

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

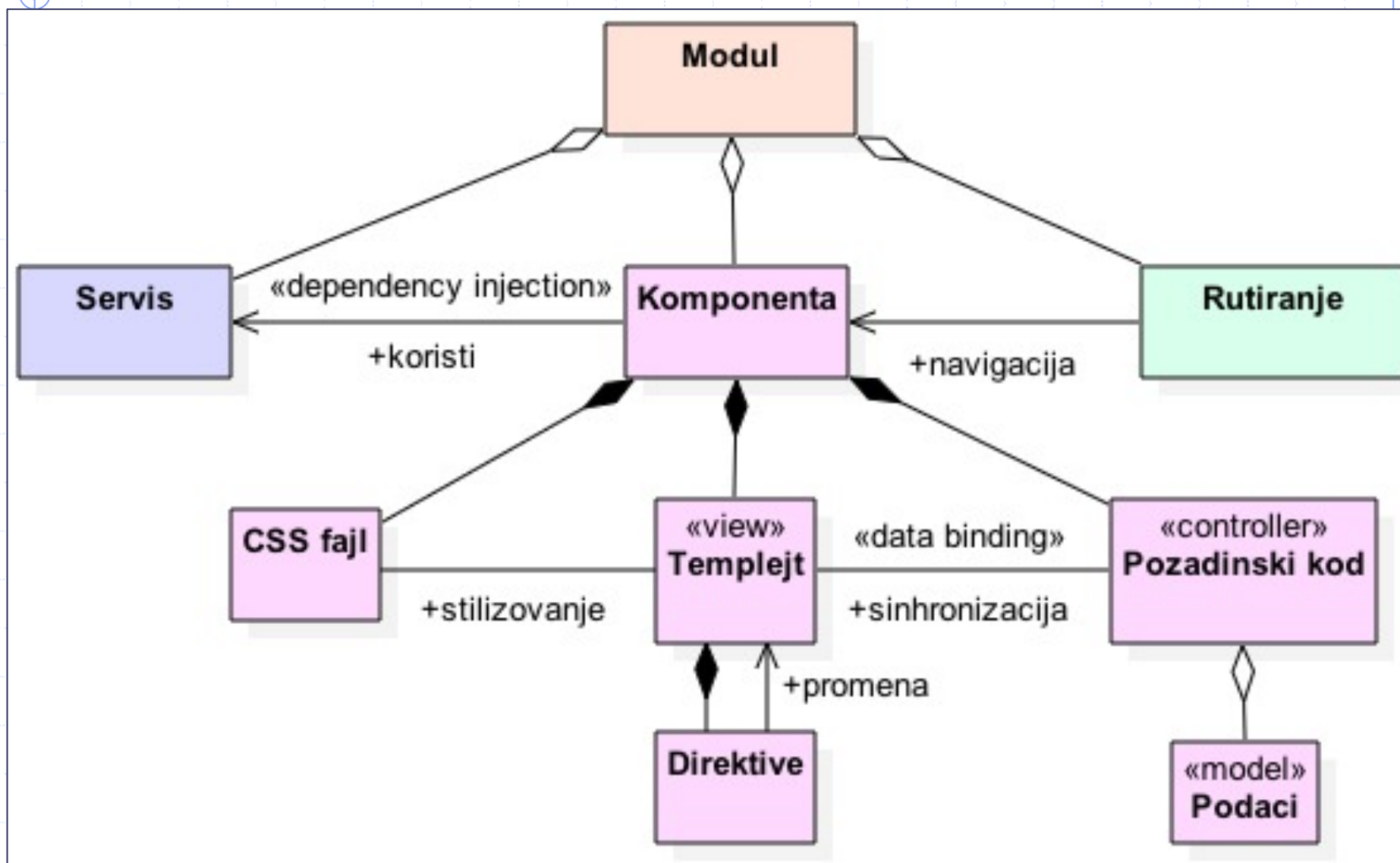
AngularJS

◆ 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 grupiše 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 implementiraju 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

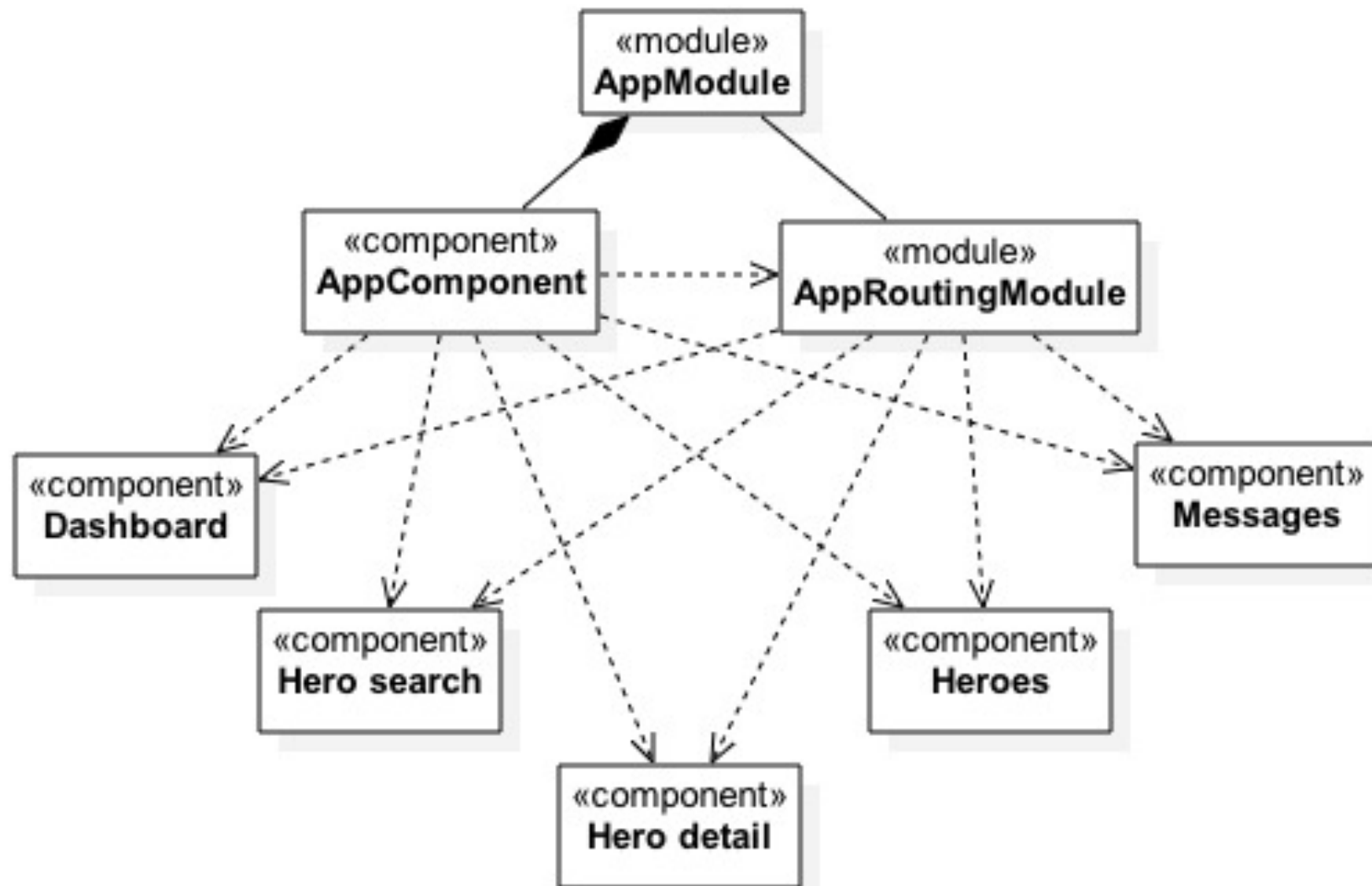
Rutiranje (*routing*)

- ◆ 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

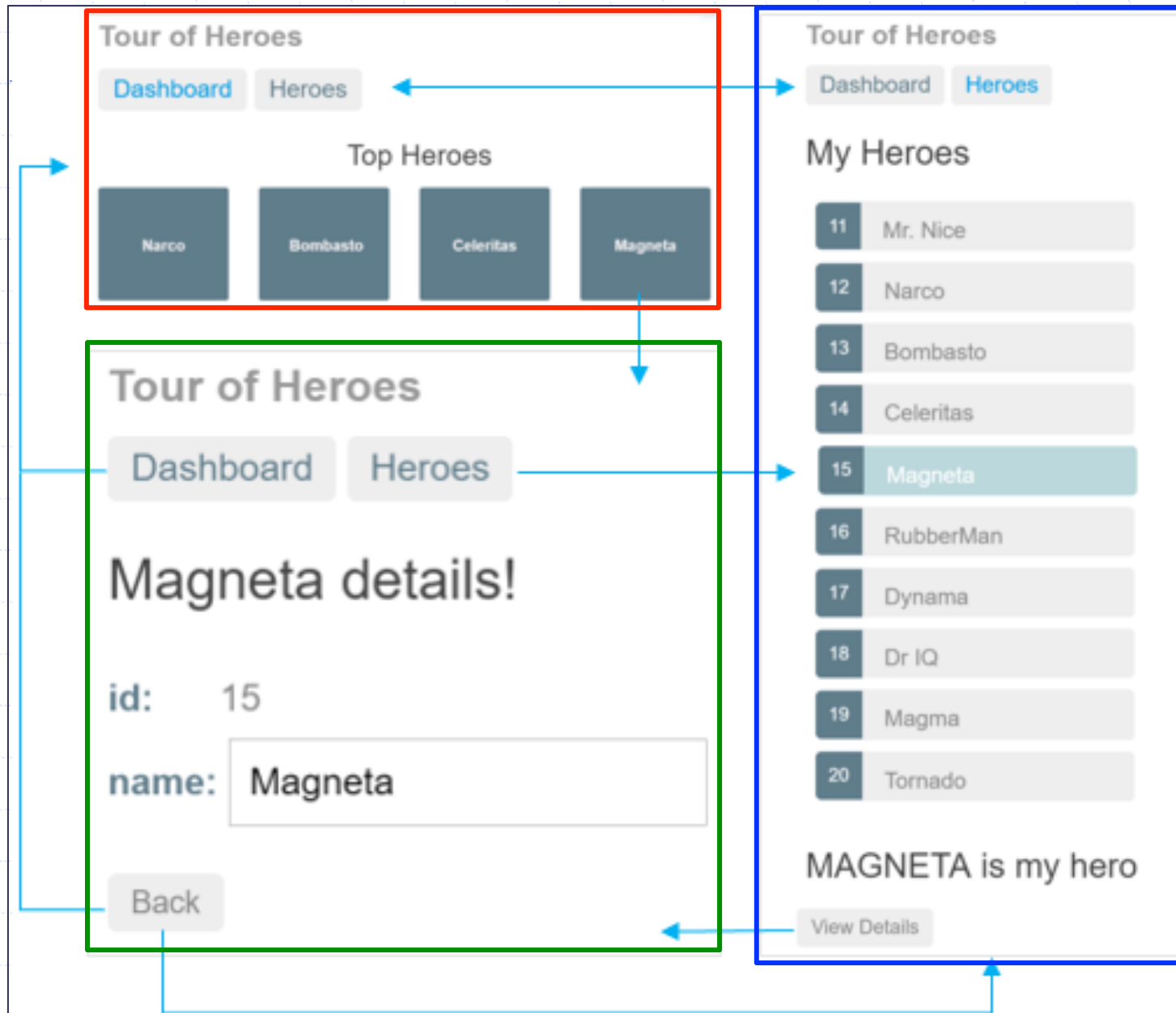
Dekoratori

- ◆ 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:
- ◆ Jedan koreni (*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 koreni 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

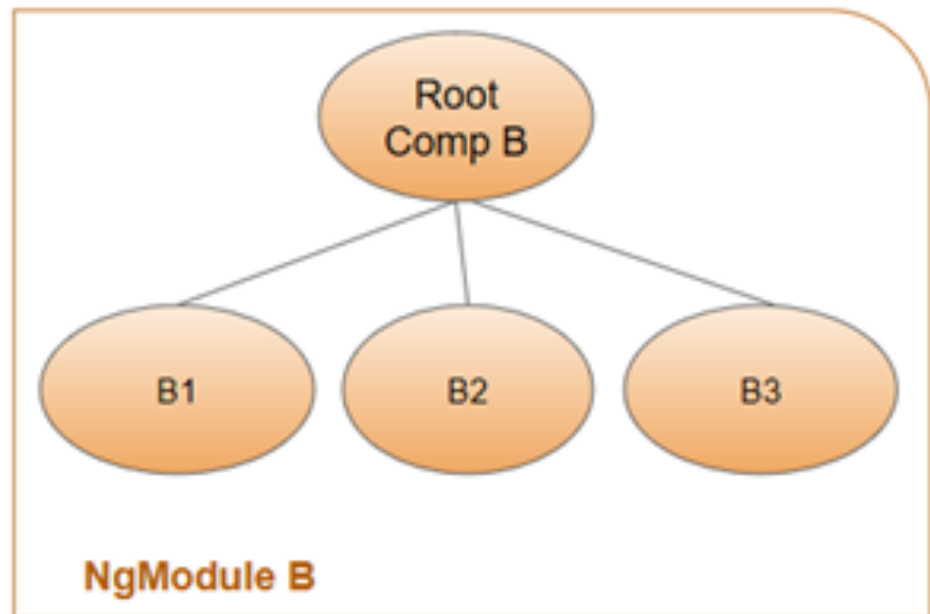
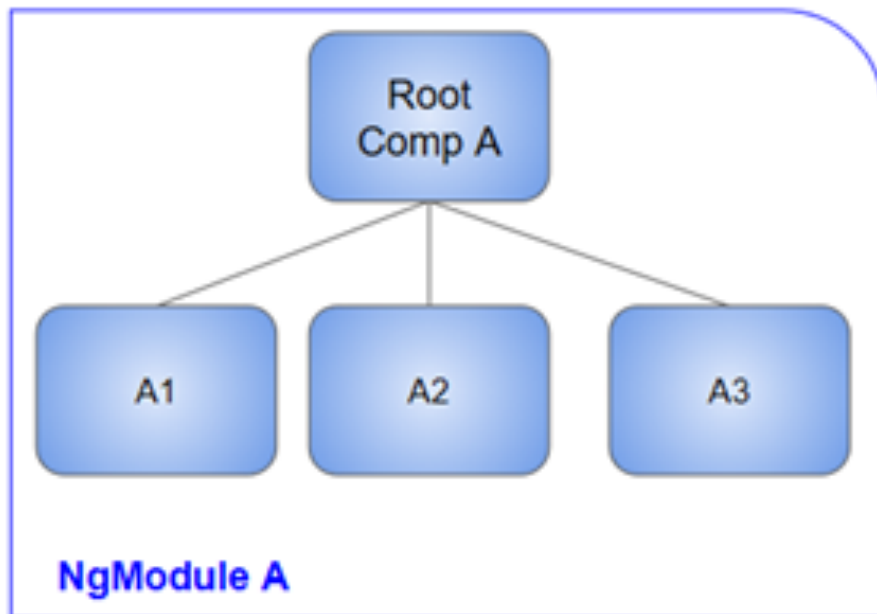
Rezultat:

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 korenu komponentu (*root*) koja predstavlja vrh hijerarhije stabla komponentenata
- ◆ Generiše se automatski prilikom kreiranja Angular aplikacije, tj. projekta

 app.component.css
 app.component.html
 app.component.ts

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 (templejt)

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 pronade 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';

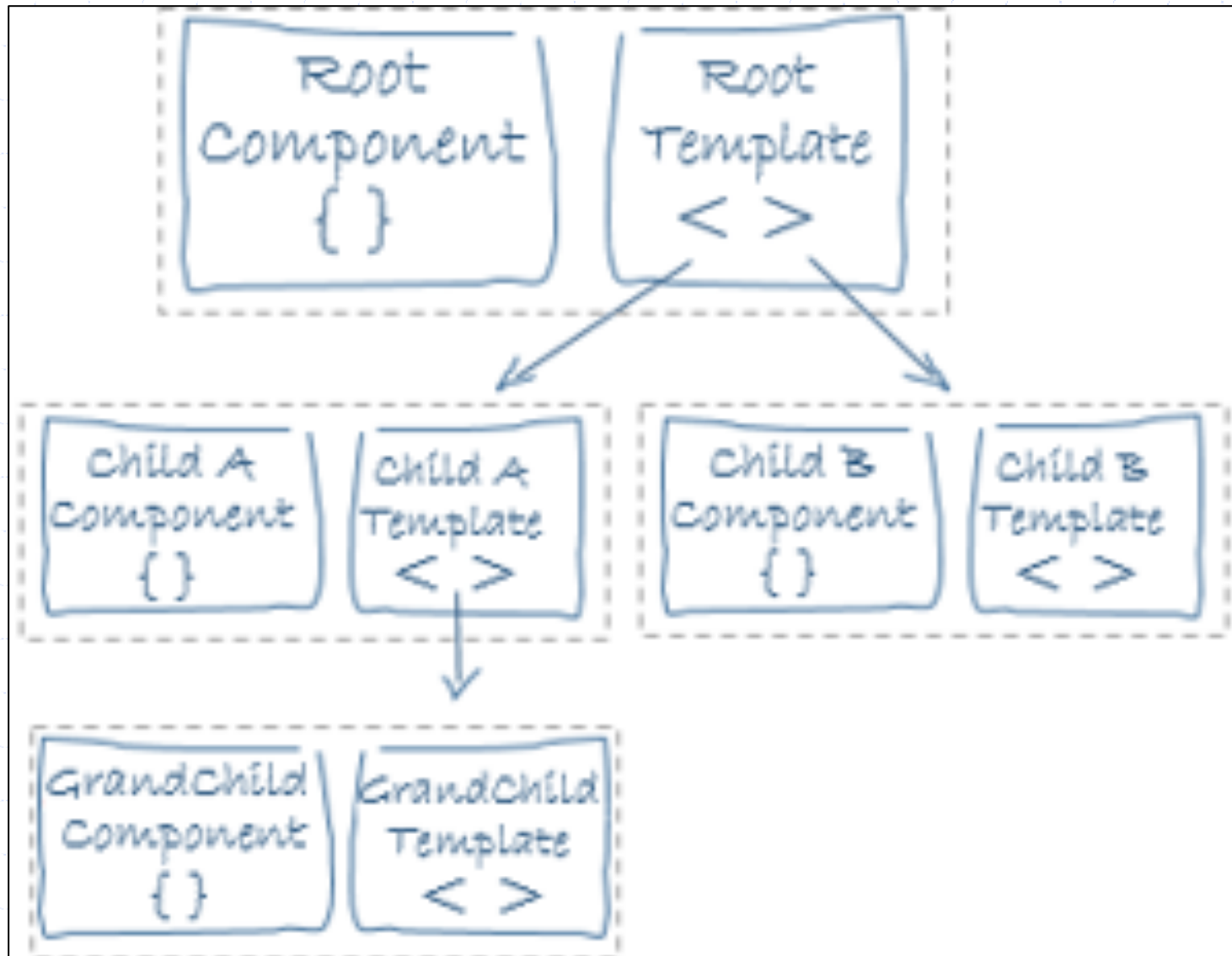
@Component({
  selector: 'app-heroes',
  templateUrl: './heroes.component.html',
  styleUrls: ['./heroes.component.css']
})
export class HeroesComponent implements OnInit {
  heroes: Hero[];

  constructor(private heroService: HeroService) { }

  ngOnInit() {
    this.getHeroes();
  }
}
```

A diagram consisting of a red box around the `ngOnInit()` method in the `HeroesComponent` class and a red bracket under the `implements OnInit` part of the class declaration. A red arrow points from the `ngOnInit()` box to the `implements OnInit` bracket, illustrating that the `ngOnInit()` method implements the `OnInit` interface.

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; }  
}
```

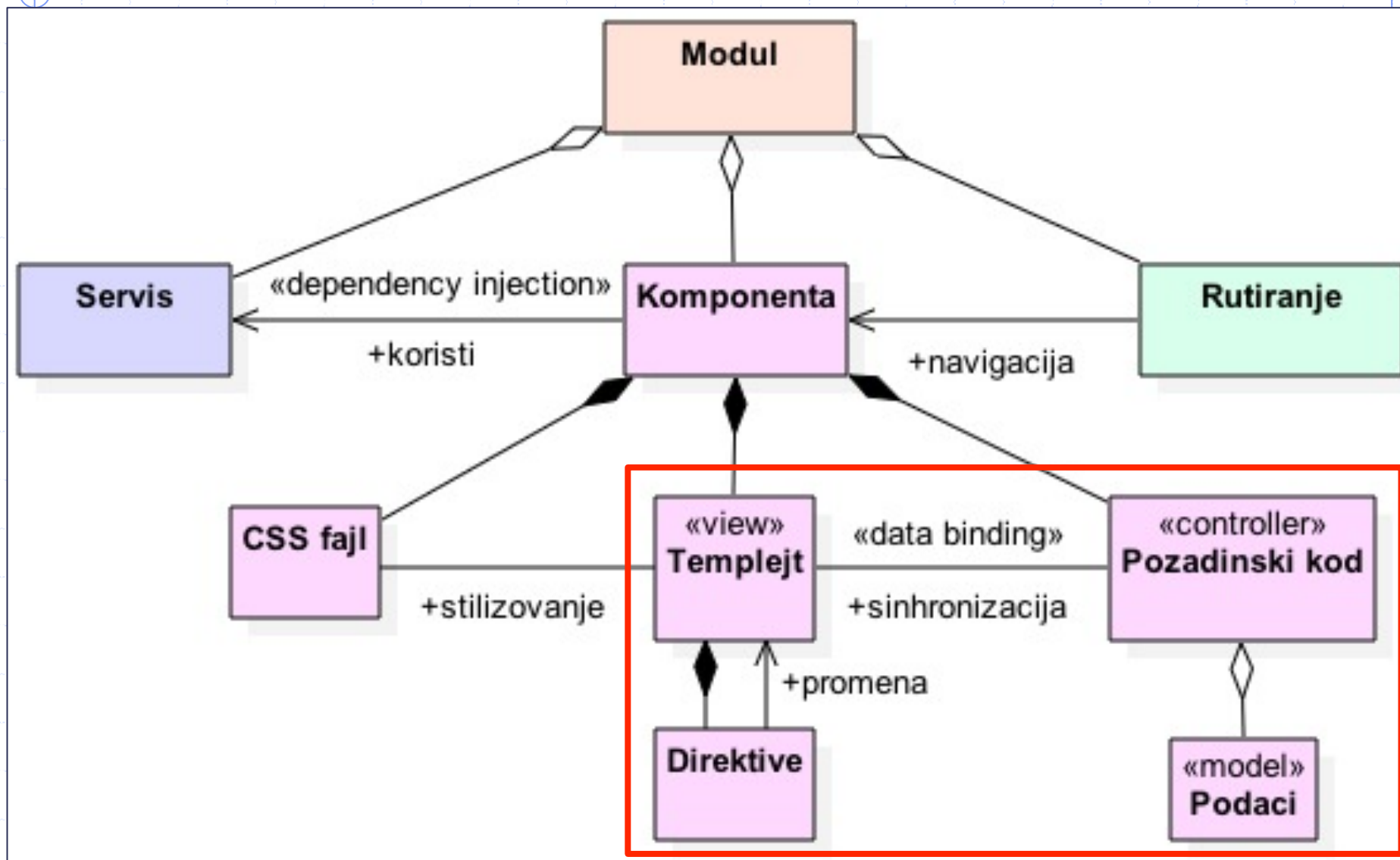
Primeri komponenti u praksi

- ◆ 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 ?
- ◆ Upotrebom **direktiva** koje realizuju programsku logiku u okviru tagova i **binding** mehanizama 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

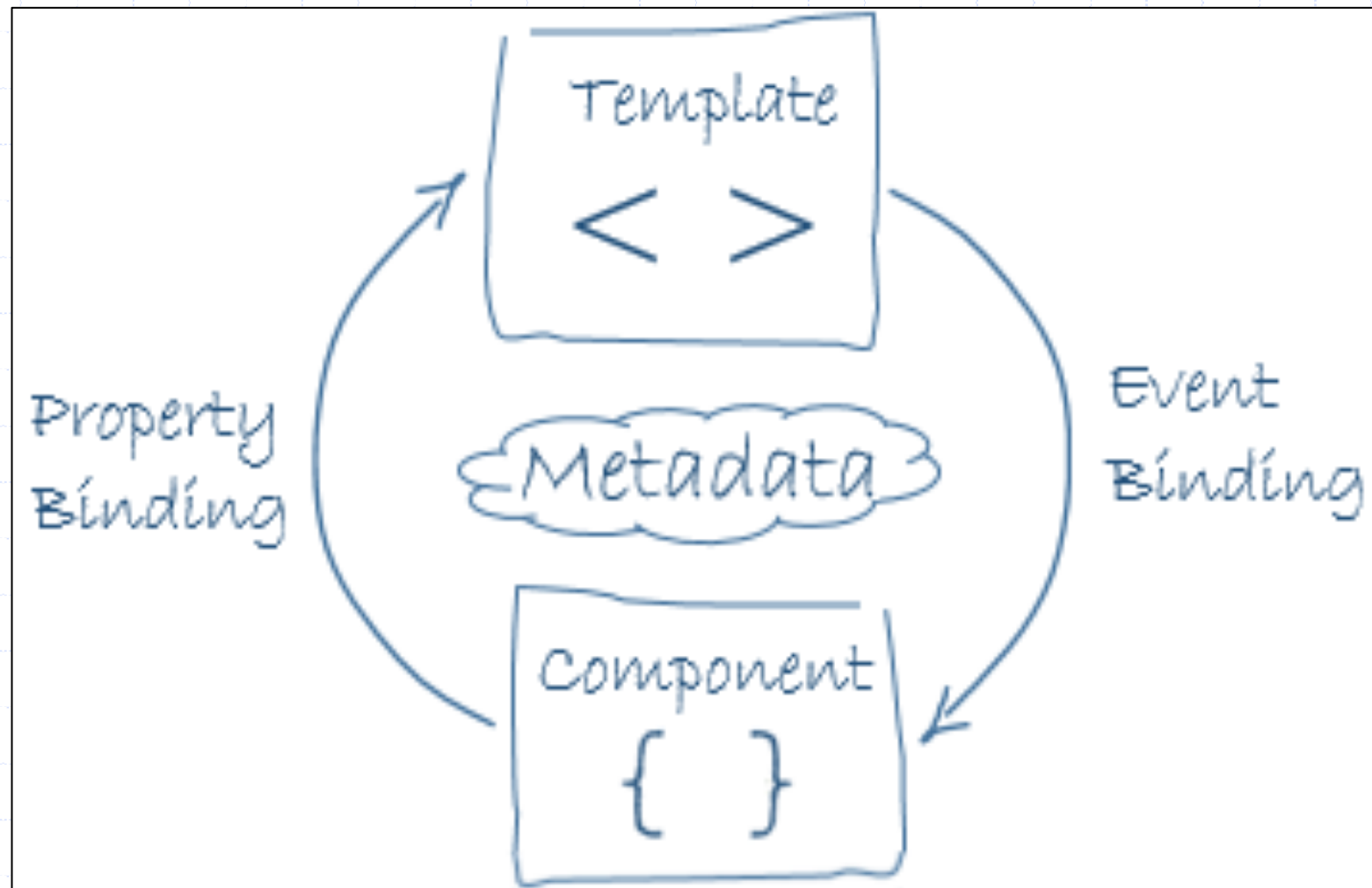
Logička šema gradivnih elemenata Angular-a



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"`

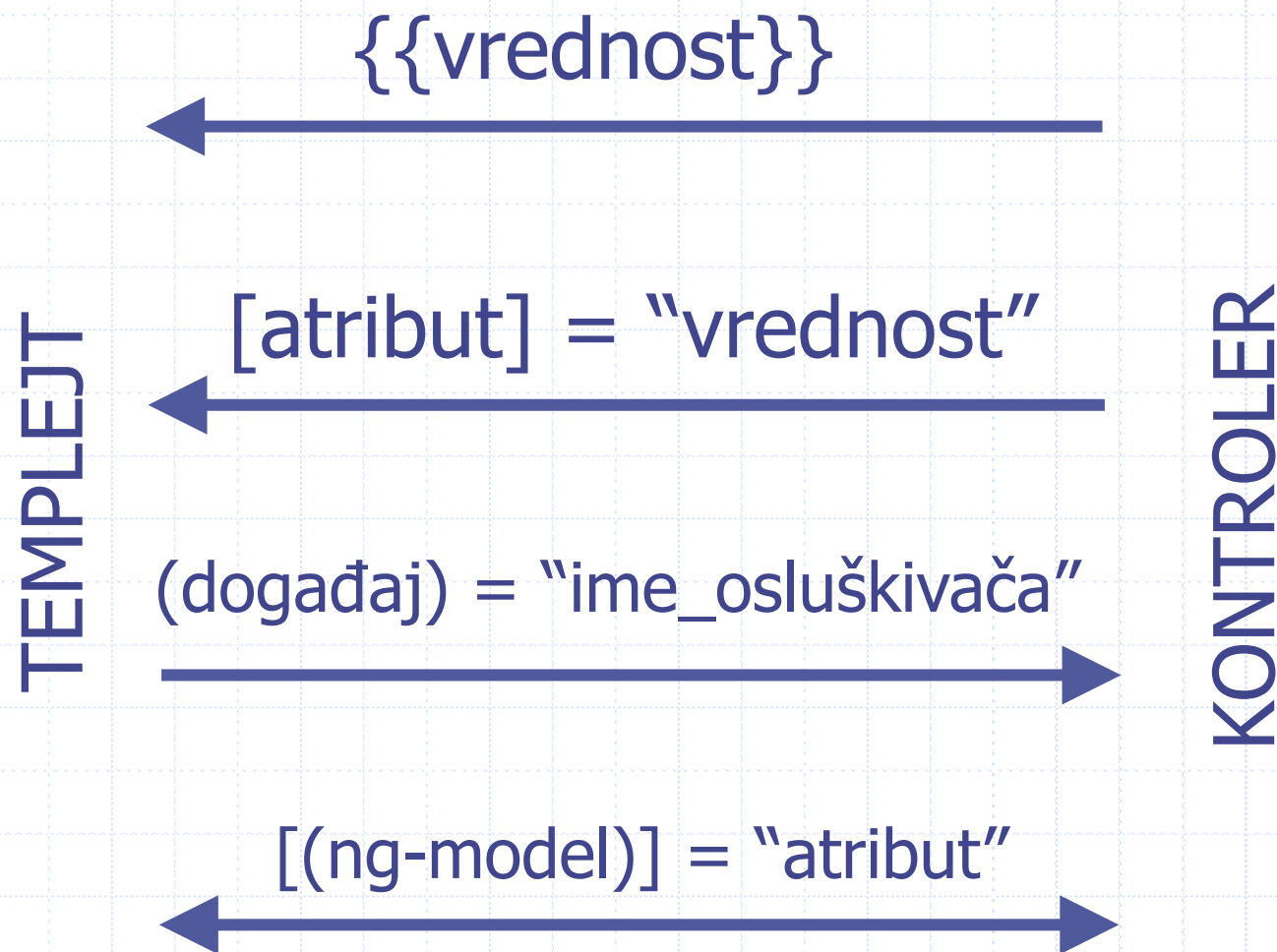
3. Događaj korisnika (*event*)

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



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 smeru 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 (zagrada ())

- ◆ Koriste se za **simboličke nazive 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 direktive (*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 modifikuje na osnovu specifičnih tipova naredbi koje se navodeu 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** direktive
 - Direktive [**atributa**]

Strukturne direktive

- ◆ Menjaju izgled komponente tako što dodaju, uklanjaju 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_tag </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 ispisaće se sadržaj div taga, tj. ime

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


*ngFor direktiva

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

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

Koristi se za ispisivanje kolekcija preko html lista, tj. *tag* je najčešće **

*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 templatejt

```
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 atributa (*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
- ◆ Prihvat 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*

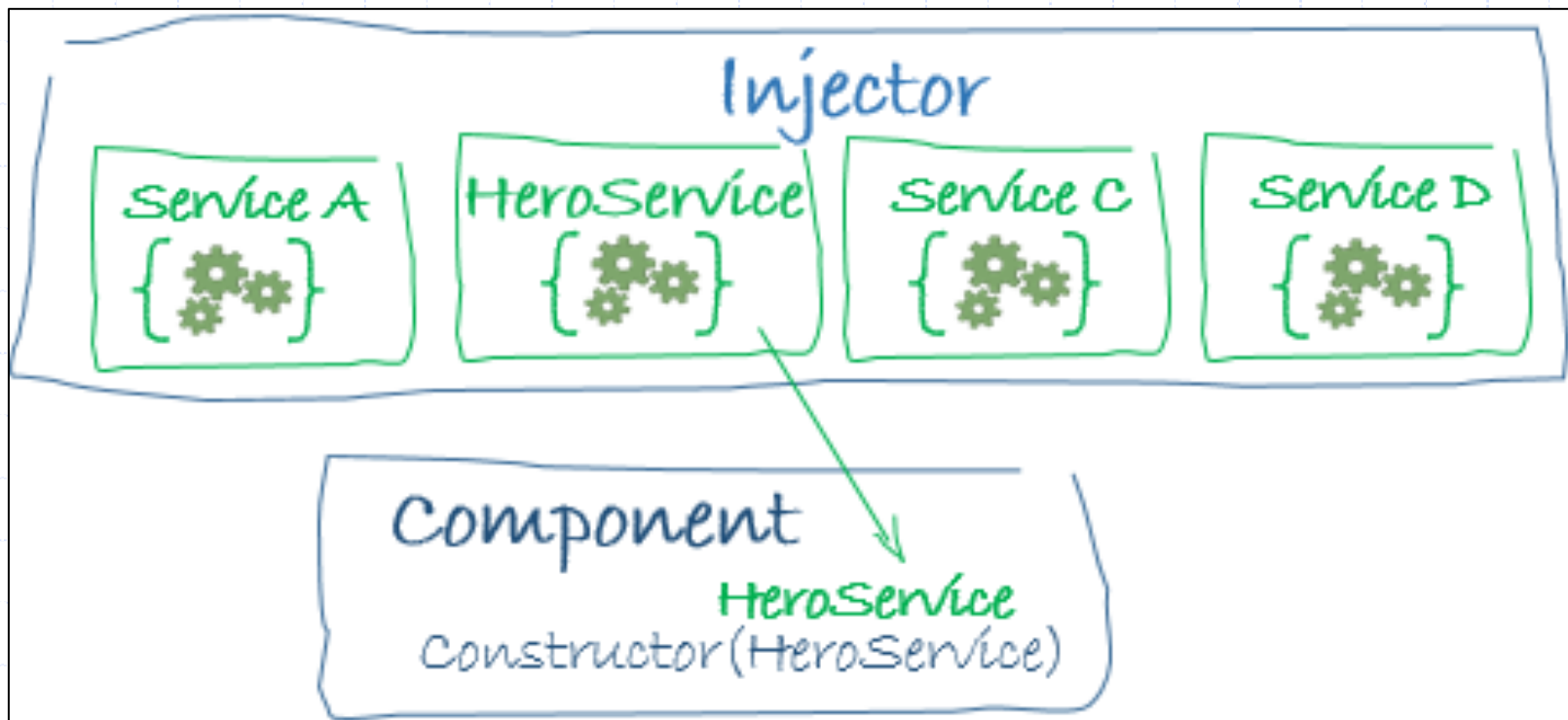
Dependency Injection (DI)

- ◆ 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 **konstruktor**a komponente (za komponentu koja koristi servis)
- ◆ Prilikom kreiranja instance komponente, Angular analizira parametre konstruktor a i proverava da li instance pronađenih servisa postoje
- ◆ Ukoliko ne postoje, one se kreiraju i prosleđuju kao argumenti konstruktor a
 - 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 !

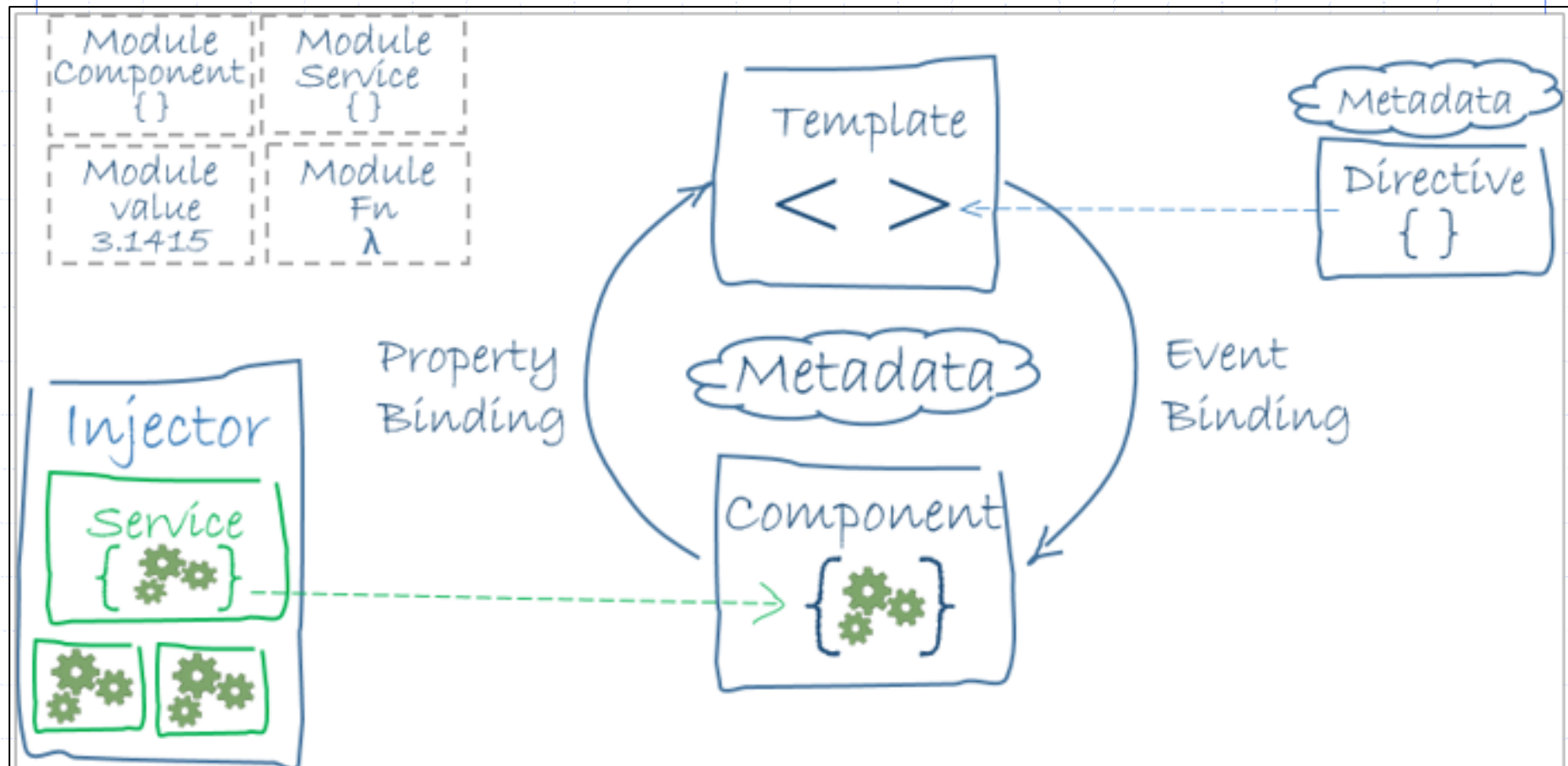
Načini registracije servisa ^{3/3}

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