**Everything there is to know about components - Angular**

**The structure of a component –**

Any component should be composed from number of parts, the first part is the \*component-name\*.component.ts file, this file will have an import of the component from the angular/core, a component selector that gets an object that contains kind of important data to the component and export of the component as a class. It looks like that:



**selector –** it’s the HTML tag that we will use when we want to use our new component. The selector mechanism is like the css one, but selecting by ID or pseudo selector won't work.

**templateUrl / template –** the actual HTML code that we insert into the component and will be shown whenever we use the new component we are creating. If we want to use a file we will choose - templateUrl else for inline HTML style we will use - template.

**stylesUrl/styles –** same as the template selector but only with CSS.

**How to create a new component?**

There are 2 ways to create a new component, manually or with the CLI, let's go through them:

**The manual way:**

In this way, we will create the files ourselves, first of all lets create a new folder inside our app folder, the folder name will be our component name. Example: warning-alert, inside that folder lets create these files: warning-alert.component.ts/html/css in each of them we will find only the relevant pieces of code as we expect by the file name.

After that, we should import our new component to the app.module.ts file to the declarations array.

**The CLI way:**

we can you the following commands:

ng generate component \*component-name\* / ng g c \*component-name\*.

**Component Lifecycle:**

Inside the export class we can see that our new component implements OnInit, and has a constructor. Inside the constructor, we can write code that will run at the point of time when this component is being created by Angular.

**ng-content:**

ng content is a directive we can use in our component to pass data, but it looks like an element selector (its serves as a hook). This directive will come handy if we want to pass some HTML data that is in between the opening and closing selectors of our custom component, and it will be placed in between our ng-content selector in our custom component. Example: This is the parent component -

תמונה שמכילה טקסט, צג, מקורה, צילום מסך

התיאור נוצר באופן אוטומטי

This is the child that receive the template and implements it inside the decorator

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

It would be nice when we will use it for building re-usable components and want to inject different codes in there.

Lifecycle Hooks:

**ngOnChanges** **–** Called after a bound input property (properties using the decorator @Input()) changes. It means that once at the beginning, and everytime the relevant properties has a new value. It’s the only hook that receives an argument and its receive changes argument from type SimpleChanges (imported from angular/core)

**ngOnInit –** Called once the component is initialized. It will run after the constructor, it doesn’t mean that the component has been added to the DOM or anything, just that Angular finished the basic process and the properties can be accessed and initialized for example.

**ngDoCheck –** Called during every change detection run. It runs a lot, everytime that Angular has to check if there any change need to be happening.

**ngAfterContentInit –** Called after content (ng-content) has been projected into view.

**ngAfterContentChecked –** Called every time the projected content has been checked. Like ngDoCheck but for the ng-content.

**ngAfterViewInit –** Called after the component’s view (and child views)has been initialized (once the view has been rendered).

**ngAfterViewChecked –** Called every time the view (and child views) has been checked. So once we are sure that either all changes which had to be done were displayed in the view or no changes were detected by angular.

**ngOnDestroy –** Called once the component is about to be destroyed.

**It’s a good practice to specify which hook our components implements!**