# Posielanie dát medzi komponentami

- robí sa to pomocou property a event binding. Tieto veci je možné bindovať aj na komponenty.

- vieme teda nabindovať dáta na naše vlastne HTML properties (custom properties), čo sú vlastne properties nášho komponentu.

# Bindovanie na custom properties

## Príklad:

Do server-element componentu chceme posielať info o samotných serveroch z app componentu. Musíme si nabindovať custom property v server-element componente na jednotlivé položky z app componentu.

```

<app-server-element

\*ngFor="let serverElement of serverElements"

[element]="serverElement"> // nabidnovanie custom property "element" na jednotlivé položky

</app-server-element>

```

V server-element componente TS súbore si musíme vytvoriť premennú, ktorá nám bude reprezentovať daný server. Do tejto premennej sa uložia informácie o serveri, ktoré si potom pomocou string interpolation vieme vypísať v server-element componente.

```

@Input() element: {type: string, name: string, content: string};

```

Decorator Input slúži na to, aby sme spravili túto property viditeľnú pre všetky komponenty hostované našou aplikáciou.

## Druhý príklad

Ak chceme bindovať na inú property, ako je názov danej property v server-componente, tak môžeme určit alias, ako je vidieť na druhom príklade

```

<app-server-element

\*ngFor="let serverElement of serverElements"

[srvElement]="serverElement"> // nabidnovanie custom property "element" na jednotlivé položky

</app-server-element>

```

```

@Input('srvElement') element: {type: string, name: string, content: string};

```

# Bindovanie na custom eventy

Ak chceme informovať náš parent component, že nastala nejaká zmena, alebo nejaká akcia.

Musíme si vytvoriť vlastné eventy v componente, ktorý chce odoslať informácie. Robí sa to nasledovne:

```

@Output() serverCreated = new EventEmitter<{serverName: string, serverContent: string}>();

@Output() blueprintCreated = new EventEmitter<{serverName: string, serverContent: string}>();

```

EventEmitter je objekt, ktorý je generického typu (davame mu datatype, teda čo bude posielať ako informácie eventu do <>).

@Output je decorator, ktorý slúži na to, aby sme tento event mohli zavolať aj z iného komponentu.

Takisto si musíme vytvorit funkcie, ktoré nám vyvolávajú daný event. Tieto funkcie si potom zavoláme nappríklad na klik nejakého buttonu.

```

onAddServer() {

this.serverCreated.emit({

serverName: this.newServerName,

serverContent: this.newServerContent

});

}

onAddBlueprint() {

this.blueprintCreated.emit({

serverName: this.newServerName,

serverContent: this.newServerContent

});

}

```

Keď máme vytvorené eventy, pomocou event binding si ich dáme na element komponentu, ktorý chce odoslať informácie

```

<app-cockpit

(serverCreated)="onServerAdded($event)"

(blueprintCreated)="onBlueprintAdded($event)"></app-cockpit>

```

Funkcie, ktoré volá daný event definujeme v parent komponente, kam posielame info.

\*\*Alias funguje tak isto ako pri custom property binding\*\*

#View encapsulation

Angular narúša defaultné správanie browseru tým, že css súbory jednotlivých komponentoch aplikujú štýly len an elementy z daného komponentu. Defaultné správanie je také, že ak includneme nejaký css súbor na web, tak sa štýly aplikujú na každý element. Roí to tak že pridáva rovnaké atribúty k elementom, ktoré sú v rovnakom komponente.

Riadenie enkapsulácie:

Ak do derocatora dame property "encapsulation tak vieme riadit enkapsuláciu.

máme 3 možné hodnoty:

- none - štýly budú globálne

- native - používa shadow DOM technology

- emulated - default - štýly sú viditelné len pre daný komponent

```

@Component({

selector: 'app-server-element',

templateUrl: './server-element.component.html',

styleUrls: ['./server-element.component.css'],

encapsulation: ViewEncapsulation.None

})

```

# Local reference

Oznčujeme ju znakom '#'. Ak ňou označíme element, tak táto referencia drží celý ten element s všetkými properties.

\*\*Môžeme ju použiť len v template, v TS kóde nebude fungovať!!!!\*\*

```

<input

type="text"

class="form-control"

#serverNameInput> // zadefinovanie local reference

<label>Server Content</label>

<input type="text" class="form-control" [(ngModel)]="newServerContent">

<br>

<button

class="btn btn-primary"

(click)="onAddServer(serverNameInput)">Add Server</button> // použitie local reference

```

# @ViewChild

Je do decorator, ktorý slúži na priamy prístup k elementom v template z TS kódu. Takisto ho môžeme použiť aj na prístup k componentom. používa sa v spojení s local reference. Ako argument mu musíme poslať local reference na daný element. Ak selektujeme component, tak musíme dať názov komponentu.

Do elementu sa uloží hodnota dátového typu ElementRef. Aby sme vedeli pristúpiť k samotného html elementu, musíme pristupovať cez atribút nativeElement.

\*\*Neodporúča sa pristupovať k DOM týmto spôsobom, angular poskytuje lepšie možnosti ako to spraviť (pomocou directives)\*\*

```

@ViewChild('serverContentInput') serverContentInput;

```

```

this.serverContentInput.nativeElement.value

```

# ng-content

Všetko čo dáme medzi tagy, ktoré reprezentujú komponent, je stratené. Aby sme tomu predišli, tak môžeme pomocou tagu <ng-content></ng-content> určiť, kde sa tento kód v rámci komponentu má appendnúť.

Používa sa to napríklad vtedy, ak máme nejaký HTML kód, ktorý chceme dať do stránky odniekaľ zvonka.

# Component lifecycle

Počas vytvárania komponentu nám angular poskytuje dostať sa do istých jednotlivých fáz vytvárania

- ak chceme nasledovné metódy implementovať, musíme si pridať ku classe prislúchajúci interface (najdi v dokumentacii).

Treba si pozriet skor dokumentaciu.

Fázy: (metódy spustené pri týchto fázach)

1. ngOnChanges

- zavolané pri vytvorení nového komponentu

- keď sa nejaká input property (property v TS classe daného componentu s decoratorom Input) v tomto elemente zmení

2. ngOnInit

- zavolané pri inicializácii komponentu - nie pri pridaní do DOMU, ale vtedy keĎ je vytvorený samotný komponent

3. ngDoCheck

- zavolaný pri každej zmene v template (change detection)

- napr ak sa nejaká property zmení z 1 na 2 a je vypísaná do templatu

- je takisto zavolaný aj keď napríklad klikneme na nejaký button v componente a nič nezmení len sa vyvolá event

- použijeme keď chceme zavolať nejaký kód pri každej zmene alebo akcii v komponente

4. ngAfterContentInit

- zavolané po tom ako sa content (ng-content) dal do view

... je ich ešte viac