**Angular**

* Angular is for Single Page Applications. It has all the capabilities of a server side deployment.
* [Angular (forked) - StackBlitz](https://stackblitz.com/edit/angular-nvmjgf?file=src%2Fmain.ts) is the sandbox for Angular
* In Angular , UI functionality is encapsulated in components. A component can represent anything from a piece of text, button, form etc., and can contain another component. Components communicate with one another with the help of well defined interfaces.
* A TypeScript class is a component using the @Component decorator. Decorators provide additional information by annotating or modifying classes or class members. In this case, the component decorator provides additional metadata to Angular about how the class will behave.



* A ‘selector’ attribute tells this component an accessed with a HTML tag ‘app-root’
* A ‘standalone’ attribute tells that this component doesn’t reside in separate NgModule
* A ‘template’ attribute tells or shows us the actual HTML markup
* The ‘imports’ tag tells us that this component makes use of functionality inside ‘CommonModule’
* The executable portion of the code is found inside the class definition.
* Even though everything is present in a single file still code, mark, styles are all separated from one another. This is called addressing ‘Separation of Concerns’
* Inside the HTML above we can see {{ name }} and during rendering phase Angular sees the expression and binds the value from variable of the class. This is called ‘One-way’ binding in Angular.
* **Import vs imports** 🡪 The TypeScript ‘import’ imports classes, functions, types and other exported items from another typescript or javascript modules. From Angular 15, the @Component decorator has ‘imports’ which includes the components we want to import components, directives, pipes and NgModules into the HTML mark of our current component. **HelloComponent.ts main.ts** A computer screen shot of a program code

  Description automatically generated A computer screen shot of a program code

  Description automatically generated
* The Angular CLI is an indispensable tool for Angular developers, providing a comprehensive set of commands that simplify and streamline the process of creating, scaffolding, and managing Angular applications. The CLI abstracts away much of the complexity and boilerplate code that comes with setting up a new Angular project, allowing you to focus on writing your application logic

npm install –global @angular/cli@13.0.3

(If “ng v” doesn’t display anything or says “the term ‘ng’ is not recognized as the name…’ then in the environment variables add this under ‘PATH’ variable 🡺 *%AppData%\npm* make sure that path stays in the top)

* ng new <options...> (ng new <project\_name> --routing false --style css --skip-git --skip-tests)

Some of the more common options are

* --dry-run (Boolean) (Default: false) – Run through without making any changes.
* --skip-install (Boolean) (Default: false) – Skip installing packages.
* --skip-git (Boolean) (Default: false) – Skip initializing a git repository.
* --standalone (Boolean) (Default: false) – Creates an application based upon the standalone API, without NgModules.
* --style (String) (Default: css) – The style file default extension. Your choices are css, scss, sass, or less.
* --prefix (String) (Default: app) – The prefix to use for all component selectors.
* --routing (Boolean) (Default: false) – Generate a routing module.
* ng new <options...>

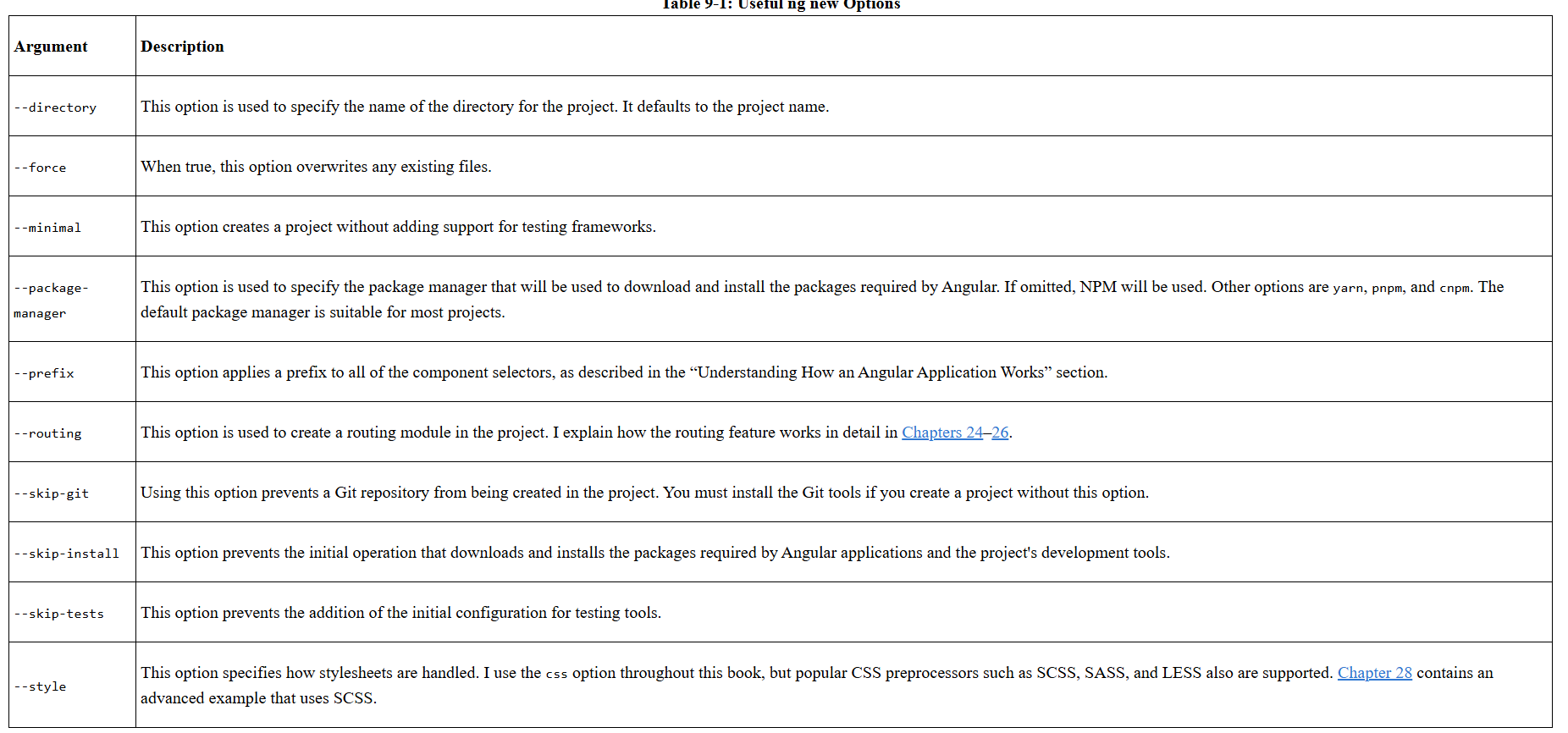
Some of the more common options are :

* --dry-run (Boolean) (Default: false) – Run through without making any changes.
* --skip-install (Boolean) (Default: false) – Skip installing packages.
* --skip-git (Boolean) (Default: false) – Skip initializing a git repository.
* --standalone (Boolean) (Default: false) – Creates an application based upon the standalone API, without NgModules.
* --style (String) (Default: css) – The style file default extension. Your choices are css, scss, sass, or less.
* --prefix (String) (Default: app) – The prefix to use for all component selectors.
* --routing (Boolean) (Default: false) – Generate a routing module.

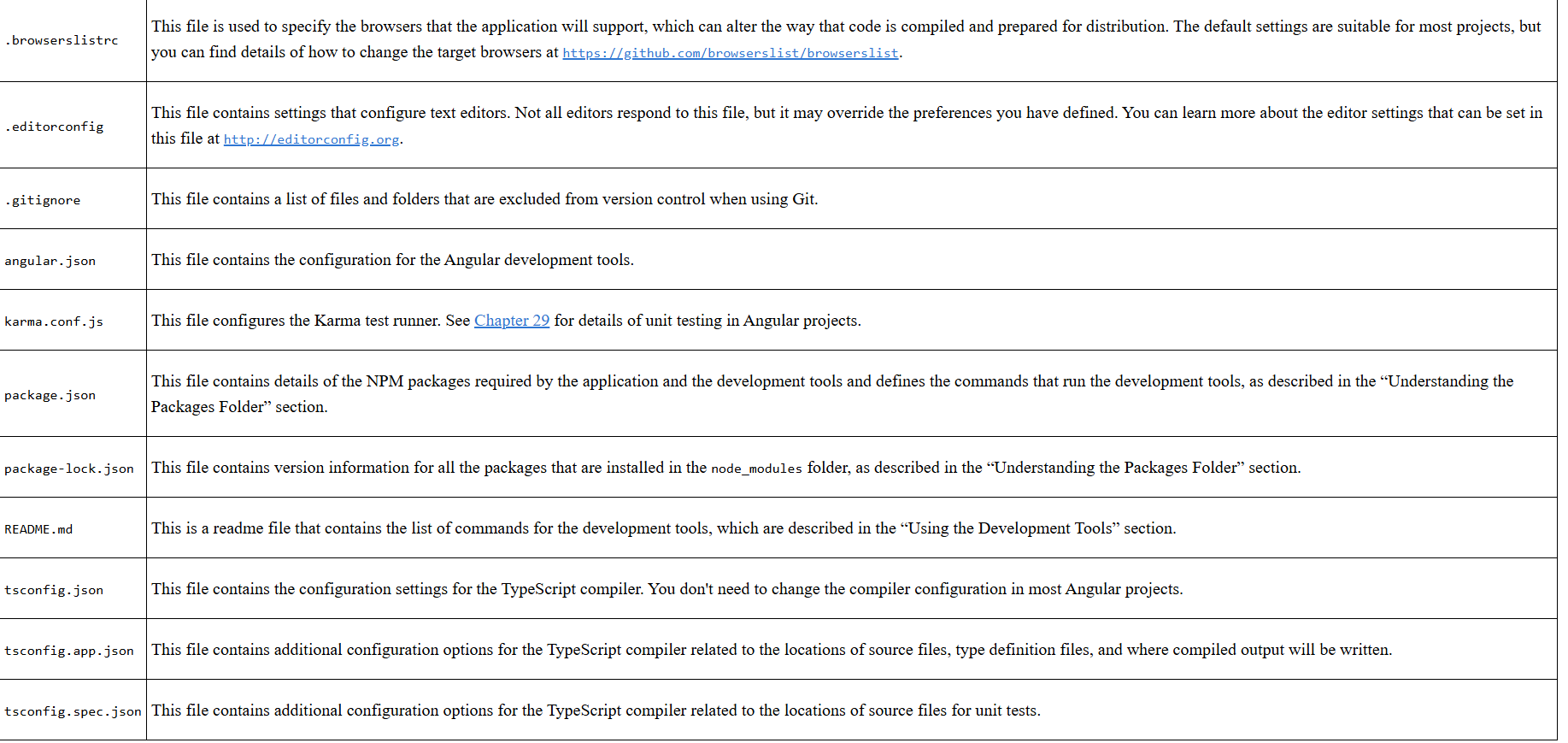
## Common Flags for Schematics

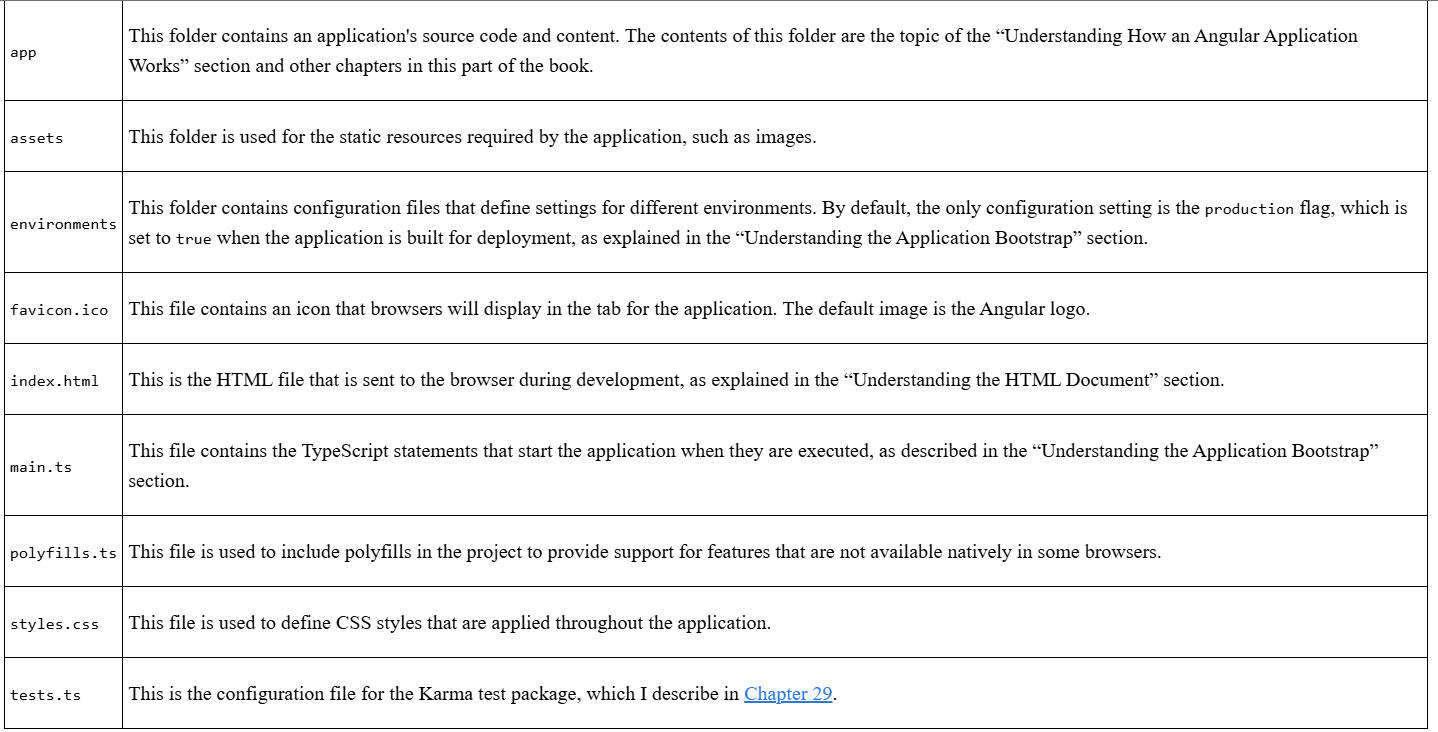
These flags are commonly used with most of the schematics:

* --flat (Boolean): Flag to indicate if a directory is created. By default, ng CLI will create a separate folder for most blueprints, even if there is only one file. If you know you won’t be creating multiple files (templates, tests, etc.), you can pass true to this flag, and the CLI won’t create the separate folder.
* --spec (Boolean): Specifies if a spec file is generated. Pass false to this flag to prevent the CLI from generating test files for you. Use this with caution. There are very few reasons not to have unit tests for your code.
* --app (String): Specifies app name to use.
* --standalone (Boolean): Specifies whether a component or directive should be created without an existing NgModule.
* --module (String): Allows specification of the declaring module. By default, the item being generated will be attached to the “closest containing module.” The CLI will walk up the folder tree, looking for a module. Specifying a different module here will override that behavior.
* npx ng generate component Loading –skip-tests –dry-run
* ng serve <options…>
* --host (String): Allows you to change the host being served.
* --port (Number): Allows you to override the port served.
* --open: Causes the CLI to open your default browser automatically.
* ng serve 🡪 displays or serves the output from memory not from disk.
* The @Input() decorator tells us Angular to expose the variable denoted with @Input() decorator as an attribute tag for the HTML element.
* **Guards:**
  + **Route Guard 🡪** It acts a gatekeeper to decide whether navigation to a requested route is permissible or can it be blocked. Guards are basically functions that do the below :
    - **Value Proposition:** 
      1. *User Access Control* : Guards can control to display the page or component based on user’s role and permissions.
      2. *Data Protection* : When user navigates from one page to another guards can protect the data from being lost.
      3. *Load Optimization* : Guards can prevent lazy loaded modules from loading until certain conditions are met.
      4. *Data Pre****-****Fetching* : Guards can fetch the data required for specific route in advance using *Resolve* Guards.
* **Data Binding:**
  + Displaying data in a template is done using {{ }} braces. The {{ and }} characters are an example of *data binding* that creates a relationship between templates and data.
  + An Angular component is responsible for managing the template and providing it with the data and logic it needs.
  + ** @angular/core 🡪** It contains most of the core functionality of Angular including support for components. The import statements don’t include file extensions that’s because the relationship between the target of an import and the file that is loaded by the browser is handled by Angular’s build tools.
  + The decorator provides metadata for the class. This is a @Component decorator that tells angular that it is a component.
* **Styles :** For Styling we can use in built ‘Angular Material’ package that many inbuilt modules with styles included.
  + ng add @angular/material@18.1.0 –defaults
* **Attribute Binding:** The square brackets [ ] denote attribute binding which is used to set an element attribute.
* **ng-container:** It is used to group content together
* **Two-way Binding: ** Events are represented by parenthesis and attributes are represented by square brackets, combination of both is called *banana in a box* syntax. ngModel is a angular feature that is used to set two way binging in form elements.
* **RxJS** 🡪 It is a library for asynchronous and event based programs by using observable sequences. It provides one core type, the Observable, satellite types (Observer , Schedulers , Subjects) and operators inspired by *Array* methods (map, reduce, filter, every etc.,) to allow handling asynchronous events as collections.
  + It combines the observer pattern, iterator pattern and collections to fill the need for an ideal way of managing sequences of events.
  + The essential concepts in RxJS which solve async event management are :
    - Observable 🡪 a basic idea of invokable collection of future values or events.
    - Observer 🡪 It is a collection of callbacks that knows how to listen to values delivered by the Observable.
    - Subscription 🡪 represents the execution of an Observable , this is primarly useful for cancelling the execution.
    - Operators 🡪 These are pure functions that enable a functional programming style of dealing with collections.
    - Subject 🡪 This emits the event about state of change like multicasting a value or event to multiple observers.
    - Schedulers 🡪 These are centralized dispatchers to control concurrency , allowing us to coordinate when computation happens.
* **ng new 🡪** This command is used to create projects along with arguments. Below are the arguments for ng new:



Below are the details for the folder structure:

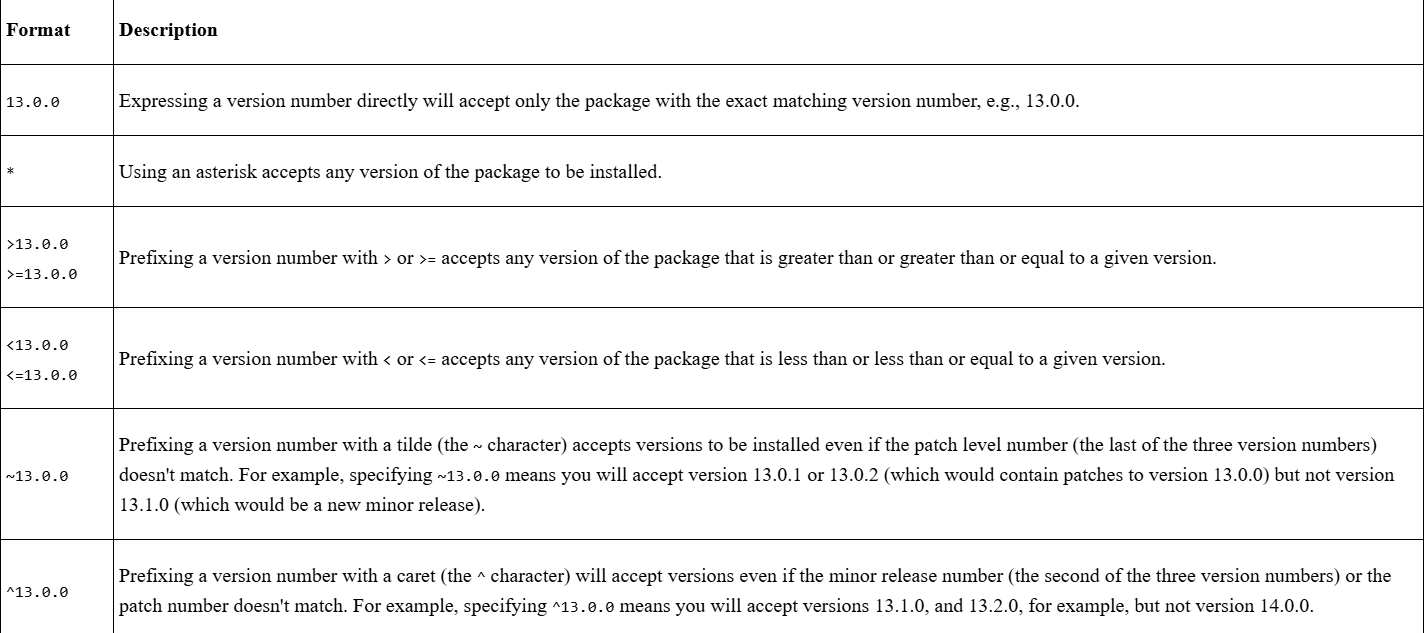




The JavaScript application development depends on lot of packages so of which are related to angular frameworks that need to sent to browser in small packages. There is a complex hierarchy of dependencies in between them which is really difficult to manage manually and hence we need a package manager.

Some JavaScript packages take advantage of *schematics API* provided by the *@angular/cli* package to automate the integration process. This is because the package provides angular specific functionality such as Angular material package. The npm install doesn’t understand the schematics API so the ng add command is used to download these packages.

The schematics API allows package authors to ask the user questions and use the responses during the integration process



**Hot Reloading**: During Development phase, Angular development tools add a feature called hot reloading. This means that when build is run once and URL created and if user does any code changes and saves the file the javascript code added to the bundle opens a connection back to the development server.

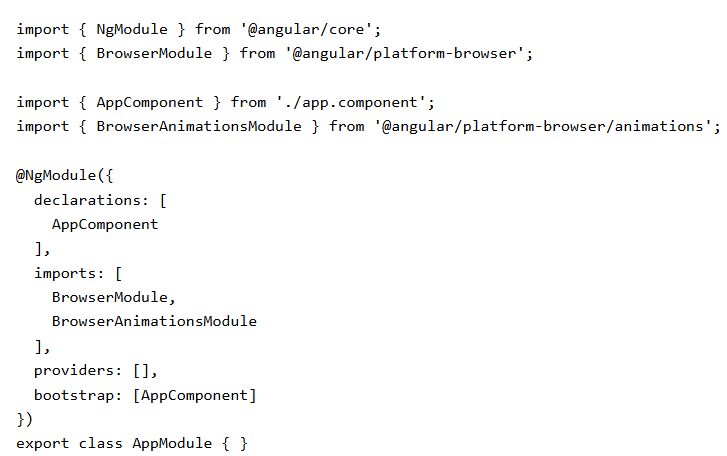
**Main.ts**

* In main.ts we use platformBrowserDynamic module from @angular/platform-browser-dynamic package to kick start Angular application in browser. Angular is created to run in different environments and hence we will need to inform it.

platformBrowserDynamic().bootstrap(AppModule).catch(error => console.error(error));

**Module:**

* The term *Module* does double duty Angular. It refers to both JavaScript module and Angular Module. JavaScript module is used to track the dependencies of the application and ensures that browser only received the code it needs whereas Angular module is used to configure a part of the application.
* Angular application has root module. If we use *ng new* command to create the application then AppModule will be root application. *@NgModule* decorator has few configuration need to inform Angular like bootstrap which tells Angular that it should ‘AppComponent’ as part of application startup process.

****

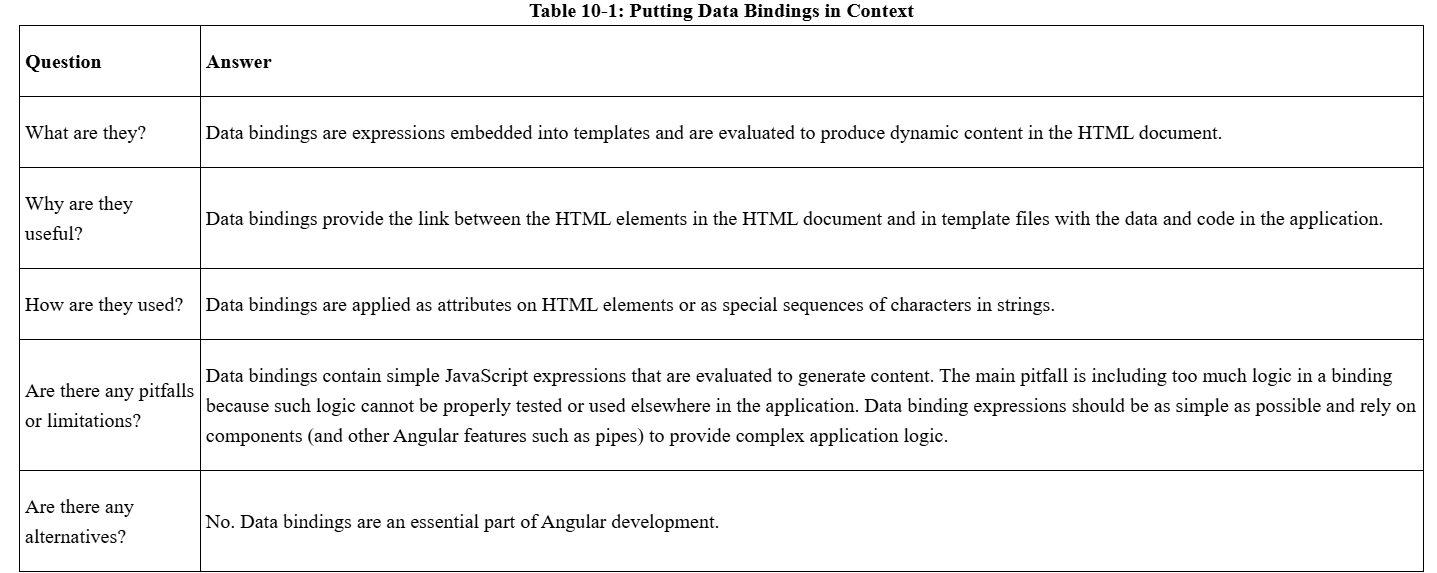
Components are main building block in Angular and content provided by the component AppComponent will be displayed to the user.

When application starts Angular processes index.html and looks for *app-root* selector to replace the html content from *root component’s* template URL along with style URL properties.

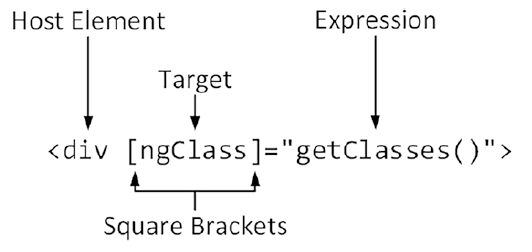
**Production Process**

* ng serve command doesn’t bundle the files and hence in development mode the file sizes are bit larger.
* ng build performs the production compilation process and it does bundles and shrinks the files in small sizes.

**Data Binding**:

****

*One-way data bindings* are used to generate content for the user and are the basic feature used in Angular templates. The term one-way refers to the fact that the data flows in one direction, meaning that data flows from the component to the data binding so that it can be displayed in a template.



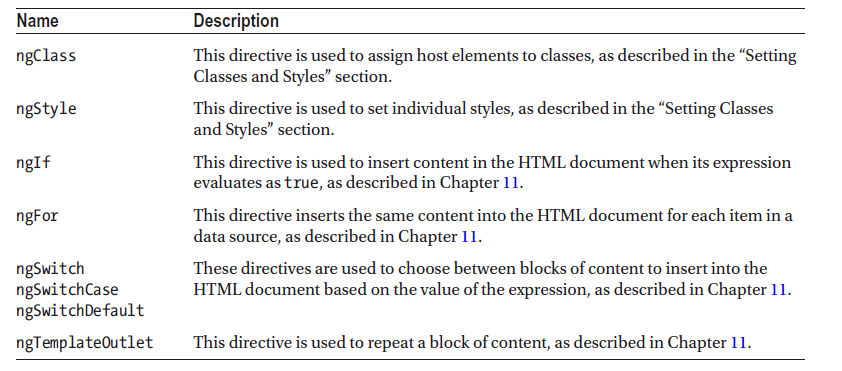
**Host Element** 🡪 HTML element

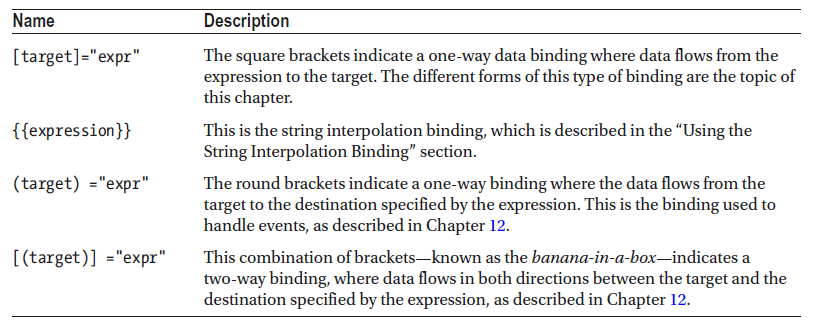
**Target**🡪 It specifies what the binding will do, it is either property or directive binding.

**Square Brackets [] 🡪** It specifies one way binding and Angular evaluates the expression and passes result to the target. If no square brackets exist the expression inside will be sent as literal value to the browser.

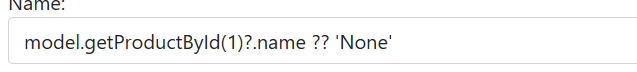
**Expression 🡪** It’s a fragment of JavaScript that is evaluated using the template’s component to provide context to the target.

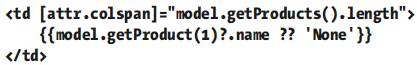
If we keep more complex logic in expressions then this can’t be checked by TypeScript complier nor can they be tested which bugs are more likely to remain undetected until the application is deployed. Hence expressions should be as simple as possible ideally only used to get the data from the component and may be format it for display.



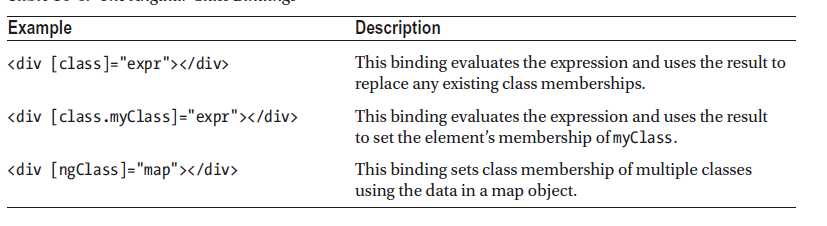


**Property Binding:** The property binding let’s us set an expression for the property as a target (one way binding) for an element.

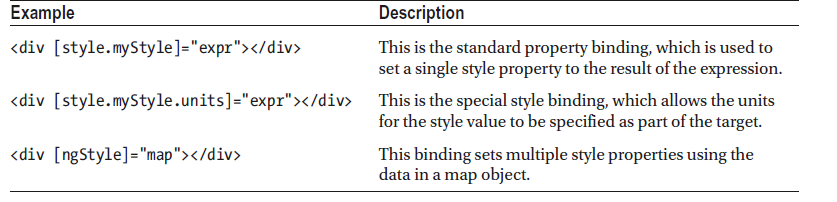
 and output will be like this 🡪 .The square brackets indicate one-way binding and if we are going to remove the square brackets the output will be like this 🡪 

**Attribute Binding:** Angular provides the attribute binding which is used to set an attribute on the host element rather than setting a value of the JavaScript object. The attribute binding is applied by defining a target that prefixes the value along with word “attr.” like below: 

**Class Bindings :**

****

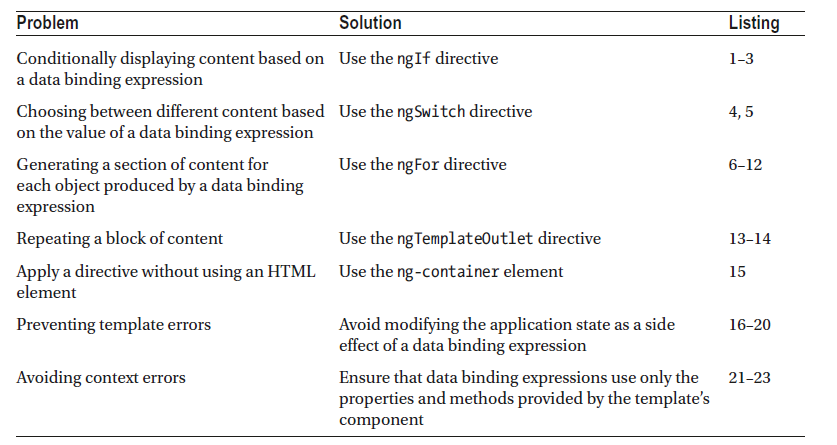
**Style Bindings:**

****

Bindings are worth understanding because their expressions are re-evaluated when the data they depend on changes.

When Angular bootstraps the application it creates an *ApplicationRef* object to represent the application. The tick method starts the Angular change detection process, where Angular looks at the data in the application and the expressions in the data binding and processes any changes. The data bindings in the template use specific array indexes to display data, and now that an object has been removed from the model, the bindings will be updated to display new values.

**Built-in Directives:** These are responsible for selectively displaying content, repeating content and choosing between different fragment contents in the HTML which are common tasks in web apps. These directives are applied to HTML elements in templates.



**ngIf** is the simplest of the built-in directives and is used to include a fragment of HTML in the document when an expression evaluates as true. It adds and removes elements from the HTML document, rather than just showing or hiding them. Use property or style bindings if you want to leave elements in place and control their visibility.

**Micro-Template Directive or Structural Directives:** Some directives, such as ngFor, ngIf, and the nested directives used with ngSwitch, are prefixed with an asterisk, as in \*ngFor, \*ngIf, and \*ngSwitch. The asterisk is shorthand for using directives that rely on content provided as part of the template, known as a *micro-template*. Directives that use micro-templates are known as *structural directives* and its content as the micro-template for each of the objects that it processes. Behind the scenes, Angular expands the micro-template and the directive like this:

<ng-template ngIf="model.getProductCount() > 4">

<div class="bg-info p-2 mt-1">

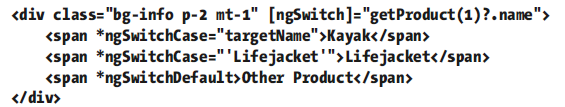
There are more than 4 products in the model

</div>

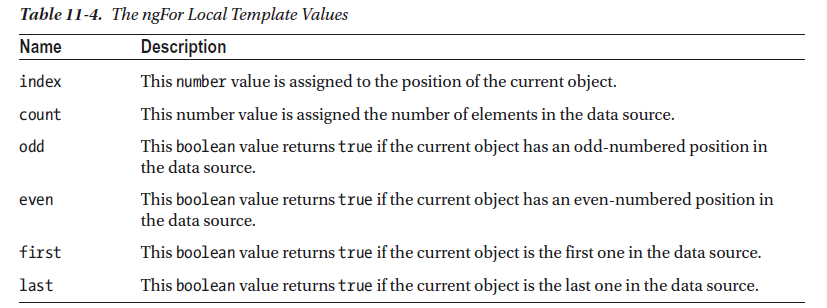
</ng-template>

You can use either syntax in your templates, but if you use the compact syntax, then you must remember to use the asterisk.

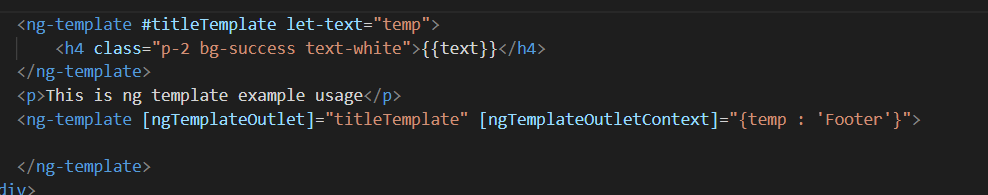
**ngSwitch:** The values assigned to the **ngSwitchCase** directives are also expressions, which means you can invoke

methods. If you want to compare a result to a specific string, then you must double quote it, like this: 

**ngFor:** This directive repeats a section of content for each object in an array, providing the template equivalent of a foreach loop.



**ngTemplateOutlet:** This directive is used to repeat the block of content at specified location which can be useful when we generate same content in different places and want to avoid duplication.

****

**Events & Forms:**

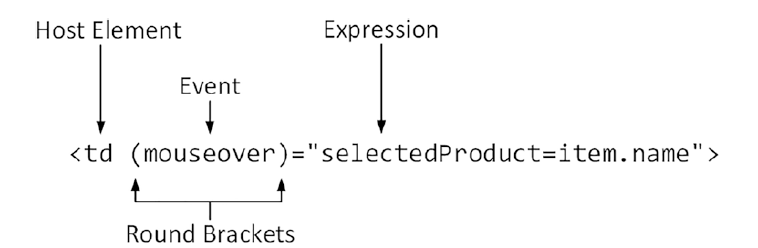
**Event Binding 🡪**  It is used to respond to events by the user on the host element. An event binding has these four parts:

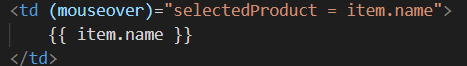
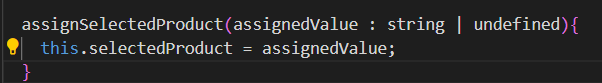
• The host element is the source of events for the binding.

• The round brackets tell Angular that this is an event binding, which is a form of oneway

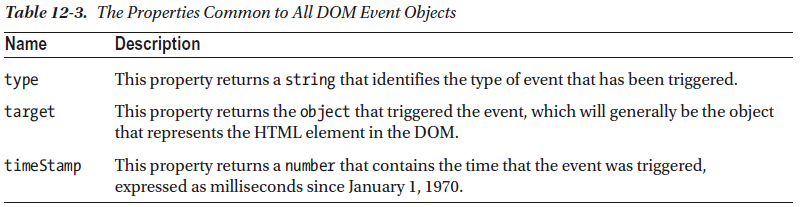
binding where data flows from the element to the rest of the application.

• The event specifies which event the binding is for.

• The expression is evaluated when the event is triggered. 

Example : In template 🡪  OR  assign the value in template itself and write respective method in component.ts 🡪 

When browser triggers an event it provides an event object that describes it. Below are the properties that describe



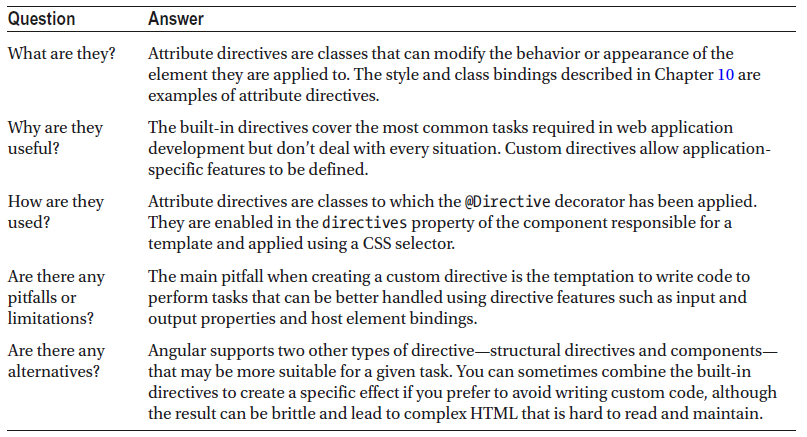
The Event object is assigned to a template variable $event. When event is triggers browser’s DOM API creates *InputEvent* object and this is assigned to $event.

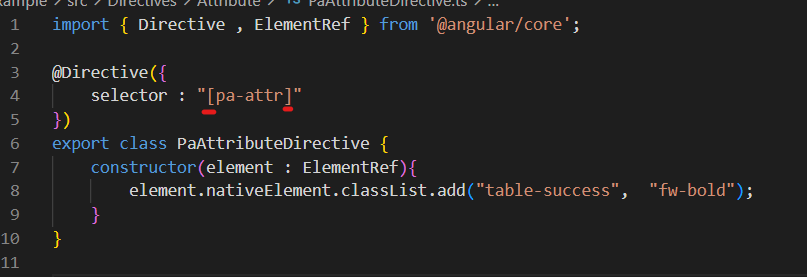
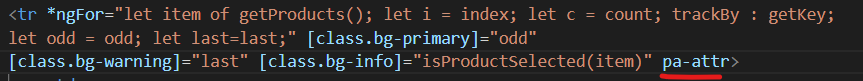
*InputEvent* has a target element which is *HTMLInputElement* object which is how DOM represents the input element that triggered the event. The *HTMLInputElement.value* returns the target value which is how $event.target.value is obtained.

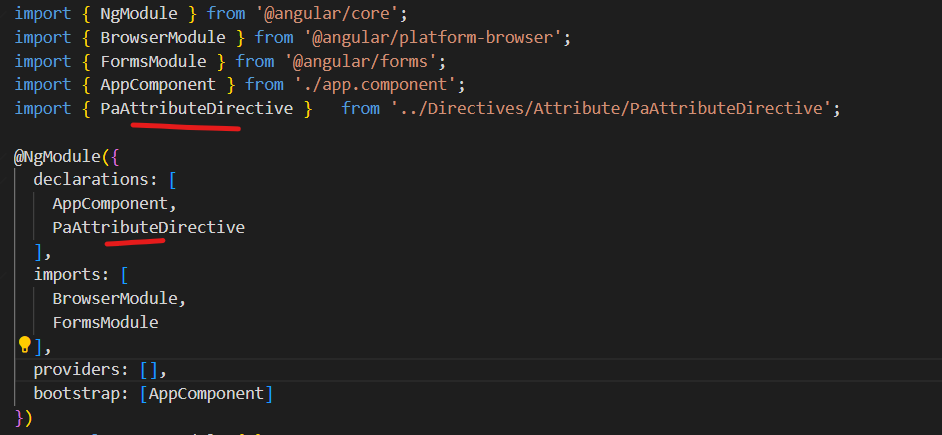
Angular by default doesn’t allow type assertions in templates so we will have to use **$any** template variable to convert the event object like this 🡪 $any($event).target.value But remember that **$any** template variable disables compiler type checks.

…

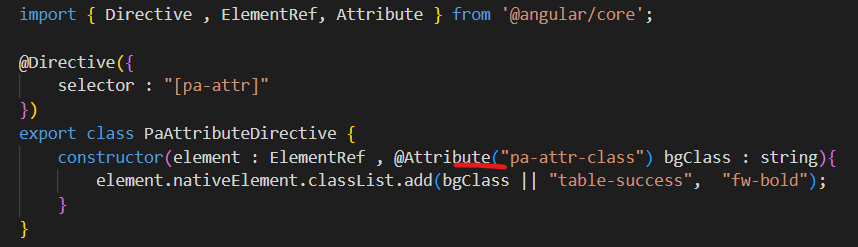
**Attribute Directives:** These are used to change the appearance or behaviour of a single element.



Attribute directive should always have  square brackets in their names and they can be added to any element like below : 

And add the directive in app.module.ts 🡪 

The @Attribute decorator defined in angular/core specifies the name of the attribute that should be used to provide a value in the constructor parameter when a new instance of directive class is being created.

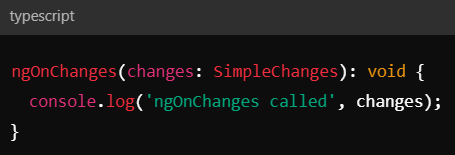
Angular creates a new instance of the decorator for each element that matches the selector and uses the element’s attributes to provide the value for the directive constructor arguments that have been decorated with @Attribute. 

The null coalescing operator (||) says that it want to use a fallback value if an element defines **pa-attr-class** attribute but doesn’t assign it a value in which case the bgClass will be assigned to empty string and the operator ( || ) evaluates to false.

Input Properties or Data Bound Input Properties: Directives receive expressions using *data-bound input properties*, also known as *input properties* or, simply, *inputs*. Input properties are defined by applying the @Input decorator to a property and using it to specify the name of the attribute that contains the expression.We don’t need to provide an argument to the @Input decorator if the name of the property corresponds to the name of the attribute on the host element. So, if you apply @Input() to a property called myVal, then Angular will look for a myVal attribute on the host element.

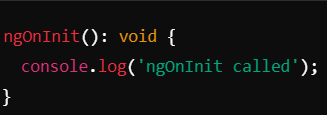
Life Cycle of components (or directive):  
**1. ngOnChanges()**

* **When**: Called before **ngOnInit** and whenever one or more data-bound input properties change.
* **Purpose**: Respond to changes in input properties. This is useful for reacting to changes that need to be handled before the component's view is updated.



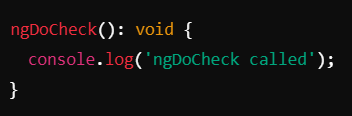
**2. ngOnInit()**

* **When**: Called once, after the first ngOnChanges. It’s used to initialize component properties.
* **Purpose**: Perform component initialization, like fetching data or setting up initial state.



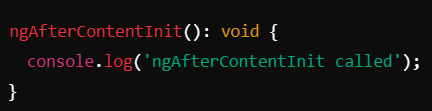
**3. ngDoCheck()**

* **When**: Called during every change detection run, immediately after ngOnChanges and ngOnInit.
* **Purpose**: Implement custom change detection. Useful if you need to perform additional checks or update component state based on changes.



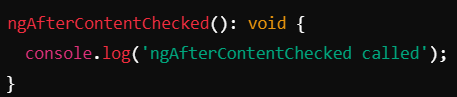
**4. ngAfterContentInit()**

* **When**: Called once after the first ngDoCheck, after Angular has projected content into the component’s view.
* **Purpose**: Respond to the initialization of projected content. This is useful when working with content projection (ng-content).



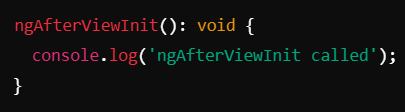
**5. ngAfterContentChecked()**

* **When**: Called after ngAfterContentInit and every subsequent ngDoCheck.
* **Purpose**: Respond to changes in projected content. It’s useful for reacting to changes in content that Angular has projected into the component’s view.



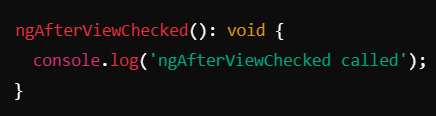
**6. ngAfterViewInit()**

* **When**: Called once after the first ngAfterContentChecked, after Angular has fully initialized the component's view and child views.
* **Purpose**: Respond to the initialization of the component's view and its child views. Ideal for DOM manipulation and view-related operations.



**7. ngAfterViewChecked()**

* **When**: Called after ngAfterViewInit and every subsequent ngAfterContentChecked.
* **Purpose**: Respond to changes in the component's view and its child views. Useful for reacting to view changes after Angular has completed its change detection.



**8. ngOnDestroy()**

* **When**: Called once, just before Angular destroys the component.
* **Purpose**: Clean up resources, such as unsubscribing from Observables, detaching event handlers, or other cleanup tasks.

