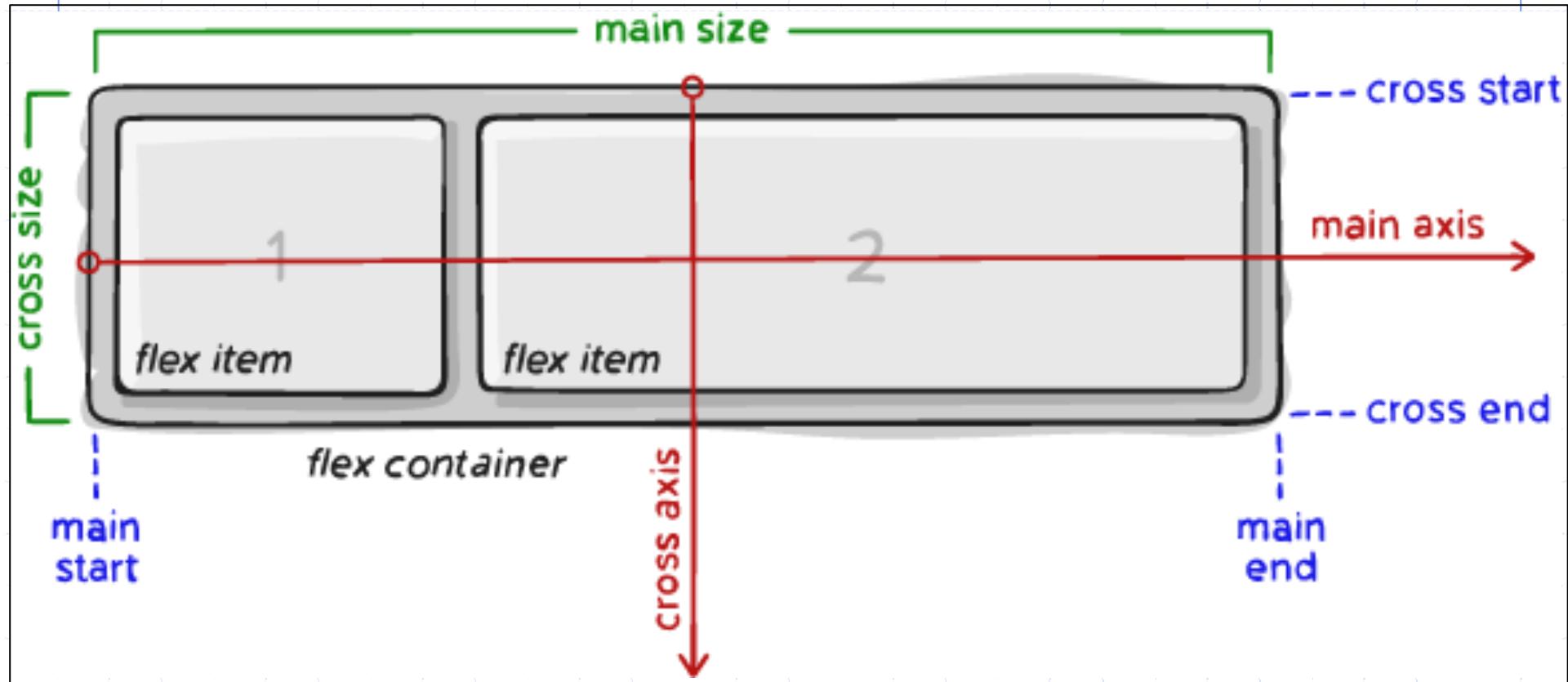
@angular/flex-layout biblioteka:

- ◆ **Static API**
- ◆ **Responsive API**

package.json

```
"dependencies": {  
  "@angular/animations": "^7.0.0",  
  "@angular/cdk": "^7.0.0",  
  "@angular/common": "^7.0.0",  
  "@angular/compiler": "^7.0.0",  
  "@angular/core": "^7.0.0",  
  "@angular/flex-layout": "7.0.0-beta.19",  
}
```

Flexlayout elementi raspoređeni su nasleđeni od Flexbox-a

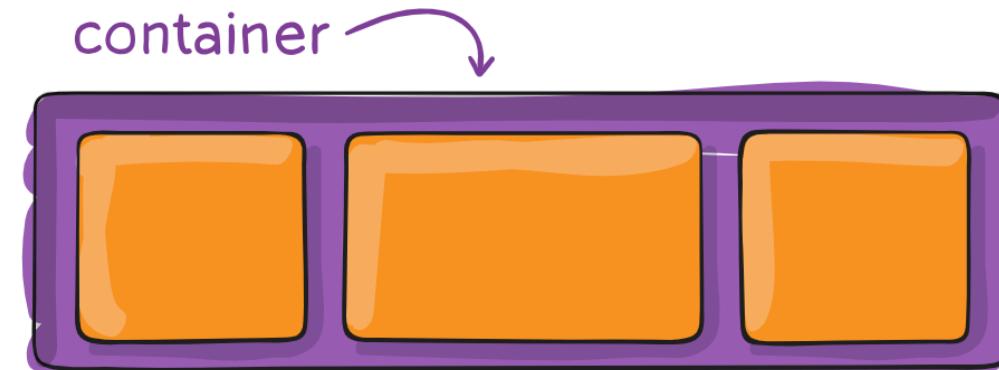


Flexlayout – elementi rasporeda

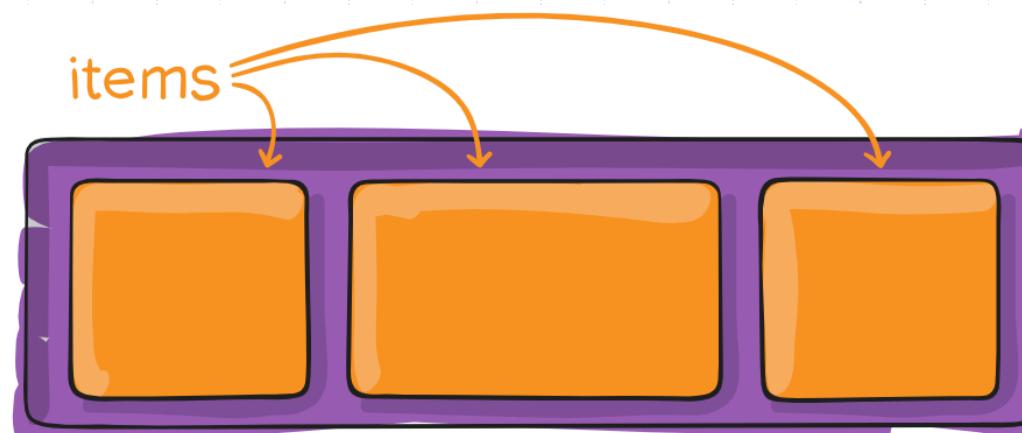
- ◆ Deca (items) se raspoređuju po glavnoj osi (**main-axis**) ili upravnoj osi (**cross-axis**)
- ◆ Glavna osa je **podrazumevano horizontalna** i deca se raspoređuju sa leva (**main-start**) na desno (**main-end**)
 - Orijentacija se može menjati programski (**flex-direction**)
- ◆ Upravna osa je uvek normalna na glavnu osu.
Njena orijentacija zavisi od orijentacije glavne ose
- ◆ Deca se (podrazumevano) raspoređuju odozgo (**cross-start**) na dole (**cross-end**)
- ◆ Raspored se definiše pomoću naredbi ili direktiva templejta

Osnovne komponente

◆ Roditelj (*container*)



◆ Dete (*item*)



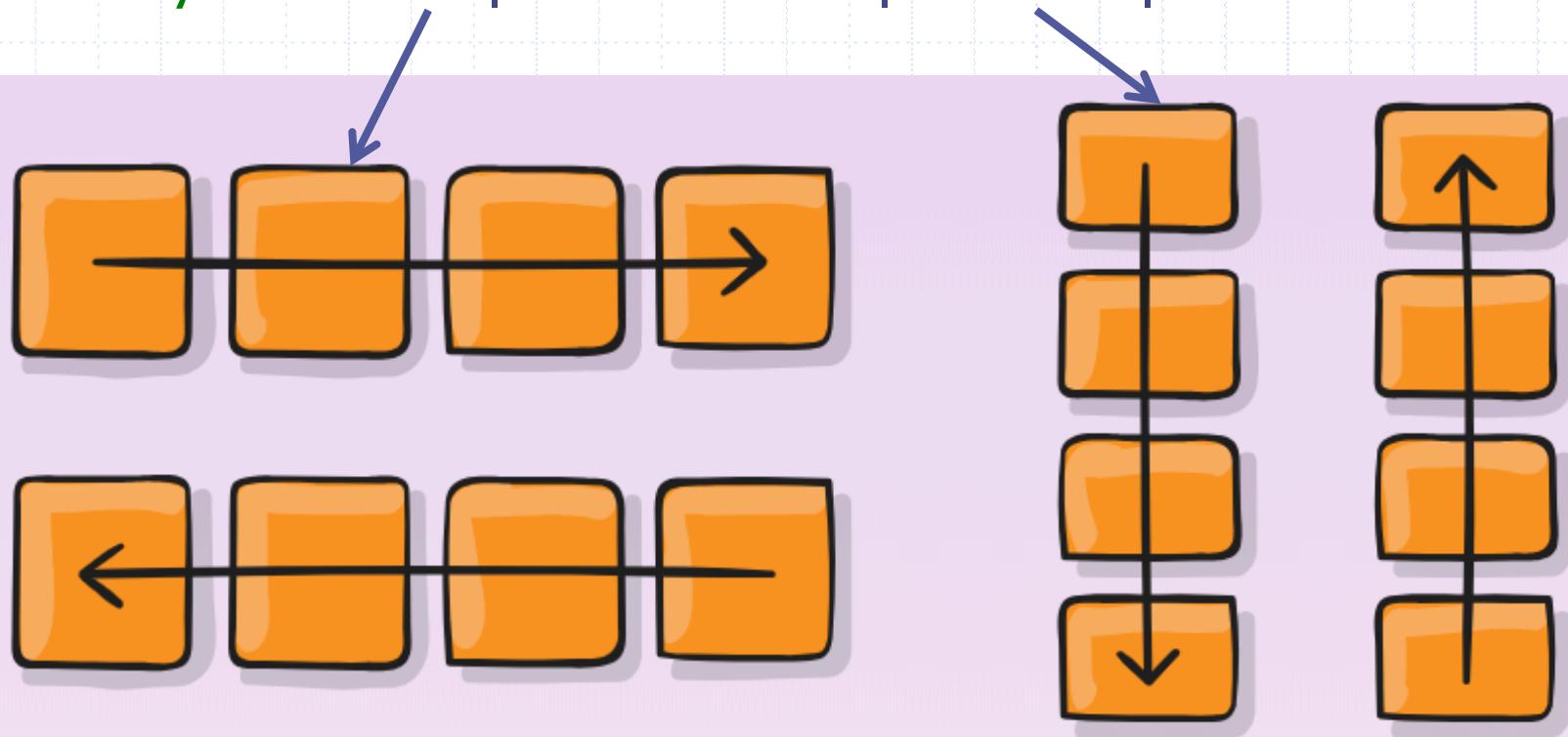


Direktive rasporeda roditeljskih elemenata

Roditeljski element – **fxLayout**

- ◆ Definiše orijentaciju i smer glavne ose (**main-axis**) na osnovu kojih se raspoređuju deca
- ◆ Flexbox ekvivalent: **flex-direction**

```
<div fxLayout="row | row-reverse | column | column-reverse">
```



fxLayout primer 1/2

```
<div fxLayout="row">  
  <div>1. One</div>  
  <div>2. Two</div>  
  <div>3. Three</div>  
  <div>4. Four</div>  
</div>
```

1. One

2. Two

3. Three

4. Four

fxLayout primer 2/2

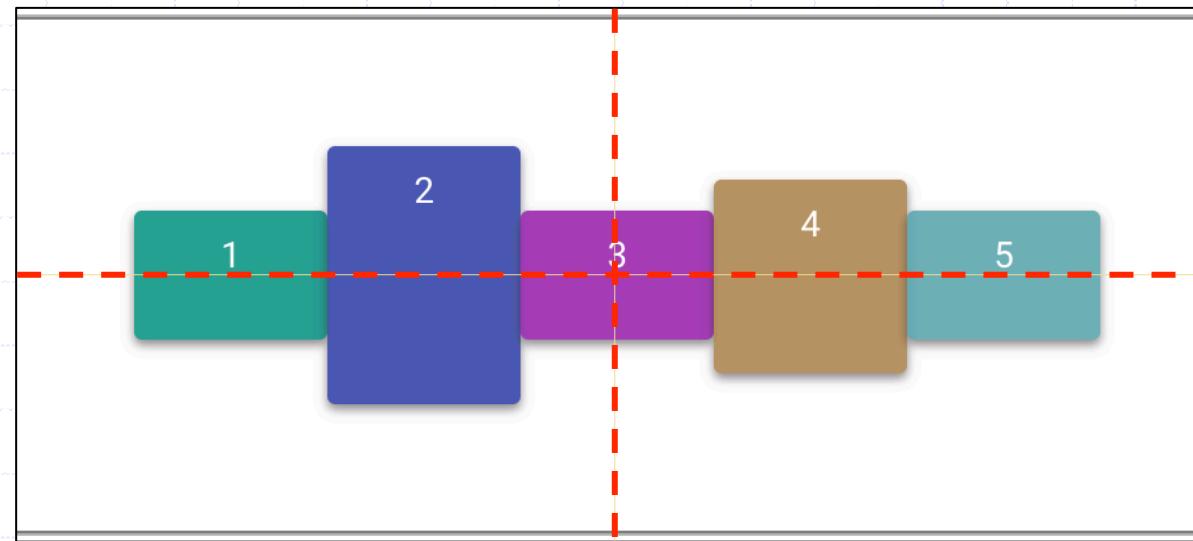
```
<div fxLayout="column">  
  <div>1. One</div>  
  <div>2. Two</div>  
  <div>3. Three</div>  
  <div>4. Four</div>  
</div>
```



Roditeljski element – **fxLayoutAlign**

- ◆ Definiše ravnjanje elemenata dece po glavnoj osi (**main-axis**) i opcionalno upravnoj osi (**cross-axis**)
- ◆ Prva vrednost koja se navodi predstavlja ravnjanje po glavnoj osi
- ◆ Druga vrednost koja se navodi predstavlja ravnjanje po upravnoj osi (opcionalno)

```
<div fxLayout="row" fxLayoutAlign="center center">...</div>
```



Roditeljski element – fxLayoutAlign

◆ Flexbox ekvivalent:

justify-content

1. vrednost – glavna osa



align-items

align-content

2. vrednost – upravna osa

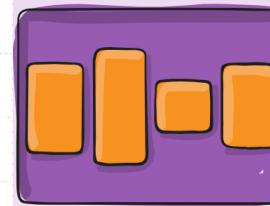
flex-start



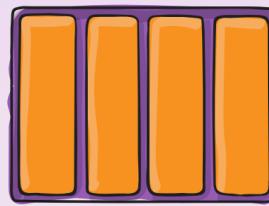
flex-end



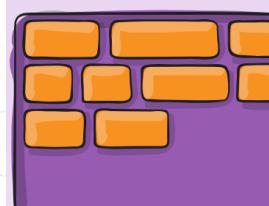
center



stretch



flex-start



flex-end



center



stretch



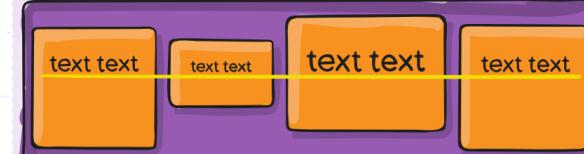
space-between



space-around



baseline



fxLayoutAlign primeri – nazivi i značenja ravnanja su identični kao za Flexbox !

Layout Children with 'layout-align'

1

2

3

4

5

Layout Direction

- row
- column

Alignment in Layout Direction (horizontal)

- none
- start (default)
- center
- end
- space-around
- space-between
- space-evenly

Alignment in Perpendicular Direction (vertical)

- none
- start
- center
- end
- stretch (default)

```
<div fxLayout="row" fxLayoutAlign="space-around center" >
```

Roditeljski element – **fxLayoutGap**

- ◆ Definiše **margin** oko elemenata dece kontejnera
- ◆ Koje margin se dodaju zavisi od orijentacije glavne ose:

`fxLayout="row" => horizontalne margine`

`fxLayout="column" => vertikalne margine`

Jedinice mere:

%

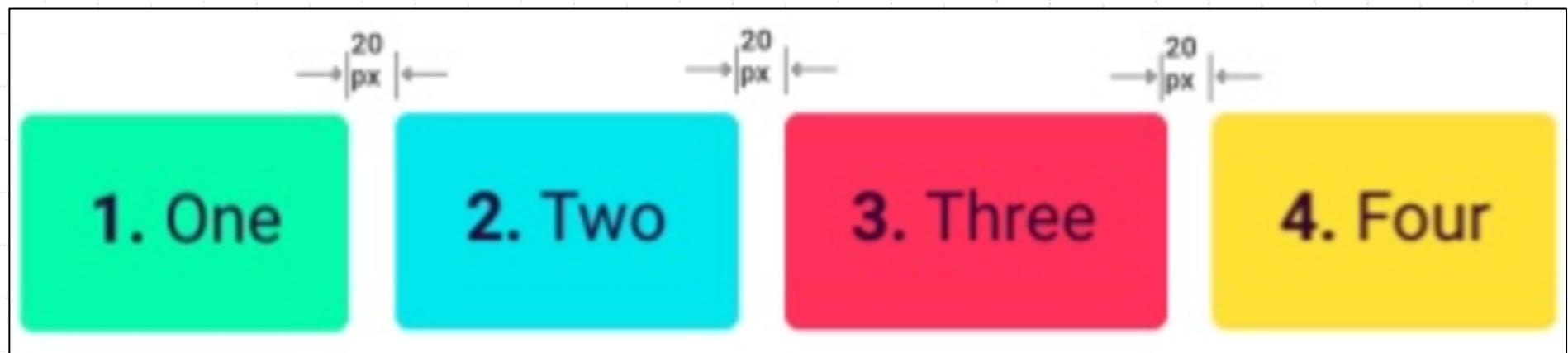
px

$1 \text{ vw} = 1\% \text{ širine prozora pretraživača (viewport width)}$

$1 \text{ vh} = 1\% \text{ visine prozora pretraživača (viewport height)}$

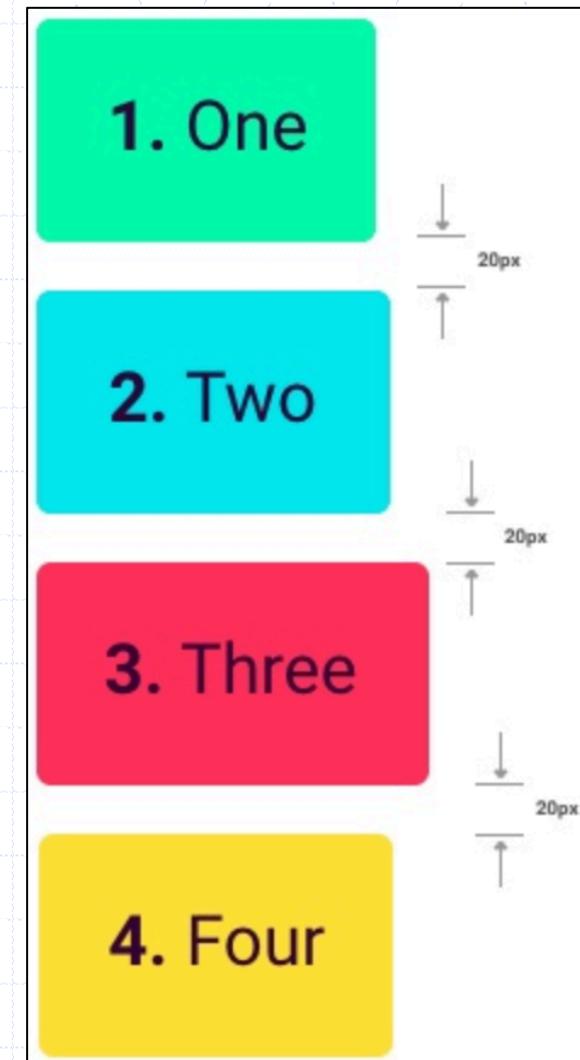
fxLayoutGap primer 1/2

```
<div fxLayout="row" fxLayoutGap="20px">  
  <div>1. One</div>  
  <div>2. Two</div>  
  <div>3. Three</div>  
  <div>4. Four</div>  
</div>
```



fxLayoutGap primer 2/2

```
<div fxLayout="column" fxLayoutGap="20px">  
  <div>1. One</div>  
  <div>2. Two</div>  
  <div>3. Three</div>  
  <div>4. Four</div>  
</div>
```





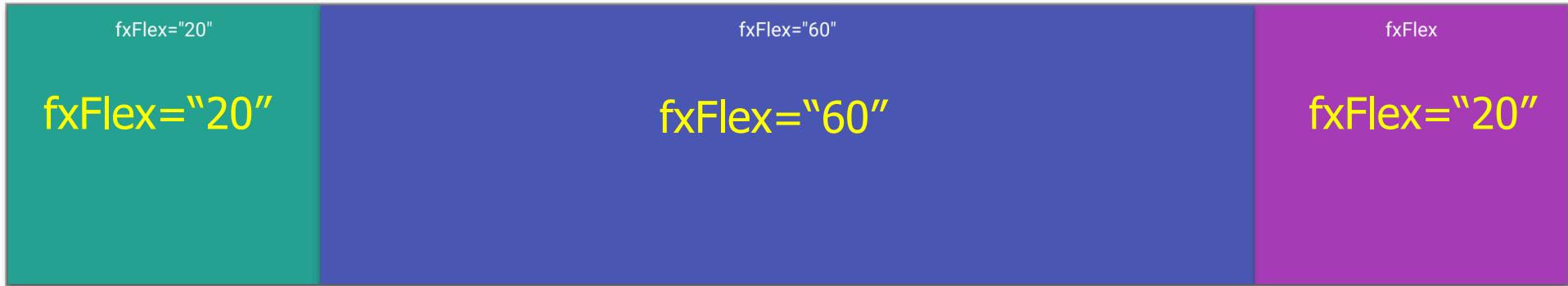
**Direktive rasporeda
elemenata dece**

Element dete – **fxFlex** 1/2

- ◆ Definiše promenu veličine elementa u kontejneru u odnosu na glavnu osu (**main-axis**) kao podrazumevanu veličinu, mogućnost povećanja veličine, i mogućnost smanjenja veličine
- ◆ Flexbox ekvivalent: **flex-grow**
- ◆ **Skraćena notacija:** navodi se jedna vrednost koja definiše podrazumevanu veličinu elementa i ravnomerno povećanje/smanjenje veličine
 - Ukoliko se ne navede vrednost, dete će zauzeti sav raspoloživ prostor po glavnoj osi roditelja
 - Ukoliko više elemenata dece navede direktivu bez vrednosti, ravnomerno će podeliti raspoloživ prostor
- ◆ Puna notacija: navode se tri vrednosti kao povećanje (**grow**), smanjenje (**shrink**) i podrazumevana veličina (**basis**)
- ◆ Jedinice mere: % px
 - 1 vw = 1% širine prozora pretraživača (viewport width)
 - 1 vh = 1% visine prozora pretraživača (viewport height)

Element dete – **fxFlex** 2/2

<div fxLayout="row">



<div fxLayout="column">

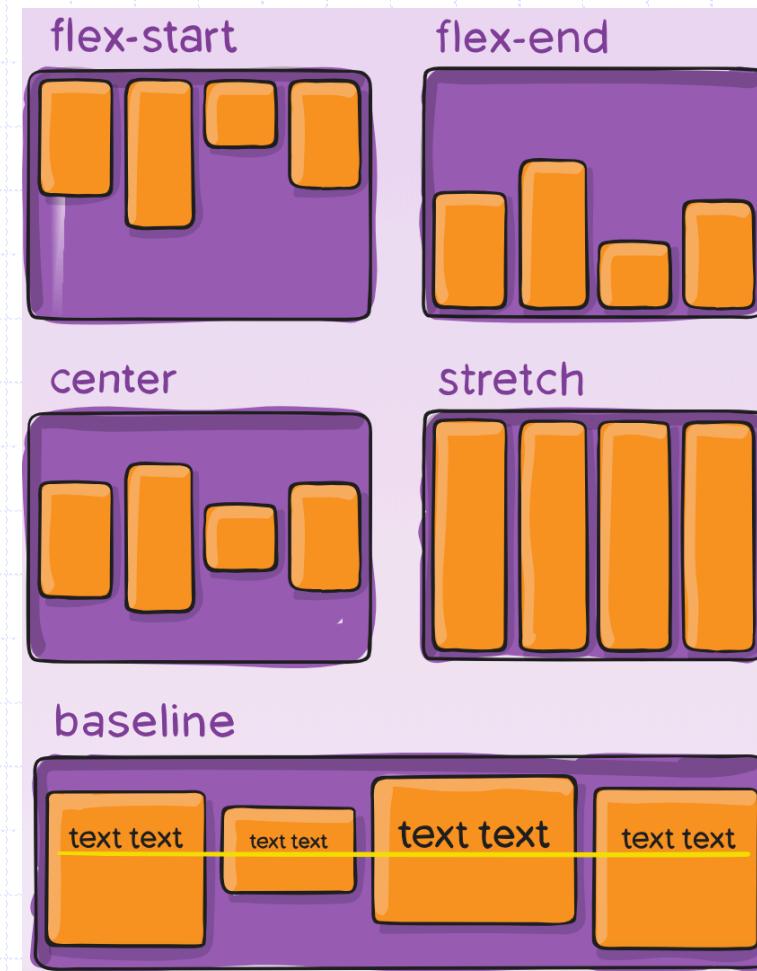


fxFlex predefinisane vrednosti (alias)

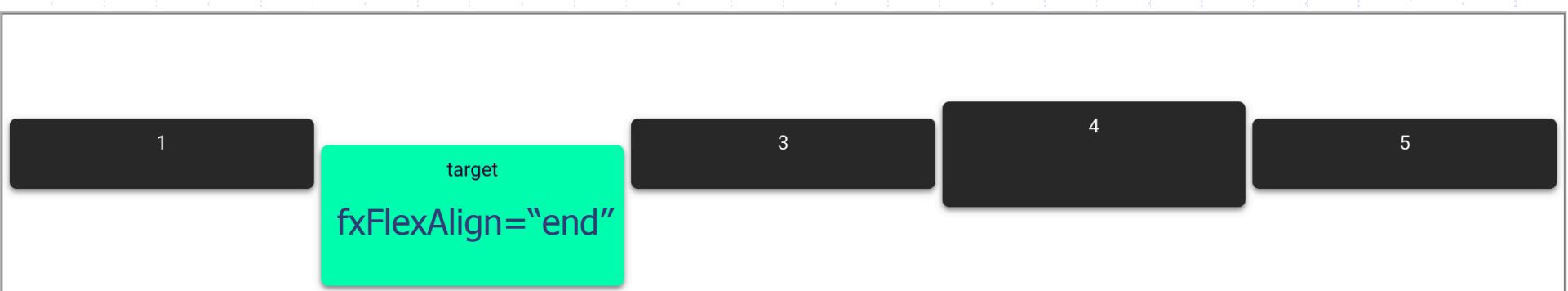
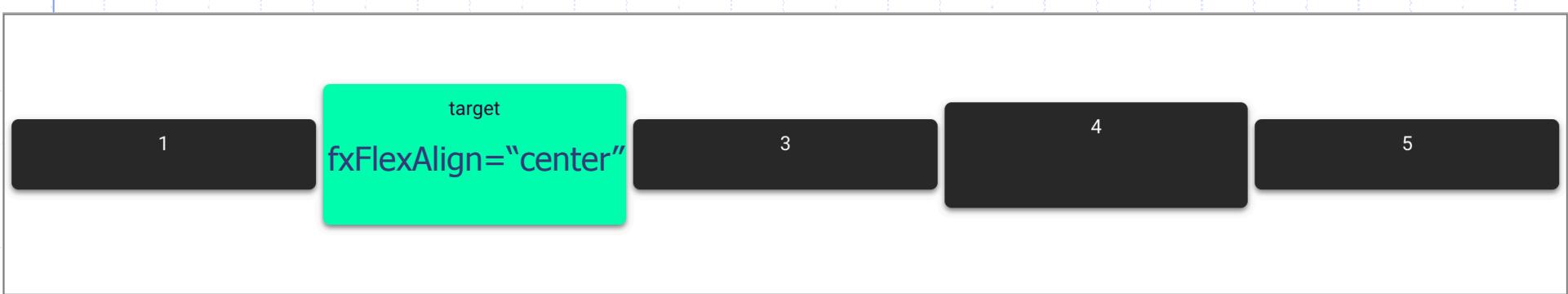
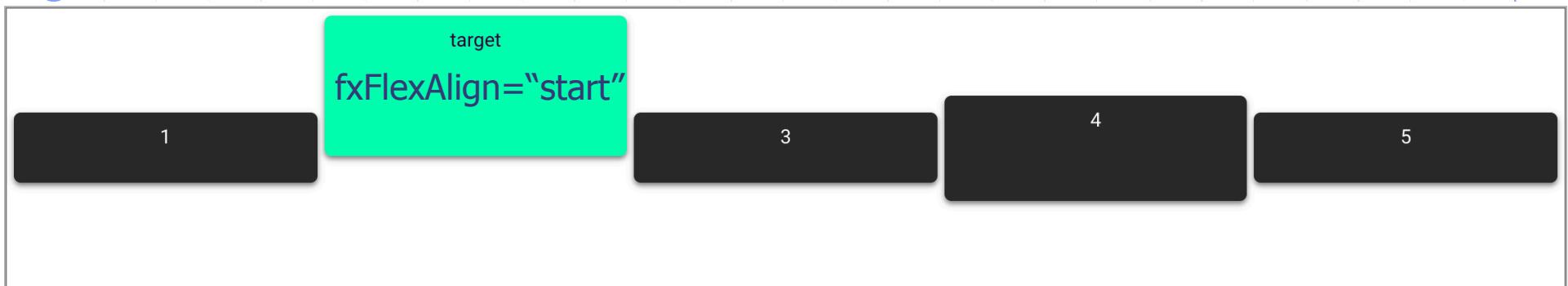
| alias | Flexbox (css) značenje |
|--------------|-------------------------------|
| grow | <code>flex: 1 1 100%</code> |
| none | <code>flex: 0 0 auto</code> |
| nogrow | <code>flex: 0 1 auto</code> |
| noshrink | <code>flex: 1 0 auto</code> |

Element dete - **fxFlexAlign**

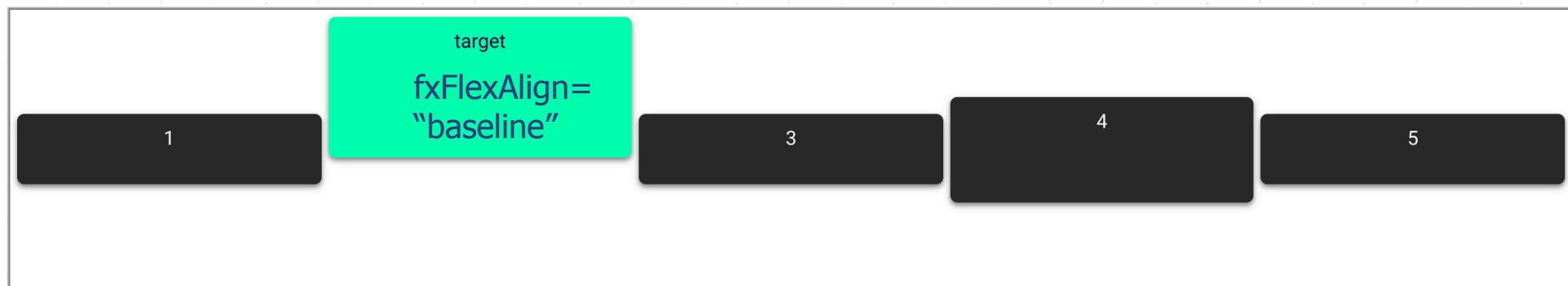
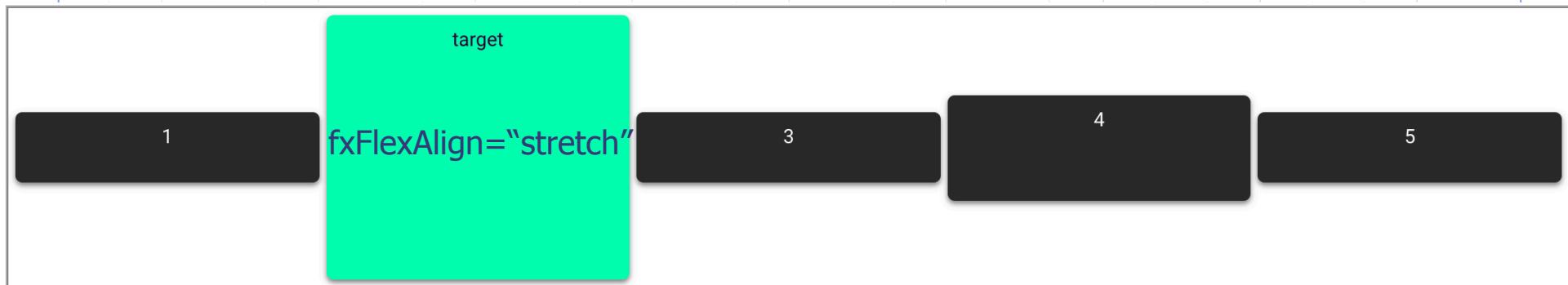
- ◆ Ravnanje na nivou deteta, redefiniše vrednost ravnjanja roditelja po uporednoj osi (**cross-axis**)
- ◆ Vrednosti:
 - start (= flex-start)
 - end (= flex-end)
 - center
 - baseline
 - stretch



fxFlexAlign primeri 1/2



fxFlexAlign primeri 2/2



Flexlayout - sažetak

- ◆ Flexlayout naredbe (direktive) se navode u templejtu Angular Material komponente (html datoteka)
 - Princip rada je identičan kao Flexbox
- ◆ Naredbe koje se odnose na roditelja (kontejner)
- ◆ Naredbe koje se odnose na decu

Materijal pripremljen na osnovu:

<https://github.com/angular/flex-layout/wiki/Declarative-API-Overview>

Flexlayout resursi

- ◆ Zvanična dokumentacija:

[https://github.com/angular/flex-layout/wiki/
Declarative-API-Overview](https://github.com/angular/flex-layout/wiki/Declarative-API-Overview)

- ◆ Online demo editor:

[https://tburleson-layouts-demos.firebaseio.com/#/
docs](https://tburleson-layouts-demos.firebaseio.com/#/docs)

TypeScript Flexbox Flexlayout

Dr. Mlađan Jovanović
mjovanovic@singidunum.ac.rs