**Property binding** - używa się, gdy chcę dynamicznie zmieniać wartości Events, Directives, Other components.

Poperty bind syntax:   
**[atribute to change]="changes"**  
W składni property binding musi znaleźć się kod TypeScript zrozumiały dla Angulara  
  
String interpolation działa tylko w kodzie normalnego template, w składni TypeScript powoduje błąd.  
Używa się tylko do wyświetlenia tekstu, można to zrobić za pomocą metody  
  
Property binding i String Interpolation to dwa sposoby na wyświetlanie danych w Template - dla użytkownika  
  
Two-way binding, połączenie event i property binding.  
  
(nameOfEventBinding) - event binding; nie powinny zawierać zbyt dużo logiki. Lepiej zrobić metodę, która przez zmianę jakiejś zmiennej bool będzie wywoływać wiele metod  
[nameOfPropetyBinding] - property binding  
[(nameOfDirective)] - two-way binding, połączenie event i property binding - następuje akcja event i bierząco zmienia informacje wraz z wysyłaniem ich do użytkownika  
  
**4 sposoby data binding:**  
event - listen to the click,  
string interpolation - output data,  
property - enable button after time,  
two-way - fetch output data.  
  
Dyrektywy @Directive - dodaje je po to żeby kontrolować kiedy coś ma się zadziać. Np.: \*ngIf - jeśli chcę kontrolować w jakim stanie ma się coś zadziać - mogę używać w przypadku zmiennych albo metod, które zwracają true or false.  
Typowe dyrektywy, są wywoływane poprzez selektory (podobnie jak @component) i istnieje kilka rodzajów ich wywołań (jak w przykładach succes and warning alert).  
Komponenty, są specjalnymi dyrektywami.  
Są różne typy tyrektyw:  
- Structural directives: wprowadzają zmiany w strukturę DOM (dodają, usuwają elementy)  
\*ngIf - to dyrektywa strukturalna, gwiazdki używam odwołując się do dyrektyw strukturalnych. Zmiana struktury DOM dokonuje się w sposób dynamiczny. Np.: Elementu strony nie ma, lecz po wykonaniu jakiejś akcji, zostaje utworzony element.  
- Attribute directives: nie wprowadzją zmian w DOM, wyłącznie zmieniają elementy, które już istnieją. Odwołuje się do nich bez użycia \*.  
[ngStyle] - pozwala na dynamiczną zmianę wartości selektorów CSS  
[ngFor] - pozwala na  
[ngClass] - pozwala na dynamiczne dodawanie bądź usuwanie klas CSS

Za pomocą dekoratora (oznaczenie: @) np.:   
**@Input(‘srvElement’) element: {type: string, name: string, content: string};**

– ustanawiam możliwość odwołania się do property „element” z innych komponentów aplikacji.   
W tym konkretnym przypadku, możliwość wprowadzenia danych na podstawie ustanowionego wzorca. ‘srvElement’ to alias – znazwa za pomocą której będę się odnosił do elementu w innym komponencie.

**@output(‘bpCreated’) blueprintCreated = new EventEmitter<{serverName: string, serverContent: string}>();**

newServerName = ‘’;

newContentName = ‘’;

**onAddBlueprint(){**

**this.blueprintCreated.emit({**

**servername: this.newServerName,**

**serverContent: this.newServerContent});**

**}**

EventEmitter to typ wbudowany w JavaScript – ustawia się za jego pomocą properte jako typowo przyjmującą dane z eventu, w tym przypadku obiekt o podanych właściwościach. Na koniec wywołuje się () konstruktor tworzący nowy obiekt typu eventEmitter który jest przechowywany w naszej nazwie zmiennej (blueprintCreated).

EventEmitter buduje obiekt w Angular Framework, który pozwala na emisję eventu poza komponent w którym event został utworzony .

Żeby obiekt typu event mógł być nasłuchany, należy dodać dekorator @Output i zaimportować odpowiednią ‘bibliotekę’

**Encapsulation –** nadpisywaniedomyślnego postępowania z dzidziczeniem stylów css. Dodaje się ją poprzez import biblioteki z @angular/core. Do wyboru są 3 opcje:

* *Native* – „**ShadowDom**” technology. Działa tak samo jak standardowa enkapsulacja stylów.
* *None* – Angular nie wymusza enkapsulacji stylów (**view encapsulation**) (nie dodaje do znaczników atrybutów świadczących o wymuszeniu wzorca stylów umieszczonego w konkretnym komponencie dla wszystkich jego elementów). W przypadku użycia tej opcji, zmiany w pliku css dla komponentu będą zastosowane globalnie do całego projektu.
* *Emulated* – opcja default – emuluje działanie ShadowDom, ponieważ nie wszystkie przeglądarki ją obsługują. Nadaje wszystkim elementom komponentu taki styl jaki był zadeklarowany w pliku .css. komponent-rodzic nie nadpisuje tych stylów.

**Local references**

Etykiety dla szablonu HTML: dodaje się poprzez dodanie atrybutu dla znacznika rozpoczynającego się od # (@serverName). Dodanie takiego atrybutu tworzy referencje na znacznik HTML i ewentualnie przechowywane przezeń info, jeśli na przykład jest to input. Tą referencję można przekazywać do wszystkich elementów szablonu (chyba jedynie w obrębie dokumentu html, w którym została utworzona). Referencję tą można przekazać do kodu TypeScript poprzez wywoałanie metody za pomocą Eventu, np.: poprzez kliknięcie buttona.

<input  
 type="text"  
 class="form-control"  
 #***serverNameInput***>

<button  
 class="btn btn-primary"  
 (click)="onAddServer(***serverNameInput***)">Add Server</button>  
<button

onAddServer(**nameInput.value**) {}

**Dynamiczne przekazywanie danych** **pomiędzy elementami komponentu (html/ts) /komponentami** może się odbywać poprzez:

* **two-way binding,**
* **local references pass to method,**
* **local references fetched through directive @ViewChild()**;
* **ng-content directive** – hook which i can use in another component, to execute code there

@ViewChild('serverContentInput', {static: true}) serverContentInput: ElementRef;

ElementRef to typ danych Angulara, który trzeba zaczytam poprzez dopisanie kolejnej biblioteki.

Nie powinno się zmieniać wartości „feczowanego” elementu.

**Nie powinno się ingerować** w strukturę DOM poprzez wykorzystanie zmiennych Angulara, np.:   
z dyrektywą @ViewChild(). Lepiej jest to zrobić wykorzystując String interpolation albo Property binding.

**<ng-content></ng-content>** użyte w kodzie html komponentu bazowego, umożliwia wykonywanie kodu zawartego w innym komponencie po użyciu selektora komponentu bazowego (np.: <app-server-element>).

**Lifecycle Hooks**

- okres życia elementów tworzonych w Angularze. Zawiera 8 metod w które można ingerować w uzasadnionych przypadkach. Może się to okazac korzystne.

* **ngOnChanges()** – wywoływana jest w momencie stworzenia nowego komponentu, następnie jest wywoływana za każdym razem, gdy następuje zmiana w zmiennych związanych z komponentem (np.: @input() element: string; – kiedy następuje zmiana)
* **ngOnInit** – wywoływana w chwili inicjalizacji komponentu. Komponent nie jest jeszcze dodany do DOM ale Angular już wykonuje na nim swoje podstawowe operacje. Otrzymujemy na tym etapie dostęp do propertów. Metoda zostaje wywołana po konstruktorze.
* **ngDoCheck** – jest wywoływana za każdym raze, gdy w template się coś zmienia. Za każdym razem, gdy jakieś działanie jest wykonywane. Może się przydać gdy chce się wykonywać jakąś operację za każdym razem gdy następuje zmiana w programie
* **ngAfterContentInit** – wywoływana w momencie inicjalizacji projekcji jednego komponentu w drugim (<ng-content>)
* **ngAfterContentChecked** – za każdym razem, gdy w projektowanym elemencie następuje zmiana.
* **ngAfterViewInit** –
* **ngAfterViewChecked –**
* **ngOnDestroy** – uruchamiana w chwili niszczenia jakiegoś elementu w Angularze.