Angular 8

Presenter: Venkatesh Reddy Madduri

Date: 7th Nov, 19

Angular - Introduction

- Angular is a Framework of **JavaScript** used to build web and mobile applications. Angular 8 is a client-side TypeScript based structure which is used to create dynamic web applications. Its first version was released by **Google** in **2012** and named as Angular JS. Angular 8 is the updated version of Angular2.
- Before starting Angular, we must have a good understanding of **JavaScript**, **HTML**, **CSS, AJAX, and TypeScript.**
- Angular 8 is a great **UI** (User Interface) library for the developers. Angular is a reusable UI component helps us constructing attractive, consistent, and functional web pages and web application. Angular 8 is a JavaScript framework which makes us able to create an attractive **Single Page Applications** (SPAs). *"A single page application is a web application or a website which provides a fluid, reactive, and fast application same as a desktop application. It contains menu, buttons, and blocks on a single page and when a user clicks on them, it dynamically rewrites the current page without loading new pages from the server so that its speed is fast."*

Angular – Introduction – Continu..

- Angular 8 has entirely based on component and consist of some tree structures with parent and child component. Angular 8 classes are created in such a way that the web page can fit in any screen size so that they are fully compatible with mobiles, tablets, laptops, and large systems.

Angular Version History

Version	Released
Angular JS	**October 2010**
Angular2.0	**September 2016**
Angular4.0	**March 2017**
Angular5.0	**November 2017**
Angular6.0	**May 2018**
Angular 8.0	**October 2018**

Angular Key Terms

What is Angular CDK?

The **Component Dev Kit (CDK**) is a set of tools that implement common interaction patterns while being preserve about their presentation. It represents an abstraction of the core functionalities found in the Angular Material library, without any styling specifically to Material Design.

What is Angular CLI?

Angular CLI is known as **Angular Command Line Interface**. It is a command-line tool for creating angular apps. It is mentioned to use angular CLI for creating angular apps as if we do not need to spend time to **install and configur**e all the required dependencies and wiring everything together. Angular CLI is a helpful tool to create and work with Angular Applications efficiently.

What is Ng in Angular?

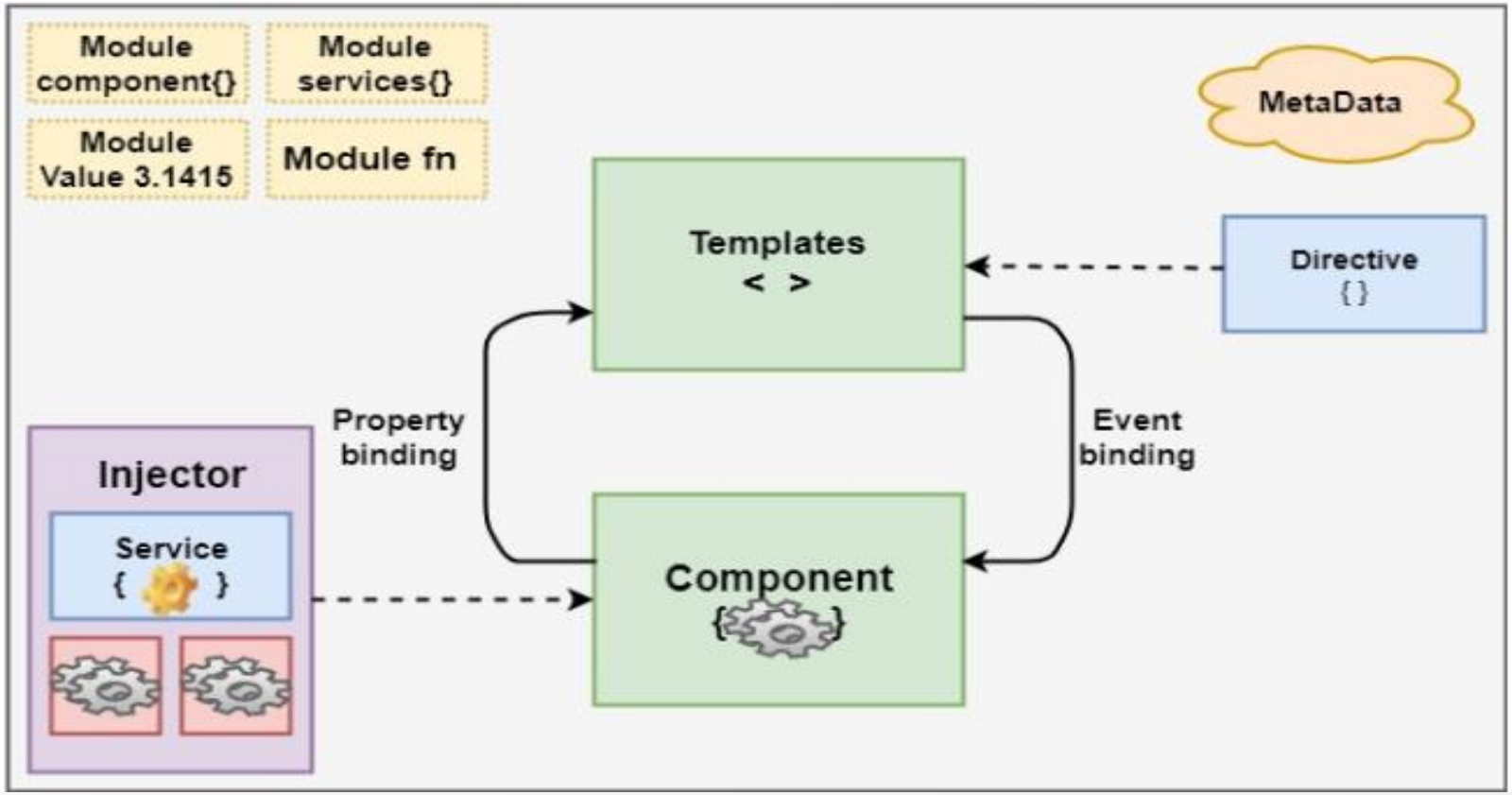
The prefix ng stands for “Angular;” all of the built-in directives that craft with Angular use that prefix. Similarly, it is suggested that you do not use the ng prefix on your instructions to avoid possible name impacts in future versions of Angular.

Architecture

- Angular 8 is a platform and framework which is used to create clients applications in HTML and Typescript. *Angular 8 is written in Typescript*. Typescript is a superset of JavaScript.
- Angular 8 implements core and optional functionality as a set of Typescript libraries which we can import in our application.
- **NgModules** are the *basic building blocks of Angular 8 application*. They provide a compilation context for components.

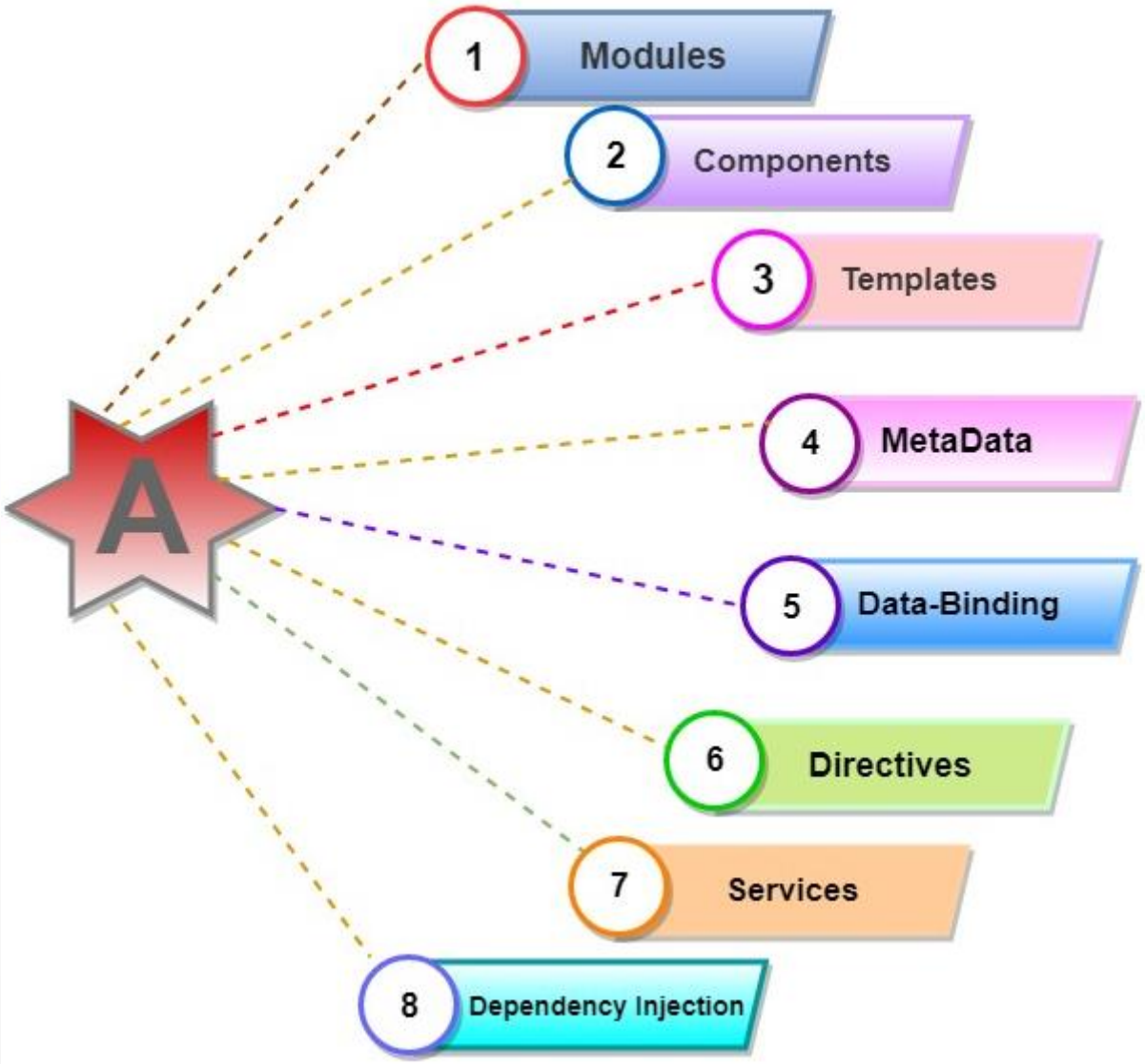
Various units are combined to build an angular application, which is as follows.

Architecture



- Angular 8 always has at least a root module which enables bootstrapping.
- Components define views, which are set of screen element that is modified according to your program logic and data in Angular 8.

Architecture essentials



previous next

Architecture essentials – Contin…

1. Modules:

- Every Angular 8 app has provided the bootstrap mechanism that launches the application.
- Generally, every angular 8 modules contain many functional modules.
- Some interactive features of Angular 8 modules:
- NgModules import the functionality from another NgModules just like other JavaScript modules.
- NgModules allow their functionality to be imported and used by other modules, e.g., if we want to use route service in our app, we can import the Routing Ng module.

2. Components:

- Every angular project has at least 1 component, the root component, and the root components connect the component with a page Document Object Module (DOM). Each component defines a class which contains data, application, logic, and it is binding with the HTML template.

Click here for the answer

```
@Component({
selector: 'app-root',
TemplateUrl: .' /app.component.html',
StyleUrls: ['. /app.component.css']
})
```

Architecture essentials – Contin…

3. Templates :

- The angular template integrates the HTML with Angular mark-up that can modify HTML elements before they are displayed. It provides program logic, and binding mark-up connects to your application data and the DOM.

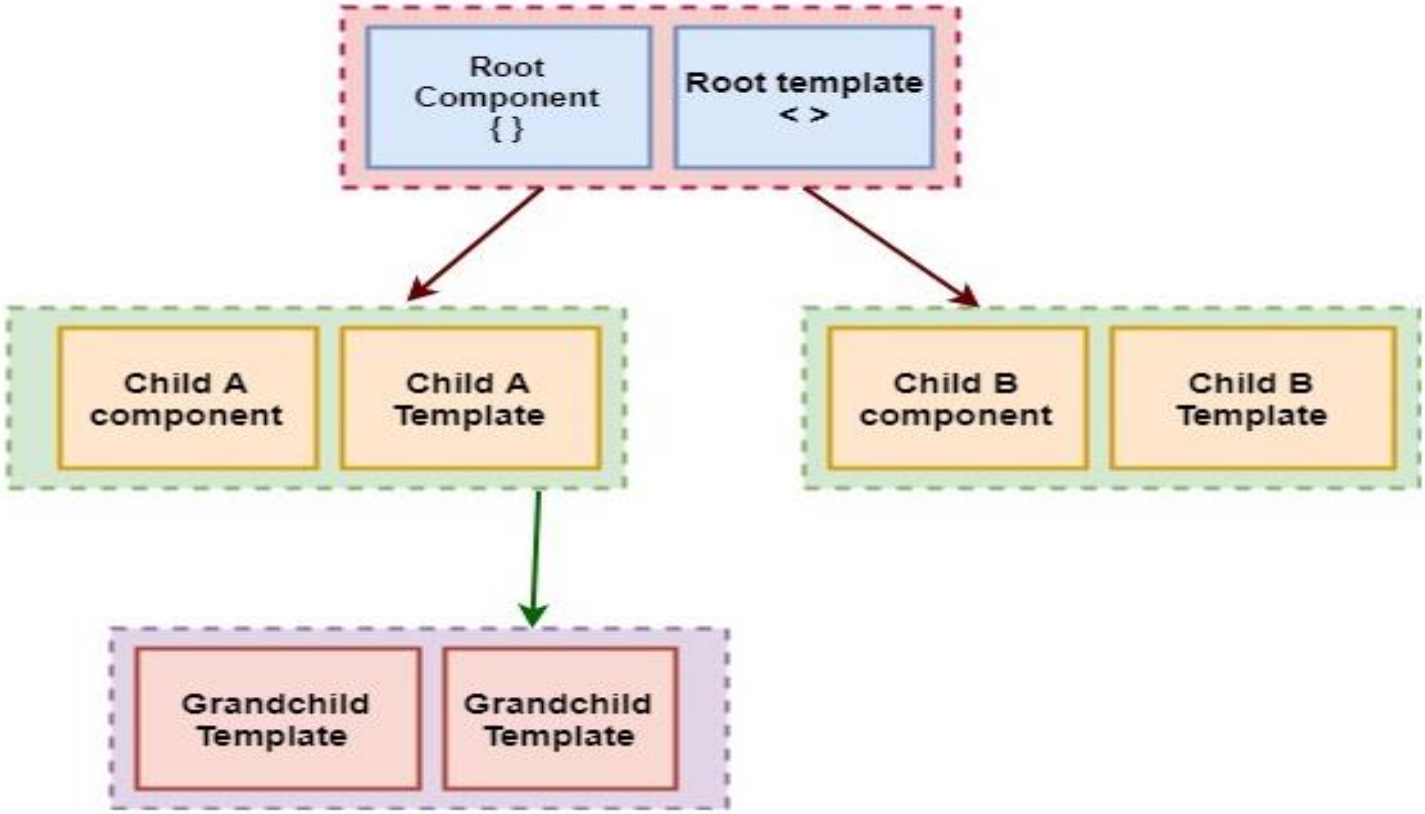
```
<div style="text-align: center">
<h1>
{{2| power: 5}}
</h1>
</div>
```

Click here for the answer

previous next

Architecture essentials – Contin…

In the above HTML file, we have been using a template. We have used the pipe inside the template to convert the values to its desired output.



previous next

Architecture essentials – Contin…

4. Metadata :

- Decorators are the metadata in Angular. It is used to enhance the class so it can configure the expected behavior of a class. Decorators are the core concept of when developing with Angular. User can use metadata in a class to tell Angular app that app component is a component. Metadata can attach to the Typescript through the decorator.

```
@Component ({
 selector: 'app-root',
 templateUrl: './app.component.html',
 styleUrls: ['./app.component.css']
 }]
```

- @pipe, @directive

Click here for the answer

previous next

Architecture essentials – Contin…

5. Data binding :

- Angular allows communication between a component and a DOM. Making it very easy to define interactive application without pulling and pushing the data.

  There are two types of data binding.

- **Event Binding:** Our app responds to user input in the target environment by updating our application data.
- **Property Binding:** Interpolates values that are computed from application data into the HTML.
- Interpolation: {{value}} Interpolation adds the value of the property to the component.

```
<p>Name: {{student.name}} </p>
<p>College: {{student. college}} </p>
```

Property binding: [property]=”value”

A value has been passed from a component to a special property, with property binding which can be a simple html attribute.

```
<input type=""text"" [value]=""student.name”/">
<input type=""text"" [value]=""student." college”="">
```

previous next

Architecture essentials – Contin…

5. Data binding :

- Angular allows communication between a component and a DOM. Making it very easy to define interactive application without pulling and pushing the data.

  There are two types of data binding.

- **Event Binding:** Our app responds to user input in the target environment by updating our application data.
- **Property Binding:** Interpolates values that are computed from application data into the HTML.
- Interpolation: {{value}} Interpolation adds the value of the property to the component.

```
<p>Name: {{student.name}} </p>
<p>College: {{student. college}} </p>
```

Property binding: [property]=”value”

Click here for the answer

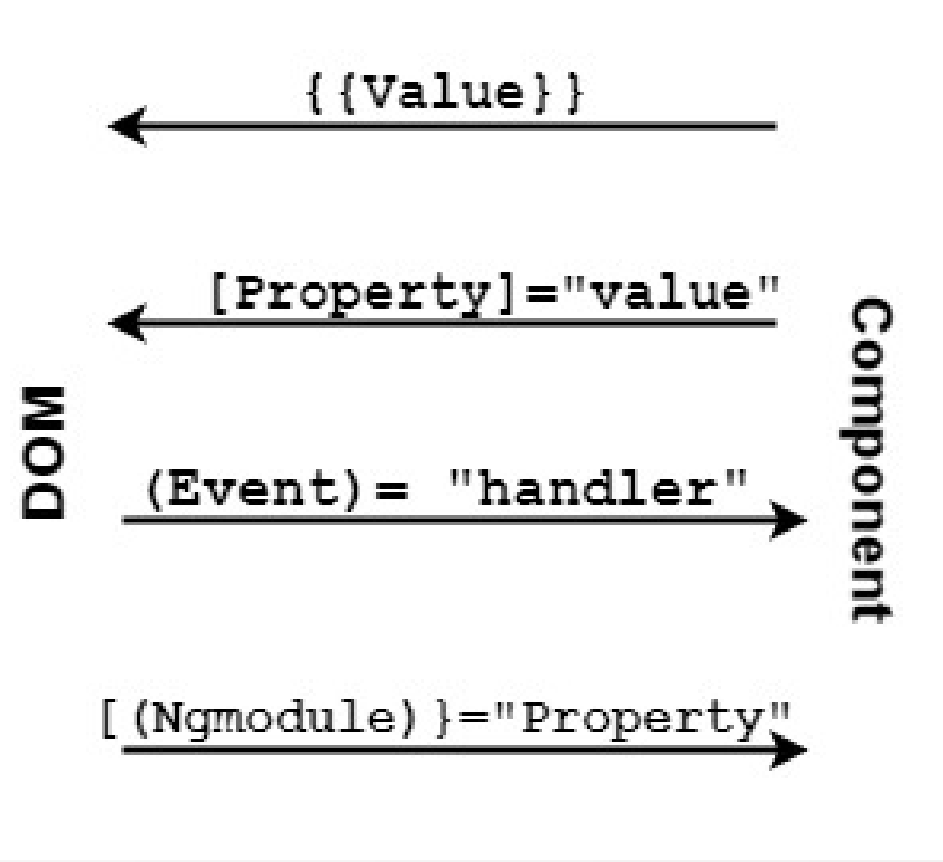
A value has been passed from a component to a special property, with property binding which can be a simple html attribute.

```
<input type=""text"" [value]=""student.name”/">
<input type=""text"" [value]=""student." college”="">
```

previous next

Architecture essentials – Contin…

5. Data binding :



Architecture essentials – Contin…

6. Directives:

- Directives used for expanding the functionality of HTML elements. There are three types of directives in Angular are Component , Structural Directive and Attribute directive.

  - Components—directives with a template.
  - Structural directives—change the DOM layout by adding and removing DOM elements.
  - Attribute directives—change the appearance or behavior of an element, component, or another directive.

- Components are the most common of the three directives.
- Structural Directives change the structure of the view. Two examples are NgFor and NgIf.
- Attribute directives are used as attributes of elements. Angular directives like: ngClass which is a better example of excising angular attribute directive.

```
<p [ngClass= "{'coffee' =true, 'red'=false}">
 Angular8  Directives Example
</p>
<style>
.coffee{color: coffee}
.red{color: red}
</style>
```

```
<div *ngIf="hero" class="name">{{hero.name}}</div>
```

previous next

Architecture essentials – Contin…

7. Services:

Services used to reuse the code. The services can be used in more than one component. The decorator provides the meta-data that allows our services to be injected into the client component as a dependency.

Why Services:

- Components shouldn't fetch or save data directly and they certainly shouldn't knowingly present fake data. They should focus on presenting data and delegate data access to a service.

```
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root',
})
export class HeroService {

  constructor() { }

}
```

```
constructor(private heroService: HeroService) { }
```

Architecture essentials – Contin…

8. Dependency Injection:

- It is a design pattern for efficiency and modularity. **DI** is wired into Angular into an Angular framework and used everywhere to provide new component with new services.

```
constructor(private heroService: HeroService) { }
```

Pipes

- A class with the @Pipe decorator defines a function which transforms an input value to output value for display in an amount.
- Angular defines various pipes like data pipe and currency pipe; for a complete list and we can also define new pipes. To specify a value transformation in a HTML template, use the **pipe operator (|).**

```
{{interpolated_value | pipe_name}}
```

Installation –

To install Angular 8, we require the following things

- OS (Windows 10/Linux)
- Node.js (12.6.0)
- NPM
- Angular CLI(Command Line Interface)
- An IDE for writing our code(VS code)
- Git

After installing these in our system, we have to know what these things are:

Node.js: Node.js is open-source, cross-platform JavaScript runtime environment that executes JavaScript code outside a browser. And it is developed by **Ryan Dahl** wrote in C, C++, and JavaScript. It is free to use and run on various platforms (Windows, Linux, UNIX, Mac, etc. It can generate dynamic page content and can add, delete, and modify data in the database. Node.js file have extension **".js".**

NPM: NPM is a package manager in node.js packages. **www.npmjs.com** hosts millions of free packages to download and use. The NPM program is installed in our computer when we install node.js. A package in node.js contains the entire package we need for a module. Downloading a package is very compatible. Open the command-line interface and say NPM to download the packages which we want. If we want to download a package called **"upper-case":**

```
C:\Users\e>npm install upper-case
```

Installation –

To install Angular 8, we require the following things

Angular CLI: Angular CLI is known as Angular Command-line interface. An Angular CLI is used for creating a project. It can be used to create content, services, pipes, directives, and many more; also, it helps in building, serving, testing, etc. It makes Angular Development workflow much easier and faster.

IDE (Visual Studio): Microsoft Visual Studio is an integrated development environment developed by **Microsoft** and it is written in C++ and C#. It is used to create computer programs, as well as websites, web apps, and mobile applications. The visual studio uses Microsoft Software Development platforms such as Windows API, Windows Forms, Windows Presentation Foundation, Windows Store, and Microsoft Silver light. It has produced both native codes and managed code.

VS code is light and easy to set up. It is free to use and provide a massive number of extensions that increase your productivity.

We can download the VS Code from here: **https://code.visualstudio.com**.

Git: Git is a distributed version control system used for tracking changes in source code during software development. It is designed for spreading work among programmers. But it is used to track any set of change in files. **Linus Torvalds** created it in 2005 for developing the Linux kernel. Git is free and open-source software distribution system under the term of the GNU(General Public License) version 2.

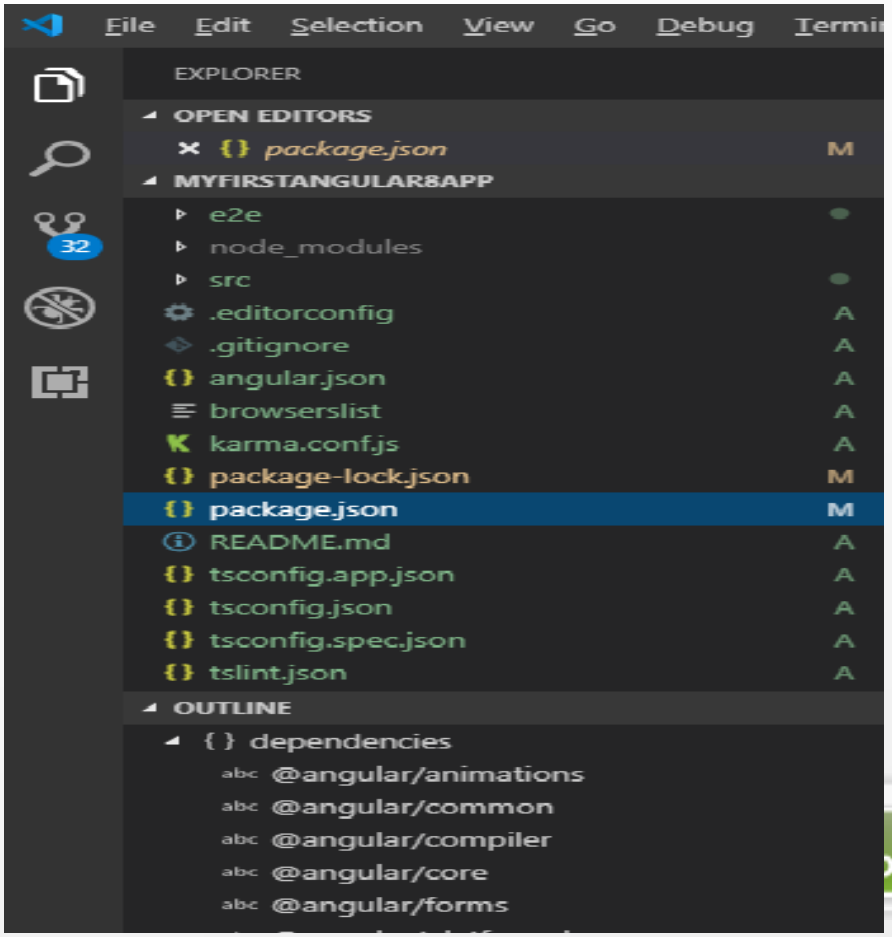
Angular Project Creation –

ng new ng-training

```
C:\Venkys\SME Training\ng\training>ng new ng-training
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? CSS
CREATE ng-training/angular.json (3465 bytes)
CREATE ng-training/package.json (1284 bytes)
CREATE ng-training/README.md (1027 bytes)
CREATE ng-training/tsconfig.json (470 bytes)
CREATE ng-training/tslint.json (1985 bytes)
CREATE ng-training/.editorconfig (246 bytes)
CREATE ng-training/.gitignore (629 bytes)
CREATE ng-training/browserslist (429 bytes)
CREATE ng-training/karma.conf.js (1023 bytes)
CREATE ng-training/tsconfig.app.json (210 bytes)
CREATE ng-training/tsconfig.spec.json (270 bytes)
CREATE ng-training/src/favicon.ico (5430 bytes)
CREATE ng-training/src/index.html (297 bytes)
CREATE ng-training/src/main.ts (372 bytes)
CREATE ng-training/src/polyfills.ts (2838 bytes)
CREATE ng-training/src/styles.css (80 bytes)
CREATE ng-training/src/test.ts (642 bytes)
CREATE ng-training/src/assets/.gitkeep (0 bytes)
CREATE ng-training/src/environments/environment.prod.ts (51 bytes)
CREATE ng-training/src/environments/environment.ts (662 bytes)
CREATE ng-training/src/app/app-routing.module.ts (245 bytes)
CREATE ng-training/src/app/app.module.ts (393 bytes)
CREATE ng-training/src/app/app.component.html (1152 bytes)
CREATE ng-training/src/app/app.component.spec.ts (1110 bytes)
CREATE ng-training/src/app/app.component.ts (215 bytes)
CREATE ng-training/src/app/app.component.css (0 bytes)
CREATE ng-training/e2e/protractor.conf.js (810 bytes)
CREATE ng-training/e2e/tsconfig.json (214 bytes)
CREATE ng-training/e2e/src/app.e2e-spec.ts (640 bytes)
CREATE ng-training/e2e/src/app.po.ts (251 bytes)
[..................] / fetchMetadata: sill pacote version manifest for webpack-subresource-integrity@1.1.0-rc.6 fetched in 2674ms
```

Angular Project File Structure –

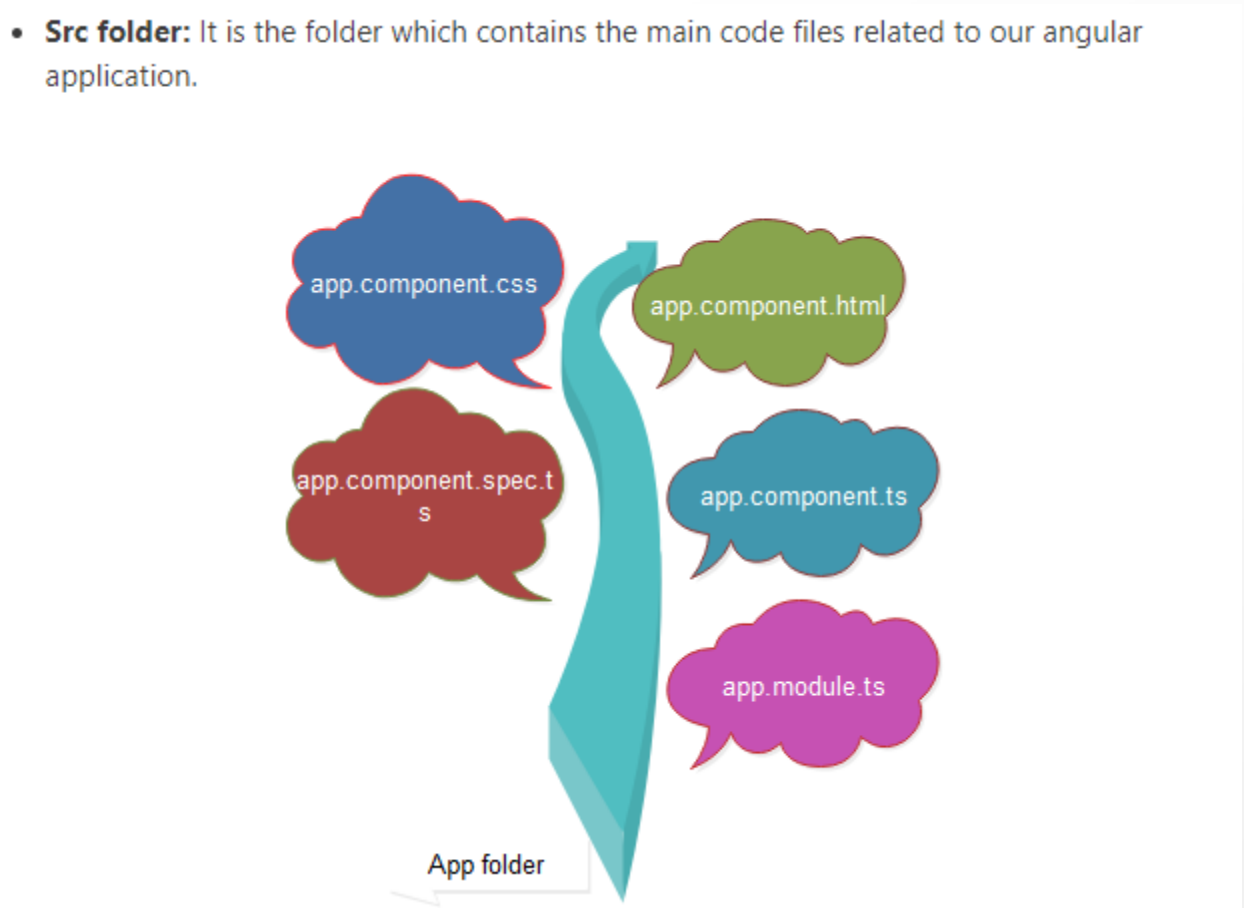
- A file structure contains the files for one or more projects. A project is the set of files that comprise an application on a shareable library. An angular module is used to group related angular components, services, directives, etc.
- Let see the structure of the Angular 8 app on VS code IDE for Angular development, and we can use either Web Storm IDE or Visual Studio IDE. Both are better. Here, we are using the VScode IDE.



Angular Project File Structure –

➢ Src folder

- **Src folder:** It is the folder which contains the main code files related to our angular application.



Angular Project File Structure –

- **Src folder**

•**app folder:** It contains the files which we have created for app components.
•**app.component.css**: The file contains the **CSS**(cascading style sheets) code in our app component.
•**app.component.html:** The file contains the **HTML** file related to its app component. Itis the template file which is specially used by angular to the data binding.
•**app.component.spec.ts:** This file is a unit testing file is related to the app component. This file is used across with more other unit tests. It is run from angular CLI by command ng test.
•**app.component.ts:** It is the essential typescript file which includes the view logic beyond the component.
•**app.module.ts:** It is also a typescript file which consists of all dependencies for the website. The data is used to define the needed modules has been imported, the components to be declared, and the main element to be bootstrapped.

Angular Project File Structure –

Other Important Files

- **package.json**: It is the npm configuration file. It includes details of our website's and package dependencies with more information about our site being a package itself.
- **package-lock.json:** This is an auto-generated and transforms file that gets updated when npm does an operation related to the node_modules or package.json file.
- **angular.json:**It is a necessary configuration file related to our angular application. It defines the structure of our app and includes any setting to accomplish with the application.
- **.gitignore**: The record is related to the source code git.
- **.editorconfig:** This is a standard file which is used to maintain consistency in code editors to organizing some basics. such as indentation and whitespaces.
- **Assets folder:** This folder is a placeholder for the resource files which are used in the application such as images, locales, translations, etc.
- **Environment folder:** The environment folder is used to grasp the environment configuration constants that help when we are building the angular application.
- **Browser list:** This file specifies a small icon that appears next to the browser tab of a website.
- **favicon.ico:**It defines a small image that appears next to the browser tab of any website.

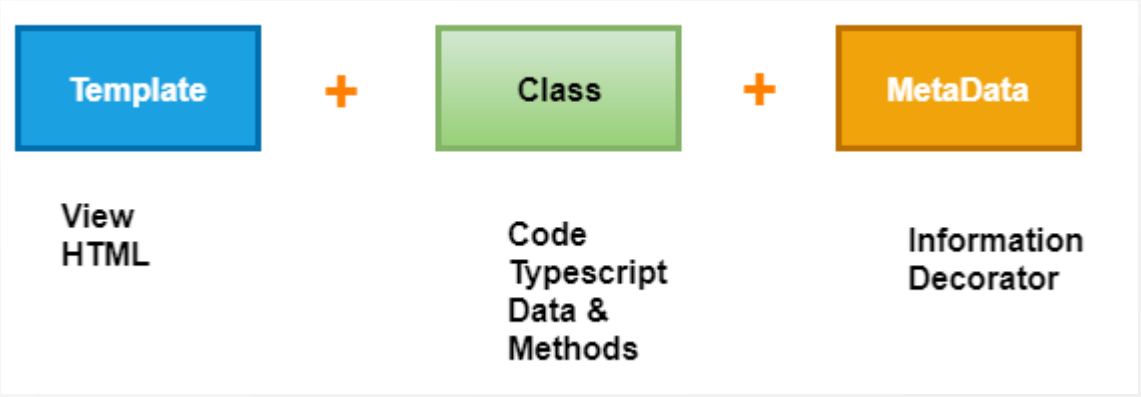
Angular Project File Structure –

Other Important Files

- **Index.html:** It is the entry file which holds the high-level container for the angular application.
- **karma.config.js:** It specifies the config file in the karma Test runner, Karma has been developed by the AngularJS team and can run tests for AngularJS and Angular 2+.
- **main.ts**: This is the main ts file that will run; first, It is mainly used to define the global configurations.
- **polyfills.ts**: The record is a set of code that can be used to provide compatibility support for older browsers. Angular 8 code is written in **ES6+** specifications**.**
- **test.ts:**It is the primary test file that the Angular CLI command ng test will apply to traverse all the unit tests within the application.
- **styles.css:** It is the angular application uses a global CSS.
- **tsconfig.json:**This is a typescript compiler of the configuration file.
- **tsconfig.app.json:** It is used to override the ts.config.json file with app-specific configurations.
- **tsconfig.spec.json:** It overrides the tsconfig.json file with the app-specific unit test cases.

Angular Components

- Angular is used for *building mobile and desktop web applications*. The component is the **basic building block of Angular**. It has a selector, template, style, and other properties, and it specifies the metadata required to process the component.
- Components are defined using a **@component** decorator and a tree of the angular component. It makes our complex application in reusable parts which we can reuse easily.
- The component is the most critical concept in Angular and a tree of components with a root component. The root component is one contained in the bootstrap array in the main **ngModule** module defined in the **app.module.ts** file.
- One crucial aspect of components is re-usability. A component can be reused throughout the application and even in other applications. Standard and repeatable code that performs a specific task can be encapsulated into a reusable component.



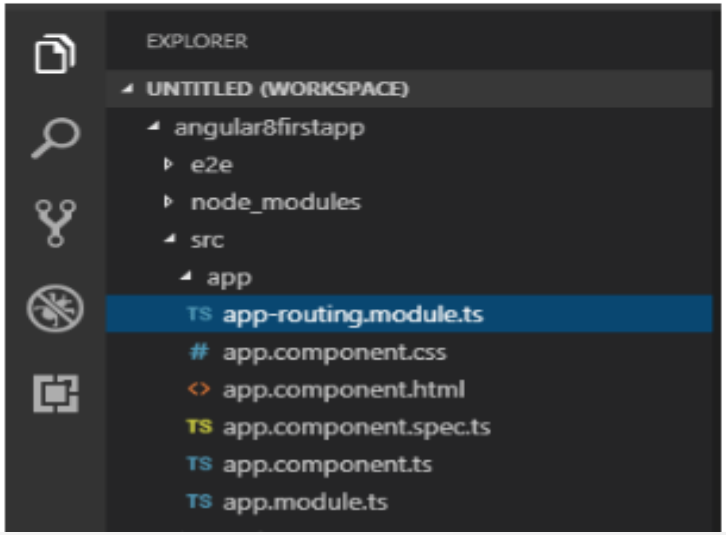
Angular Components Contin…

What is Component-Based Architecture?
An Angular application is build of some components which form a tree structure with parent and child components.
A component is an independent block of an extensive system that communicates with the other building blocks of the systems using inputs and outputs. It has an associated view, data, and behavior and has parent and child components.
The component allows maximum re-usability, secure testing, maintenance, and separation of concerns. Go to our Angular project folder and open the src/app folder, which we will see the following files.
App folder: The app folder contains the data we have created for app components.

Angular Components Contin…

App folder: The app folder contains the data we have created for app components.

- **app.component.css**: The component CSS file.
- **app.component.html**: This component is used for HTML view.
- **app.component.spec.ts**: The HTML view of the component
- **app.component.ts**: Component code (data and behavior)
- **app.module.ts**: The main application module.



Angular Components Contin…

Firstly, we import the Component decorator from **@angular/core** and then we use it to preserve the Typescript AppComponent class. The Component decorator takes an object with multiple parameters, however:

•**selector:** It specifies the tag that can be used to call this component in HTML templates just like the standard HTML tags

•**templateUrl:** It indicates the path of the HTML template that will be used to display this component.

•**styleUrls:** It specifies an array of URLs for CSS style-sheet in the component.

The export keyboard is used to export the component, and it has imported from other components and modules in the application file.

The title variable is a member variable which holds the string' app,' and it is not the part of the legal definition of any Angular component.

```
@Component({
  selector: 'load-dashboard',
  template: `
    <ruf-page-header
      title="Dashboard"
      description="This page will contain various charts and figures."
      color="default"
    >
    </ruf-page-header>
  `,
  styles: [``]
})
```

Angular Components Contin…

How to create a new Component?

This is the basic building block of Angular.

Open **VS code** then go to your project source folder then expand the app directory and create a new list named '**server**.'

Now, create a component in the server directory. Right-click on server directory and create a new file named as 'server.component.ts.' It is the recently created component.

Components are used to build webpages in all versions of Angular, but they require modules to bundle them together. Now, we have to register our new component in the module.

Creating a component with CLI

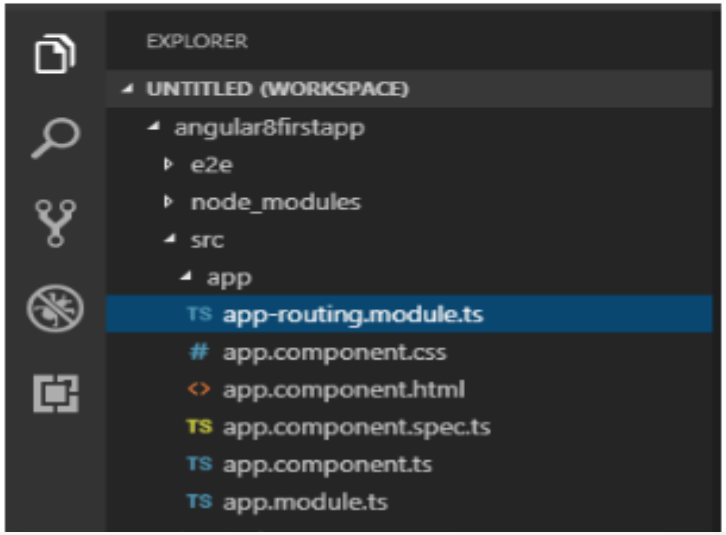
Syntax:

```
ng generate component component_name
Or
ng g c Component-name
```

Angular Components Contin…

App folder: The app folder contains the data we have created for app components.

- **app.component.css**: The component CSS file.
- **app.component.html**: This component is used for HTML view.
- **app.component.spec.ts**: The HTML view of the component
- **app.component.ts**: Component code (data and behavior)
- **app.module.ts**: The main application module.



Angular Components Contin…

Life cycle hooks

- A component has a lifecycle managed by Angular.
- Angular creates and renders components along with their children, checks when their data-bound properties change, and destroys them before removing them from the DOM.
- Angular offers lifecycle hooks that provide visibility into these key life moments and the ability to act when they occur.
- A directive has the same set of lifecycle hooks.

Angular Components Contin…

Life cycle hooks

Hook	Purpose and Timing
`ngOnChanges()`	Respond when Angular (re)sets data-bound input properties. The method receives a `SimpleChanges` object of current and previous property values. Called before `ngOnInit()` and whenever one or more data-bound input properties change.
`ngOnInit()`	Initialize the directive/component after Angular first displays the data-bound properties and sets the directive/component's input properties. Called *once*, after the *first* `ngOnChanges()`.
`ngDoCheck()`	Detect and act upon changes that Angular can't or won't detect on its own. Called during every change detection run, immediately after `ngOnChanges()` and `ngOnInit()`.
`ngAfterContentInit()`	Respond after Angular projects external content into the component's view / the view that a directive is in. Called *once* after the first `ngDoCheck()`.

Angular Components Contin…

Life cycle hooks

`ngAfterContentInit()`	Respond after Angular projects external content into the component's view / the view that a directive is in. Called *once* after the first `ngDoCheck()`.
`ngAfterContentChecked()`	Respond after Angular checks the content projected into the directive/component. Called after the `ngAfterContentInit()` and every subsequent `ngDoCheck()`.
`ngAfterViewInit()`	Respond after Angular initializes the component's views and child views / the view that a directive is in. Called *once* after the first `ngAfterContentChecked()`.
`ngAfterViewChecked()`	Respond after Angular checks the component's views and child views / the view that a directive is in. Called after the `ngAfterViewInit()` and every subsequent `ngAfterContentChecked()`.

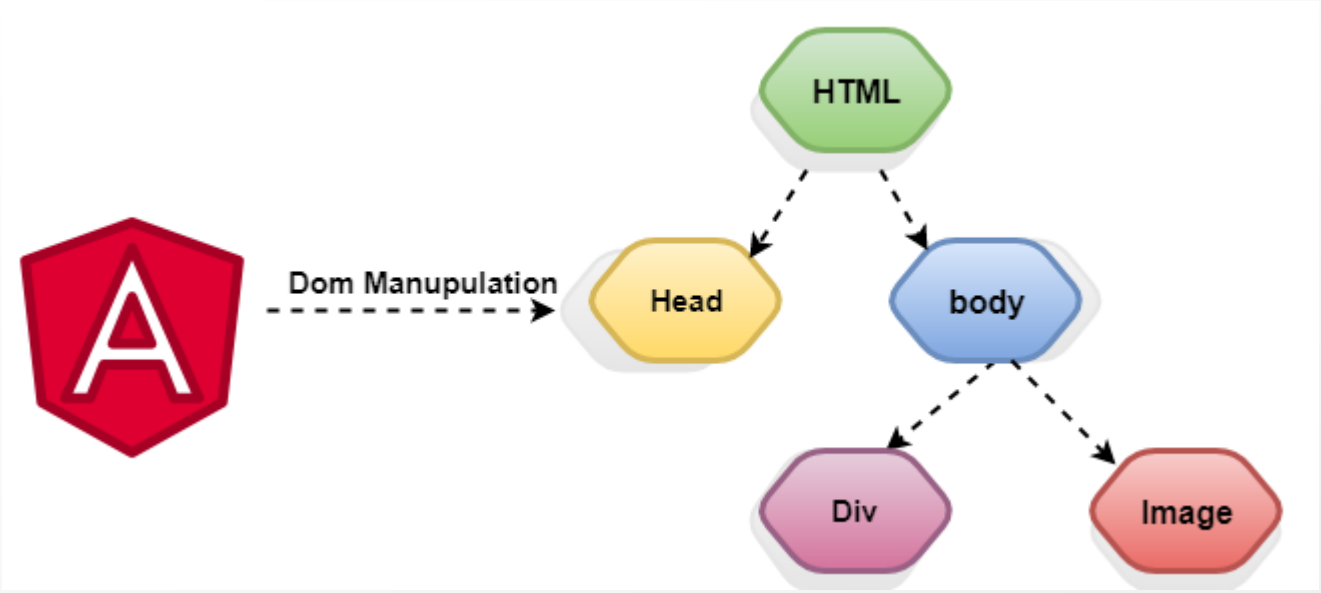
Angular Components Contin…

Life cycle hooks

`ngOnDestroy()`	Cleanup just before Angular destroys the directive/component. Unsubscribe Observables and detach event handlers to avoid memory leaks. Called *just before* Angular destroys the directive/component.

Angular Directives

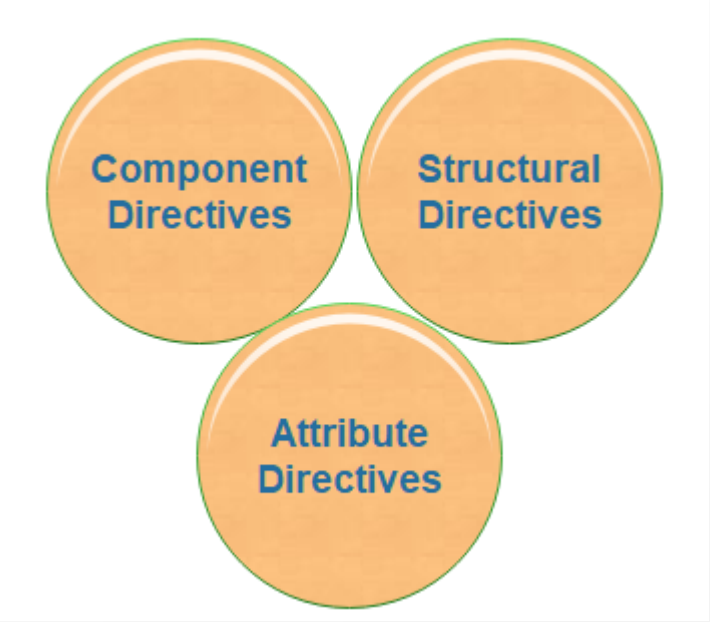
- Directives are instructions in the **DOM** (Document Object Model). It specifies how to place our business logic in Angular. The directive is markers on a DOM element that tell Angular to attach a specified behavior to that DOM element or even transform the DOM element and its children. Mostly directives in Angular starts with ng- where **ng** stands for **Angular,** and it extends the HTML.



Angular Directives

There are three kinds of directives:

1. **Component Directives**
2. **Structural Directives**
3. **Attribute Directives**



Angular Directives

Component Directives

Components are the most common of the directives. It contains the details of how the component should be processed, instantiated, and used at runtime. The component comprises meta-data.

Structural Directives

Structural Directivesaredone in the elements section. These directives are used to manipulate and change the structure of the DOM elements. Structural directives have a star (*) sign before the directive. **Like as,* ngIf**, ***ngFor, and *ngSwitch directive.**

***ngIf Directive:** The *ngIf allows us to Add/Remove DOM Element.

***ngSwitch Directive:** The *ngSwitch will enable us to Add/Remove DOM element. It is same as the switch statement of C#.

***ngFor Directive:** The *ngFor directive is used to repeat a part of HTML template once per each item from an iterable list (Collection).

Angular Directives

Attributes Directives

Itdeals with changing the look and behavior of the DOM element. For example: **ngClass**, **ngStyle** etc.

• **NgClass Directive:** The ngClass Directive is used to add or remove CSS classes to an element.

•**NgStyle Directive:** The ngStyle Directive facilitates you to modify the style of an HTML element using the expression. We can also use the ngStyle Directive to change the style of our HTML element dynamically.

How to Create Custom Directives?

We can create our custom directives to use in Angular components with the help of the command line. The command which is used to develop the Directive using the command line is as follows-

```
ng g directive name of the Directive
e.g
ng g directive change text
```

Angular Directives

ngIf Directive is the part of structural directives.The NgIf is the most straightforward structural Directive and comfortable to understand. The ngIf Directives is used to add and remove HTML elements according to the expression. The expression returns a Boolean value. If the appearance is false, then the removed, otherwise portion is inserted.
It is same as the ng-if Directive of AngularJS. The ngIf directive doesn't hide elements. It adds and removes them physically from the DOM. We can confirm it by using browser developer tools to inspect the DOM. When the condition is false, NgIf removes the host element from the DOM, which detection, and destroys it.

ngIf Syntax

ngIf Syntax

```
<p *ngIf="condition">
  Condition is true and ngIf is true.
</p>
<p *ngIf="!condition">
 Condition is false and ngIf is false.
</p>
```

The *ngIf directive form with an "else" block

```
<div *ngIf="condition; else elseblock">
```

Angular Directives

Same template example with else block

```
@Component({
  selector :'ng-if-else';
 template:
<button (click) =”show = !show”>{{show ? ‘hide’ : ’show’}}</button>
 show={{show}}
<br>
<div *ngIf=”show; else elseBlock’>Text to show</div>
<ng-template #elseBlock>Alternate text while primary
text is hidden </ng-template>
 `
 }]
 export class NgIfElse{
 show: boolean = true;
 }
```

Angular Directives

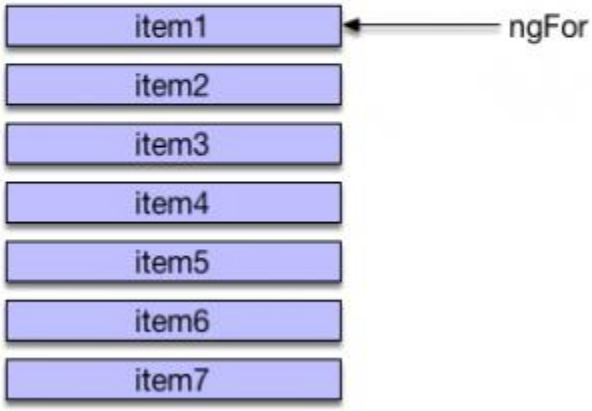
Angular 8 ngFor Directive

The *ngFor directive is used to repeat to repeat a portion of HTML template once per each item from an iterable list (collection). The ngFor is an Angular structural directive and is similar to ngRepeat in AngularJS. Some local variables like Index, First, Last, odd and even are exported by *ngFor directive.

Syntax of ngFor

Syntax of ngFor

```
<li *ngFor="let item of items ;">….</li>
```



previous next

Angular Directives

Angular 8 ng-Switch Directive

The ng-Switch Directive hides and shows the HTML elements depending on an expression.

Child elements with the ng-switch-when directive will be displayed if it gets a match; otherwise, the component and its children will be removed.

We could also define a default section, by using the ng-switch default directive, to show a section if no any other part gets a match.

Syntax:

```
<element ng-switch="expression">
<element ng-switch-when="value"></element>
<element ng-switch-when="value"></element>
<element ng-switch-when="value"><element>
<element ng-switch-default></element>
</element>
```

Angular Directives

Angular 8 ngClass Directive

ngClass Directive is a type of attribute directive. Angular provided built-in directive, and it helps in adding or removing CSS classes on an HTML element. The ngClass directive allows us to apply CSS classes dynamically based on expression evaluation.

[ngClass] selector uses the NgClass directive, and ngClass offers three simple ways through which we can update CSS classes in the DOM.

Syntax:

```
<element ng-class=""expression""></element>
```

Angular Pipes

- Pipes are a useful feature in Angular. These are the simple way to transform values in an Angular template. It takes the integers, strings, array, and dates as input separated with | to be converted in the format as required and display same as an in the browser.
- Inside the interpolation expression, we define the pipe and use it based on the situation because there are many types of pipes. We can use in our angular applications.
- A pipe holds in data as input and transforming it into the desired output. Some values benefit for a bit of editing. We may notice that we want many of the same transformations repeatedly, both within and across many applications.
- We can almost think of them as styles, and In fact, we might like to apply them in our HTML templates.

Syntax:

```
{{title | uppercase}}
```

```
<h1>
{{ title | uppercase}} <br>
</h1>
<h1>
{{ title | lowercase }} <br>
</h1>
```

Parameterizing a pipe in Angular 8

We can also move a parameter to the pipe; we can use the HTML code to pass the parameter.

app.component.html

```
<h1>
Rohan's birthday is {{ birthday | date:"dd/mm/yyyy"}}
</h1>
```

Chaining pipes

We can chain pipes together and creates useful combinations and also can use the lowercase and upper case pipe in our example.

app.component.html

```
<h1>
Rohan's birthday is {{birthday | date | uppercase}}
</h1>
```

Now, our date is in upper case letter.

Pure and Impure pipes

There are two categories of pipes:

1. Pure
2. Impure

By default, pipes of angular are pure. Every pipe we have seen are pure and built-in pipes. We can make the pipe impure by setting the pure flag into false.

Pure pipes

Angular executes the pure pipe only when if it detects the perfect change in the input value. The real difference is either in the shift to the primitive input value (Number, Boolean, Symbol, String) or a changed object reference (Array, Function, Object, Date).

Impure pipes

Angular executes the corrupted pipe during every component change detection cycle. The impure pipe is called often every keystroke or mouse-move.

How to Create a Custom Angular Pipe?

To create a custom pipe, we create a new ts file and use the code according to work, and we have to import Pipe, Pipe Transform from Angular/Core.

Create a sqrt custom pipe.

component.ts file:

```
import {Pipe, PipeTransform} from
'@angular/core';
@Pipe ({
name: 'sqrt'
})
Export class SqrtPipe implements PipeTransform {
transform (val: number) : number{
Return Math.sqrt(val);
}
}
```

We have to make changes in the app.module.ts. Create a class named as SqrtPipe.

This class will implement PipeTransform. The transform method defined in the class can take arguments as the number and will return the number after taking the square root.

String Interpolation

String Interpolation is a **one-way data-binding** technique which is used to output the data from a TypeScript code to HTML template (view).

It uses the template impression in double curly braces to show the data from the component to the view.

For example:

```
{{data}}
```

String interpolation sums the value of a property from the component.

Syntax:

```
<li>Name: {{user.name}}</li>
<li>Email: {{user. Email}}</li>
```

Property Binding

Property binding is a technique, which will help to bind values to the properties of HTML elememts. It is also a one-way data binding approach.

In property binding, we bind a characteristic of a DOM element in a field which defined property in our component Typescript code.

Property binding helps us to bind the values to the target property of the element property surrounded within the square brackets.

For example:

```
<img [src]=""imgUrl"/">
```

Syntax:

```
<input type=""email"[value]="user.email""">
```

Angular Data Binding

Event binding

Event binding is used to hold the events lifted from the DOM such as button click, mouse move, etc. in Angular 8. When the DOM event happens (e.g., Click, change, key up), it calls the specified method in the component. In the following example, the cookbacon() method in the component is called when the button is clicked:

For example:

```
<</code>button (click)="cookBacon()"></button>
```

Two-way Data Binding

In one-way data binding, any changing in the template (view) was not considered in the component Typescript code. To solve this problem, Angular produces two-way data binding.

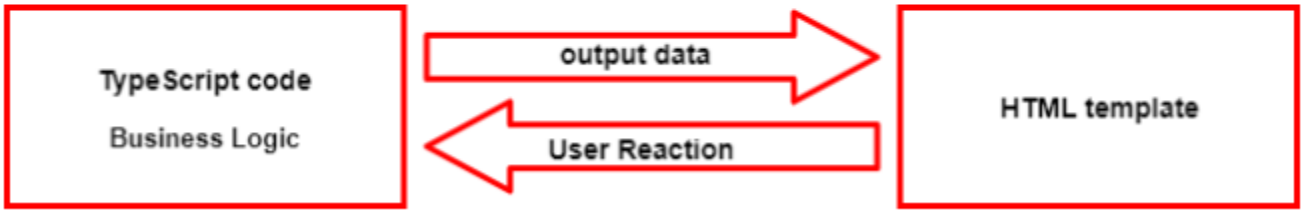
The two-way binding has a vital feature to update data from component to view and view to the element.

In two way data binding, property binding and event binding are combined. For two-way data binding, we have to enable the ngModel directive.

Angular Data Binding

Syntax:

```
[(ngModel)]="[property of our component]"
```



Two-way data Binding

Angular Forms

Reactive forms

Angular forms are used to handle the user's input. We use Angular form in our application to authorize users to log in, to update profile, to enter information, and to perform many other data-entry tasks.

In Angular 8, there are two approaches to handle the user's input through forms:

- Both methods are used to collect user input event from the view, validate the user input, create a form model to update, and provide a way to track changes.
- Reactive and template-driven forms process and manage form data differently. Each offers different advantages.

Reactive forms

Reactive forms are more robust, and they are more scalable, reusable, and testable. If forms are a vital part of our application or are already using reactive patterns for building our app, handle responsive forms.

Angular Forms

Template-driven forms
It is useful for adding a simple form to an app, such as an email list signup form to an app, or login form. It is easy to add to an app, but they don't scale as well as reactive forms. If we have fundamental form requirements and logic that can be managed in the template and use template-driven forms.

Practice examples
https://stackblitz.com/angular/rlqdvkmerpde?file=src%2Fapp%2Fprofile-editor%2Fprofile-editor.component.ts

https://stackblitz.com/angular/ymagmgbovvq?file=src%2Fapp%2Fhero-form%2Fhero-form.component.html

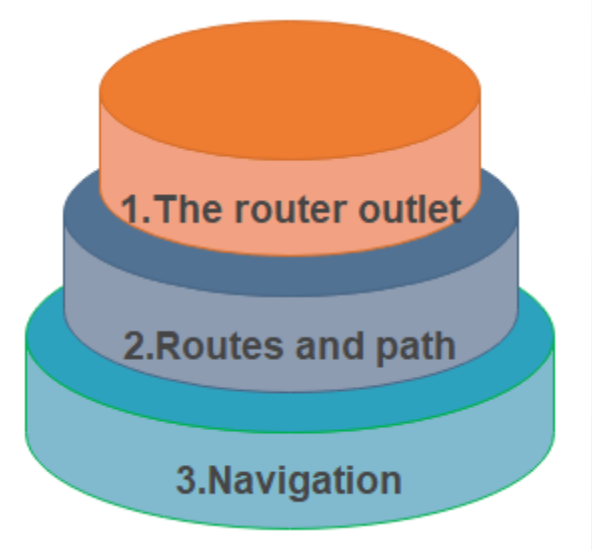
Angular Routing

Angular Router is a powerful JavaScript router is built and maintained by the Angular core team that can install from the **package @angular/router**. Routing provides a complete routing library with the possibility of multiple router outlets, different path strategies.

- The angular Router is the main part of Angular platform. It allows developers to build single-page applications with multiple views and allow navigation between views.
- Angular supports SPA using routing module ngRoute. The routing module acts based on the URL. It works as a browser navigation's bar, and it was navigating between the pages.
- Enter URL in the address bar, and then the browser will navigate to the corresponding page.
- Click on the link to the page and the browser will navigate to the new page.
- Click on the browser on the back or forward, and the browser will navigate backward or forward according to the history pages.

Angular Routing

Angular brings many improved modules to the Angular ecosystem, including a new router called the component router. The component router is highly configurable and features packed Router. Features included are standard view routing, nested child routes, named routes, and route parameters. The concept related to the Router are:



Angular Routing

The Router-outlet

The Router-outlet is a directive accessible from the router library where the Router inserts the component and gets matched based on the current browser's URL. We can add multiple outlets in our angular application, which enables us to implement advanced routing scenarios.

```
<router-outlet></router-outlet>
```

Routes And Paths

Routes are definitions comprised of a path and component attributes. The path mention to the part of the URL that determines a unique view that can be displayed, and refer to the Angular component that needs to be associated with a path.

In a component, each route maps a URL path.

The path could take a wildcard string (**). The Router selects this route if the called URL doesn't match the explained routes. It can be used for displaying a **"Not Found"** view or redirecting to a specific view if there is no match.

Example:

```
{path:'contacts', component: ContactListComponent}
```

If the route definition is provided to the router configuration, the router will render **ContactListComponent** when the browser URL for the web application **/contacts.**

Angular Routing

Route Params

To creating the routes with parameters is a common feature in web apps. The angular Router allows us to access parameters in different ways.

We can create a router parameter using the colon syntax.

```
{path:'contacts/:id', component: contactDetailcomponent}
```

Route Guard

A route guard is the feature of the Angular Router which allows the developer to run logic if a router is requested, and it is based on the logic. It allows and denies the user access to the route. We can add a route guard by implementing the CanActivate interface available from **@angular/router** package. It **can activate()** method which holds the logic to allow and deny access to the route.

```
Class MyGuard implements CanActivate {
can activate () {
return true;
}
}
```

Angular Routing

Route Params

To creating the routes with parameters is a common feature in web apps. The angular Router allows us to access parameters in different ways.

We can create a router parameter using the colon syntax.

```
{path:'contacts/:id', component: contactDetailcomponent}
```

Route Guard

A route guard is the feature of the Angular Router which allows the developer to run logic if a router is requested, and it is based on the logic. It allows and denies the user access to the route. We can add a route guard by implementing the CanActivate interface available from **@angular/router** package. It **can activate()** method which holds the logic to allow and deny access to the route.

```
Class MyGuard implements CanActivate {
can activate () {
return true;
}
}
```

Angular Routing

<base href>

Most routing applications should add a <base> element to the index.html as the first child in the <head> tag to tell the router how to compose navigation URLs

<base href="/">

import { RouterModule, Routes } from '@angular/router';

Configuration

A routed Angular application has one singleton instance of the *Router* service. When the browser's URL changes, that router looks for a corresponding `Route` from which it can determine the component to display.

A router has no routes until you configure it. The following example creates five route definitions, configures the router via the `RouterModule.forRoot()` method, and adds the result to the `AppModule`'s `imports` array.

Angular Routing

```
const appRoutes: Routes = [
  { path: 'crisis-center', component: CrisisListComponent },
  { path: 'hero/:id',      component: HeroDetailComponent },
  {
    path: 'heroes',
    component: HeroListComponent,
    data: { title: 'Heroes List' }
  },
  { path: '',
    redirectTo: '/heroes',
    pathMatch: 'full'
  },
  { path: '**', component: PageNotFoundComponent }
];
```

Angular Routing

```
@NgModule({
  imports: [
    RouterModule.forRoot(
      appRoutes,
      { enableTracing: true } // <-- debugging purposes only
    )
    // other imports here
  ],
  ...
})
export class AppModule { }
```

Angular Routing

- The appRoutes array of routes describes how to navigate. Pass it to the RouterModule.forRoot() method in the module imports to configure the router.
- The :id in the second route is a token for a route parameter. In a URL such as /hero/42, "42" is the value of the id parameter.
- The data property in the third route is a place to store arbitrary data associated with this specific route. The data property is accessible within each activated route. Use it to store items such as page titles, breadcrumb text, and other read-only, static data.
- The empty path in the fourth route represents the default path for the application, the place to go when the path in the URL is empty, as it typically is at the start.
- The ** path in the last route is a wildcard. The router will select this route if the requested URL doesn't match any paths for routes defined earlier in the configuration.
- The order of the routes in the configuration matters and this is by design. The router uses a first-match wins strategy when matching routes.
- If you need to see what events are happening during the navigation lifecycle, there is the enableTracing option as part of the router's default configuration.

Angular Routing

Navigation Directive

The Angular Router provides the router link directive to create the navigation links. This directive generates the path associated with the component to navigate.

```
<a[routerLink]= " '/contacts' ">Contacts</a>.
```

https://stackblitz.com/angular/jyydopxboqn?file=src%2Fapp%2Fapp-routing.module.ts

For more information of example explanation

https://angular.io/guide/router#the-basics

Angular HTTP Client

- The HttpClient in @angular/common/http offers a simplified client HTTP API for
- Angular applications that rests on the XMLHttpRequest interface exposed by browsers.
- Additional benefits of HttpClient include testability features, typed request and response objects,request and response interception, Observable apis, and streamlined error handling.

Setup

```
import { NgModule }         from '@angular/core';
import { BrowserModule }    from '@angular/platform-browser';
import { HttpClientModule } from '@angular/common/http';

@NgModule({
  imports: [
    BrowserModule,
    // import HttpClientModule after BrowserModule.
    HttpClientModule,
  ],
```

Angular HTTP Client

Having imported `HttpClientModule` into the `AppModule`, you can inject the `HttpClient` into an application class as shown in the following `ConfigService` example.

app/config/config.service.ts (excerpt)

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Injectable()
export class ConfigService {
  constructor(private http: HttpClient) { }
}
```

HTTP headers

Many servers require extra headers for save operations. For example, they may require a "Content-Type" header to explicitly declare the MIME type of the request body; or the server may require an authorization token.

Adding headers

The `HeroesService` defines such headers in an `httpOptions` object that will be passed to every `HttpClient` save method.

app/heroes/heroes.service.ts (httpOptions)

```
import { HttpHeaders } from '@angular/common/http';

const httpOptions = {
  headers: new HttpHeaders({
    'Content-Type':  'application/json',
    'Authorization': 'my-auth-token'
  })
};
```

Angular HTTP Client

HTTP headers

Many servers require extra headers for save operations. For example, they may require a "Content-Type" header to explicitly declare the MIME type of the request body; or the server may require an authorization token.

Adding headers

The `HeroesService` defines such headers in an `httpOptions` object that will be passed to every `HttpClient` save method.

app/heroes/heroes.service.ts (httpOptions)

```
import { HttpHeaders } from '@angular/common/http';

const httpOptions = {
  headers: new HttpHeaders({
    'Content-Type':  'application/json',
    'Authorization': 'my-auth-token'
  })
};
```

https://angular.io/generated/live-examples/http/stackblitz

Observables & RxJS

RxJS (Reactive Extensions for JavaScript) is a library for reactive programming using observables that makes it easier to compose asynchronous or callback-based code

RxJS provides an implementation of the Observable type. The library also provides utility functions for creating and working with observables. These utility functions can be used for:

- Converting existing code for async operations into observables
- Iterating through the values in a stream
- Mapping values to different types
- Filtering streams
- Composing multiple streams

RxJS Operators

- Combination (concat, endWith, merge etc)
- Conditional (defaultIfEmpty, sequenceequal, iif, every)
- Creation (ajax, create, defer, from , of, range etc)
- Error Handling (catch/catchError, retry etc)
- Filtering (filter, find , first, last, skip, take etc)
- Multicasting (multicast, publish, share etc)
- Transformation (buffer, expand, map, scan etc)
- Utility (delay, tap/do , toPromise , repeat etc)

RxJS Subjects

A Subject is a special type of Observable which shares a single execution path among observers.

We can think of this as a single speaker talking at a microphone in a room full of people. Their message (the subject) is being delivered to many (multicast) people (the observers) at once. This is the basis of **multicasting**. Typical observables would be comparable to a 1 on 1 conversation.

There are 4 variants of subjects:
Subject - No initial value or replay behavior.
AsyncSubject - Emits latest value to observers upon completion.
BehaviorSubject - Requires an initial value and emits its current value (last emitted item) to new subscribers.
ReplaySubject - Emits specified number of last emitted values (a replay) to new subscribers.

Observables & RxJS

Subject:

A special type of Observable which shares a single execution path among observers

```
// RxJS v6+
import { Subject } from 'rxjs';

const sub = new Subject();

sub.next(1);
sub.subscribe(console.log);
sub.next(2); // OUTPUT => 2
sub.subscribe(console.log);
sub.next(3); // OUTPUT => 3,3 (logged from both subscribers)
```

Observables & RxJS

AsyncSubject:

Emits its last value on completion

```
// RxJS v6+
import { AsyncSubject } from 'rxjs';

const sub = new AsyncSubject();

sub.subscribe(console.log);

sub.next(123); //nothing logged

sub.subscribe(console.log);

sub.next(456); //nothing logged
sub.complete(); //456, 456 logged by both subscribers
```

Observables & RxJS

BehaviourSubject:

Requires an initial value and emits the current value to new subscribers

```
// RxJS v6+
import { BehaviorSubject } from 'rxjs';

const subject = new BehaviorSubject(123);

// two new subscribers will get initial value => output: 123, 123

subject.subscribe(console.log);
subject.subscribe(console.log);

// two subscribers will get new value => output: 456, 456
subject.next(456);

// new subscriber will get latest value (456) => output: 456
subject.subscribe(console.log);

// all three subscribers will get new value => output: 789, 789,
 789
subject.next(789);

// output: 123, 123, 456, 456, 456, 789, 789, 789
```

Observables & RxJS

ReplaySubject:

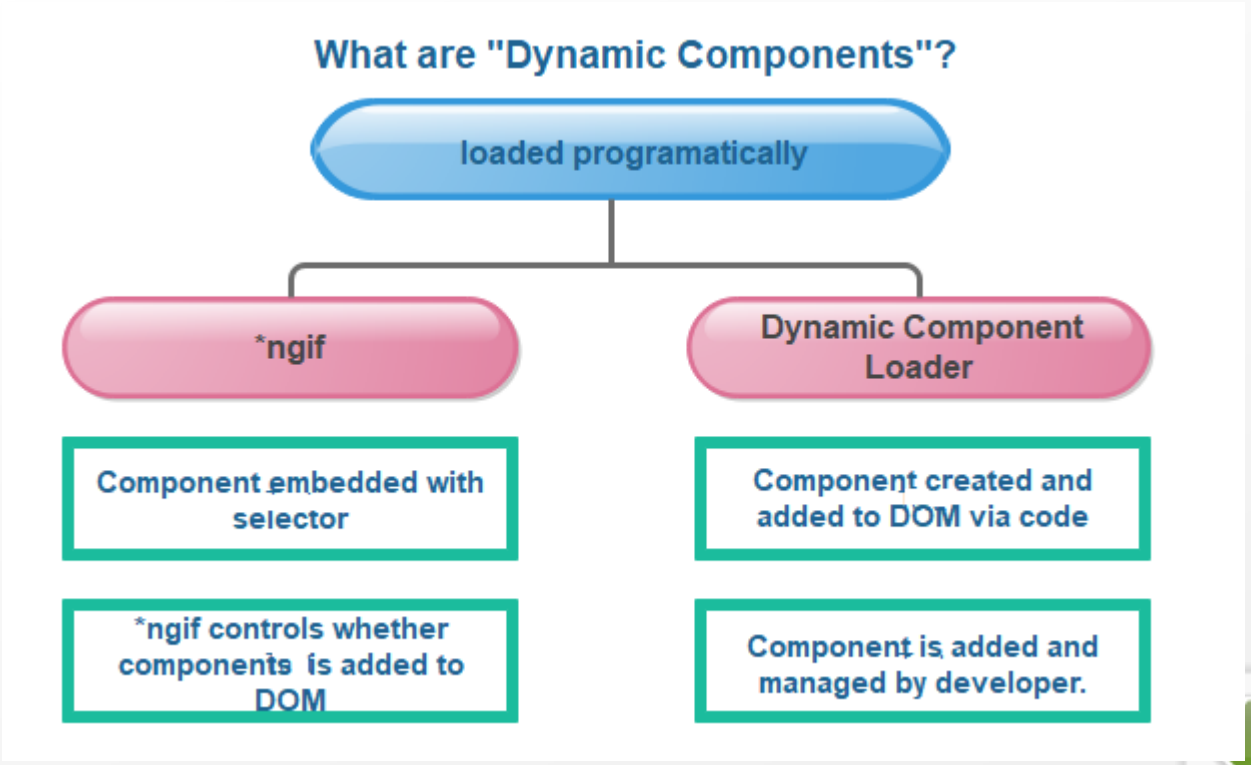
"Replays" or emits old values to new subscribers

```
// RxJS v6+
import { ReplaySubject } from 'rxjs';

const sub = new ReplaySubject(3);

sub.next(1);
sub.next(2);
sub.subscribe(console.log); // OUTPUT => 1,2
sub.next(3); // OUTPUT => 3
sub.next(4); // OUTPUT => 4
sub.subscribe(console.log); // OUTPUT => 2,3,4 (log of last 3 va
lues from new subscriber)
sub.next(5); // OUTPUT => 5,5 (log from both subscribers)
```

For additional reference: https://www.learnrxjs.io/

Angular Dynamic Components

The dynamic component is one of the versatile and core concept introduced in Angular, Component template is not fixed. An application needs to load new elements at runtime in various scenarios.
The dynamic component is the component which is created dynamically at the runtime. Angular has its API for loading components dynamically.



previous next

Angular Dynamic Components

Dynamic component loading

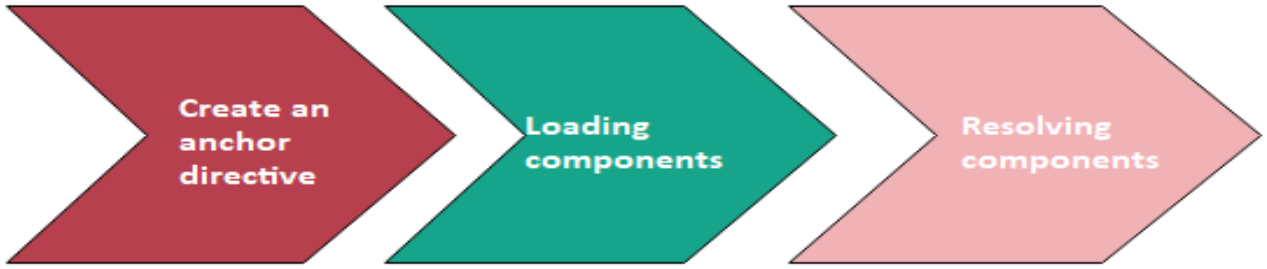
In the given example, we can see how to build a dynamic ad-banner.

The hero agency is planning an ad-campaign with several different ads cycling through the banner, where new ad components are added often by different teams. We need to load a new component without a fixed reference to the component in the ad banner's template.

Angular comes with its API for loading components dynamically.

Steps required to create Dynamic Component in Angular 8

1. Create an anchor directive
2. Loading components
3. Resolving components



Create an Anchor Directive

We should know where to include this anchor point into components. Create helper directive called **NewsFeedDirective** to create the anchor to insert anywhere to the component. The ad banner uses a directive called **AdDirective** to make a valid insertion point in the template bar.

src/app/ad.directive.ts

```
import { Directive, ViewContainerRef } from '@angular/core';
@Directive({
selector: '[ad-host]',
})
export class AdDirective {
constructor(public viewContainerRef: ViewContainerRef) { }
}
```

AdDirective injects **ViewContainerRef** to access to the view container of the element that host the dynamic added component.

In the @Directive decorator, observe the selector name, ad-host; that is, we use to apply the directive in the element.

Loading Components

Most of the ad banner implemented in **ad-banner.component.ts**. To keep things simple in the example, the HTML is in the @Component decorator's template properly as a template string.

The <ng-template> element is where we apply the directive we just made. To ask the AdDirective, recall the selector from ad.directive.ts and ad-host. Apply the <ng-template> without the square brackets.

src/app/ad-banner.component.ts(template)

```
template: `
  <div class="ad-banner-example">
  <h3>Advertisements</h3>
  <ng-template ad-host=""></ng-template>
  </div>
  `
```

The **<ng-template>** element is good choice for dynamic component because it doesn't render any additional output.

Resolving components

In Resolving component, **AdBannerComponent** takes an array of **AdItem** objects as input, which finally comes from the **AdService**. AdItem objects generate the type of component to load and any data to bind in the **component.AdService** returns the actual ad making up the ad campaign.

Passing an array of a component to **AdbannerComponent** allows for a dynamic list of ads without static element in the template.

For example: https://stackblitz.com/edit/angular-rtzd4w?file=src%2Fapp%2Fad-banner.component.ts

Angular Animations

The animation is a process of drawing, designing , making layouts, and preparation of photographic sequences which are integrated into the multimedia and gaming products.

A reflection of movement created for displaying a series of frames, or pictures. Cartoons on television are on of the examples of animation.

Angular Animation
Animation provides the illusion of motion. HTML elements change styling over time. A well designed animation can make our application more fun and easier to use.

Angular Animations

Set-up of Animation in Angular 8

Previously, we have to import new animation package through the command "**npm install –save@angular/animations**."

Then, add the **BrowserAnimationsModule** to our **imports[]** array in **AppModule**.

The module needs to be imported from **@angular/platform-browser/animations'** => import **{ BrowserAnimationsModule }** from **'@angular/platform-browser/animations'**(in AppModule).

After that, we import **trigger, state, style**, etc. from **@angular/animations** instead of **@angular/core.**

Angular Animations

Step 1: Enabling the animations module

Import **BrowserAnimationsModule**, which introduces the animation capabilities in our angular root application module.

src/aap/app.module.ts

```
 import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
@NgModule({
imports: [
BrowserModule,
BrowserAnimationsModule
],
declarations: [ ],
bootstrap: [ ]
})
export class AppModule { }
```

Step 2: Importing animation functions into component file

If we plan to use specific animation functions in component files, import those functions from **@angular/animations.**

src/app/app.component.ts

```
import { Component, HostBinding } from '@angular/core';
import {
trigger,
state,
style,
animate,
transition,
// ...
} from '@angular/animations';
```

Step 3: Adding the animation metadata property

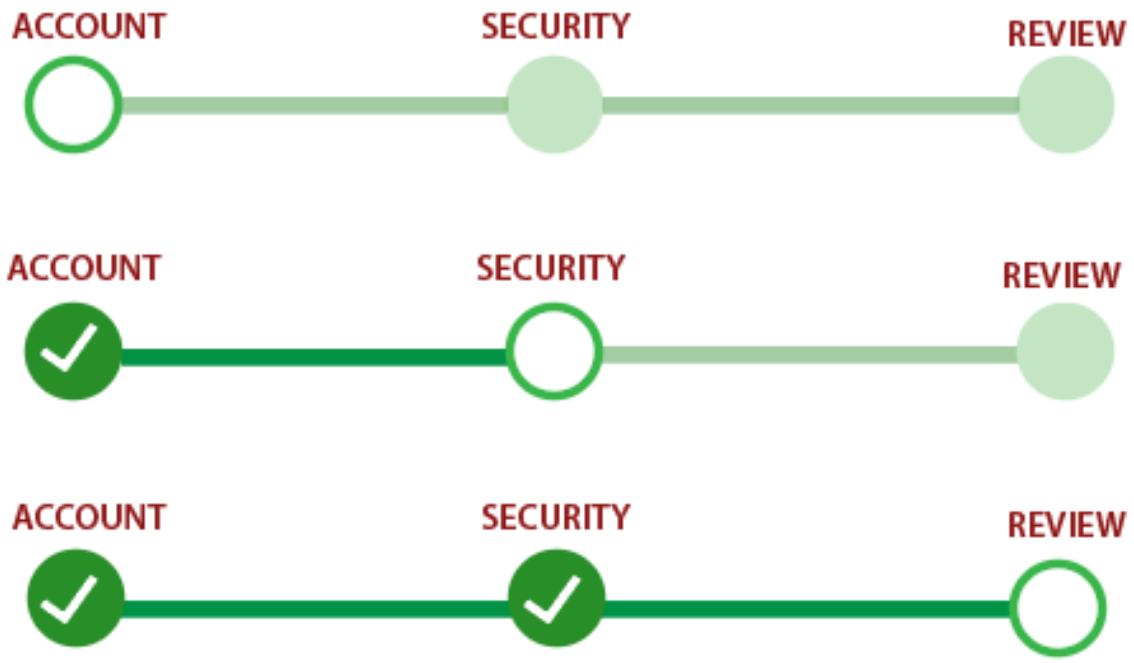
In the component file, add a metadata properly called animations in the **@Component ()** decorator. We put the trigger which defines an animation within the animations metadata property.

src/app/app.component.ts

```
@Component({
selector: 'app-root',
templateUrl: 'app.component.html',
styleUrls: ['app.component.css'],
animations: [
// animation triggers go here
]
})
```

Angular Animations

For Example:



Animating a simple transition in Angular

We can specify that a button displays either open or closed based on the user's last action. When the button is in the free state, it's visible and yellow. When it is the closed state, it's transparent and green.

In HTML, these attributes are set using common CSS styles such as color and opacity. In Angular, use the style () functions to specify a set of CSS styles for use with animations.

Animation state and styles

We can use Angular's state() function to define different states to call at the end of each transition. This function has two arguments: **a unique name** such as open, **closed**, and **style( )** function.

Use style() function to define a set of style to associate with a given state name.

In open state, the button has a height of 200 pixels, an opacity of 1, and a background color of yellow.

src/app/open-close.component.ts

```
// ...
state('open', style({
height: '200px',
opacity: 1,
backgroundColor: 'yellow'
})),
```

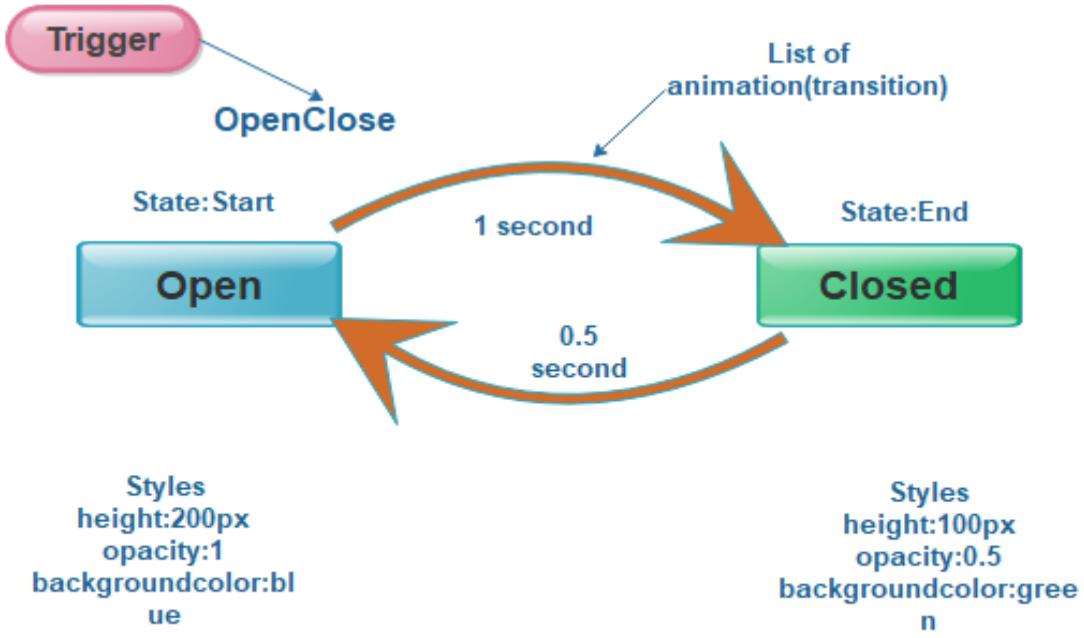
```
state('closed', style({
height: '100px',
opacity: 0.5,
backgroundColor: 'green'
})),
```

In the closed state, the button has height of 100 pixels, an opacity of 0.5, and a background color of green.

Triggering the animation in Angular

An animation needs a trigger, so that it knows when to start it. The trigger () function collect the states and transitions, and gives the animation a name, so we can attach to the triggering element in HTML template.

The trigger() function describes the property name to watch for the changes. When a change occurs, the trigger commences the action included in its definition.



Angular Animations

Attaching animation to HTML template in Angular

The animation is defined in the metadata of the component which controls the HTML element, which is animated. Put the code that defines our animation under the animations: **property** within the **@Component ()** decorator.

src/app/open-close.component.ts

```
@Component(
selector: 'app-open-close',
animations: [
trigger('openClose', [
// ...
state('open', style({
height: '200px',
opacity: 1,
backgroundColor: 'yellow'
})),
state('closed', style({
height: '100px',
opacity: 0.5,
backgroundColor: 'green'
})),
transition('open => closed', [
animate('1s')
]),
transition('closed => open', [
animate('0.5s')
]),
```

Angular Animations

When we have defined an animation trigger for a component, we can attach it to an element in the component's template by wrapping the trigger name in the bracket and introduce it with an @symbol.

When triggerName is the name of the trigger, and expression evaluates to a defined animation state.

```
<div[@triggerName]="expression">...</div>;
```

The animation will be executed or triggered when the expression value changes into a new state.

The given code binds the trigger to the value of the is open property.

src/app/open-close.component.html

```
<div [@openClose]="isOpen ? 'open' : 'closed'" class="open-close-container">
<p>The box is now {{ isOpen ? 'Open' : 'Closed' }}!</p>
</div>
```

Angular Animations

Example:
https://stackblitz.com/angular/mkqrxnyekjx?file=src%2Fapp%2Fapp.component.ts

Angular CLI Commands

- Angular CLI is a command-line interface which is used to initialize, develop, and maintain Angular applications. We can use these Command on command prompt or consequentially by an associated UI. i.e., Angular Console.

Command	Alias	Description
add		It used to add support for an external library to the Project.
build	b	It compiles an Angular app in an output directory "dist" at the given output path. Must be executed within a workspace directory.
config		It retrieves Angular configuration values in the angular.json file for the workspace directory.
doc	d	It opens the official Angular site (angular.io) in the browser and also searches for any given keywords.
e2e	e	It builds and serves an Angular app, and then runs the end-to-end test using a protractor.
generate	g	It generates or modifies files based on a schedule.
help		It provides a list of possible commands and short descriptions.
lint	l	It runs linting tools on the Angular app in a given project folder.
new	n	It creates a workspace and a first Angular app.

Angular CLI Commands

run		It runs an architecture target with an optional custom builder configuration in our descriptions.
serve	s	It builds and helps our app, rebuilding on files changes.
test	t	It runs unit tests in a project.
update		It updates our application and its dependencies.
version	v	It updates the Angular CLI version. shows
Xi18n		It extracts i18n messages from source code.

Angular CLI Commands

1. ng add Command

The **ng add** Command add support of an external library in our Project. It combines the npm package for a published library to your workspace and makes your default app project to use that library.

The default app project is the primary value of the default project in angular.json.

Syntax:

```
ng add <collection>
 [options]
```

Parameter Explanation:

Options

–default=true|false: Whentrue, it disables interactive input prompts for option with a default.

–help=true|false|json|JSON: It generates a help message in the console. Default:false.

–interactive= true|false: it disables interactive input prompts,when false.

Angular CLI Commands

2.ng build Command

The ng build command is used to compile an angular app into an output directory file "dist" at the following output path. It may be executed from within a workspace directory.

Syntax:

```
ng build < Project>
 [options]
```

Parameter Explanation:

< **Project**>:It describes the name of the Project, which builds. It can be an app or library.

Options

–aot=**true|false**: It builds ahead of Time compilation.

Default: false

–baseHref=**baseHref**: It specifies the base URL for the application being built.

–buildEventLog=**buildEventLog**: It is an output file path for build event protocol event.

–buildOptimizer=**true|false**: It enables "@angular-devkit/build-optimizer" for optimization when using the "aot" option.

Angular CLI Commands

Default: false

–commonChunk=true|false: It has used a separate bundle contains code used across multiple packages. Default: true

–configuration=configuration: It is a named build target, which specified in the "configurations" section of angular.json and each designated target is accompanied by a configuration of options default for the goal.

Aliases: -c

–deleteOutputPath= true|false: It was used to delete the output path before it is building. Default: true

–deployUrl=deployUrl: The URL where files will be deployed.

–es5BrowserSupport= true|false: Enables dependency loaded ES2015 polyfills.

Default: false

–extractCss=true|false: It has used to extract CSS from the global styles into CSS files instead of js ones.

Default: false

Angular CLI Commands

–forkTypeChecker=true|false: It has been used to run the Typescript typechecker in a forked process. Default: true

–help= true|false|json|JSON: It is mainly used to show a help message for this Command in the console. Default: false

–i18nFile= i18nFile: i18n is used to locale.

–i18nMissingTranslation=i18nMissingTranslation: How to handle missing translation for i18n.

–index=index: The name of index HTML file.

–lazyModules: List of additional ngModules files will be lazy loaded. Lazy router modules have been discovered unwittingly.

–main=main: The full path in the main entry point to the app is completely relative to the current workspace.

–namedChunks=true|false: Use filename for lazy loaded chunk Default: true

–ngswConfigPath=ngswConfigPath: Path to ngsw-config.json.

–outputHashing=none|all|media|bundles: Define the outputs of filename cache-busting and hashing mode.

Default: none

Angular CLI Commands

–outputPath=**outputPath:**The full path of the new output directory directly relative to the current workspaces.

By default, write output to a folder named "dist \" in the current Project.

–poll: Enables and define the file watching poll time in milliseconds.

–polyfills=**polyfills:** The full path for the polyfill files is relative to current workspaces.

–preserveSymlinks=**true|false:** It do not use the real way when resolving modules.

Default: false

–prod=**true|false**: Sets the build configuration in the production target. All builds make use of packaging and little tree-shaking. A production build runs limited dead code elimination, when true.

–profile=**true|false**: Output profile events is used in Chrome profiler.

Default: false

–progress=**true|false:** it progress to the console during building.

–resourcesOutputPath= **resourcesOutputPath:**The path where style resources would place is combined to outputPath.

Angular CLI Commands

–serviceWorker=true|false: It generates a service worker config in the production build. Default: false.

–showCircularDependencies=true|false: Show circular dependency warning on the build. Default: true

–sourceMap=true|false: It is showed Output source maps.

–statsJson=true|false: It generates a "stats.json" file which can be analyzed using tools such as: 'webpack-bundle-analyzer' or **http://webpack.github.io/analyse.**

–subresourceIntegrity=true|false: It qualifies the use of subresource integrity and validation.

–tsConfig=tsConfig: The full path of the Typescript configuration file is comparative to the current workspaces.

–vendorChunk=true|false: It mostly uses a separate bundle which contains the vendor libraries.

Default: true

–verbose=true|false: It adds more details of output logging.

Default: false

–watch= true|false: It builds when a file changes.

Default: false

Angular CLI Commands

3. ng config Command

The ng command is used to retrieve and set Angular configuration values in the angular.json file of the workspaces.

Syntax:

```
ng config <jsonPath><value>\
 [options]
```

Parameter Explanation:

<jsonPath>: The configuration key is used to set or query, in JSON path format. For example: "a[3].foo.bar[2]". If there is no new value is provided, results the current value of that key.

<value>: A new value for the given configuration key, if provided.

Options

–global=true|false: It accesses the global configuration in the caller's home directory.

Default: false

Aliases: -g

–help= true|false|json|JSON: It has used to show a help message for this Command in the console.

Default: false

Angular CLI Commands

4. ng doc Command

The ng doc command is used to open Angular documentation site "angular.io" in a browser and searches for a given keyword.

Syntax:

```
ng doc <keyword>
 [options]
```

Parameter Explanation:

<keyword>: It is used to identify the keyword to search for, as provided in the search bar in angular.io.

Options

–help= true|false|json|JSON: It generates a helping message for this Command in the console.

Default: false

Angular CLI Commands

5.ng e2e Command

It has used to build and serve an Angular application, that runs end-to-end tests using Protractor.

Syntax:

```
ng e2e <project>
 [options]
```

It must be executed form within a workspace directory. When we don't specify the project name, it executes for all projects.

Parameter Explanation:

< **Project**>: It identifies the name of the Project we want to build. It can be any app or library.

Options:

–baseUrl=**baseUrl:** It specifies the base URL of the Protractor to connect.

–configuration=**configuration:** It is used to determine build target, as is specified in the <configurations> section of angular.json. An arrangement of option defaults accompanies each named target for the goal.

Aliases: -c

Angular CLI Commands

–devServerTarget=devServerTarget: It specifies dev server which target to run tests.

–elementExplorer=true|false: It start Protractor's Element used to explorer for debugging process.

Default: false

–help= true|false|json|JSON: It has seen a help message for this Command in the console line.

Default: false

–host=host: it is used to listen on the host.

Default: localhost

–port: It modifies the port to serve the application.

–prod=true|false: it sets the build configuration to the production target when it is true. All build uses bundling and limited tree-shaking techniques. A production build runs limited dead code elimination.

–protractorConfig= protractorConfig: It implies the name of the Protractor configuration file.

–specs: It overrides spec in protractor config.

–suite=suite: It overrides suite on the protractor configuration.

–webdriverUpdate=true|false: It is worn to update the web driver. Default: true

Angular CLI Commands

6. ng generate Command

Syntax:

```
ng generate
<schematic>
```

Parameter Explanation:

It specifies the schematic or collection: which we want to generate, and it can take one of the following sub-commands.

- appShell
- application
- class
- component
- directive
- enum
- guard
- interface
- library
- module

Angular CLI Commands

- pipe
- service
- service Worker
- universal

Schematic Command

appShell:

It was used to generate an app shell for creating a server-side version of an app.

Syntax:

```
Ng generate appShell
[options]<strong>or</strong>
Ng g appshell
[options]
```

application

It is used to create a basic app definition in the "projects" subfolder of the workspace.

Angular CLI Commands

Syntax:

```
ng generate application<name>
<strong>or</strong>
ng g application
<name> [options]
```

class

It creates a new generic class in the given or default project.

Syntax:

```
ng generate class
<name>
[options]<strong>or</strong>
ng g class <name>
[options]
```

Angular CLI Commands

component

It can create a new generic component definition in the given or default project.

Syntax:

```
ng generate
component <name><strong>or</strong>
ng g component
<name> [options]
```

directive

It is mainly used to create a new generic directive definition in the given or default project.

Syntax:

```
ng generate directive
<name>[options]<strong>or</strong>
ng g directive
<name> [options]
```

previous next

Angular CLI Commands

enum

It create a new, generic enum definition for the given or default project.

Syntax:

```
ng generate enum
<name>[options]<strong>or</strong>
ng g enum  <name>
[options]
```

guard

It is used to generate a new and generic route definition in the given or default project.

Syntax:

```
ng generate enum
<name> [options]<strong>or</strong>
ng g enum <name>
[options]
```

Angular CLI Commands

interface

It create a new generic interface definition in the given or default project.

Syntax:

```
ng generate interface
<name><type> [options] <strong>or</strong>
ng g interface
<name><type>
[Options]
```

Library

It can used to create a new generic library project in the current workspace.

Syntax:

```
 ng generate library
<name> [options] <strong>or</strong>
 ng g library <name>
 [options]
```

Angular CLI Commands

Module

It creates a new generic NgModule definition in the given or default project.

Syntax:

```
ng generate module
<name> [options]<strong> or</strong>
ng g module <name>
[options]
```

Pipe

It creates a new generic pipe definition in the given or default project.

Syntax:

```
ng generate pipe
<name> [options]
```

Angular CLI Commands

Service

It is used to create a new and generic service definition in the given or default project.

Syntax:

```
 ng generate service
<name> [options]
```

serviceWorker

It is used to pass this to "run" Command to create a service worker.

Syntax:

```
ng generate
serviceWorker
```

Angular CLI Commands

universal

This Command is used to pass this simplified to the "run" command to set up server-side rendering for an app.

Syntax:

```
ng generate
 universal  [options]
```

Options

–default=true|false: When true, it disables interactive input prompts for options with a default.

-dryRun=true|false: When true, it runs through and reports activity without writing out results.

Default: false

Aliases: -d

–force=true|false: It forces overwriting of existing files, when true

Default: false

Aliases: -f

Angular CLI Commands

Aliases: -f

–help=true|false|json|JSON: Used to show a help message in the console.

Default: false

–interactive=true|false: When it is false, it disables interactive input prompts.

Angular additional references

Angular State management: https://redux.js.org/
https://blog.angular-university.io/angular-ngrx-store-and-effects-crash-course/
Angular Storage : https://www.npmjs.com/package/angular-web-storage
Debugging : https://augury.rangle.io/

Architecture of angular application : https://nx.dev/angular/getting-started/what-is-nx
Style Guide: https://angular.io/guide/styleguide

Reuse components in your application

https://material.agnular.io
https://primefaces.org/primeng/#/

Questions?

Thank you
Venkatesh Reddy Madduri