

# 3. Komponente

---

# Sadržaj

---

- UI Angular aplikacija
- Projekcija sadržaja komponente
- Životni ciklus komponente
- Pristup komponentama

# Komponente u Angular aplikaciji

- Komponente čine UI Angular aplikacija
- UI ima strukturu stabla čiji je koren (*root* komponenta) u fajlu `index.html`
- *Root* komponenta sadrži direktno ili indirektno sve druge komponente

The screenshot displays a web application for managing taxpayers ('Obveznik'). The sidebar on the left contains navigation links: 'Finansije', 'Pregled', 'Ažuriranje', and 'Obveznik'. The main content area shows a table of taxpayers with the following data:

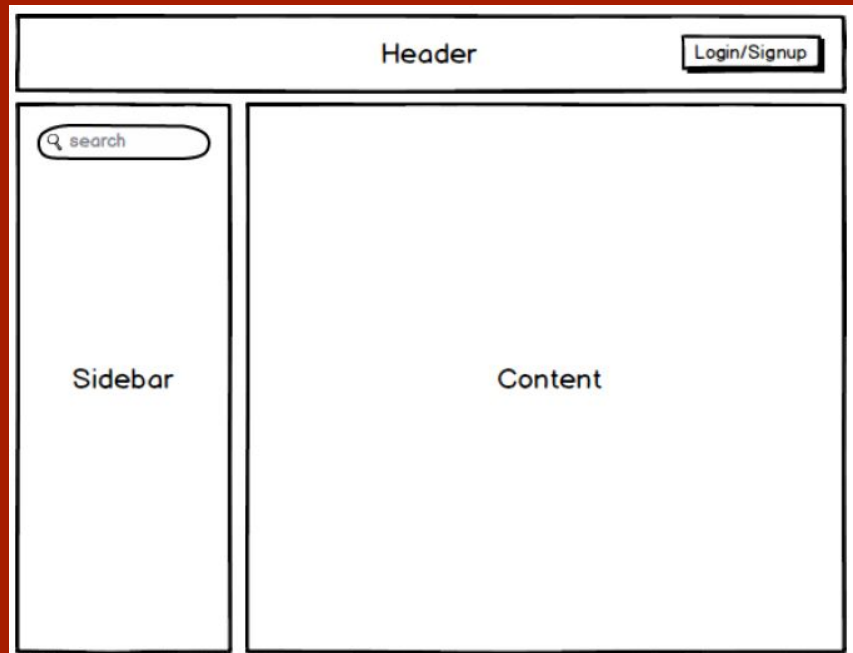
Id	Matični broj	AOP	Tip obveznika	naziv1	naziv2	naziv3	adresa1	adresa2	adresa3	Telefon	PIB	Reon	Pošta	Žiro račun	Napomena
996	9181792	90699	Fizičko lice - rezident	PPTU GLOBUS PEČ			V.RAIČEVIĆA 25	PEČ			2		2		
28335	2611919386505	24839	Fizičko lice - rezident	JAKOBUŠIĆ	NADA	PETAR	BIJELA GLOŽUN 36	HERCEG NOVI			2		2		pasiv - objekat - po komisiji nema o
28704	2703954175047	102625	Fizičko lice - nerezident	OBUČINA	DANICA		FRANCUSKA 35	BEOGRAD			2		2		
39468	6192000012466	113993	Fizičko lice - nerezident	LOBUSEVA	IRINA				RUSKA FEDERACIJA		2		2		

The interface also includes a search bar at the top, a toolbar with add, edit, and delete icons, and a pagination bar at the bottom showing 'Items per page: 10' and '1 - 4 of 4'.

# Arhitektura aplikacije

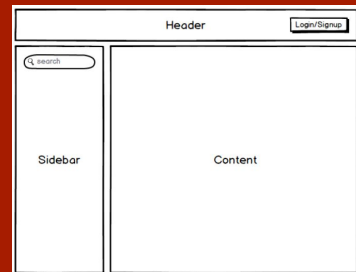
---

- Razložiti aplikaciju na komponente
- Definisati odgovornosti komponenti
- Definisati ulaze i izlaze komponenti



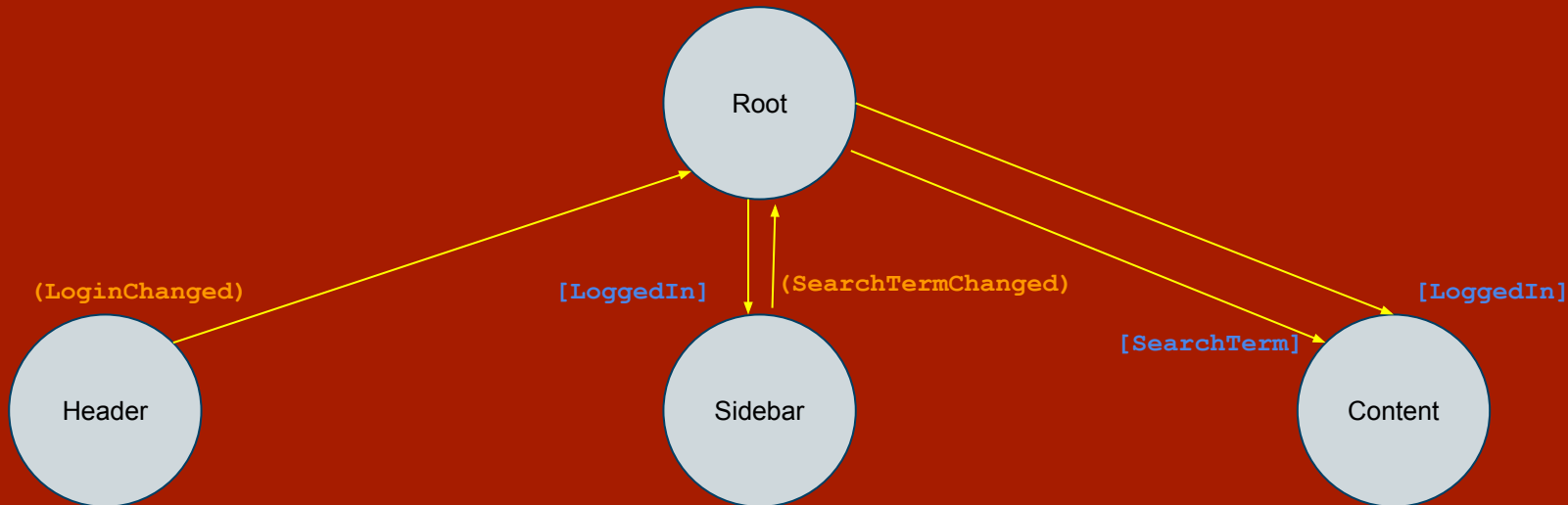
# Odgovornosti komponenti

- HeaderComponent
  - Odgovornosti: Omogućiti korisniku login i logout
  - Ulazi: Nema
  - Izlazi: Događaj `LoginChanged` se okine kada se korisnik prijavi ili odjavi
- SidebarComponent
  - Odgovornosti: Pokretanje pretrage
  - Ulazi: `LoginChanged` koji određuje da li će pretraga biti omogućena
  - Izlazi: Događaj `SearchTermChanged` se okine kada korisnik pokrene pretragu, `SearchTerm` sadrži izraz koji se pretražuje
- ContentComponent
  - Odgovornosti: Prikaz rezultata pretrage
  - Ulazi: `LoginChanged` koji menja prikaz sadržaja i `SearchTerm` koji se koristi za izvršenje pretrage
  - Izlazi: Nema



# Tok podataka

- Kada se komponente povežu preko *root* komponente, tok podataka između njih se može predstaviti na sledeći način:



# Uvezivanje komponenti

---

- Uvezivanje komponenti se dešava u HTML-u
- *Template root* komponente može izgledati ovako:

```
<app-header (loginChanged)="loggedIn=$event"></app-header>
<app-sidebar (searchTermChanged)="searchTerm=$event"></app-sidebar>
<app-content [searchTerm]="searchTerm"></app-content>
```
- *Root* sluša `loginChanged` koji emituje `header`
  - svoje obeležje `loggedIn` postavlja na vrednost koja se emituje pri događaju
- *Root* sluša `searchTermChanged` koji emituje `sidebar`
  - **svoje** obeležje `searchItem` postavlja na vrednost koja se emituje pri događaju
  - postavlja komponenti `content` **njeno** obeležje `searchTerm` na vrednost **svog** obeležja `searchTerm`

- ```
@Component({
  selector: 'app-puzzle-form',
  templateUrl: './puzzle-form.component.html'
})
...
```



# Stil komponente

---

- Komponenta može imati definisan sopstveni stil
- Potrebno je definisati obeležje `styles` i definisati niz stringova koji predstavljaju stilove

```
@Component({
  selector: 'app-puzzle-form',
  templateUrl: './puzzle-form.component.html',
  styles: [
    .mat-card {
      background-color: yellow;
    }
  ]
})...
```

# Enkapsulacija prikaza

---

- Iako smo u primeru sa prethodnog slajda boju pozadine za `.mat-card` promenili na žutu, samo se promenila boja pozadine kartice forme za dodavanje
- Očekivalo bi se da ako se promeni CSS klasa da se ta promena odrazi kroz celu aplikaciju
- Ono što se u ovom slučaju desilo naziva se enkapsulacija prikaza (engl. *view encapsulation*)

# Web Components i shadow DOM

---

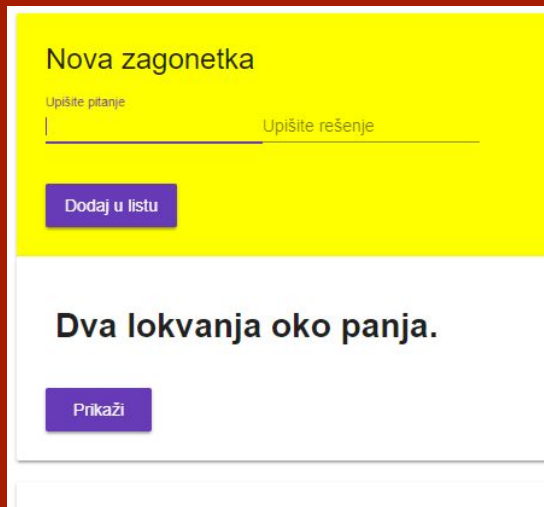
- Enkapsulacija prikaza je posledica postojanja *Web Components* specifikacije i jedne od njenih osnovnih mogućnosti koja se naziva *shadow DOM*
- *Shadow DOM* omogućava uključivanje stila u komponentu bez “curenja” van te komponente
- Angular pomenutu mogućnost obezbeđuje komponentama i možemo kontrolisati enkapsulaciju prikaza konfigurisanjem obeležja `encapsulation` na neku od vrednosti:
  - `ViewEncapsulation.Emulated`
  - `ViewEncapsulation.ShadowDom`
  - `ViewEncapsulation.None`

# ViewEncapsulation.Emulated

- Podrazumevana vrednost za obeležje `encapsulation`
- Stilovi koji se definišu za komponentu ne reflektuju se na druge komponente
- Komponenta i dalje nasleđuje stilove koji su globalno definisani

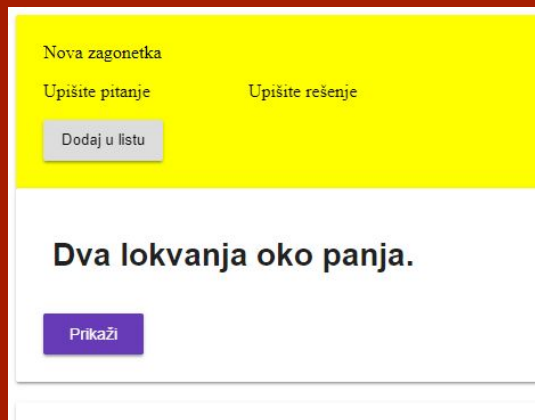
```
<style>.mat-card[_ngcontent-c1]{  
  background-color: yellow;  
}</style>  
▶<style>...</style>  
▶<style>...</style>  
▶<style>...</style>  
</head>  
▼<body>  
  ▼<app-root _ngghost-c0 ng-version="6.1.4">  
    ▼<puzzle-list _ngcontent-c0>  
      ▼<puzzle-form _ngghost-c1>  
        ▼<mat-card _ngcontent-c1 class="mat-card">
```

Developer pane, Elements tab



# ViewEncapsulation.ShadowDom

- Stilovi koji se definišu za komponentu ne reflektuju se na druge komponente
- Komponenta ne nasleđuje stilove koji su globalno definisani
- Ima smisla da se koristi kada se pravi *3rd party* komponenta koju želimo da neko drugi koristi u izolaciji



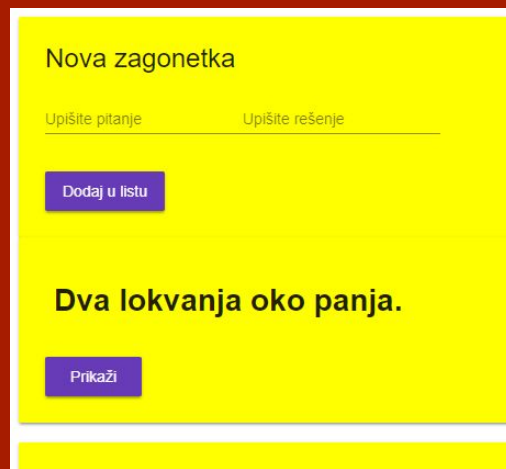
The screenshot shows a web application interface. The top section has a yellow background and contains the text "Nova zagonetka" followed by two input fields labeled "Upišite pitanje" and "Upišite rešenje". Below these is a button labeled "Dodaj u listu". The bottom section has a white background and contains the text "Dva lokvanja oko panja." followed by a button labeled "Prikaži".

```
@Component({
  selector: 'app-puzzle-form',
  templateUrl: './puzzle-form.component.html',
  styles: [`.mat-card {
    background-color: yellow;
  }`],
  encapsulation: ViewEncapsulation.ShadowDom
}) ...
```

# ViewEncapsulation.None

- Stil definisan za komponentu će u stranici biti deklarisan kao globalni
- Komponenta nasleđuje stilove koji su globalno definisani

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Lekcija1</title>
    <base href="/">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="icon" type="image/x-icon" href="favicon.ico">
    <style type="text/css">...</style>
    <style></style>
    ..
    <style>
      .mat-card {
        background-color: yellow;
      }</style> == $0
    <style>...</style>
    <style>...</style>
    <style>...</style>
  </head>
  <body>
    <app-root _ngghost-c0 ng-version="6.1.4">...</app-root>
```



# Obeležje styleUrls

---

- Kao i `templateUrl`, obeležje `styleUrls` omogućava da se CSS definiše eksterno u posebnim fajlovima
- Obeležje `styleUrls` referencira niz fajlova

```
@Component({  
  selector: 'app-puzzle-form',  
  templateUrl: './puzzle-form.component.html',  
  styleUrls: [  
    'puzzle-form.component.css'  
  ]  
})  
...
```

# Projekcija sadržaja

---

- Pretpostavimo da neko drugi želi da koristi našu `PuzzleComponent` ali na način da za prikaz obeležja `question` ne želi da koristi `<h1>` već recimo `<h3>`
- Trenutna implementacija `PuzzleComponent` ne omogućava takvu fleksibilnost
- Primena projekcije sadržaja (engl. *content projection*) omogućava prilagođavanje komponente pri korišćenju



# Realizacija projekcije sadržaja

- Dodavanjem `<ng-content></ng-content>` u *template* naše komponente (`puzzle`) omogućavamo nadređenoj komponenti (`puzzle-list`) da na to mesto umetne sadržaj naveden između selektora za našu komponentu

```
<mat-card>
  <mat-card-header>
    <ng-content></ng-content>
  </mat-card-header>
  ...

```

template komponente `puzzle`

```
...
<app-puzzle *ngFor="let p of puzzles" [puzzle] = "p">
  <mat-card-title><h3>{{ p.question }}</h3></mat-card-title>
</app-puzzle>
...

```

template komponente `puzzle-list`

# Nedostatak projekcije sadržaja

---

- Pri projektovanju sadržaja nadređena komponenta nema pristup obeležjima i metodama podređene komponente
- Prema tome, u sadržaju koji projektujemo ne možemo koristiti *input property binding* niti *output event binding*
- U našem primeru, u sadržaju koji projektujemo u kodu između `<app-puzzle>` tagova možemo pristupiti samo obeležjima i metodama `puzzle-list` komponente

# Faze životnog ciklusa komponente

---

- Komponenta tokom izvršenja aplikacije prolazi kroz različite faze životnog ciklusa
- Angular omogućava da u različitim fazama životnog ciklusa komponente kontrolišemo izvršavanje aplikacije pozivanjem predefinisanih metoda
- Ove metode se često nazivaju *callback* ili *hooks* metode

# Callback metode komponente

---

- `constructor`
  - poziva se kada Angular kreira komponentu ili direktivu pozivanjem `new` za klasu
- `ngOnChanges`
  - poziva se **svaki put** kada se desi izmena jednog od **input** obeležja komponente
- `ngOnInit`
  - poziva se kada se inicijalizuje komponenta; poziva se **samo jednom** nakon prvog poziva `ngOnChanges`
- `ngDoCheck`
  - poziva se kada je pozvan *change detector* komponente; omogućava implementaciju sopstvenog algoritma detekcije promene za komponentu
- `ngOnDestroy`
  - poziva se neposredno pre nego što Angular uništi komponentu

# Callback metode za podređene komponente

---

- Pozivaju se samo za komponente, ne i direktive
- `ngAfterContentInit`
  - poziva se nakon što Angular izvrši projektovanje sadržaja
- `ngAfterContentChecked`
  - poziva se svaki put nakon što se sadržaj komponente proveri od strane *change detector*-a
- `ngAfterViewInit`
  - poziva se kada se prikaz komponente potpuno inicijalizuje
- `ngAfterViewChecked`
  - poziva se svaki put kada se prikaz komponente proveri od strane *change detector*-a

# Dodavanje *callback* metoda

- Da bismo videli kada se izvršavaju *callback* metode, dodaćemo svaku od njih u klasu `PuzzleComponent` i u implementaciji ispisati odgovarajuću poruku na konzolu

```
...
export class PuzzleItemComponent {
  @Input() puzzle: Puzzle;
  constructor() {
    console.log('new - data = ${this.puzzle}');
  }
  ngOnChanges() {
    console.log('ngOnChanges - data = ${this.puzzle}');
  }
  ...
}
```

# Provera pozivanja *callback* metoda (1)

- Uklonićemo formu za dodavanje nove zagonetke
- Dodaćemo dva dugmeta u `PuzzleListComponent`
  - jedno kreira novu *hardcoded* zagonetku
  - drugo prazni kompletnu listu zagonetki

template: `

```
<!-- <app-puzzle-form (puzzleCreated)="addPuzzle($event)"></app-puzzle-form> -->
<div class="button-row">
  <button mat-raised-button color="primary" (click)="addHardcodedPuzzle()">Dodaj u listu</button>
  <button mat-raised-button color="primary" (click)="clearPuzzles()">Isprazni listu</button>
</div>
...
`
```

# Provera pozivanja *callback* metoda (2)

- Prilikom dodavanja ispisuju se poruke koje ukazuju na hronologiju faza životnog ciklusa i vreme inicijalizacija podataka
- Možemo da vidimo da je obeležje `puzzle` u konstruktoru `undefined`
- U `ngOnChanges` obeležje `puzzle` je inicijalizovano
- Najpogodnija metoda za inicijalizaciju komponente je `ngOnInit`
  - izvršava se jednom dok se `ngOnChanges` izvršava pri svakoj promeni *input* obeležja
- Prilikom brisanja poziva se `ngOnDestroy`

```
new - data = undefined
ngOnChanges - data = [object Object]
ngOnInit - data = [object Object]
ngDoCheck
ngAfterContentInit
ngAfterContentChecked
ngAfterViewInit
ngAfterViewChecked
```



# Detekcija promena i SimpleChange

---

- Metoda `ngOnChanges` prihvata parametar `changes` koji je mapa gde je ključ naziv obeležja a vrednost `SimpleChange`

```
class SimpleChange {  
    constructor(previousValue: any, currentValue: any)  
    previousValue: any  
    currentValue: any  
    firstChange: boolean  
    isFirstChange(): boolean  
}
```

# Prolazak kroz mapu promena

- U metodi `ngOnChanges` možemo proći kroz mapu `changes` i proveriti promene vrednosti obeležja

```
ngOnChanges(changes: SimpleChanges) {  
  console.log(`ngOnChanges - data = ${this.puzzle}`);  
  for (let key in changes) {  
    console.log(`${key} promenjen.  
                Trenutno: ${changes[key].currentValue}  
                Prethodno: ${changes[key].previousValue}`);  
  }  
}
```

# Interfejsi

---

- Proveru TypeScript koda pre izvršenja možemo koristiti uz interfejse kako bi bili sigurni da smo ispravno deklarirali *callback* metode
- Na raspolaganju su nam interfejsi
  - OnChanges
  - OnInit
  - DoCheck
  - AfterContentInit
  - AfterContentChecked
  - AfterViewInit
  - AfterViewChecked
  - OnDestroy

# Referenciranje prikaza *child* komponente

---

- Nekad je potrebno referencirati prikaz podređene komponente iz nadređene komponente
- Tada je u nadređenoj komponenti potrebno deklarirati obeležje tipa podređene komponente uz navođenje dekoratora `@ViewChild(<tipKomponente>)`
- Nakon toga je u kodu nadređene komponente moguće koristiti deklarisanu referencu za pristup podređenoj komponenti
- Ukoliko nadređena komponenta sadrži više od jedne podređene komponente navedenog tipa, referenciraće se samo prva podređena komponenta

# Primer @ViewChild

---

```
export class PuzzleListComponent {  
  puzzles: Puzzle[] = [new Puzzle("Dva lokvanja oko panja.", "Glava i uši."),  
                        new Puzzle("Bele koke ispod strehe vire.", "Zubi.")];  
  
  @ViewChild(PuzzleComponent) puzzleViewChild : PuzzleComponent;  
  
  constructor() {  
    console.log(`new - puzzleViewChild is ${this.puzzleViewChild}`);  
  }  
  
  ngAfterViewInit() {  
    console.log(`ngAfterViewInit - puzzleViewChild is ${this.puzzleViewChild}`);  
  }  
  ...  
}
```

# Dekorator @ViewChildren

```
export class PuzzleListComponent {  
    puzzles: Puzzle[] = [new Puzzle("Dva lokvanja oko panja.", "Glava i uši."),  
        new Puzzle("Bele koke ispod strehe vire.", "Zubi.")];  
  
    @ViewChildren(PuzzleComponent) puzzleViewChildren: QueryList<PuzzleComponent>;  
  
    constructor() {  
        console.log(`new - puzzleViewChildren is ${this.puzzleViewChildren}`);  
    }  
  
    ngAfterViewInit() {  
        console.log(`ngAfterViewInit puzzleViewChildren is ${this.puzzleViewChildren}`);  
        let puzzlesFromView: PuzzleComponent[] = this.puzzleViewChildren.toArray();  
        console.log(puzzlesFromView);  
    }  
}
```

# Referenciranje lokalne promenljive

- Ako se kao parametar u `@ViewChild` prosledi string koji odgovara nazivu lokalne promenljive u *template*-u, dobićemo referencu na tu promenljivu

```
@Component({
  selector: 'app-puzzle-list',
  template: `
    <h1 #header>Zagonetke</h1>
    ...
  `
})
export class PuzzleListComponent {
  ...
  @ViewChild('header') headerEl: ElementRef;
  ...
  ngAfterViewInit() {
    this.headerEl.nativeElement.textContent = 'Zaglavlje liste (header)';
  }
}
```

## Zaglavlje liste (header)

Dodaj u listu

Isprazni listu

Dva lokvanja oko panja.

Prikaži

Bele koke ispod strehe vire.

# Referenciranje projektovanog sadržaja

- Analogno dekoratorima `@ViewChild` i `@ViewChildren` postoje i dekoratori `@ContentChild` i `@ContentChildren`

```
@Component({  
  selector: 'app-puzzle-list',  
  template: `...  
    <h1>Content zagonetka</h1>  
    <ng-content></ng-content>  
  })
```

Iz root komponente  
projektovan sadržaj  
puzzle u puzzle-list

```
export class PuzzleListComponent {
```

```
  ...
```

```
  @ContentChild(PuzzleComponent) puzzleContentChild: PuzzleComponent;
```

```
  ngAfterContentInit() {
```

```
    console.log(`ngAfterContentInit - puzzleContentChild = ${this.puzzleContentChild.puzzle.question}`)
```

```
  }...
```

```
new - data = undefined  
ngOnChanges - data = [object Object]  
ngOnInit - data = [object Object]  
ngDoCheck  
ngAfterContentInit  
ngAfterContentChecked  
ngAfterContentInit - puzzleContentChild = content pitanje
```



# Rezime

---

- UI Angular aplikacije čine komponente organizovane u strukturu stabla
- *Template* komponente određuje prikaz komponente
- Enkapsulacija prikaza omogućava različite načine primene stilova
- *Callback* metode omogućavaju upravljanje izvršenjem aplikacije u različitim fazama životnog ciklus komponente
- Dekoratori `@ViewChild` i `@ViewChildren` omogućavaju pristup podređenim komponentama
- Dekoratori `@ContentChild` i `@ContentChildren` omogućavaju pristup komponenti čiji je sadržaj projektovan