**Component Comunication**

@Input - properties bindable from outside, from parent component

@Output - enable parent components to listen to your own custom events which you can create with EventEmitter

-parameters where you can rename basic property name

If you have 2 components next to each other you build chains of Inputs and Outputs

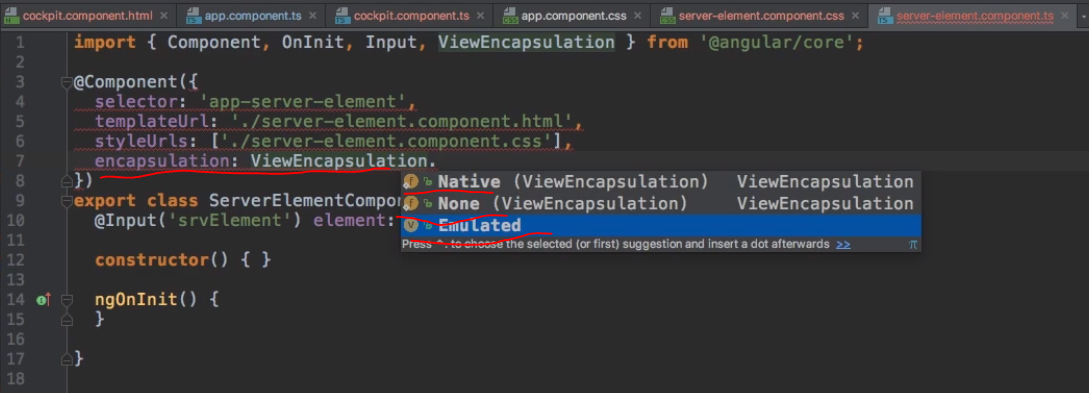
Some use cases where components are too far - long chains there is other approach which will be shown later.

**CSS Angular forced behavior – Style encapsulation**

Styles apply for the component where a are placed

Angular gives a specific attr for all elements in a component

This encapsulation **can be overridden** – in a component definition:



* None – we wont see Angular attr. (The others use view encapsulation)
* Native – uses Shadow DOM technology

When we disabled encapsulation in affects other Components.

**Property binding**

**Event binding**

**Two way binding**

**String interpolation**

**Attribute VS Structural Directives**

Attr. – vztahuju san a konkretny element

Struct. – -||-, ale tiez ovplyvnuje priamo DOM okolo daneho elementu, vieme vynechat/zobrazit iba niektore elementy (napr. \*ngIf zobrazi iba jeden z odsekov), obsahuje \* na zaciatok,

**Vlastne Attr Directives** (ng g d nazov)

-Implements onInit

-pouzivanie Rendereru na menenie properties, napr farby pozadia (<https://angular.io/api/core/Renderer2>)

-namiesto toho sa da pouzit **HostBinding** – vieme specifikovat na ktoru property hosting elementu chceme bindovat (napr. @HostBinding(‘style.backgroundColor’) backgroundColor : string = “blue”)

**-HostListener –** pocuva na eventy ktore sa deju na elemente na ktorom je umiestneni directive (napr. Pri Mouseenter a mouseleave sa zmeni farba)

-custom binding of directive properties…

**Structural Directives:**

\*ngFor

\*ngIf

ngSwitch – use : [ngSwitch]=”value”, <p \*ngSwitchCase=”5”>…, <p2 \*ngSwitchCase=”2”>… ,…,

<pN \*ngSwitchDefault>…

**Services** (<https://angular.io/guide/architecture-services>)

* can clean the app up
* make code leaner, more centralized and easier to maintain
* we do not need to build complex Input/Output Chains
* Basic use cases: duplication of code and data storage
* Acts as central repository
* E.g. log service, user service (to manage data storage)

**Dependency injector**

* Tool for accessing services
* Injects an instance in our component automatically
* Hierarchical injector – if we provide service in appModule the same instance of the service is available in our whole app (in all components, directives, services..), if in Appcomponent all children of this have same instance (instance don’t propagate up)
* If we provide same service on higher level – **it overrides!**
* We can stop the overriding with deleting the Service from providers in child classes, **not from the constructor!**
* E.g of use: (We create a Logging service – print a console log in it)

constructor(private loggingService: LoggingService){}

* This informs a component that we will need an instance of LoggingService
* When angular comes to our selector (in html) it gives us instances of our components. Similarily now it tries to gives us this component
* providers: [LoggingService]
* typescript needs to know where this logging service comes from (if we want to operate with the same instance we just import service, **NOT THE PROVIDERS!**

We can use (inject) a service in a service too – we specify the providers in appModule, use the constructor the same way, we have to attach metadata – we attach @Injectable (this tells angular that something can be injected in there)

-it is good for centralization of the services

**Replacing Input/Output**

* in the service add EventEmitter property
* in one component emit:

this.accountsService.statusUpdated.emit(status);

* In the other component subscribe:

constructor(private loggingService: LoggingService, private accountsService: AccountsService){

this.accountsService.statusUpdated.subscribe(

(status : string) => alert('New status: '+ status)

);

}

* **For details check Services-asignment-start User Service!**

In angular 6+ application wide services can be done this way too: (cons only for big apps)

* @Injectable({providedIn: 'root'})
* export class MyService { ... }

General Notes:

//slice returns a copy of the array in this service, otherwise it is passed by reference