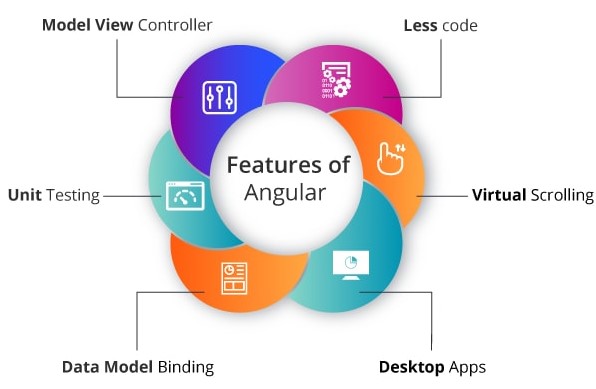
# 1. What is Angular ?

The name Angular derives simply from the fact that the HTML tags are enclosed by angle brackets.

Originally called AngularJS, Angular is Google’s JavaScript (TypeScript-based) open-source front-end web application framework. It is designed specifically for creating dynamic web applications. With this framework, you can develop front-end based applications without having to use other plug-ins or frameworks.

Angular is very similar to the [JavaScript](https://www.besanttechnologies.com/training-courses/web-designing-training/javascript-training-institute-in-chennai) framework as it is open-source. As there is great support from th**Angular CLI and Angular App**e Google team and many ideas are being imported, Angular is said to be up to date. All the latest trends available in today’s market are incorporated with this wonderful framework.



#### Less code:

Plenty of scripts is necessary just to design an application when performing DOM manipulation. But the very minimal amount of code is enough for DOM manipulation if using Angular.

#### Model View Controller:

The angular framework is constructed on a well-known idea called Model-View-Controller (MVC). I hope you all are aware that MVC is a design pattern that is used in the entire web applications in today’s modern trend.

#### Unit Testing:

We have an excellent test framework known as Karma designed by the Google development team. This is very much helpful for performing designing unit tests for AngularJS applications.

#### Data Model Binding:

While binding data to the HTML controls, special code is not required. Simply by adding very few snippets of code is possible to bind data, which is done through Angular.

#### Desktop Apps:

Using Angular, you can easily create applications that are desktop-installed across various operating systems such as Windows, Mac, and Linux.

#### Virtual Scrolling:

To load and unload items from DOM, Virtual Scrolling in Angular is principally used. This process is completely based on visible parts of lists.

# 2. What is TypeScript ?

TypeScript is a new language developed by Microsoft that extends JavaScript. It is a superset of JavaScript ES2015 and incorporates the new version of JavaScript functionality. Using TypeScript, you can write state-of-the-art JavaScript without using Babel. TypeScript also has a powerful typing system that allows static analysis of code through type annotations and type interfaces.

Because TypeScript is heavily influenced by Java and the .NET framework (C # and VB), TypeScript is easier to learn than pure JavaScript if developers are experienced with these languages. Among the major frameworks, Angular was the first to have actively adopted TypeScript.

# 3. What is ES6 or ECMAScript ?

ES5 is an abbreviation of **ECMAScript 5** and also known as **ECMAScript** 2009. The sixth edition of the **ECMAScript** standard is **ES6** or **ECMAScript 6**. It is also known as ECMAScript 2015. [ES6](https://www.javatpoint.com/es6) is a major enhancement in the [JavaScript](https://www.javatpoint.com/javascript-tutorial) language that allows us to write programs for complex applications.

## **Difference between ES5 and ES6**

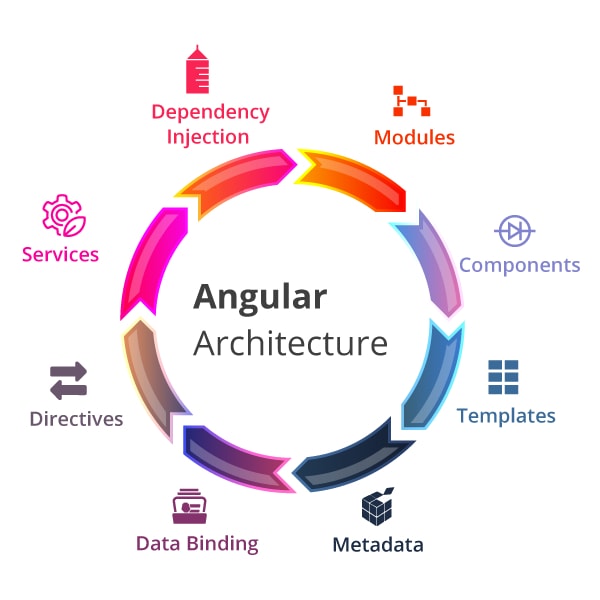
|  |  |  |
| --- | --- | --- |
| **Based on** | **ES5** | **ES6** |
| **Definition** | ES5 is the fifth edition of the ECMAScript (a trademarked scripting language specification defined by ECMA International) | ES6 is the sixth edition of the ECMAScript (a trademarked scripting language specification defined by ECMA International). |
| **Release** | It was introduced in 2009. | It was introduced in 2015. |
| **Data-types** | ES5 supports primitive data types that are **string, number, boolean, null,** and **undefined**. | In ES6, there are some additions to JavaScript data types. It introduced a new primitive data type **'symbol'** for supporting unique values. |
| **Defining Variables** | In ES5, we could only define the variables by using the **var** keyword. | In ES6, there are two new ways to define variables that are **let** and **const**. |
| **Performance** | As ES5 is prior to ES6, there is a non-presence of some features, so it has a lower performance than ES6. | Because of new features and the shorthand storage implementation ES6 has a higher performance than ES5. |
| **Support** | A wide range of communities supports it. | It also has a lot of community support, but it is lesser than ES5. |
| **Object Manipulation** | ES5 is time-consuming than ES6. | Due to destructuring and speed operators, object manipulation can be processed more smoothly in ES6. |
| **Arrow Functions** | In ES5, both **function** and **return** keywords are used to define a function. | An arrow function is a new feature introduced in ES6 by which we don't require the **function** keyword to define the function. |
| **Loops** | In ES5, there is a use of **for** loop to iterate over elements. | ES6 introduced the concept of **for...of** loop to perform an iteration over the values of the iterable objects. |

# 4. What is diff between ES6 ,TypeScript, Javascript ?

# 5. What are the differences between Angular and AngularJS

| **Parameters** | **AngularJS** | **Angular** |
| --- | --- | --- |
| Script | Based on JavaScript | Based on TypeScript |
| To use a property or event | you must remember the correct ng directive | uses () for event binding and [] for property binding . |
| Mobile features | Not focused on this | Supports mobile features |
| Syntax for routing | Uses $routeProvider.when()to configure routing | Uses @RouteConfig{(...}). |
| Performance | Drawing speed of the view is moderate | Drawing speed of the view is 3 to 5 times faster than Angular JS. Template pre-compilation and view caching are used to reduce memory usage and CPU load. |
| Template engine | Has many directives and every developer can also specify new custom directive. | Angular also has standard directives, but they are used in a bit different way. E.g.: ng-model in AngularJS means that you want to create two-way binding. If you want to create one-way binding, you should use ng-bind. |
| Features that affect supporting by various browsers | Have controllers | Controllers are replaced by components. |
| One-way data binding | ng-bind is used to link data in one direction (one-way data binding) | Replaced by one-way data binding.[property] |
| Two-way data binding links | ng-model | replaced by [(ngModel)] |

# 6. How does an Angular application work?



Every Angular app consists of a file named angular.json. This file will contain all the configurations of the app.

While building the app, the builder looks at this file to find the entry point of the application.

Following is an image of the angular.json file:

"build": {

"builder": "@angular-devkit/build-angular:browser",

"options": {

"outputPath": "dist/angular-starter",

"index": "src/index.html",

"main": "src/main.ts",

"polyfills": "src/polyfills.ts",

"tsConfig": "tsconfig.app.json",

"aot": false,

"assets": [

"src/favicon.ico",

"src/assets"

],

"styles": [

"./node\_modules/@angular/material/prebuilt-themes/deeppurple-amber.css",

"src/style.css"

]

}

}

Inside the build section, the main property of the options object defines the entry point of the application which in this case is **main.ts**.  
The main.ts file creates a browser environment for the application to run, and, along with this, it also calls a function called **bootstrapModule**, which bootstraps the application. These two steps are performed in the following order inside the main.ts file:

import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

platformBrowserDynamic().bootstrapModule(AppModule)

In the above line of code, **AppModule** is getting bootstrapped.  
The AppModule is declared in the app.module.ts file. This module contains declarations of all the components.  
Below is an example of app.module.ts file:

import { BrowserModule } from '@angular/platform-browser';

import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';

@NgModule({

declarations: [

AppComponent

],

imports: [

BrowserModule

],

providers: [],

entryComponents: [],

bootstrap: [AppComponent]

})

export class AppModule { }

As one can see in the above file, **AppComponent** is getting bootstrapped.  
This component is defined in **app.component.ts** file. This file interacts with the webpage and serves data to it.  
Below is an example of app.component.ts file:

import { Component } from '@angular/core';

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class AppComponent {

title = 'angular';

}

Each component is declared with three properties:  
1. **Selector** - used for accessing the component  
2. **Template/TemplateURL** - contains HTML of the component  
3. **StylesURL** - contains component-specific stylesheets  
  
After this, Angular calls the **index.html** file. This file consequently calls the root component that is **app-root**. The root component is defined in **app.component.ts**.  
This is how the index.html file looks:

<!doctype html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>Angular</title>

<base href="/">

<meta name="viewport" content="width=device-width, initial-scale=1">

</head>

<body>

<app-root></app-root>

</body>

</html>

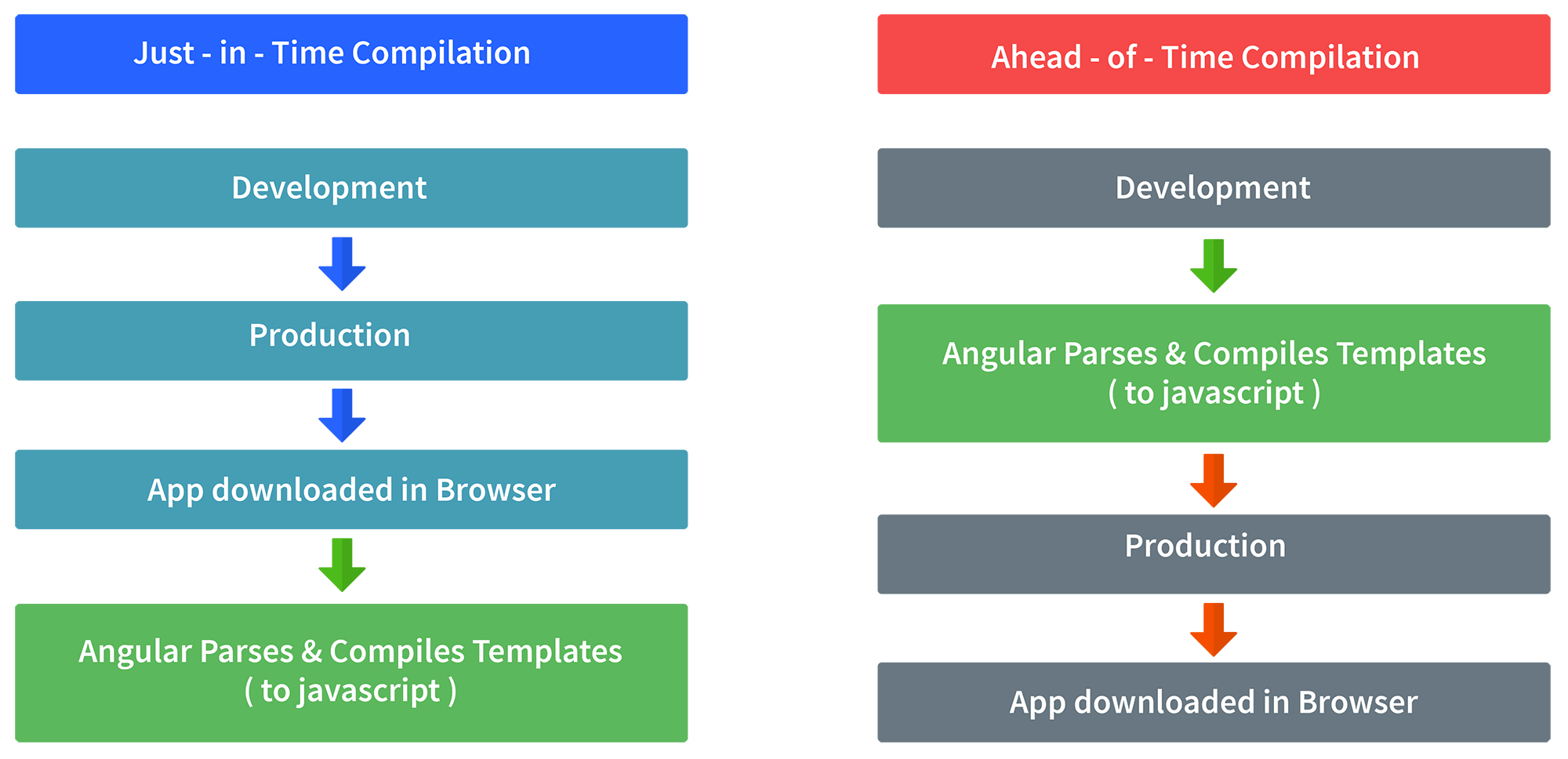
The HTML template of the root component is displayed inside the <app-root> tags.  
  
This is how every angular application works.

# 7. What is AOT compilation? What are the advantages of AOT?

Every Angular application consists of components and templates which the browser cannot understand. Therefore, all the Angular applications need to be compiled first before running inside the browser.  
  
Angular provides two types of compilation:

 JIT(Just-in-Time) compilation

 AOT(Ahead-of-Time) compilation



In JIT compilation, the application compiles inside the browser during runtime.  
Whereas in the AOT compilation, the application compiles during the build time.  
  
The advantages of using AOT compilation are:

 Since the application compiles before running inside the browser, the browser loads the executable code and renders the application immediately, which leads to **faster rendering**.

 In AOT compilation, the compiler sends the external HTML and CSS files along with the application, eliminating separate AJAX requests for those source files, which leads to **fewer ajax requests**.

 Developers can detect and handle errors during the building phase, which helps in **minimizing errors**.

 The AOT compiler adds HTML and templates into the JS files before they run inside the browser. Due to this, there are no extra HTML files to be read, which provide **better security** to the application.

By default, angular builds and serves the application using JIT compiler:

ng build  
ng serve

For using AOT compiler following changes should be made:

ng build --aot  
ng serve --aot

# 8.Explain Components, Modules and Services in Angular ?

create an Angular application by running the following inside the command terminal:

For better understanding, I would like you to create an Angular application by running the following inside the command terminal:

ng new angularApp

The above command will create an angular application in the directory.  
Next, let's move on to understand Components, Modules, and Services.  
  
**Components**  
In Angular, components are the basic building blocks, which control a part of the UI for any application.  
A component is defined using the **@Component** decorator. Every component consists of three parts, the template which loads the view for the component, a stylesheet which defines the look and feel for the component, and a class that contains the business logic for the component.  
For creating a component, inside the command terminal, navigate to the directory of the application created, and run the following command:

ng generate component test

Or

ng g c test

One can see the generated component inside src/app/test folder. The component will be defined inside test.component.ts and this is how it looks:

import { Component, OnInit } from '@angular/core';

@Component({

selector: 'app-test',

templateUrl: './test.component.html',

styleUrls: ['./test.component.css']

})

export lass TestComponent implements OnInit {

constructor() {}

ngOnInit() {

}

}

As we can see in the above image, our component is defined with **@Component** decorator.

**Modules**  
A module is a place where we can group components, directives, services, and pipes. Module decides whether the components, directives, etc can be used by other modules, by exporting or hiding these elements. Every module is defined with a @NgModule decorator.  
By default, modules are of two types:

 Root Module

 Feature ModuleEvery application can have only one root module whereas, it can have one or more feature modules.  
A root module imports **BrowserModule**, whereas a feature module imports **CommonModule**.  
In the application that we created before, one can see that the root module is defined inside **app.module.ts** and this is how it looks:

import { BrowserModule } from '@angular/platform-browser';

import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';

import { TestComponent } from './test/text.component';

@NgModule({

declarations: [

AppComponent,

TestComponent

],

imports: [

BrowserModule

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

We can see in the above image that the component we created earlier is already imported in the declarations array.  
  
To create a feature module, run the following command:

ng g m test-module

The module is created inside the src/app/test-module/test-module.module.ts file:

import { NgModule } from '@angular/core';

import { CommonModule } from '@angular/common';

@NgModule({

declarations: [],

imports: [

CommonModule

]

})

export class TestModuleModule { }

As one can see, **CommonModule** is imported since this is a feature module.

**Services** Services are objects which get instantiated only once during the lifetime of an application. The main objective of a service is to share data, functions with different components of an Angular application.  
A service is defined using a **@Injectable** decorator. A function defined inside a service can be invoked from any component or directive.  
  
To create a service, run the following command:

ng g s test-service

The service will be created inside src/app/test-service.service.ts:

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

@Injectable({

providedIn: 'root'

})

export class TestServiceService {

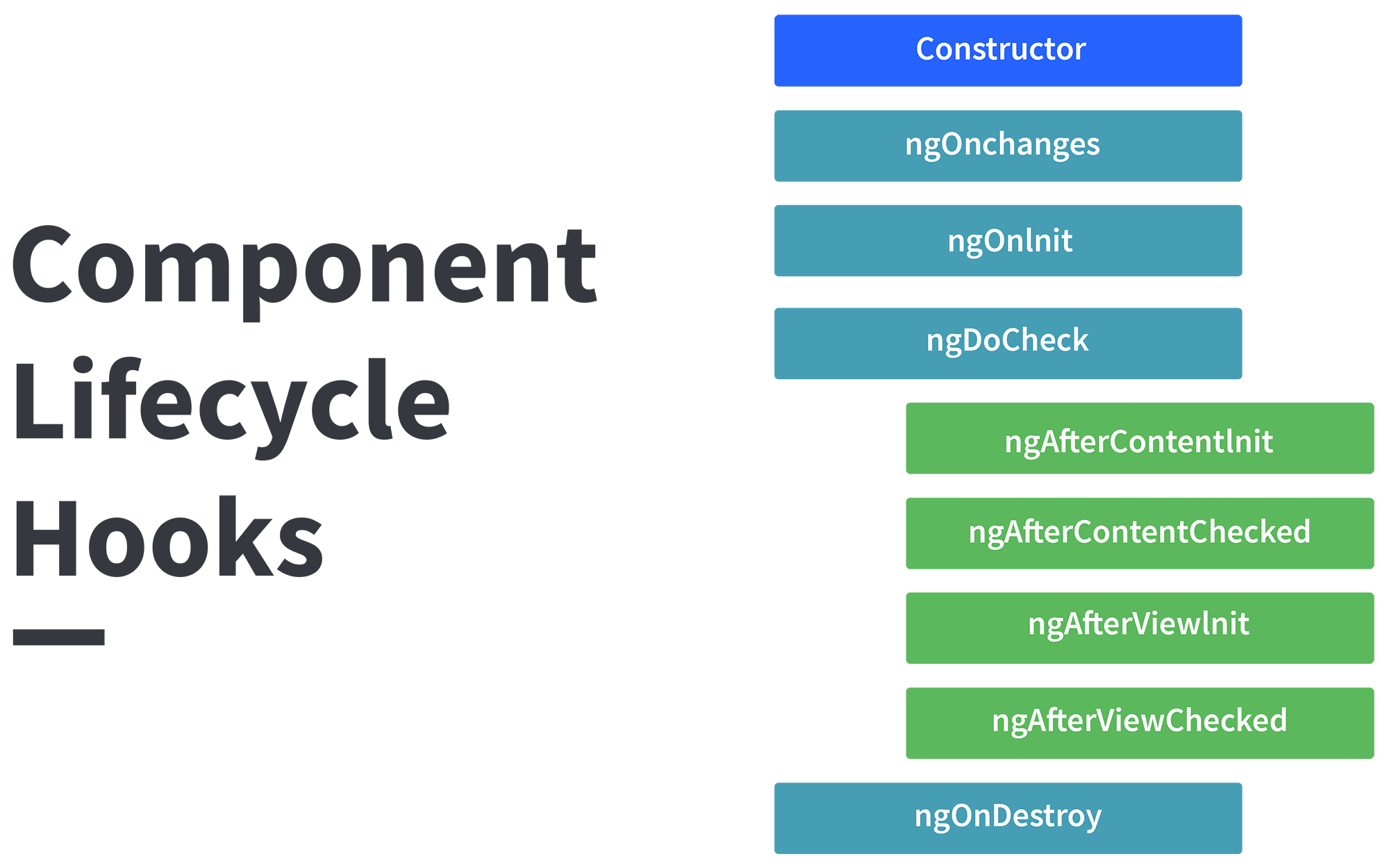
constructor() { }

}

Any method/function defined inside the TestServiceService class can be directly used inside any component by just importing the service.

# 9. What are lifecycle hooks in Angular? Explain a few lifecycle hooks.

Every component in Angular has a lifecycle, different phases it goes through from the time of creation to the time it's destroyed. Angular provides **hooks** to tap into these phases and trigger changes at specific phases in a lifecycle.



**ngOnChanges( )** This hook/method is called before **ngOnInit** and whenever one or more input properties of the component changes.  
This method/hook receives a SimpleChanges object which contains the previous and current values of the property.  
  
**ngOnInit( )** This hook gets called once, after the **ngOnChanges** hook.  
It initializes the component and sets the input properties of the component.  
  
**ngDoCheck( )** It gets called after **ngOnChanges** and **ngOnInit** and is used to detect and act on changes that cannot be detected by Angular.  
We can implement our change detection algorithm in this hook. **ngAfterContentInit( )** It gets called after the first **ngDoCheck** hook. This hook responds after the content gets projected inside the component.  
  
**ngAfterContentChecked( )** It gets called after **ngAfterContentInit** and every subsequent **ngDoCheck**. It responds after the projected content is checked.  
  
**ngAfterViewInit( )** It responds after a component's view, or a child component's view is initialized.  
  
**ngAfterViewChecked( )** It gets called after **ngAfterViewInit**, and it responds after the component's view, or the child component's view is checked.  
  
**ngOnDestroy( )** It gets called just before Angular destroys the component. This hook can be used to clean up the code and detach event handlers.  
  
Let’s understand how to use **ngOnInit** hook, since it’s the most oftenly used hook. If one has to process lot of data during component creation, it’s better to do it inside **ngOnInit** hook rather than the constructor:

import { Component, OnInit } from '@angular/core';

@Component({

selector: 'app-test',

templateUrl: './test.component.html',

styleUrls: ['./test.component.css']

})

export class TestComponent implements OnInit {

constructor() { }

ngOnInit() {

this.processData();

}

processData(){

// Do something..

}

}

As you can see we have imported OnInit but we have used **ngOnInit** function. This principle should be used with the rest of the hooks as well

# 10. Explain string interpolation and property binding in Angular?

String interpolation and property binding are parts of **data-binding** in Angular.  
Data-binding is a feature in angular, which provides a way to communicate between the component(Model) and its view(HTML template).  
Data-binding can be done in two ways, **one-way** binding and **two-way** binding.  
In Angular, data from the component can be inserted inside the HTML template. In one-way binding, any changes in the component will directly reflect inside the HTML template but, vice-versa is not possible. Whereas, it is possible in two-way binding.  
  
String interpolation and property binding allow only one-way data binding.  
String interpolation uses the double curly braces **{{ }}** to display data from the component. Angular automatically runs the expression written inside the curly braces, for example, {{ 2 + 2 }} will be evaluated by Angular and the output 4, will be displayed inside the HTML template. Using property binding, we can bind the DOM properties of an HTML element to a component's property. Property binding uses the square brackets **[ ]** syntax.

# 11. How are observables different from promises?

The first difference is that an Observable is **lazy** whereas a Promise is **eager**.

|  |  |
| --- | --- |
| Promise | Observable |
| Emits a single value | Emits multiple values over a period of time |
| Not Lazy | Lazy. An observable is not called until we subscribe to the observable |
| Cannot be cancelled | Can be cancelled by using the unsubscribe() method |
|  | Observable provides operators like map, forEach, filter, reduce, retry, retryWhen etc. |

Consider the following Observable:

const observable = rxjs.Observable.create(observer => {

console.log('Text inside an observable');

observer.next('Hello world!');

observer.complete();

});

console.log('Before subscribing an Observable');

observable.subscribe((message)=> console.log(message));

When you run the above Observable, you can see messages being displayed in the following order:

Before subscribing an Observable  
Text inside an observable  
Hello world!

As you can see, observables are lazy. Observable runs only when someone subscribes to them hence, the message “Before subscribing…” is displayed ahead of the message inside the observable.  
  
Now let’s consider a Promise:

const promise = new Promise((resolve, reject) => {

console.log('Text inside promise');

resolve('Hello world!');

});

console.log('Before calling then method on Promise');

greetingPoster.then(message => console.log(message));

Running the above promise, the messages will be displayed in the following order:

Text inside promise  
Before calling then method on Promise  
Hello world!

As you can see the message inside Promise is displayed first. This means that a promise runs before the **then** method is called. Therefore, promises are **eager**.  
  
The next difference is that Promises are always **asynchronous**. Even when the promise is immediately resolved. Whereas an Observable, can be both **synchronous** and **asynchronous**.  
  
The above example of an observable is the case to show that an observable is synchronous. Let’s see the case where an observable can be asynchronous:

const observable = rxjs.Observable.create(observer => {

setTimeout(()=>{

observer.next('Hello world');

observer.complete();

},3000)

});

console.log('Before calling subscribe on an Observable');

observable.subscribe((data)=> console.log(data));

console.log('After calling subscribe on an Observable');

The messages will be displayed in the following order:

Before calling subscribe on an Observable  
After calling subscribe on an Observable  
Hello world!

You can see in this case, observable runs asynchronously.  
  
The next difference is that Observables can emit **multiple** values whereas Promises can emit only one value.  
  
The biggest feature of using observables is the use of **operators**. We can use multiple operators on an observable whereas, there is no such feature in a promise.

# 12. What are directives in Angular?

A directive is a class in Angular that is declared with a @Directive decorator.

Every directive has its own behaviour and can be imported into various components of an application.

When to use a directive?

Consider an application, where multiple components need to have similar functionalities.

The norm thing to do is by adding this functionality individually to every component but,

this task is tedious to perform. In such a situation, one can create a directive having the

required functionality and then, import the directive to components which require this functionality.

Types of directives

Component directives

These form the main class in directives. Instead of @Directive decorator we use @Component decorator to declare these directives. These directives have a view, a stylesheet and a selector property.

Structural directives

These directives are generally used to manipulate DOM elements.

Every structural directive has a ‘ \* ’ sign before them.

We can apply these directives to any DOM element.

In the above example, we can \*ngIf and \*ngFor directives being used.

\*ngIf is used to check a boolean value and if it’s truthy,the div element will be displayed.

\*ngFor is used to iterate over a list and display each item of the list.

Attribute Directives

These directives are used to change the look and behaviour of a DOM element.

Let’s understand attribute directives by creating one:

How to create a custom directive?

We’re going to create an attribute directive:

In the command terminal, navigate to the directory of the angular app and type the following command to generate a directive:

ng g directive blueBackground

The following directive will be generated. Manipulate the directive to look like this:

import { Directive, ElementRef } from '@angular/core';

@Directive({

selector: '[appBlueBackground]'

})

export class BlueBackgroundDirective {

constructor(el:ElementRef) {

el.nativeElement.style.backgroundColor = "blue";

}

}

Now we can apply the above directive to any DOM element:

<p appBlueBackground>Hello World!</p>

# 13. How does one share data between components in Angular?

A . Using Routing from one component to other component

B--->Parent/ child design flow:

1.Parent to child using @Input decorator

2.

Child to parent using @Output and EventEmitter

3.Child to parent using @ViewChild decorator:

Child component---->:

import {Component} from '@angular/core';

@Component({

selector: 'app-child',

template:`

<p>{{data}}</p>

`,

styleUrls: ['./child.component.css']

})

export class ChildComponent {

data:string = "Message from child to parent";

constructor() { }

}

Parent Component------------->:

import { Component,ViewChild, AfterViewInit} from '@angular/core';

import { ChildComponent } from './../child/child.component';

@Component({

selector: 'app-parent',

template: `

<p>{{dataFromChild}}</p>

` ,

styleUrls: ['./parent.component.css']

})

export class ParentComponent implements AfterViewInit {

dataFromChild: string;

@ViewChild(ChildComponent,{static:false}) child;

ngAfterViewInit(){

this.dataFromChild = this.child.data;

}

constructor() { }

}

In the above example, a property named “data” is passed from the child component to the parent component.

@ViewChild decorator is used to reference the child component as “child” property.

Using the ngAfterViewInit hook, we assign the child’s data property to the messageFromChild property

and use it in the parent component’s template.

----------

# 14. Explain the concept of Dependency Injection?

Dependency injection is an application design pattern which is implemented by Angular.

It also forms one of the core concepts of Angular.

So what is dependency injection in simple terms?

Let’s break it down, dependencies in angular are nothing but services which have a functionality. Functionality of a service, can be needed by various components and directives in an application. Angular provides a smooth mechanism by which we can inject these dependencies in our components and directives.

So basically, we are just making dependencies which are injectable across all components of an application.

# 15. How to make module as lazy Loading?

# 16. How to make shared module ?

# 17. What is Differance between CanActivate and CanDeactivate ? And when to use them?

Controlling Access to or from a Route:

To control whether the user can navigate to or away from a given route, use route guards.

For example, we may want some routes to only be accessible once the user has logged in or accepted Terms

& Conditions. We can use route guards to check these conditions and control access to routes.

Route guards can also control whether a user can leave a certain route. For example, say the user has typed

information into a form on the page, but has not submitted the form. If they were to leave the page,

they would lose the information. We may want to prompt the user if the user attempts to leave the route without submitting

or saving the information

canActivate and canDeactivate are guards, when your app multiple roles in application, like user, admin etc,

you use guards to protece them, also you use canActivate for pages which are accessible only for loggedin users

and vice versa for canDeactivate

Registering the Route Guards with Routes:

In order to use route guards, we must register them with the specific routes we want them to run for.

For example, say we have an accounts route that only users that are logged in can navigate to.

This page also has forms and we want to make sure the user has submitted unsaved changes before leaving the accounts page.

In our route config we can add our guards to that route:

# 18.What is resolver in angular ? why it used?

# 19.What is Differance between resolver and CanActivate and canDeactivate ?

there are a lot of guards in Angular that we can use to protect routes and provide the right page for the right user.

Guards are processed in the following order:

canDeactivate

canLoad

canActivateChild

canActivate

resolve

--1.canActivate (Guard navigation to a route):

The canActivate is called when the URL changes to the route and much the route with the Guard. This type of guards is commonly used to:

Limit route access to specific users

Ensure prerequisites are met

If you use Angular CLI, you can generate the guard you want with the following command:

'ng g g article/auth'

The first g is for generate and the second is for guard. After launching this command on the terminal,

the Angular CLI will ask you which guard do you want to implement except canDeactivate and canLoad,

I’ll talk about them later. After choosing the guard type, the CLI generates the guard with its testing file.

--2.canActivateChild (Guard navigation to a child route)

This guard type is similar to the canActivate guard except that it is called when activating a route child

and not the route itself. This guard job is to check the criteria before activating a child route. This type of guards is commonly used when:

Limit access to child route

Ensure prerequisites for the child route are met

--3.canDeactivate (Guard navigation away from a route):

The canDeactivate guard’s job is to check criteria before leaving this time a route. Some common use cases for this type

of guard are:

Check for unsaved changes

Confirm leaving an incomplete operation

Alert the user

--4.canLoad (Prevent asynchronous routing):

The canLoad guard is implemented to decide if children can be loaded or not.

You might be wondering what’s the difference between the canLoadand the canActivate?

Well, there is a difference, the canActivate exists to prevent unauthorized users from accessing a route, while canLoad

is used to prevent the application from loading an entire module or component in a lazy way (lazy loading) if the user

is not authorized.

So the main advantage of this type of guard is to optimize the app and have better performance.

--5.resolve (Prefetch data before activating a route):

The last type I’m going to talk about is the resolve guard. This guard allows us data before we navigate to a route.

You may say, wait but we can retrieve data in the ngOnInit() life cycle hook, right? I agree but this approach will lead

us to see an empty component at the beginning. If your client is okay with an empty component at the opening or a spinner

while loading data, that’s okay but if not there is a solution for that which is the resolve guard. This allows you to

render the component along with data.

# 20. What is Interceptor in angular ? why it used?

# 21. What is Differance between Interceptor and resolver and CanActivate and CanActivate ? And when to use them?

# 22. How to push data on Post method REST API ?

# 22. How to Get data on Get method REST API ?

# 23. How to Get data on Get method REST API for non Json type response ?

# 24. How to check any input is String or Object or Array ?

# 25. How to create MAP ,Set and List in angular?

# 26. How to create select and button input in angular?

# 26. What is pipe in angular ? why it used?

# 27. How is popup in angular ?

# 28.What is Differance between ng-bootstrap and primeng ?

# 28.What is amCharts in angular ?

amCharts used for responsive graph -Map, XY graph,Pie,Venn Diagram

https://www.amcharts.com/docs/v4/getting-started/

# 29.What is primeng in angular ?

PrimeNG :PrimeNG is a rich set of open source native Angular UI components.

PrimeNG Components used for responsive -widget like Button.grid,table,menu,Popup etc.

More:https://primefaces.org/primeng/showcase/#/setup