

# Component Architecture

Лектор: Петър Маламов

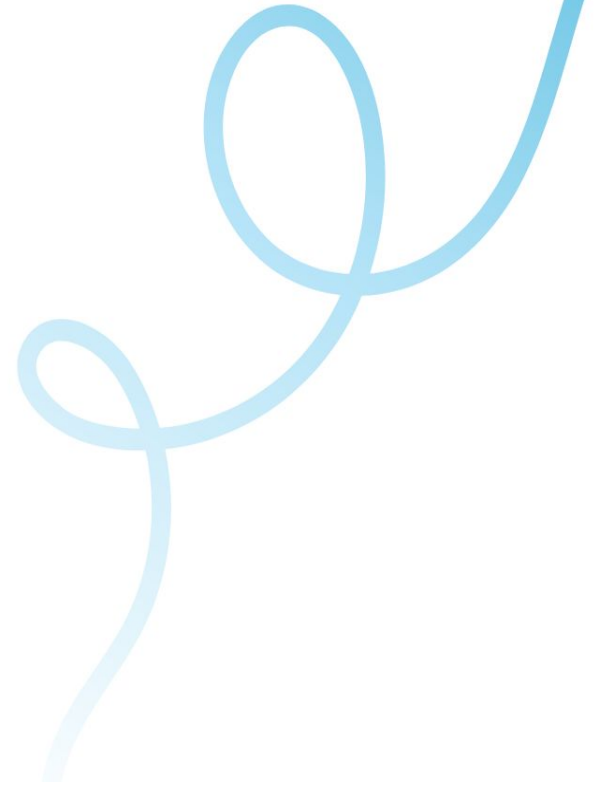
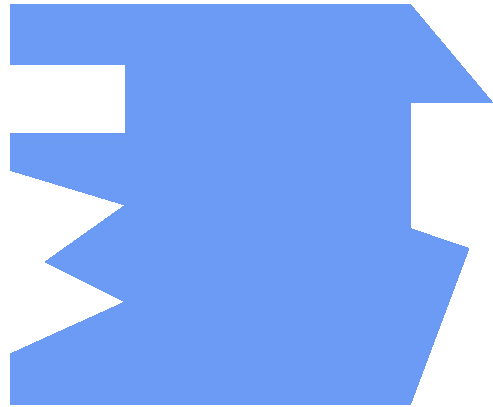
# Component-based architecture (CBA) ?

- Какво представлява ?
- Какви са ползите ?
- Какви са негативите ?
- Как да я използваме в Angular ?

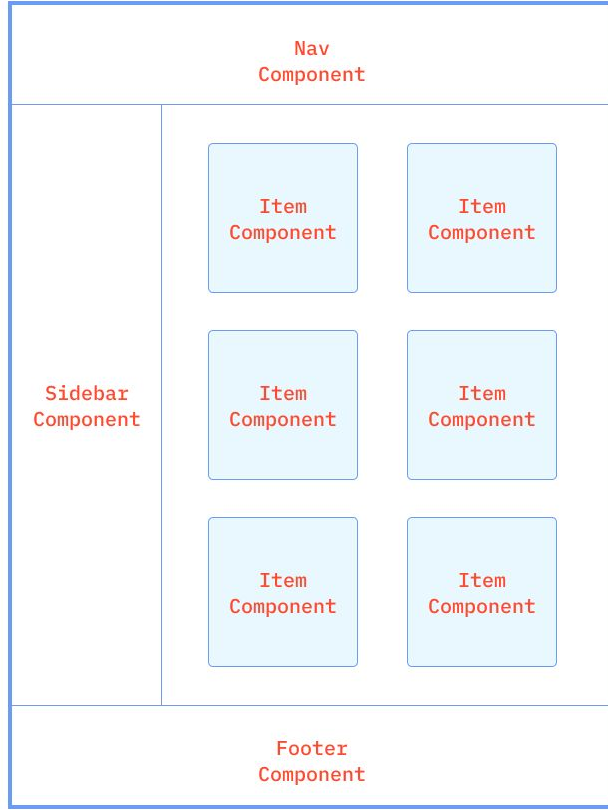


# Какво представлява СВА ?

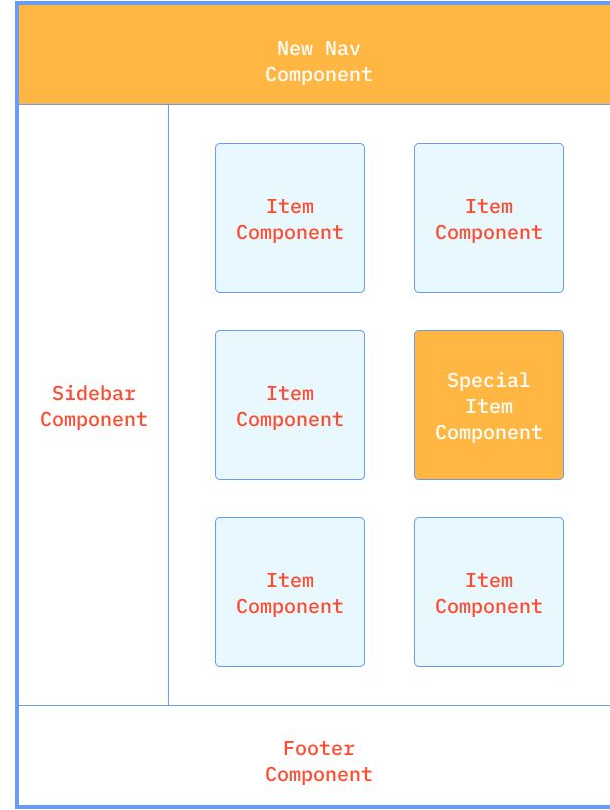
- Парадигма за проектиране на софтуер.
- Основава се на разбиването на големи и сложни системи в малки, самостоятелни и взаимозаменяеми компоненти.
- Набляга се на създаването на преизползваеми, независими софтуерни единици наречени компоненти



## App



## App



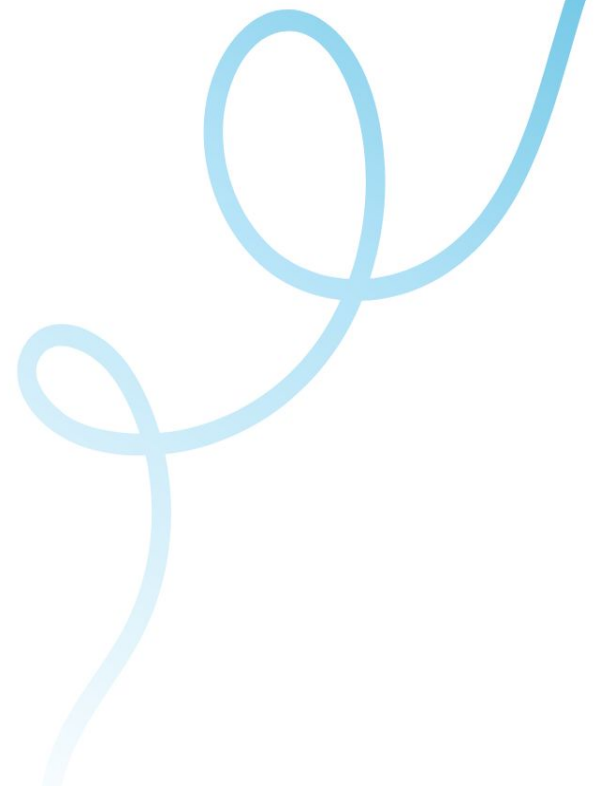
# Характеристики на СВА

- Енкапсулация
- Преизползваемост
- Взаимозаменяемост
- Composability
- Тестваемост



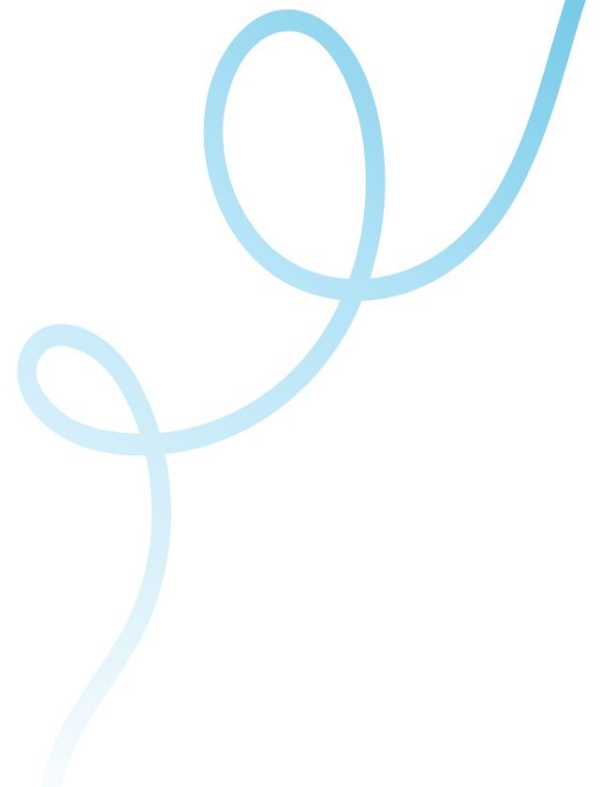
# Ползи от СВА

- Повишава преизползваемост
- Подобрява поддръжката
- Подобрява скалируемостта
- Позволява паралелна работа
- Гъвкавост



# Негативи от СВА

- Повишена сложност
- Performance Overhead
- Learning Curve
- Prop drilling





# Добри практики при създаването на компоненти

- Баланс при създаването на компоненти
- По-малко предварителна оптимизация
- По-малко зависимости към други компоненти

# Angular components

# Angular components

- Какво е компонент ?
- Жизнен цикъл на компонента



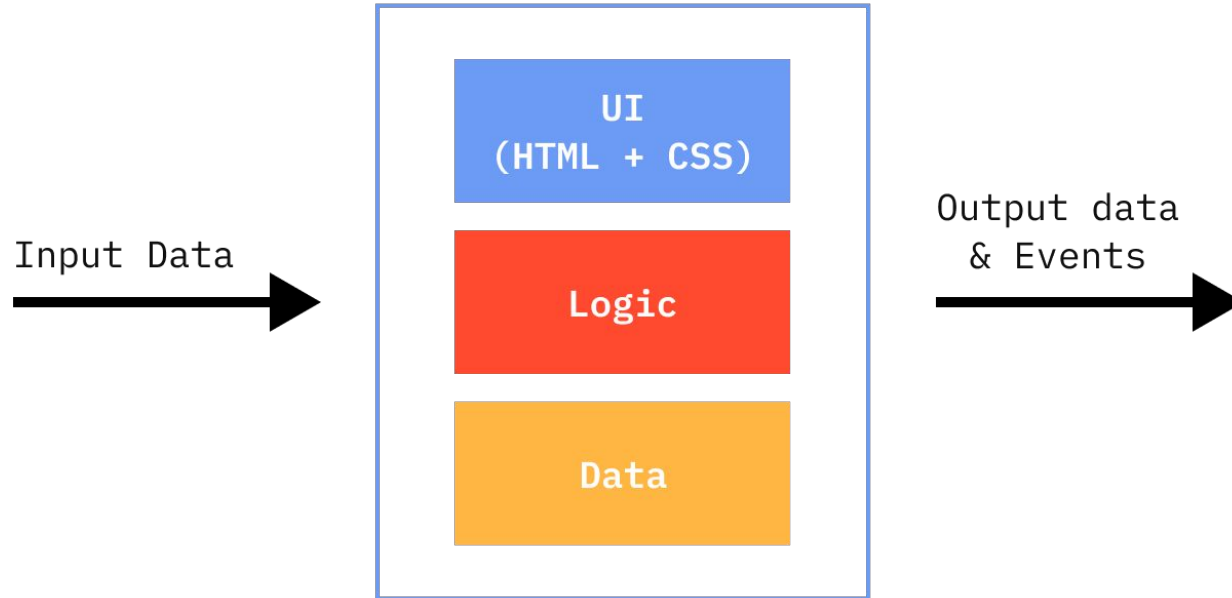
# Какво е компонент ?

- Градивен елемент за Angular приложение.
- Енкапсулира темплейта, данните и поведението

```
@Component({  
  selector: 'my-component',  
  styleUrls: ['./my-component.component.scss'],  
  templateUrl: './my-component.component.html',  
})  
export class MyComponent {}
```

```
<my-component></my-component>
```

## Component



# Жизнен цикъл на компонента

Стъпки, през които компонента минава от неговото създаване до неговото премахване от DOM-а.

В компонентите, Angular ни предоставя възможност да имплементираме **Lifecycle hooks**. Те представляват функции, чрез които можем да се “закачим” за различни стъпки от жизнения цикъл на даден компонент.



## Creation

constructor

## Change

ngOnInit

ngOnChanges

ngDoCheck

ngAfterContentInit

ngAfterContentChecked

ngAfterViewInit

ngAfterViewChecked

## Rendering

afterNextRender

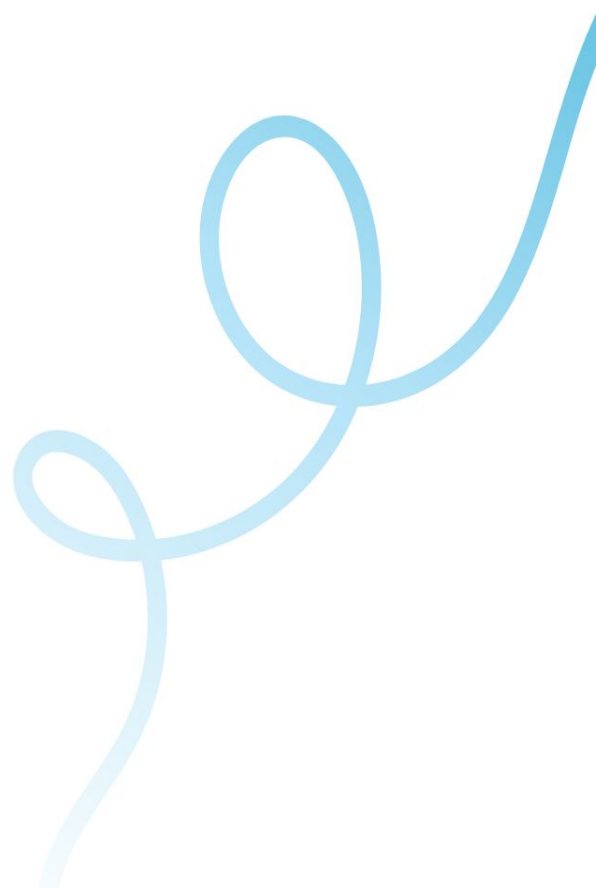
afterRender

## Destruction

ngOnDestroy

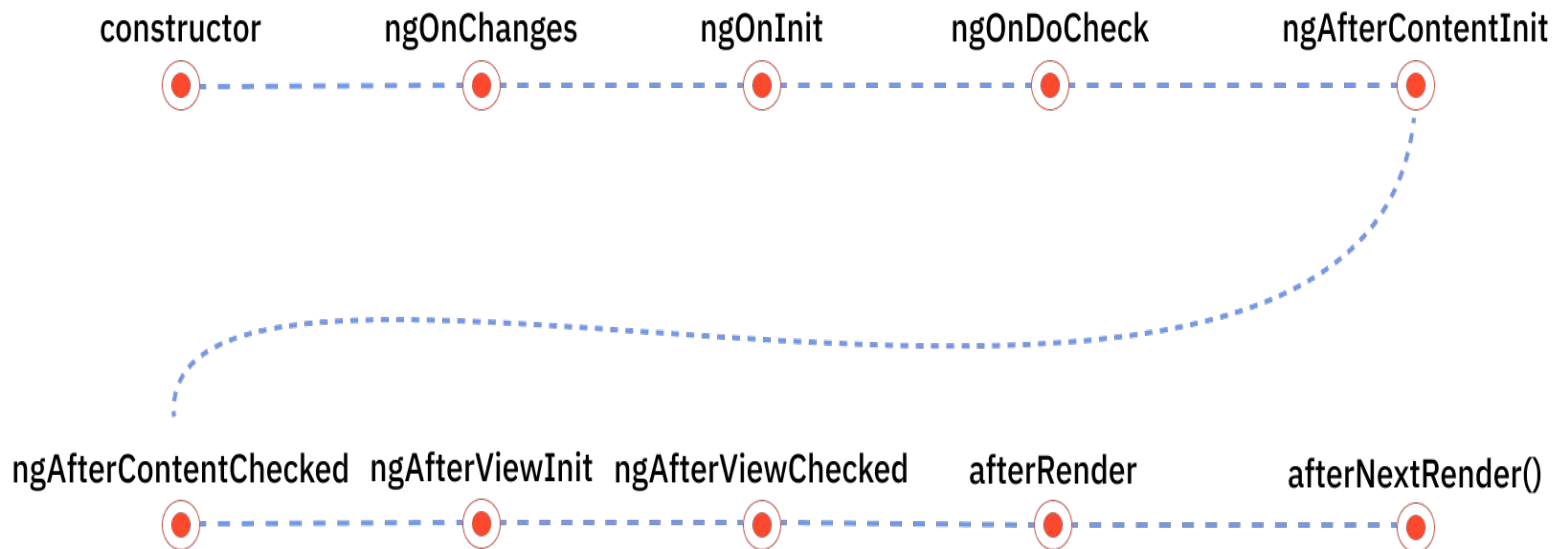
# Състояния на компонента

- При създаване
- При промени
- При унищожаване/ премахване от DOM-а

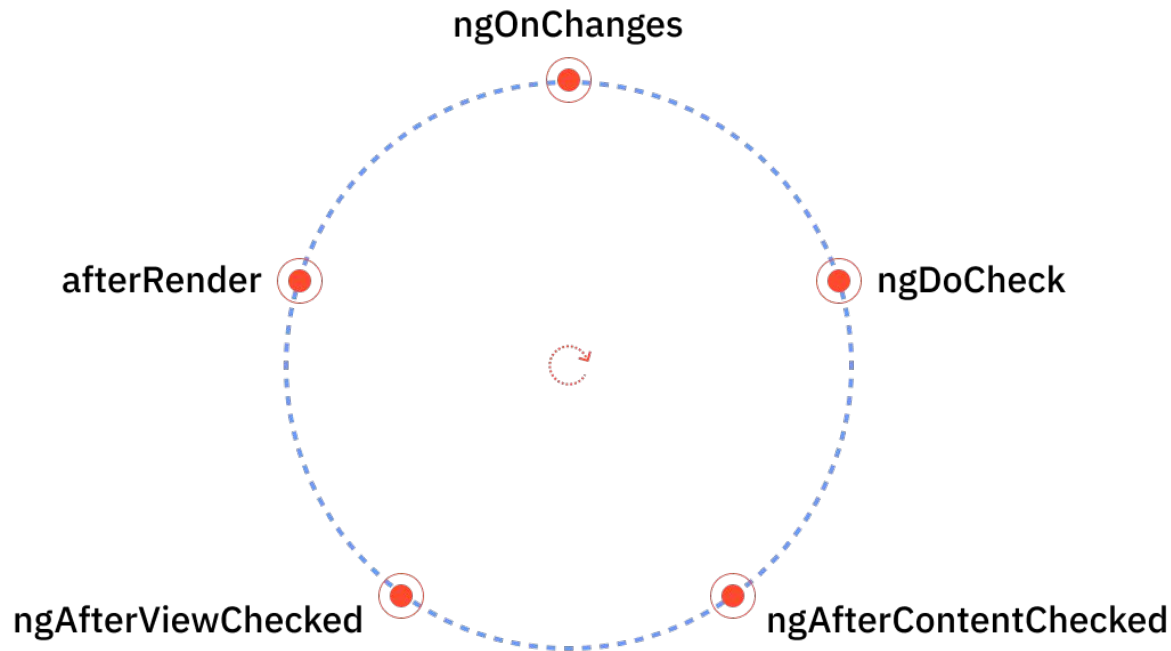




# При създаване на компонента

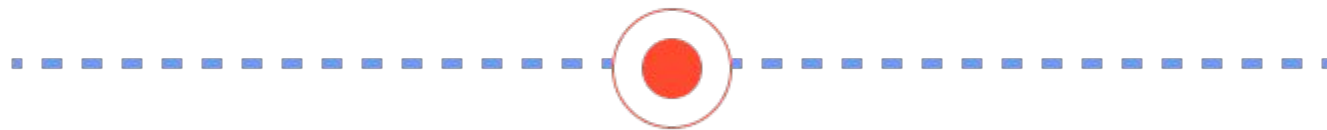


# При промени на компонента



# При унищожаване на компонента

ngOnDestroy



Remove  
component  
from DOM

# Standalone components

# Standalone components

- Какво представляват ?
- Каква е разликата с предишните компоненти ?
- Предимства
- Недостатъци
- Упражнение.



# Какво представляват Standalone component ?

Нов тип компоненти, които нямат нужда да се декларират в модул.

Могат директно да бъдат използвани в друг компонент или модул.

You, 20 hours ago | 1 author (You) | Codiumate: Options | Test this class

```
@Component({
  selector: 'app-catalog-table',
  templateUrl: './table.component.html',
  styles: [],
})
export class TableComponent {
  @Input({ required: true }) classesData!: IClass[];
}
```

You, 20 hours ago | 1 author (You) | Codiumate: Options | Test this class

```
@NgModule({
  declarations: [FilterComponent, TableComponent, CatalogComponent],
  exports: [CatalogComponent],
  imports: [CommonModule],
})
export class CatalogModule {}
```

You, 41 seconds ago | 1 author (You) | Codiumate: Options | Test this class

```
@Component({
  standalone: true,
  selector: 'app-catalog-table',
  templateUrl: './table.component.html',
  styles: [],
})
export class TableComponent {
  @Input({ required: true }) classesData!: IClass[];
}
```

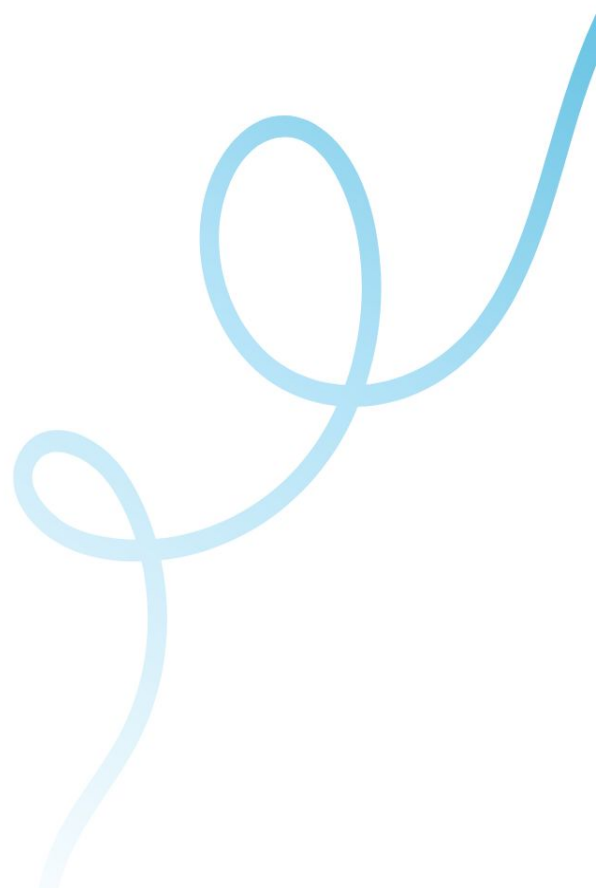
You, 54 seconds ago | 1 author (You) | Codiumate: Options | Test this class

```
@NgModule({
  declarations: [FilterComponent, CatalogComponent],
  exports: [CatalogComponent],
  imports: [CommonModule, TableComponent],
})
export class CatalogModule {}
```



# Предимства

- Подобрява developer experience
- Опростява архитектурата
- Подобрява преизползваемостта
- Улеснява lazy-loading



# Недостатъци

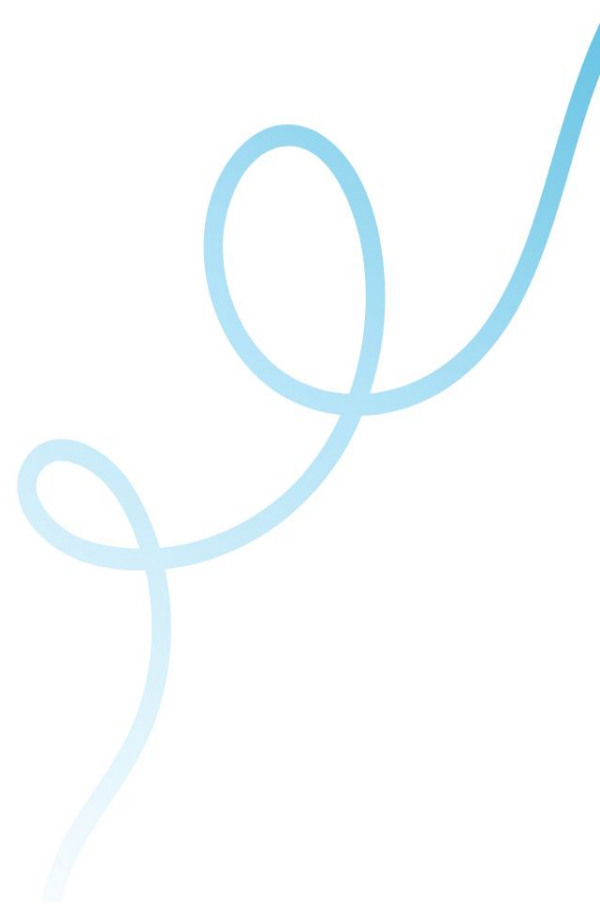
- Загуба на някой функционалности на модулите.
- Опасност от прекалено раздробяване на компоненти.

# Упражнение

# Dynamic component rendering

# Преговор

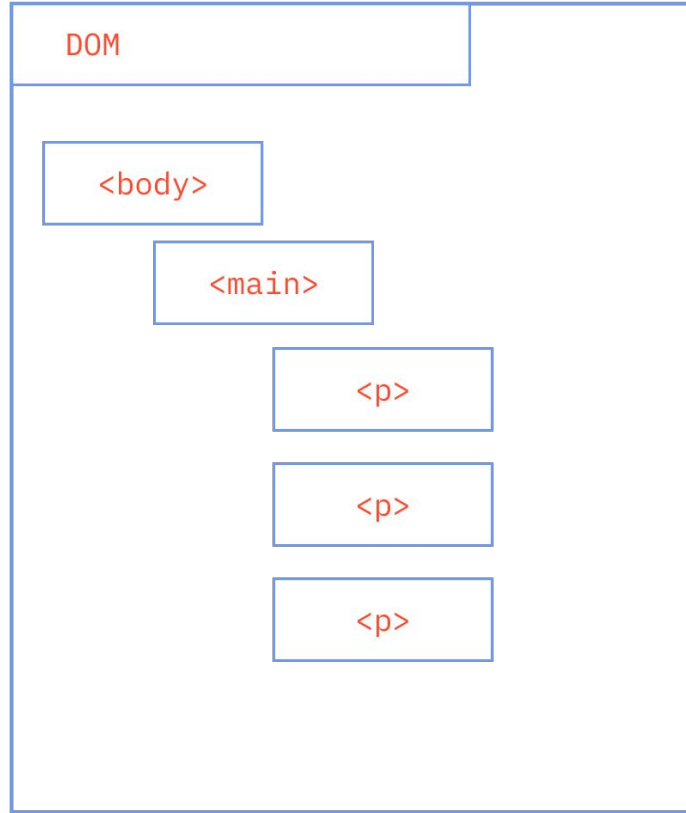
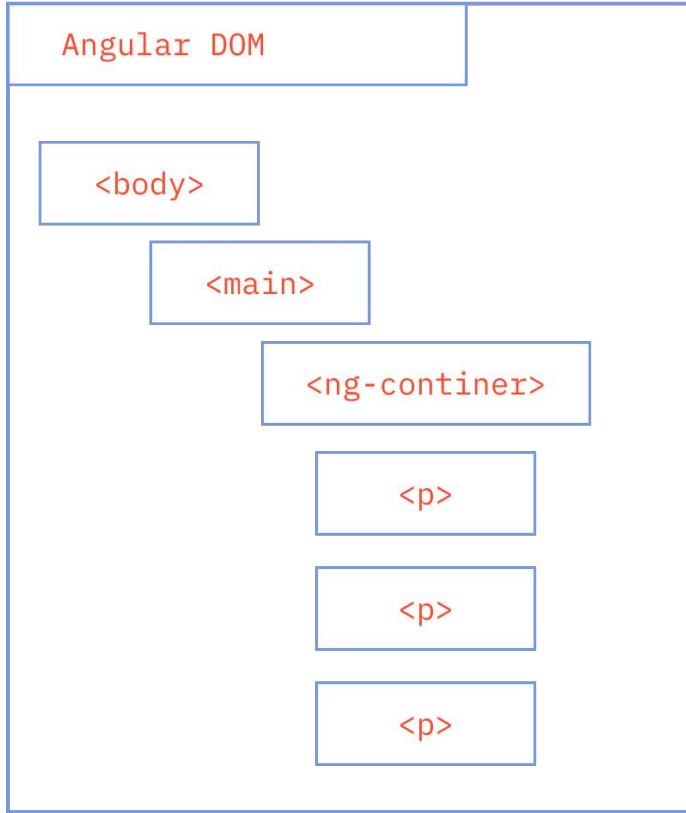
- ng-container
- ng-content
- ng-template



# ng-container

Елемент, който живее само в Angular и не добавя елемент в DOM-а.

Ползваме го най-често когато искаме да групираме елементи, без да добавяме обвиващ елемент в DOM-а или в комбинация с някоя структурна директива.



```
<div>
  <p>1</p>
  <div *ngIf="true">
    <p>2</p>
  </div>
  <p>3</p>
</div>
```

```
<app-list>
  <div>
    <p>1</p>
    <div == $0
      <p>2</p>
    </div>
    <!--bindings={
      "ng-reflect-ng-if": "true"
    }-->
    <p>3</p>
```



```
Go to component | You, 6 seconds ago | 1 author  
<div>  
  <p>1</p>  
  <ng-container *ngIf="true">  
    <p>2</p>  
  </ng-container>  
  <p>3</p>  
</div> You, 1 second ago • U
```

```
<app-list>  
  <div> == $0  
    <p>1</p>  
    <p>2</p>  
    <!--ng-container-->  
    <!--bindings={  
      "ng-reflect-ng-if": "true"  
}-->  
    <p>3</p>  
  </div>
```

# Нов Control flow

Позволява ти условно да показваш, скриваш и повтаряш елементи.

- @if, @if-else, @else
- @for, @empty
- @switch

# @if

```
<div class="flex flex-col gap-5">
  <ng-container *ngIf="classesData.length>0">
    <app-catalog-filter [courses]="coursesData" (changeFilter)="onFilterClasses($event)"></app-catalog-filter>
    <app-catalog-table [classesData]="classesData"></app-catalog-table>
  </ng-container>
</div>
```

```
<div class="flex flex-col gap-5">
  @if(classesData.length>0){
    <app-catalog-filter [courses]="coursesData" (changeFilter)="onFilterClasses($event)"></app-catalog-filter>
    <app-catalog-table [classesData]="classesData"></app-catalog-table>
  }
</div>
```

# @for

```
<ng-container *ngFor="let classData of classesData">
  <tr class="divide-x">
    <td class="p-2">{{classData.course.name}}</td>
    <td class="p-2">{{classData.course.description}}</td>
    <td class="p-2">{{classData.professor}}</td>
    <td class="p-2">{{classData.durationDays}}</td>
  </tr>
</ng-container>
```

```
<tbody class="■ bg-white divide-y ■ divide-gray-200">
  @for (classData of classesData; track $index) {
    <tr class="divide-x">
      <td class="p-2">{{classData.course.name}}</td>
      <td class="p-2">{{classData.course.description}}</td>
      <td class="p-2">{{classData.professor}}</td>
      <td class="p-2">{{classData.durationDays}}</td>
    </tr>
  }
</tbody>
```

# track

```
<tbody class="■ bg-white divide-y ■ divide-gray-200">
  @for (classData of classesData; track $index) {
    <tr class="divide-x">
      <td class="p-2">{{classData.course.name}}</td>
      <td class="p-2">{{classData.course.description}}</td>
      <td class="p-2">{{classData.professor}}</td>
      <td class="p-2">{{classData.durationDays}}</td>
    </tr>
  }
</tbody>
```

# @switch

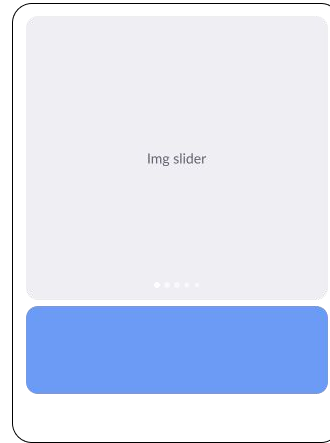
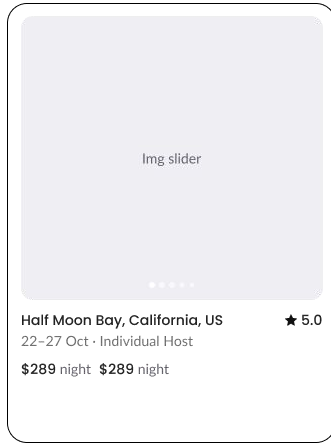
```
<ng-container [ngSwitch]="selectedOption">
  <ng-container *ngSwitchCase="'angular'">
    <p>Angular</p>
  </ng-container>
  <ng-container *ngSwitchCase="'react'">
    <p>React</p>
  </ng-container>
</ng-container>
```


```
@switch (selectedOption) {
  @case ('angular') {
    <p>Angular</p>
  }
  @case ('react') {
    <p>React</p>
  }
  @default {
    <p>jQuery</p>
  }
}
```

# ng-content

Елемент, който служи за индикация къде в темплейта на компонента може да се “вмъкне” допълнително съдържание.

Това позволява на компонента да бъде персонализиран без да се налага промяна по функционалностите му.



 == `<ng-content/>`



```
Go to component | You, 1 second ago  
<div>  
  <p>{{title}}</p>  
  <ng-content>  
    No content  
  </ng-content>  
</div>
```

Test

No content

```
Go to component | You, 5 seconds ago | 1 author (You)  
<app-blog-item title="Test">  
  This is the content of the blog item  
</app-blog-item> You, 5 seconds ago •
```

Test

This is the content of the blog item

# Multi-slot content projection

Техника, която позволява на компонентите да имат няколко именувани места (слота), в които родителският компонент може да “вмъква” допълнително съдържание.

За да създадем именувано място трябва да добавим `select` атрибут към `ng-content`

```
<ng-content select="[header]"></ng-content>
```

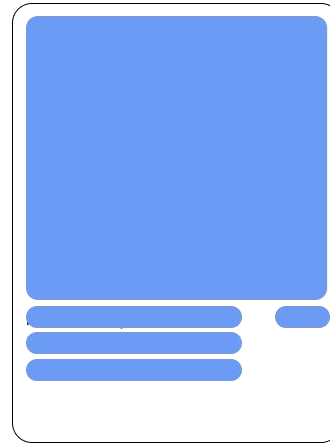
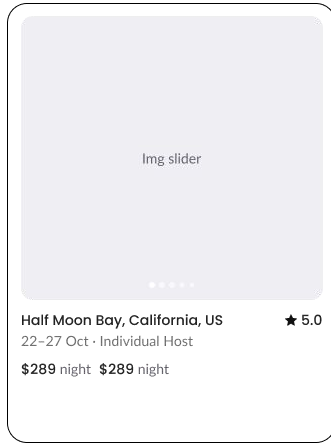
```

<div class="flex flex-col gap-4 ■ bg-slate-100 p-2 shadow-md">
  <div class="flex justify-between">
    <div class="flex gap-2 items-center">
      <button type="button" class="border rounded □ border-black p-1 text-sm"
        (click)="toggleContentVisibility()">{{showContent ? "Close":"Open"}}</button>
      <ng-content select="[title]"></ng-content>
    </div>
    <ng-content select="[extra]"></ng-content>
  </div>
  @if(showContent){
    <div class="transition-all ease-in-out duration-300">
      <div class="wrapper">
        <ng-content select="[body]"></ng-content>
      </div>
    </div>
  }
</div>

```

```
<ts-accordion>
  <div title>{{team.name}}</div>

  <div extrabody class="flex flex-col gap-2">
    body text
  </div>
</ts-accordion>
```

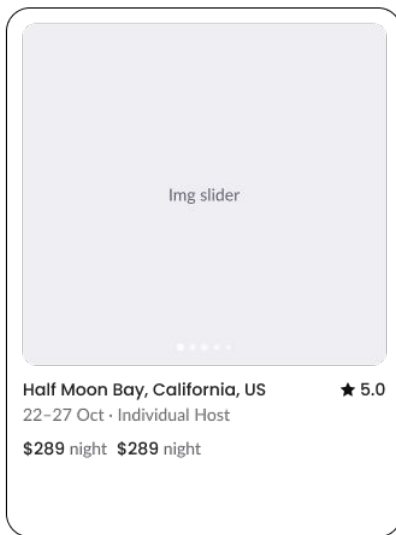


 == `<ng-content/>`

# Упражнение

- Да се създаде компонент карта
  - Трябва да може да му се подава заглавие, описание и снимка

```
type Blog = {  
  id: string;  
  title: string;  
  description: string;  
  image: string;  
};
```



# ng-template

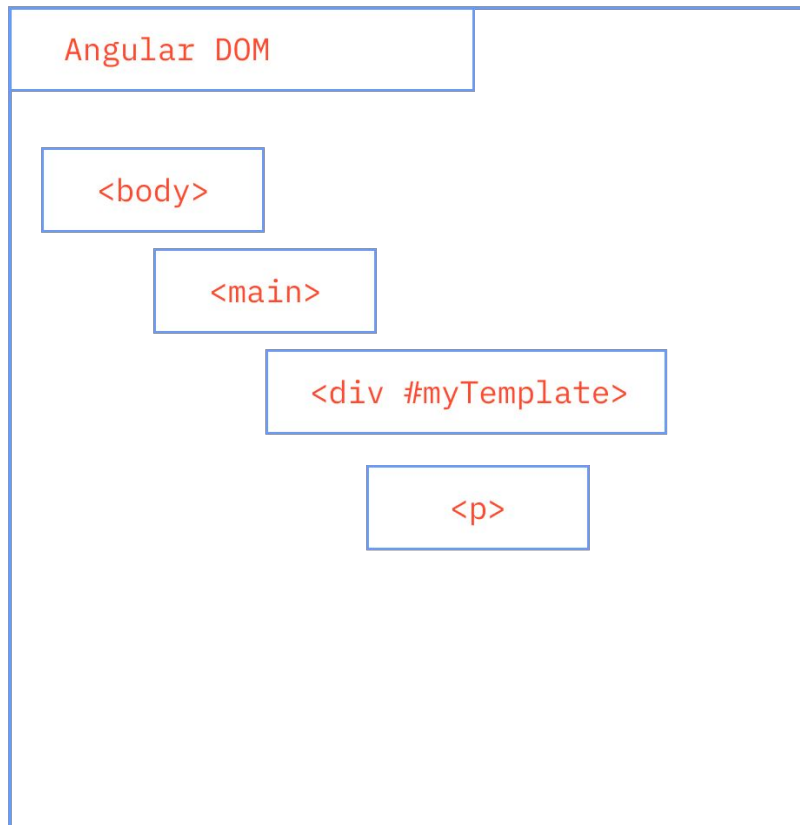
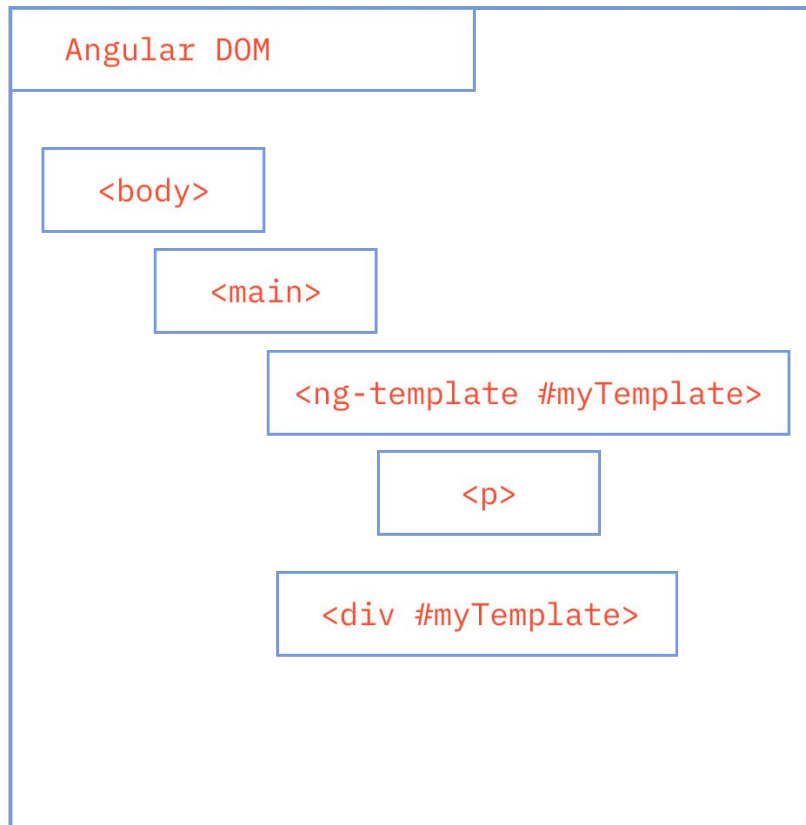
Елемент, който може динамично или програмно да бъде създаден.

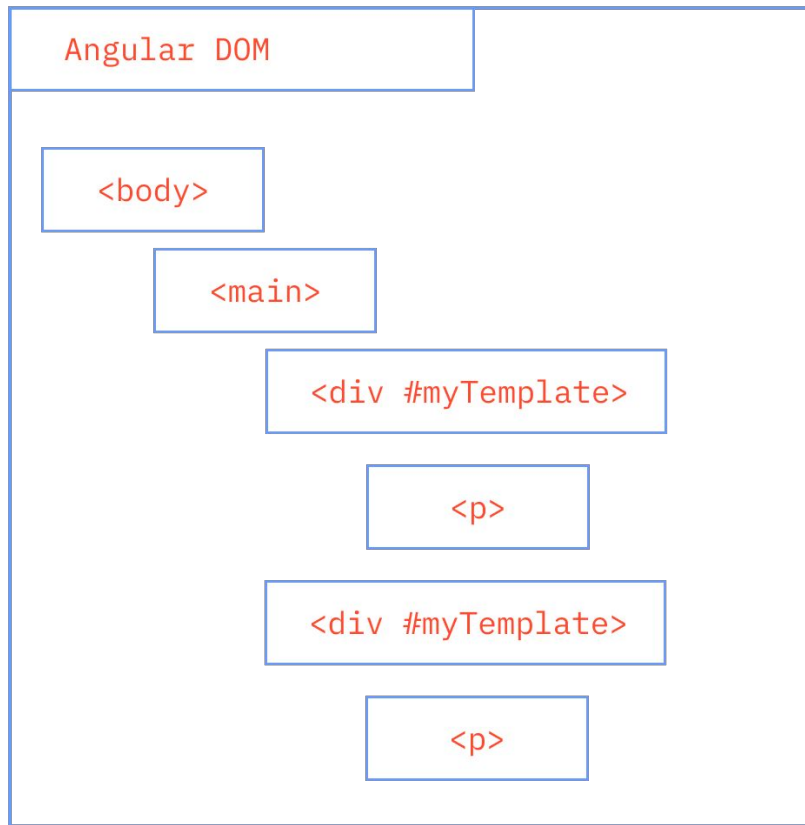
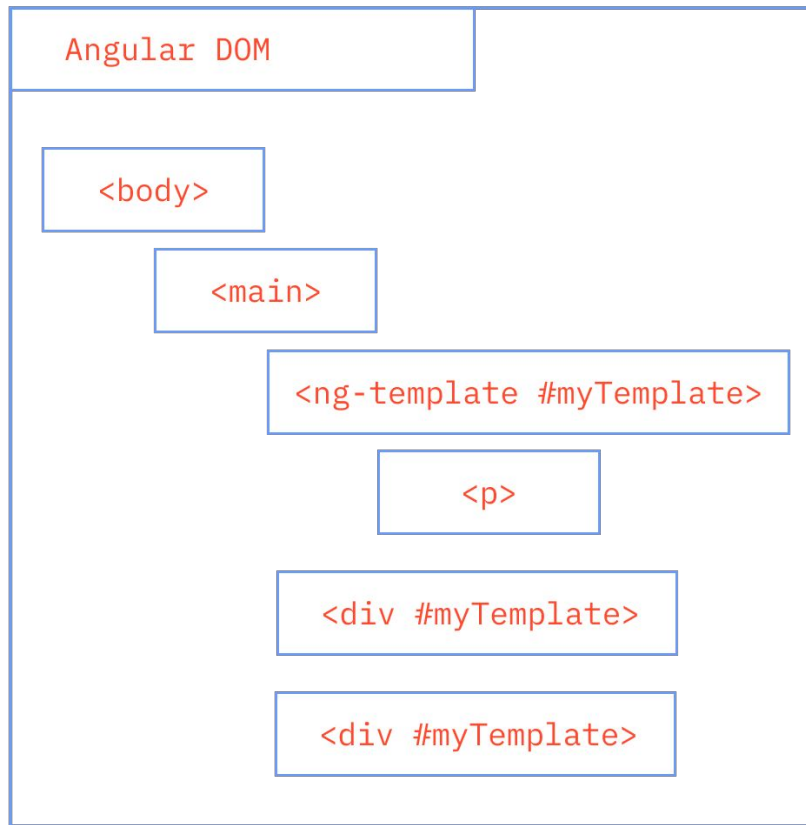
Може да се използва от други компоненти и директиви.

Ng-template има много различни начини на употреба.

```
<div>
  <div *ngIf="false; else myTemplate">
    Main content
  </div>
  <ng-template #myTemplate>
    Other content
  </ng-template>
</div>
```







# Dynamic component rendering

- Какво е dynamic component rendering ?
- Как да интегрираме dynamic component rendering ?

# Какво е dynamic component rendering ?

Техника, която ни позволява да заредим и покажем компоненти по време на изпълнението на програмата, вместо да ги деклариране предварително.

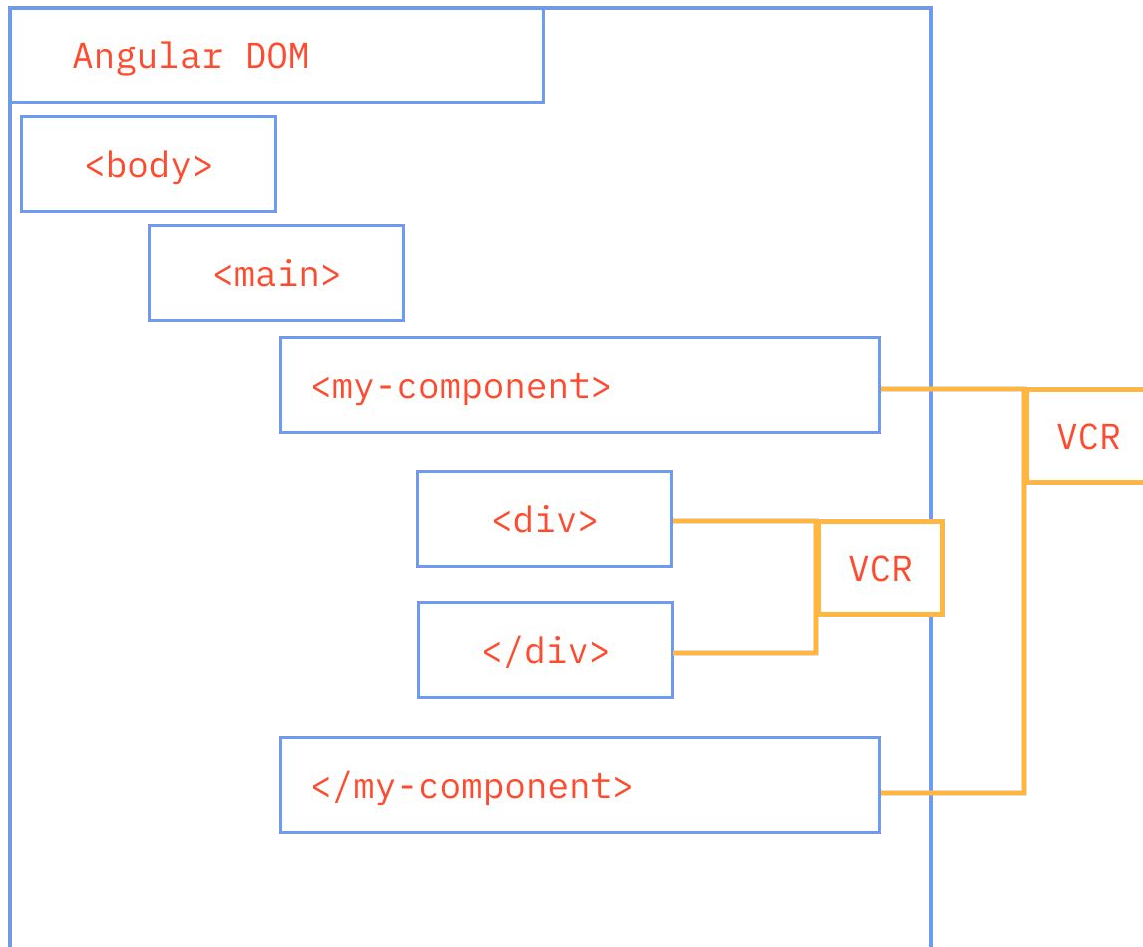
# Как да използваме dynamic component rendering ?

- `ViewContainerRef`
- `ngComponentOutlet`
- `ngTemplateOutlet`

# ViewContainerRef

Референция на контейнер, към който може динамично да добавяме компоненти.

ViewContainerRef може да реферира различни контейнери в зависимост от това как го използваме!



Go to component

```
<div>
  <div #div>div content</div>
  <span #span>span content</span>
</div>
```

```
export class ProfileComponent implements AfterViewInit {
  @ViewChild('div', { read: ViewContainerRef }) div!: ViewContainerRef;
  @ViewChild('span', { read: ViewContainerRef }) span!: ViewContainerRef;

  constructor(private vcr: ViewContainerRef) {}

  ngAfterViewInit(): void {
    console.log('host element->', this.vcr);
    console.log('div', this.div);
    console.log('span', this.span);
  }
}
```

Codeium: Refactor | Explain | Generate JSDoc | X



# Как създаваме компоненти динамично с `ViewContainerRef`

- `createEmbeddedView`
- `createComponent`

# createEmbeddedView

Създава динамични компоненти, чрез използване на ng-template

```
<div>
  <ng-container #mainDiv>
    Main Content
  </ng-container>
  <ng-template #myTemplate>My Template Content</ng-template>
</div>
```

```
export class ProfileComponent {
  @ViewChild('myTemplate', { read: TemplateRef }) myTemplate!: TemplateRef<any>;
  @ViewChild('mainDiv', { read: ViewContainerRef }) mainDiv!: ViewContainerRef;

  ngAfterViewInit(): void {
    this.mainDiv.createEmbeddedView(this.myTemplate);
  }
}
```

```
<div *ngIf="data">{{name}}</div>
```



```
<ng-template ngIf="data">  
  <div>{{name}}</div>  
</ng-template>
```

# createComponent

Създава динамично компонент, чрез неговия токен.

```
createComponent  
<div>  
  Main content  
</div>
```

```
export class ProfileComponent {  
  constructor(private vcr: ViewContainerRef) {}  
  
  Codeium: Refactor | Explain | Generate JSDoc | X  
  ngAfterViewInit(): void {  
    this.vcr.createComponent(AnalyticsComponent);  
  }  
}
```

# ngComponentOutlet

Директива, която приема тип на компонент и го създава в текущия изглед.

```
Go to component  
<div>  
  Main Content  
  <ng-template [ngComponentOutlet]="component" />  
</div>
```

```
export class ProfileComponent {  
  component: Type<AnalyticsComponent> = AnalyticsComponent;  
}
```

Main Content  
analytics works!

# Приложения

- Смяна на компонент спрямо някаква променлива или евент от потребителя
- Зареждане на компоненти от JSON схема

# Лимитации на ngComponentOutlet

- Може да създава само един динамичен компонент.
- Не може да реагираме на евенти от динамично създадения компонент

# ngTemplateOutlet

Директива, която приема темплейт и го създава в текущия изглед.

```
Go to component
<select class="w-full border border-black rounded" (change)="onSelectionChange($event)">
  @for (option of options; track $index) {
    <ng-container [ngTemplateOutlet]="selectTemplate || defaultOptions"
      [ngTemplateOutletContext]="{$implicit: option, index: $index}"></ng-container>
  }
</select>

<ng-template #defaultOptions let-option>
  <option [value]="option.value" [selected]="option.value === value? 'selected': ''">{{option.label}}</option>
</ng-template>
```



# Приложения

- Създаване на преизползваеми компоненти
- Създаване на компоненти, които лесно могат да бъдат персонализирани

# Добри практики при използването на ngTemplateOutlet

- Да имаме темплейт по подразбиране
- Може да имаме повече от един ngTemplateOutlet
- Може да подаваме темплейта като входен параметър
- Да добавим проверка на типовете

```
// @ContentChild(selectTemplate, {
  @Input(['selectTemplate'])
  selectTemplate!: TemplateRef<any>;
```

```
<ng-template #sharedTemplate let-option let-index="index">
  <option [value]="option.value" [selected]="option.value === selectedOption">
    {{index}}-{{option.label}}</option>
</ng-template>

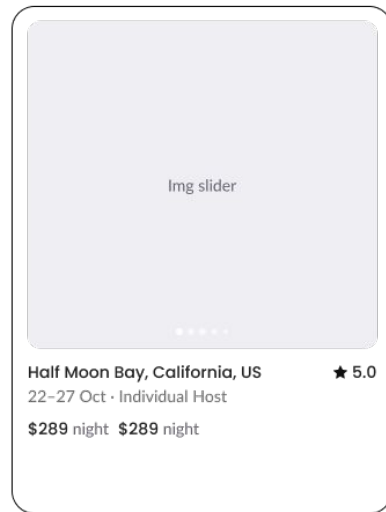
<app-select [options]="options" [value]="selectedOption" [selectTemplate]="sharedTemplate">
</app-select>

<app-select [options]="options" [value]="selectedOption" [selectTemplate]="sharedTemplate">
</app-select>
```

# Упражнение

- Да се създаде преизползваем компонент карта
  - Трябва да може да работи без да е нужно да уточняваме как да се рендерира частите на картата
  - Трябва да може да се променя изгледа на снимката
  - Трябва да може да се променя изгледа на заглавието
  - Трябва да може да се променя изгледа на описанието

```
type Blog = {  
  id: string;  
  title: string;  
  description: string;  
  image: string;  
};
```



# Бонус - проверка на типове в ngTemplateOutlet

```
interface SelectorRowContext<TData extends SelectOption> {  
  $implicit: TData;  
  index: number;  
}  
  
@Directive({  
  standalone: true,  
  selector: 'ng-template[selectorRow]',  
})  
export class SelectorRowDirective<TData extends SelectOption> {  
  @Input('selectorRow') data!: TData[];  
  
  static ngTemplateContextGuard<TContext extends SelectOption>(  
    dir: SelectorRowDirective<TContext>,  
    ctx: unknown  
  ): ctx is SelectorRowContext<TContext> {  
    return true;  
  }  
}
```

```
export class SelectComponent<TData extends SelectOption> {  
  @ContentChild(SelectorRowDirective, { read: TemplateRef })  
  selectTemplate!: TemplateRef<SelectorRowContext<TData>>;  
  
  @Input({ required: true }) value!: string;  
  @Input({ required: true }) options!: TData[];  
  
  @Output() changeSelection = new EventEmitter<string>();  
  
  onSelectionChange(e: Event) {  
    const newValue = (e.target as HTMLSelectElement).value;  
    this.changeSelection.emit(newValue);  
  }  
}
```

```
<app-select [options]="options" [value]="selectedOption">
  <ng-template [selectorRow]="options" let-option let-index="index">
    <option [value]="option.value" [selected]="option.value === selectedOption">
      {{index}}-{{option.label}}</option>
    </ng-template>
  </app-select>
```

```
<app-select [options]="options" [value]="selectedOption" (variable) option: { label: string; value: string; }>
  <ng-template [selectorRow]="options" let-option let-index="index">
    <option [value]="option.value" [selected]="option.value === selectedOption">
      {{index}}-{{option.label}}</option>
    </ng-template>
  </app-select>
```

A large, light blue decorative arc is positioned on the left side of the slide, partially cut off by the edge.

**Благодаря за вниманието!**