

Angular For Beginners



The most comprehensive beginner guide for Angular

Course Content

1. Introduction
 - a. Introduction
 - b. Version history
 - c. Advantages of Angular
 - d. Quiz
2. Project setup
 - a. Different commands
 - b. Installation (node js, Angular CLI, VS Code editor)
 - c. Creation of project
 - d. Running the project
 - e. Quiz
3. Angular project architecture & Flow of execution
 - a. A quick glance on different folders
 - b. Angular Architecture
 - c. Flow of execution
 - d. Quiz
4. Components
 - a. What are the components?
 - b. Creating components
 - i. Custom components
 - ii. Using CLI
 - c. Adding bootstrap
 - d. Quiz
5. Data Binding
 - a. String Interpolation
 - b. Property Binding
 - c. Event Binding
 - d. Two-way Binding
 - e. Quiz
6. Directives
 - a. Structural Directives
 - i. *ngIf
 - ii. *ngFor
 - b. Attribute Directive
 - i. ng style
 - ii. ngClass
 - c. Quiz

What is Angular?

Angular is a Typescript based framework used to build client-side applications. Especially Single-Page-Applications.

Typescript is a superset of JavaScript and it compiles down to JavaScript at the end.

In Single-Page-Applications Parts of the views get refreshed asynchronously without having to reload the entire page.

History of Angular

Angular is a product developed by Google and It makes use of Typescript language which is developed by Microsoft.

A single page application is the one which works inside the browser and does not require the page reloading during use.

Some of the examples for single-page applications-

Gmail, Twitter, Google Drive, Google maps

Now let us have a look into different versions of Angular.

In general, we can expect the following release cycle.

- A major release every 6 months
- 1-3 minor release for each major release
- A patch release and pre-release build almost every week.

Version	Year	Called As
1	2010	Angular JS
2	2016 (September)	Angular 2
4	2016 (December)	Angular 4
5	2017 (November)	Angular 5
6	2018 (April)	Angular 6
7	2018 (October)	Angular 7
8	2019 (May)	Angular 8
9	2020 (February)	Angular 9

Version 3 was skipped because of some active and huge development in the router section

What are the advantages of angular

1. It separates HTML code from Business logic.
2. It provides a modular approach hence the application that we build will have a clear structure.
3. Making use of components we can have a lot of reusable code.
4. Using routing, validation, services extra make development quicker and easier.

Basic setup

Project Setup

```
> npm install -g @angular/cli  
  
> ng new my-dream-app  
  
> cd my-dream-app  
  
> ng serve
```

These are different commands Used in Angular Applications.

To run above commands we need one tool so

1. Download Node.js : (<https://nodejs.org/en/>)

Node.js is a server side language. We are not writing any code but Node.js will be used by the cli (command line interface) to bundle and optimize our project.

And we use npm(node package manager) to manage different dependencies in our Angular project. Dependencies means angular framework itself and also some libraries that framework uses.

So using the above link we can install Node.js and npm tools on our machine.

Once we install these we can run the above command in our Node.js command prompt

->To check version of node installed run the command **node -v**

->To check version of npm installed run the command **npm -v**

Run above commands in Node.js command prompt.

And also install **VS code editor** to follow my course and link is given below

<https://code.visualstudio.com/>

2. Installing Angular CLI :

To create our angular project we will use the official command line interface. Using cli is the recommended and best way to create angular projects. because Angular projects are a bit more elaborate regarding their build workflow.

And there are a couple of files they need to be converted before They can run in the browser and cli does all of it and also heavily optimizes our code so we ship a highly optimized version of our code to the browser.

To install this, in Node.js command prompt run the command,

For windows

npm install -g @angular/cli (g indicates install globally on our machine.)

or **npm install -g @angular/cli@latest**

Here @latest is optional.

For mac or Linux

sudo npm install -g @angular/cli

NOTE: you may get some error when you run this command but you can ignore them and remember to check, at the end you should get like this @angular/cli@9.0.7 (9.0.7 is the version at the time when I installed it.)

Check version of angular CLI using command **ng v**

A terminal window with a black background. The prompt is 'C:\Users\vindhya hegde>ng v'. The output shows the 'Angular CLI' logo in a stylized red font. Below the logo, the text 'Angular CLI: 9.0.7' is displayed in white. Underneath that, 'Node: 12.8.0' is shown in yellow, and 'OS: win32 x64' is shown in white at the bottom.

```
C:\Users\vindhya hegde>ng v
Angular CLI
Angular CLI: 9.0.7
Node: 12.8.0
OS: win32 x64
```

3. Creating new project :

Now in Node.js command prompt navigate through the folder where you need to create your project

cd folder name

and to create new project run the command

ng new project-name

Ex:ng new hello-world

you can give any name that you want

and when we run this command it will ask two questions

```
Your environment has been set up for using Node.js 12.8.0 (x64) and npm.

C:\Users\vindhya hegde>ng new ay-dream-app
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? (Use arrow keys)
> CSS
SCSS [ https://sass-lang.com/documentation/syntax#scss ]
Sass [ https://sass-lang.com/documentation/syntax#the-indented-syntax ]
Less [ http://lesscss.org ]
Stylus [ http://stylus-lang.com ]
```

1. would you like to add Angular routing? (Y/N)

you can enter yes or no, if you are entering no then also we can create routing for our application.

if you are entering yes then it will create default file in order to add routes and its name will be **app-routing.module.ts**

2 . Which stylesheet format would you like to use?

you can choose any of these. but in course I have used CSS so you can see all my files end with .CSS extension

ex: app.component.css

if you choose SCSS your files will have .scss extension

Now it will create a folder with a couple of files and dependencies and the entire workflow setup.

4. Run your project

Now navigate in to your project typing this command

cd project-name

finally to run your app run this command

ng serve

after running this, at the end it will show compiled successfully

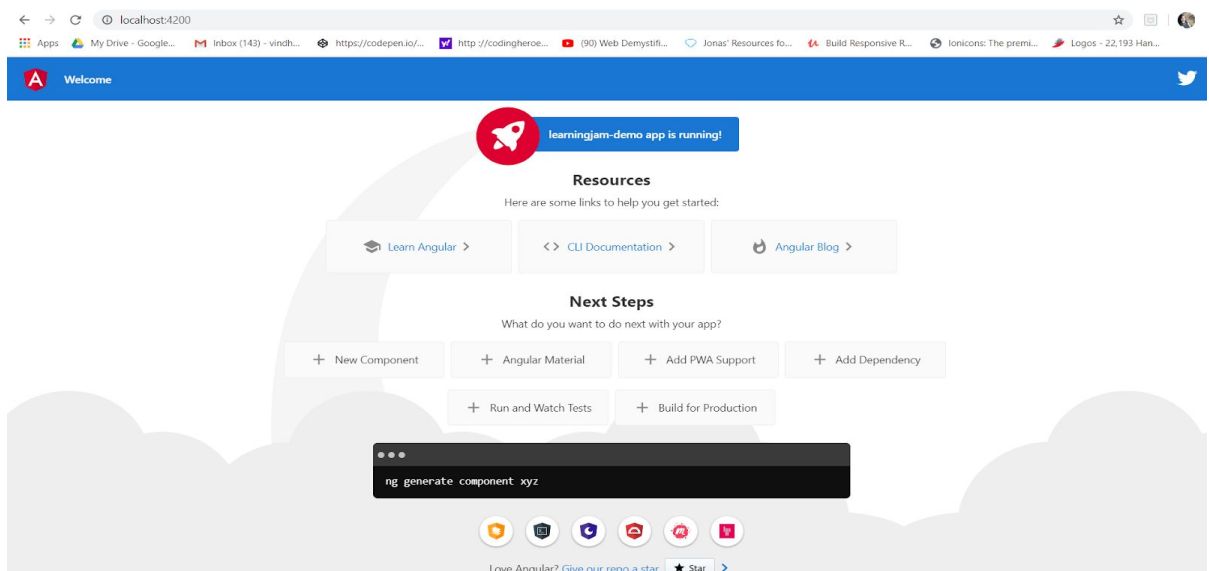
Now run **localhost 4200** on the browser.

server runs on localhost 4200 by default. When you run this it will asks question like this

```
? Would you like to share anonymous usage data about this project with the Angular Team a
t
Google under Google's Privacy Policy at https://policies.google.com/privacy? For more
details and how to change this setting, see http://angular.io/analytics. (y/N) █
```

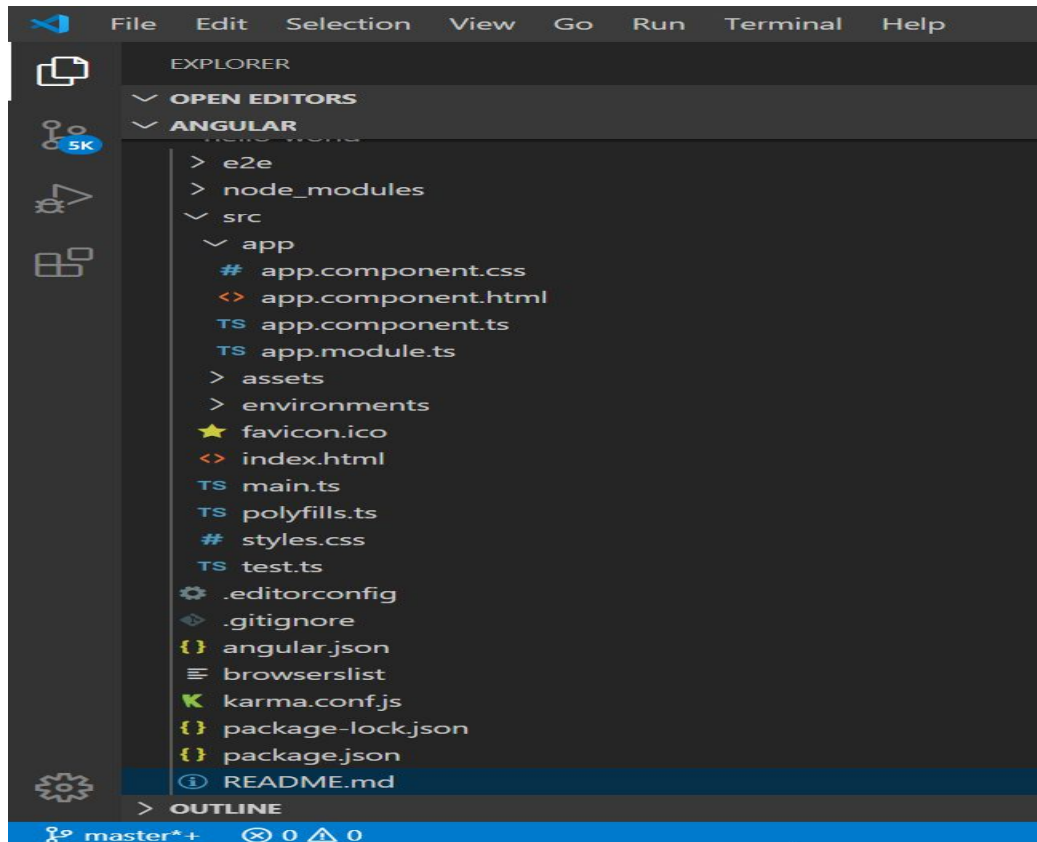
you can enter Yes/No it will not matter.

Finally you will see like this screen on the browser



Angular Architecture & code execution

Folders created



These are the different files and folders created when we run the command **ng new project-name**

Quick glance on different folders created

1. e2e :

It stands for end to end and here we write an end to end test for our application.

2. node_modules

Here we store all third party libraries that our project depends upon. It manages all our dependencies

3. src

Here we have actual source code of our application. We have app component

Its root component and every other component are nested inside the app component. Each component will have an html template (here app.component.html). HTML template is responsible for the view in the browser. And we have a typescript file (app.component.ts). It contains a class that is responsible for that particular view.

- `app.component.html` : its root template of our application. It will contain selectors of all the components that are present in our application.
- `app.component.ts` : It is the typescript file it contains Business logic of our application
- `app.component.css` : it's the file where we add styles.
- `app.component.spec.ts` : this file is used for testing purposes.

4. assets

Here we can store static assets of our application

For example if we have any image files, icons we can store here.

5. environments

here we store configuration settings for different environments in this we have 2 folders

1. One folder is for production environment
2. Another folder is for development environment

6. index.html

If we compare `index.html` in a normal project with the same `index.html` file in an angular project we can observe that here we will not have any reference for script and stylesheet here these references will be dynamically inserted into this page.

7. main.ts

This is the starting point for our application. In a lot of programming languages execution starts from the main function/method. and some concepts in our angular application also. Here we bootstrap main module of our application that is app module

8. Polyfills.ts :

This file imports some scripts that are required to run our Angular application. because Angular uses features of JavaScript that are not available in the current versions of JavaScript supported by the browsers at that time this `polyfills.ts` fill the gap between features of JavaScript that angular needs and features supported by the current version of browsers.

9. Style.css

Here we add global styles to for application

10.test.ts

This is the file which helps us to set a testing environment for our application.

11. .editorconfig

If we are working in a team environment then all the developers in the team should use the same setting in their editor. This is the place where we store settings for our editor.

12. .gitignore

This folder is for excluding certain files and folders from git repository.

13. Karma.config.js

This is a configurable file and Karma is the test runner for JavaScript code.

14. package.json

This is the standard file that every node project should have. here We have the name and version of our application and we are having dependencies. dependencies means it will have libraries on which our angular application is dependent upon.

it will also contain devDependencies.

15.tsconfig.json

It will have a bunch of settings for typescript compilers. our typescript compiler will look at these settings and based on these settings it is going to compile our typescript code into JavaScript.

16. tslint.json

It is a static analysis tool for typescript code It checks typescript code for readability, maintainability and functionality error.

Angular Architecture

a. Angular app contains one or more modules

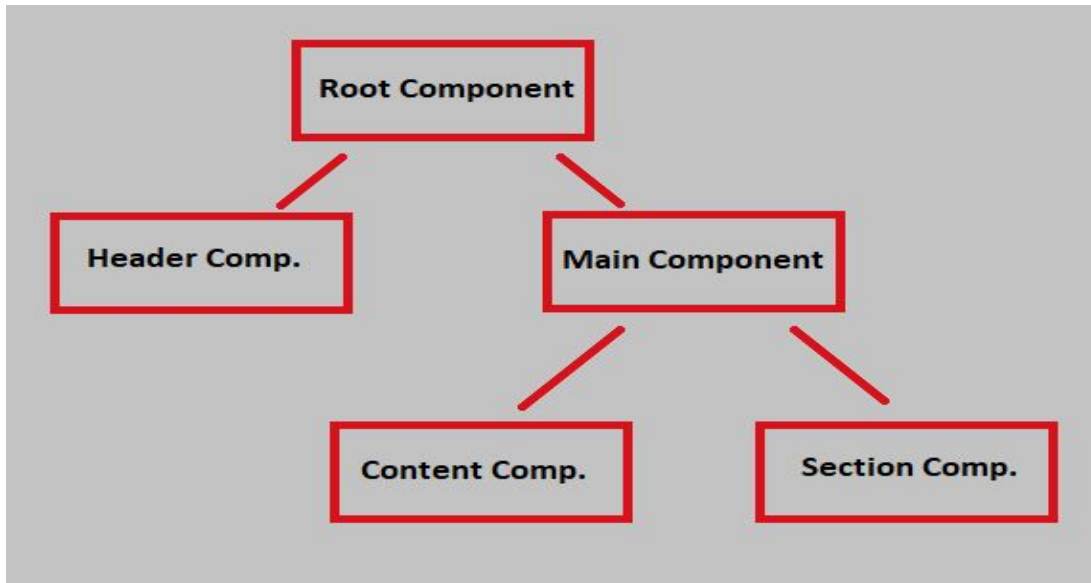
The building blocks of angular apps are modules, angular apps are modular in nature, angular application is just a collection of many individual modules. Every module represents a feature area in our application.

Every angular app has at least one module which is a Root Module by convention this is called an app **module** .

b. Each module is made up of components and services.

components controls portion of the view on the browser.

example: If we consider home-page of our application we can have one component for Header another component for main content like this,



but every angular app will have at least one component that is Root Component and this is again called App Component by convention.

Modules will also contain services that contain business logic for our application.

Understanding the code execution

You can open your project in vs code editor in two ways

1. By simply right click on the project folder-> then select open with vs code editor.
 2. or you can open vs code editor ->then in files -> select open folder ->and select your angular project folder
- in src folder if you open app.component.html and if you scroll down in app.component.html file you can see code like this

```
<span>{{ title }} app is running!</span>

<svg id="rocket-smoke" alt="Rocket Ship Smoke" x="100" y="100">
  <path id="Path_40" data-name="Path 40" d="M644.5,100.0c0.0,0.0,0.0,0.0,0.0,0.0" />
</svg>
```

Consider,

**{{title}} app is running! **

now from where that title is coming from ?

If we open app.component.ts file

```

1   import { Component } from '@angular/core';
2
3   @Component({
4     selector: 'app-root',
5     templateUrl: './app.component.html',
6     styleUrls: ['./app.component.css']
7   })
8   export class AppComponent {
9     title = 'hello-world';
10
11  }
12

```

here we can see a class and in class we have a property called title and that's value will be in my case hello-world

in your case it might be different. by default whatever project name we will give while running command

ng new project-name

that project-name will be set as value for that property in the class. Now let us move to understand how code execution takes place.

Flow of execution

when we run the command **ng serve** to start our application the execution comes to main.ts file

here we bootstrap or kickstart our **appModule**

you can see code for bootstrapping in main.ts,

```

10
11   platformBrowserDynamic().bootstrapModule(AppModule)
12   | .catch(err => console.error(err));
13

```

by looking above code we come to know that in main.ts we bootstrap AppModule now control moves to AppModule there we again bootstrap AppComponent if you look at app.module.ts the bootstrapping code will be,

```

hello-world > src > app > TS app.module.ts > AppModule
2   import { NgModule } from '@angular/core';
3   import { FormsModule } from '@angular/forms';
4   import { AppComponent } from './app.component';
5
6
7
8   @NgModule({
9     declarations: [
10      AppComponent
11    ],
12    imports: [
13      BrowserModule,
14      FormsModule
15    ],
16    providers: [],
17    bootstrap: [AppComponent]
18  })
19  export class AppModule { }

```

now control moves to AppComponent

In the App component we have html template and class to control view logic.

consider fig 2 there in class we have class containing

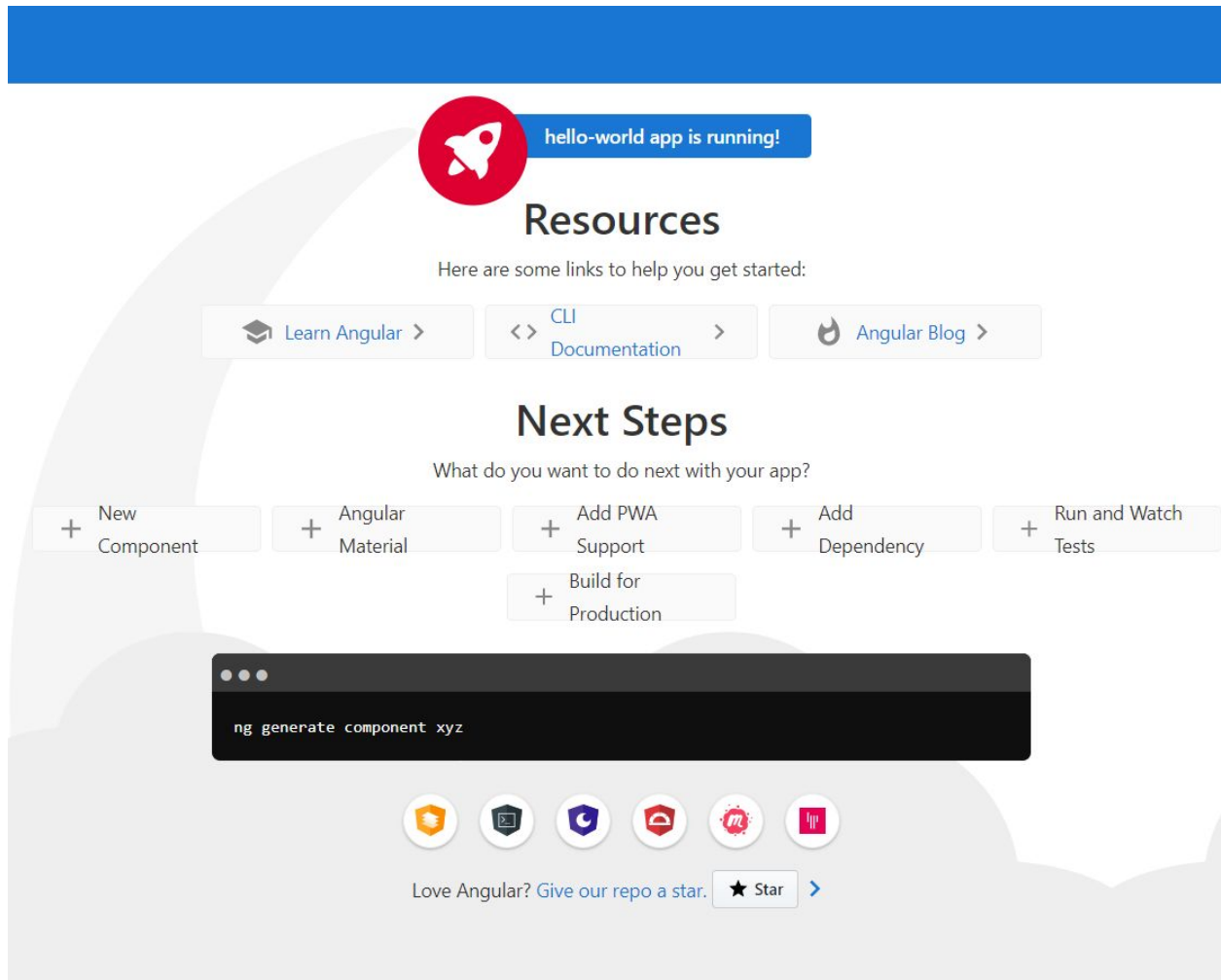
property = **title** and its **value = hello-world**

and if we consider fig 1 we have

```
<span>{{title}} app is running!</span>
```

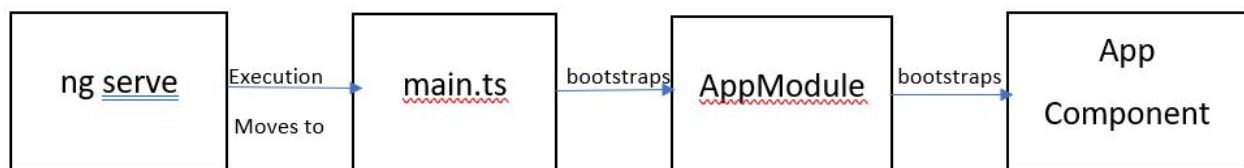
at run time this this title will be replaced by the property **hello-world**

so finally when we run **ng serve** after compilation we can see like this



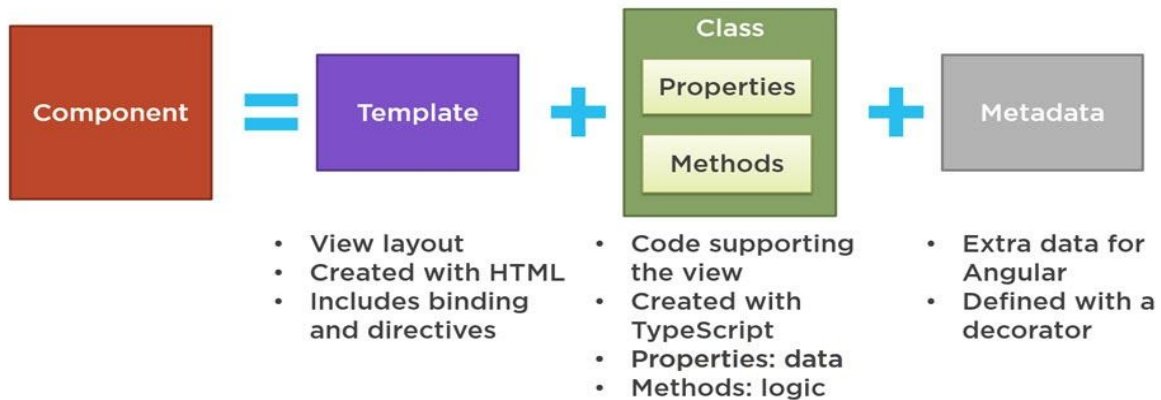
Here hello-world is the property in our class that gets bound to the html template. and in app.component.html we have a lot of default text it will also get rendered in the browser. If you change property title = "Angular" in **app.component.ts** file you can see **Angular app is running!** on the browser.

Final quick summary regarding flow of execution



In the app component property in the class gets bound to the HTML template and finally we can see the view in the browser.

Components



What are components?

components are a key feature in angular .we build our application by composing it from a couple of components which we create on our own.

Component is nothing but simply a typescript class.

Each component has its own template(HTML code), its own styling (CSS code) and its own business logic(typescript code) . We can split our complex applications into components and we can reuse these components.

How to create components

1.creating manually [Custom components]

- a. Right click on app folder
- b. Select create new folder
- c. Give name for that folder and then press enter key.

(giving same name for folder and component is the good practice)

ex: if i create a folder called **server** and i will also consider the name of the component as **server**

Create a typescript file to hold our component and syntax as follows

Component name.component.ts

In my example [server.component.ts]

After this we will have an empty file. as i mentioned component is a typescript class to create this class

follow the below steps

1. Import Component Decorator:(in server.component.ts file)


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

2. Use Decorator and Add Metadata

Now we're going to use the **@component** decorator, pass it some data and then export our component class.

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-server',
  template: ` server Component
` })
export class serverComponent {}
```

In the context of programming, decorators are a design pattern. They are an alternative to subclassing.

We've informed Angular that the class **serverComponent** is an *Angular Component*. Now, Angular will treat it like a component. It will use all the metadata we pass an **augment** to the **serverComponent** class. Our class is now an Angular component.

3. Specify Template

An Angular component is a **directive with a template**.

There are two ways you can add a template to our component

1. Inline Template

Here you add the template in the metadata. Use this when the template code is less.

Note: whenever you use an inline template remember to use backtick (``) and enclose your HTML code inside this backtick.

Example for inline template is given below

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-server',
  template: `
```

```

        <h1>Hello World</h1>
        <p>we are in server component</p>
    ` })
    export class serverComponent {}

```

2. External Template File

Use this when your template code is more . We prefer to use this anyway. It separates the logic from the code which is responsible for the view.

so to create an external template file syntax as follows

Component name.component.html

in my example it will be server.component.html and adding this inside our component decorator as follows

// adding external template to our component

```

import { Component } from '@angular/core';
@Component({
  selector: 'app-server',
  templateUrl: './server.component.html'
})

```

export class serverComponent{

Here we are not using template property instead we are using templateUrl property. Using this templateUrl property we are adding a link to an external html file. This looks a lot cleaner and it separates the code which is responsible for view from business logic.

4. Add Styles

This is optional. You don't need styles for your component to work.

Styles work a lot like templates. There are two ways you can include them inside @component decorator

1. Inline Styles

You can add inline styles using styles property.

```

@Component({
  ...
  styles: [`
    h1 {
      margin-top: 50px,
      border: 2px solid black;
    }
    p{

```

```

        font-size:10px;
    }
    `]
    ...
})
export class serverComponent {}

```

Note: when ever you will use inline styles remember to use backtick (``)

Add your styles inside backtick

Just like templates, this doesn't really look good. It's much cleaner (and more maintainable) when we extract these styles to an external file. You could also inline the styles in the template markup.

Adding external style files - Syntax to create external style file

componentname.component.css

In my example server.component.css

```

@Component({
  ...
  styleUrls: [ './server.component.css' ],
  ...
})
export class serverComponent {}

```

NOTE: when we create a component manually then in app.module.ts file import this component at the top and also include component name inside the declarations array.

```

import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { AppComponent } from './app.component';
import { ServerComponent } from './server/server.component';

@NgModule({
  declarations: [
    AppComponent,
    ServerComponent
  ],
  imports: [
    BrowserModule,
    FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule {}

```

2. Creating Components using CLI

In vs code open terminal and type the command

ng g c component-name

ng generate component component-name

In my example **ng g c server**

now by default angular cli will create Component file structure looks like this

```
✓ server
# server.component.css
<> server.component.html
TS server.component.spec.ts
TS server.component.ts
```

If we look into index.html we can see the selector for the app component.

So finally we place only the selector of the app component in index.html file and all other components selector will be nested inside the app component.

```
hello-world > src > <> index.html > html > body
1  <!doctype html>
2  <html lang="en">
3  <head>
4    <meta charset="utf-8">
5    <title>HelloWorld</title>
6    <base href="/">
7    <meta name="viewport" content="width=device-width, initial-scale=1">
8    <link rel="icon" type="image/x-icon" href="favicon.ico">
9  </head>
10 <body>
11   <app-root></app-root>
12 </body>
13 </html>
14
```

Now clean up **app.component.html** and add few content to the **server.component.html** like this

```
<h1>Demonstration for components</h1>
<p>server works!</p>
```

Now if we want to display this content in the browser, then add this components selector which is **app-server** in **app.component.html**

```
p.component.html hello-world\... X app.component.html C:\...\ay-dream-app\... TS ap ↺
world > src > app > app.component.html > app-server

<!--this is app.component.html file
we are adding server component selector here to display its content -->
<app-server>/app-server>
<app-server></app-server>
```

here I placed the same component selector twice. so in the browser we can see the content of the server component twice.



As i mentioned earlier AppComponent act as root component and all other components are nested inside this AppComponent

Now let us create another component by using command

ng g c service

and we will get file structure like this

service.component.css

service.component.html

service.component.spec.ts

service.component.ts

And now we can place this component selector inside **app.component.html** or **server.component.html** its content will be displayed.

I will place its selector inside server.component.html and now **server.component.html** files content will be (fig a)

```
<h1>Demonstration for components</h1>
<p>server works!</p>
<!--this is server.component.html file
      am placing service components selector here-->
<hr>
<app-service></app-service>
```

(fig a)

and in app.component.html file am placing **server component** selector once (fig b)

```
<!--this is app.component.html file
      we are adding server component selector here to display its content -->
<app-server></app-server>
```

(fig b)

Now if we run our project we can see the content of both components in the browser.

Demonstration for components

server works!

service works!

This is content of service component

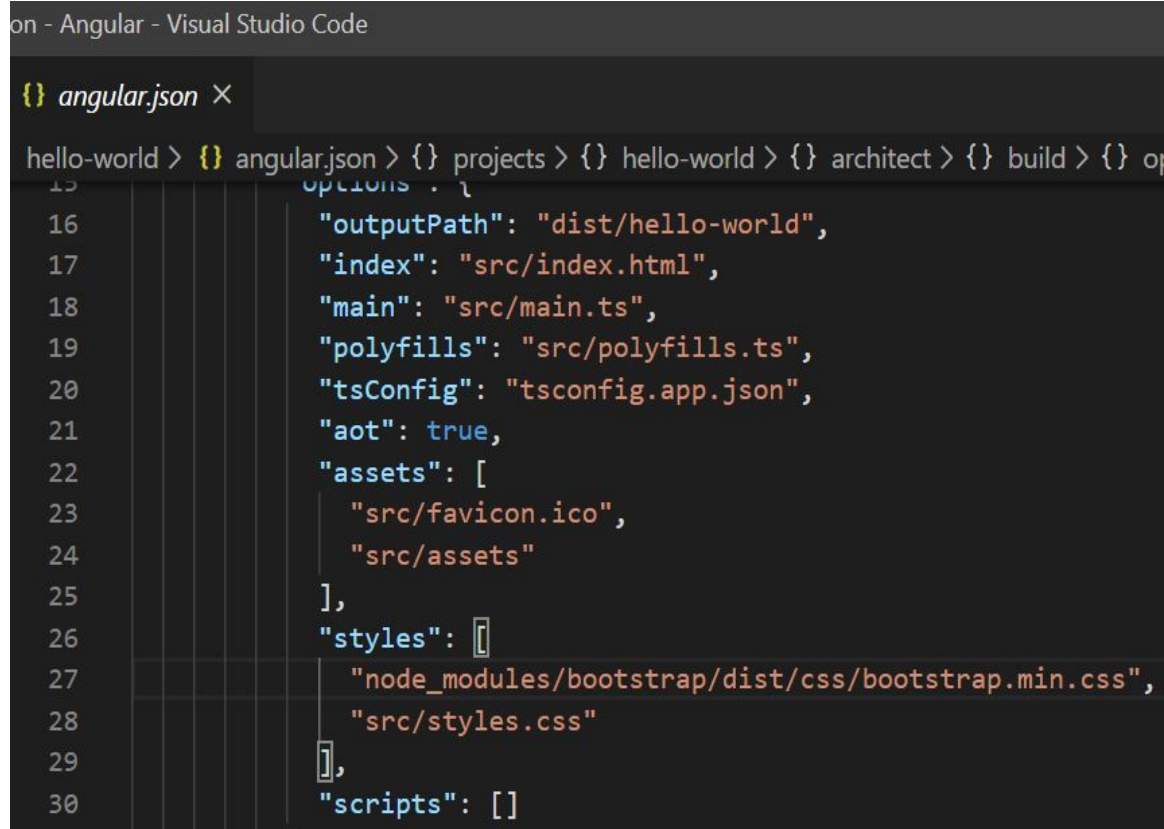
Adding bootstrap to our application

In vs code terminal run the command **npm install --save bootstrap**

And above command will install latest version of bootstrap in our application

Now we need to inform the cli that bootstrap should be included in our final bundle it creates for us to inform cli follow below steps:

1. Open angular.json file - There if you scroll down you will see the styles array. in this styles array we can add any global stylesheets we want to add our whole project. You can check where bootstrap installed will be stored. To check this,
 - a. Open node_modules folder - There lot of folders will be stored in alphabetical order. If we look into the bootstrap folder, Then in the dist folder -> We have a css folder, there our bootstrap will be saved. So now this path we need to include in the styles array like this.



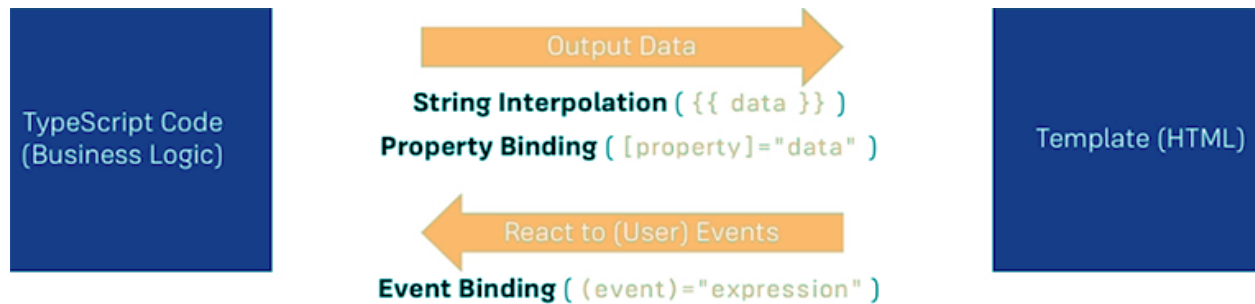
```
on - Angular - Visual Studio Code

{} angular.json X

hello-world > {} angular.json > {} projects > {} hello-world > {} architect > {} build > {} options : {
15
16     "outputPath": "dist/hello-world",
17     "index": "src/index.html",
18     "main": "src/main.ts",
19     "polyfills": "src/polyfills.ts",
20     "tsConfig": "tsconfig.app.json",
21     "aot": true,
22     "assets": [
23         "src/favicon.ico",
24         "src/assets"
25     ],
26     "styles": [
27         "node_modules/bootstrap/dist/css/bootstrap.min.css",
28         "src/styles.css"
29     ],
30     "scripts": []
```


Data Binding

Data binding is the communication between typescript code (which contains Business logic) and Template which is what the user can see.



Combination of Both: **Two-Way-Binding** ([(ngModel)]='data')

There are 4 types in data binding

1. String Interpolation
2. Property binding
3. Event Binding
4. Two-Way binding

One way binding: It is simple one way communication Where HTML template will be changed when we make changes in Typescript code.

String Interpolation, Property binding, Event Binding comes under one way binding.

1.String Interpolation

Whenever we want to display a component property the easiest way is to use string interpolation. we need to bind property names through interpolation. so that we can display the value associated with that property.

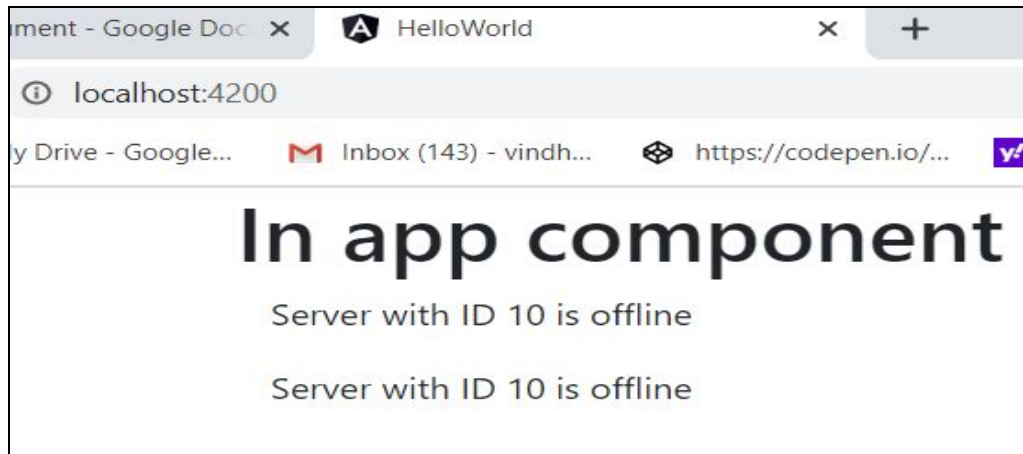
Now let us try to understand string interpolation with an example

In **server.component.ts** let us create new property as below

And to display the value of class property in the browser, in **server.component.html** We can use string interpolation. We are enclosing property within double curly braces. And whatever we enclose within curly braces that must return a string at the end. We can also call a method which returns a string at the end inside these curly braces. And we can't write multi-line expressions , if and for control structure here. We could use a ternary expression though.

```
<> server.component.html X TS server.component.ts
hello-world > src > app > server > <> server.component.html > div.container
1   <div class="container">
2       <div class="col-xs-12"><!--we used bootstrap class here-->
3       <hr>
4
5       <p>Server with ID {{ serverId }} is {{ serverStatus }} </p>
6
7   </div>
8   </div>
```

Here **serverId** is not a string it is a number. But at the end it will be converted to a string. And if we run we can see it in the browser:



I placed the server component selector twice in **app.component.html** so we are seeing content from the **server component** twice.

We can also call a method which returns a string using string interpolation - In **server.component.ts** file we can add a method like this

```
1  import { Component, OnInit } from '@angular/core';
2
3  @Component({
4    selector: 'app-server',
5    templateUrl: './server.component.html',
6    styleUrls: ['./server.component.css']
7  })
8  export class ServerComponent implements OnInit {
9
10   serverId:number = 10;
11   serverStatus:string = 'offline';
12   //getServerstatus is a method
13   getServerstatus(){
14     return this.serverStatus;
15   }
16
17
18   constructor() { }
19
20   ngOnInit(): void {
21   }
22
23 }
```

And in server.component.html we can call this method using string interpolation



```
<> server.component.html X TS server.component.ts <> app.component.html
hello-world > src > app > server > <> server.component.html > div.container
1   <div class="container">
2     <div class="col-xs-12">
```

In `server.component.ts` file,

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

@Component({
  selector: 'app-server',
  templateUrl: './server.component.html',
  styleUrls: ['./server.component.css']
})
export class ServerComponent implements OnInit {
  serverId:number = 10;
  serverStatus:string = 'offline';

  allowNewServer = false;

  constructor() {

    setTimeout(()=>{
      this.allowNewServer = true;
    }, 2000);

  }
  ngOnInit(): void {
  }
}
```

Here inside class we created another new property **allowNewServer**. And inside constructor we called a method,

setTimeout(()=>{} , 2000);

Here it takes two arguments one is a function and another is time delay, here am passing 2000milliseconds or 2seconds. My intention is to execute a function which is

passing as an argument inside **setTimeout()** after 2 seconds. And inside the arrow functions body am updating property value,

setTimeout(()=>{ this.allowNewServer = true } , 2000);

And in **server.component.html**,

```
server.component.html X TS server.component.ts TS server.component.spec.ts <> app.component.html
ello-world > src > app > server > <> server.component.html > div.container > div.col-xs-12
1   <div class="container">
2   <div class="col-xs-12"><!--we used bootstrap class here-->
3
4   <button class="btn btn-primary"
5   [disabled] ="!allowNewServer" >Add Server</button><!--example for property binding-->
6
7   <p>server with ID {{serverId}} is {{serverStatus}}</p><!--we are using string interpolation here-->
8   </div>
9   </div>
```

Here we added a button and we are placing **disabled** property of the HTML element inside **square brackets**. Square brackets in angular indicates **Property Binding**.

Here we are dynamically binding some property of the class with **disabled** property of the HTML element.

[disabled] = "!allowNewServer" here **allowNewServer** is a property in the class if this property value is **false** then we are disabling the button and after 2 seconds we are setting **allowNewServer** value to true and we are enabling the button dynamically.

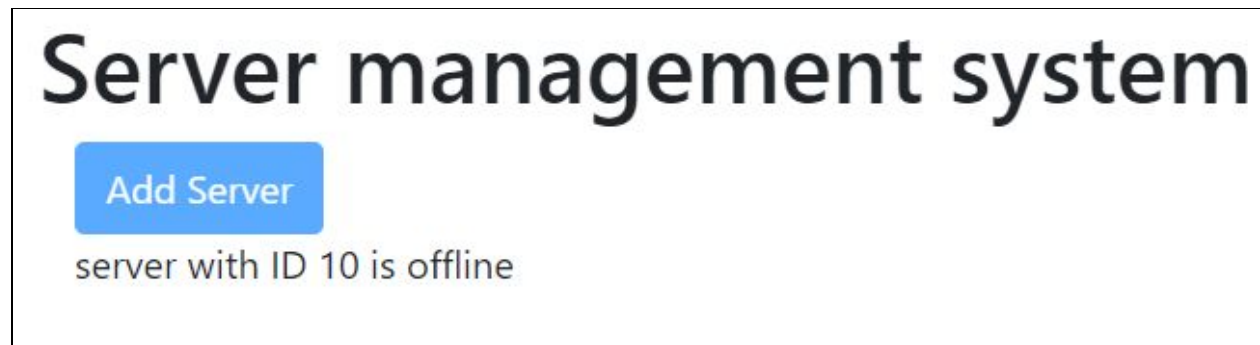
This is how property binding works.

Right now this is content of my app.component.html file

```
er.component.html # app.component.css TS server.component.ts <> app.comp
world > src > app > <> app.component.html > div.container
<div class="container">
<!--this is app.component.html file
we are adding server component selector here to display its content -->
<h1>Server management system</h1>
<app-server></app-server>

</div>
```


If i run the project we can see in the beginning for first 2 seconds like this,



And after 2 seconds our button will be enabled and result will be as below,



3. Event Binding

Event binding is used to handle the events raised from the DOM like button click, mouse move etc. When DOM event happens it calls a specified method in the component and code inside that component will be executed.

It is a one way communication mechanism between HTML template and Typescript class. Here data flows from HTML template to the typescript class.

Now consider the scenario that we need to display something when a button is clicked, then **Event binding** comes into picture. **Parentheses ()** indicates angular that it is event binding. Let us try to understand with the example, Consider **server.component.html** file,

```
hello-world > src > app > server > < server.component.html > div.container > div.col-xs-12 > p
1  <div class="container">
2  <div class="col-xs-12"><!--we used bootstrap class here-->
3
4  <button class="btn btn-primary"
5  [disabled] ="!allowNewServer" (click)="onCreateServer()">Add Server</button><!--example for property binding
6  and event binding-->
7  <p>{{serverCreationStatus}}</p>
8
9  <p>server with ID {{serverId}} is {{serverStatus}}</p><!--we are using string interpolation here-->
10 </div>
11 </div>
```

Here we added a click **event** and whenever this event occurs we are calling a method. So here we are transferring control from HTML template to Typescript class.

And in typescript class we created new property and a method as below

```
export class ServerComponent implements OnInit {
  serverId:number = 10;
  serverStatus:string = 'offline';
  allowNewServer = false;
  serverCreationStatus = 'no server was crated';

  constructor() {}

  setTimeout(()=>{
    this.allowNewServer = true;
  }, 2000);

  onCreateServer(){
    this.serverCreationStatus = 'server was created';
  }

  ngOnInit(): void {
  }
}
```

Here **serverCreationStatus** is our new property and **onCreateServer()** is our new method when the user clicks the button this method will be called and executed and the updated value of **serverCreationStatus** will be displayed in the browser.

Before the click event we can output like as below in browser,

Server management system

Add Server

no server was crated

server with ID 10 is offline

And when user clicks the button,

Server management system

Add Server

server was created

server with ID 10 is offline

So, this is how event binding works

Two-Way Binding

In two-way binding, automatic synchronization of data happens between the Model and the view. When data in the model changes the view reflects the change, and when data in the view changes the model is updated as well.

Now consider the scenario of when a user enters some value in a template at that time **input** event occurs and we need to receive it in the class. And using that received value we need to update some of our component property value and finally we want to display that in the browser at this point **Two-way binding** is very useful.


Using two way data binding we combine property and event binding.

`[()]` this syntax indicates that it is two way binding. Whenever we will use **Two-way** binding we need to use **ngModel** directive and we should enclose it like this

`[(ngModel)]`. In the next section we will learn directives in detail.

NOTE: whenever we use this **ngModel** directive we should **import FormsModule** (from **@angular/forms**) and then add **FormsModule** to our import array which will be present in the **app.module.ts** file.

```

world > src > app > TS app.module.ts >  AppModule
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core'; //while using two way binding we need to import like this
import { FormsModule } from '@angular/forms';
import { AppComponent } from './app.component';
import { ServerComponent } from './server/server.component';
import { ServiceComponent } from './service/service.component';

@NgModule({
  declarations: [
    AppComponent,
    ServerComponent,
    ServiceComponent
  ],
  imports: [
    BrowserModule,
    FormsModule // while using two-way binding we added Forms module to imports array
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

We added FormsModule now let us try to understand two-way binding with the example.

In **server.component.html**

```

<div class="container">
  <div class="col-xs-12">
```

In `server.component.ts`.

```
export class ServerComponent implements OnInit {
  serverId:number = 10;
  serverStatus:string = 'offline';
  allowNewServer = false;
  serverCreationStatus = 'no server was crated';
  serverName = 'TestServer'; //newly created property

  constructor() {

    setTimeout(()=>{
      this.allowNewServer = true;
    }, 2000);
  }
  onCreateServer(){
    this.serverCreationStatus = 'server was created';
  }
  ngOnInit(): void {
  }
}
```

Initially we assigned the value of **serverName property** and whenever a user event occurs. Its value will be updated and it sent back to the template this is how two way binding works.

And output looks like this initially,

Server management system

TestServer

Add Server

no server was crated

server with ID 10 is offline

And when we change content in the input field then updated component property will be,

Server management system

This is new server

Add Server

server was created

server with ID 10 is offline

Directives

Directives are the instructions in the DOM. Directives used to manipulate the DOM. Using directives we can change appearance, behaviour or a layout of a DOM element. We have Structural directives, Attribute directive, Component directive.

Structural directives

They start with * sign These directives are used to manipulate and change the structure of the DOM elements.

Example: ***ngIf** , ***ngSwitch**, ***ngFor**

1. *ngIf directive

It is a built-in directive. It is used to add or remove HTML Elements according to the expression. The expression must return a Boolean value it can be a method which returns true or false. If the expression is false then the element where we place this directive will be removed from DOM. otherwise the element is inserted.

Syntax

<p *ngIf = "condition"> Condition is true and ngIf is true </p>

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

Now let us learn about directives with the examples.

Consider **server.component.html** file

```
world > src > app > server > <> server.component.html > div.container > div.col-xs-12
<div class="container">
  <div class="col-xs-12">
    <input type="text" class="form-control"
      [(ngModel)] = "serverName">
    <p>{{serverName}}</p>

    <button class="btn btn-primary"
      [disabled] = "!allowNewServer" (click)="onCreateServer()">Add Server</button>
    <p *ngIf="serverCreated">Server was created, and server name is {{serverName}} </p><!--ngIf directive-->

    <p> server with ID {{serverId}} is {{serverStatus}}</p>
  </div>
</div>
```

Here we added *ngIf directive inside paragraph element and we assigned directive to an expression which returns true or false. If **serverCreated** returns true then only content of our paragraph will be displayed.

And if we look into **server.component.ts** file

```
export class ServerComponent implements OnInit {
  serverId:number = 10;
  serverStatus:string = 'offline';
  allowNewServer = false;
  serverName = 'TestServer';
  serverCreated = false;

  constructor() {

    setTimeout(()=>{
      this.allowNewServer = true;
    }, 2000);
  }
  onCreateServer(){
    this.serverCreated = true;
  }
  ngOnInit(): void {
  }
}
```

Here we assigned **serverCreated** property to false initially, and when the user clicks the button **Add Server** we are calling the method **onCreateServer()** there we are

updating the value of **serverCreated** to **true** so now the content of `<p></p>` tag will be displayed.

Server management system

TestServer

Add Server

server with ID 10 is offline

We can see this above output when ***ngIf** results false because initially we assigned **serverCreated** to false.

Server management system

TestServer

Add Server

Server was created, and server name is TestServer

server with ID 10 is offline

This will be the output when ***ngIf** results true, here It happens when user clicks

Add Server button.

And if we want to display some text if **serverCreated** was not true then we can add

Like this `<p *ngIf = "!serverCreated">No server was created</p>` this statement will get displayed When ***ngIf** results false.

We can also display this text as below,

1. Now place that **local reference** inside **ng-template** and place our paragraph inside this **ng-template** with local reference.

NOTE: in local reference we can place any text that we want.

In my example i am considering **#noServer** as local reference.

```
<ng-template #noServer>
```

```
<p> No server was created</p>
```

```
</ng-template>
```

This local reference act as a markup using that we mark certain part in our template that we want to show conditionally. To show conditionally we will add ***ngIf** and also we add **else** block. And syntax will be like this

```
<p *ngIf="serverCreated; else noServer ">server was created, and server name is  
{{ serverName }} </p>
```

```
<ng-template #noServer>
```

```
<p> No server was created</p>
```

```
</ng-template>
```


And finally it looks like this,

```
<div class="container">
  <div class="col-xs-12">
    <input type="text" class="form-control"
      [(ngModel)] = "serverName">
    <p>{{serverName}}</p>

    <button class="btn btn-primary"
      [disabled] = "!allowNewServer" (click)="onCreateServer()">Add Server</button>
    <p *ngIf="serverCreated ; else noServer">Server was created, and server name is {{serverName}} </p>
    <!--ngIf directive-->
      <ng-template #noServer>
        <p>No server was created!</p>
      </ng-template>
    <p> server with ID {{serverId}} is {{serverStatus}}</p>
  </div>
</div>
```

When ***ngIf** results to false then control comes to the else part and there using local reference control moves towards ng-template and that content will be displayed.

And on the screen we can see this,

Server management system

TestServer

Add Server

No server was created!

server with ID 10 is offline

This is an example showing using local reference how we can display else block of our code.

*ngFor:

It is a built in structural directive it helps us to render a list of elements in the browser.

It is used to repeat a portion of the HTML template once per each item from an iterable list.

Let us understand this with an example

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

@Component({
  selector: 'app-server',
  templateUrl: './server.component.html',
  styleUrls: ['./server.component.css']
})
export class ServerComponent implements OnInit {

  public fruits = ["Apple", "Mango", "orange", "grapes"];

  constructor() {

  }

  ngOnInit(): void {
  }
}
```

If we add a property in class and we set its value as an array now inorder to display individual members of the array we will use *ngFor

And code in our **server.component.html** will be

```
1 | <h1>Demonstration for ngFor </h1>
2 | <div *ngFor = "let fruit of fruits">
3 | <h2>{{fruit}}</h2>
4 |
5 | </div>
```

And if we see it in the browser output will be

Demonstration for ngFor

Apple
Mango
orange
grapes

AttributeDirectives

They used to change the look and behaviour of DOM elements.They are unlike structural directives they will not add or remove elements.

Ex: ngClass Directive, ngStyle directive etc.

ngStyle Directive

This facilitates us to modify the style of an HTML element using the expression.

We can use this directive to dynamically change the style of our HTML element.

ngClass Directive

It is used to add or remove css class to our HTML element.

