

## Angular advance course by maximillan

- Angular is a framework which allows you to create reactive, single page applications. This sounds great, but what exactly does it mean?

A Single Page Application (SPA) is an application like the one shown here

the project we're going to build throughout the course. You can navigate around and in the URL, you can see that we seem to visit different pages, but in the end, our page never changes. It's only one HTML file and a bunch of JavaScript code we got from the server and everything which you see here, every change, is rendered in the browser.

Now, why is that awesome? It gives the user a very reactive user experience (UX). JavaScript is much faster than having to reach out to a server for every page change and for every new piece of data you want to display. Therefore, this approach allows you to create web applications which look and feel almost like mobile applications. very fast! Everything happens instantly. If you do need some data from a server, you simply load it in the background so that the user never leaves this experience of having a reactive web application to use. So every click I do here simply changes this one single page we're using.

- this one HTML page. So, how is this done?

Well, JavaScript changes the DOM, changes whatever is displayed here (in the browser), by changing the HTML code during runtime (so to say). That is why you never see the refresh icon on the top-left spin. because we're only changing the currently-loaded page.

You can even see that if you inspect the source code of a page like this. That is the HTML file and as you can see, it doesn't seem to contain the content you are seeing on this page. We only have one single HTML element which doesn't seem to be a built-in one (a native one), but that's Angular doing its job. Throughout the course, you're going to learn exactly what happens here and how Angular does this job.

## Confusing Versioning in Angular

Now one thing that can be confusing when you're getting started with angular is the amount of angular versions you find out there. Now in this lecture we'll have a look at the different versions how they are connected. If that means that angular changes all the time. Spoiler It does not.

So we had angular one. It was released a couple of years ago( October 20, 2010; 10 years ago) and angular one was a big new thing back then. Now this course is not about angular one though because while said was the big new thing it also had some issues that could lead to worse performance and bigger applications ends on this question of angular angular one is nowadays referred to as angular J.S. written like this. So if you see people right angle or J.S. in some Internet forum on Reddit or anything like that they are most likely referring to angular one which is totally different to the Angular word that you're learning about because there was a complete rewrite of the angular framework between angular one and

angular two angular 2 was released in two thousand and sixteen and it was rewritten from the ground up. It works totally different than angular one to fix all the issues angular one had now since that initial release of angular two we had a couple of other versions of angular with angular four angular 3 was skipped for internal reasons . we had angular five six seven and now we have angular nine.

Why do we have all these angular weird numbers does it mean that angular was reinvented eight times since angular one ? No this is not the case instead since the release of angular to the angle or a team simply address to versioning scheme where a new word of angular is released every six months now that new version however is not a complete rewrite it does not change everything. Indeed most updates change almost nothing only some behind the scenes stuff or add some new features without breaking existing features. Indeed angular 9 is pretty much the same as angular 2 initially created .This course for angular and I updated it multiple times and it is totally up to date day but when we have a look at the core syntax at

the philosophy ends on nothing changed since angular 2 .Indeed if you learned angular in syntax. Some minor things changed and I updated the course over to time to reflect that but overall it's the same framework. We have small incremental backwards compatible changes between these versions. It's simply just a commitment which the angular team made to release such a major new version every six months. It does not break or change everything every time it's released .

## From here we Start making projects



### ng new

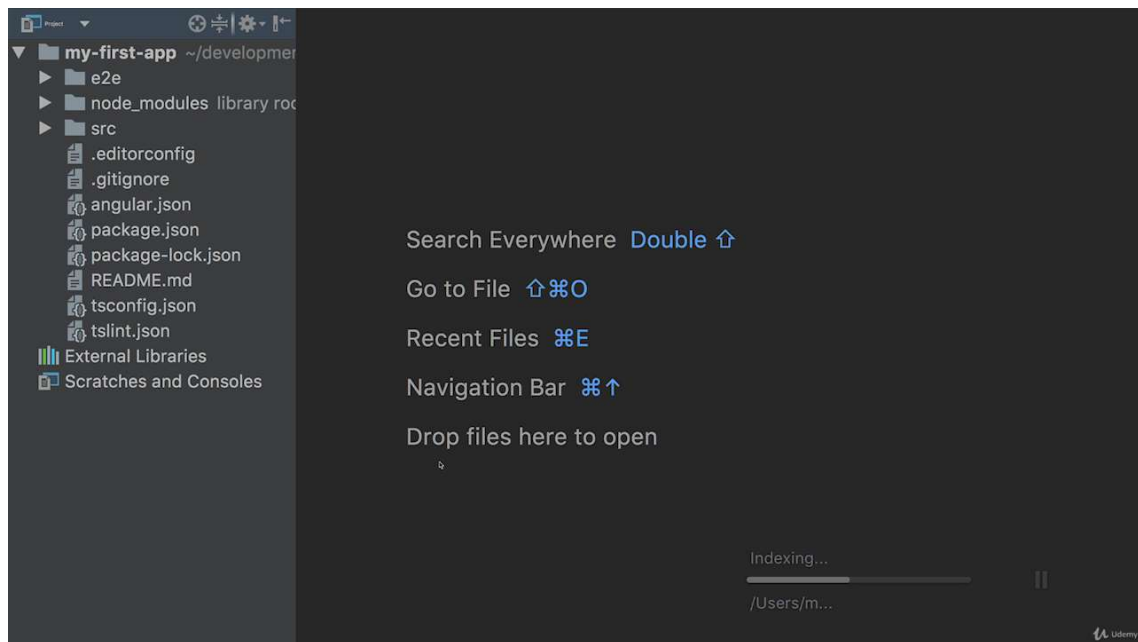
The Angular CLI makes it easy to create an application that already works, right out of the box. It already follows our best practices!

### ng generate

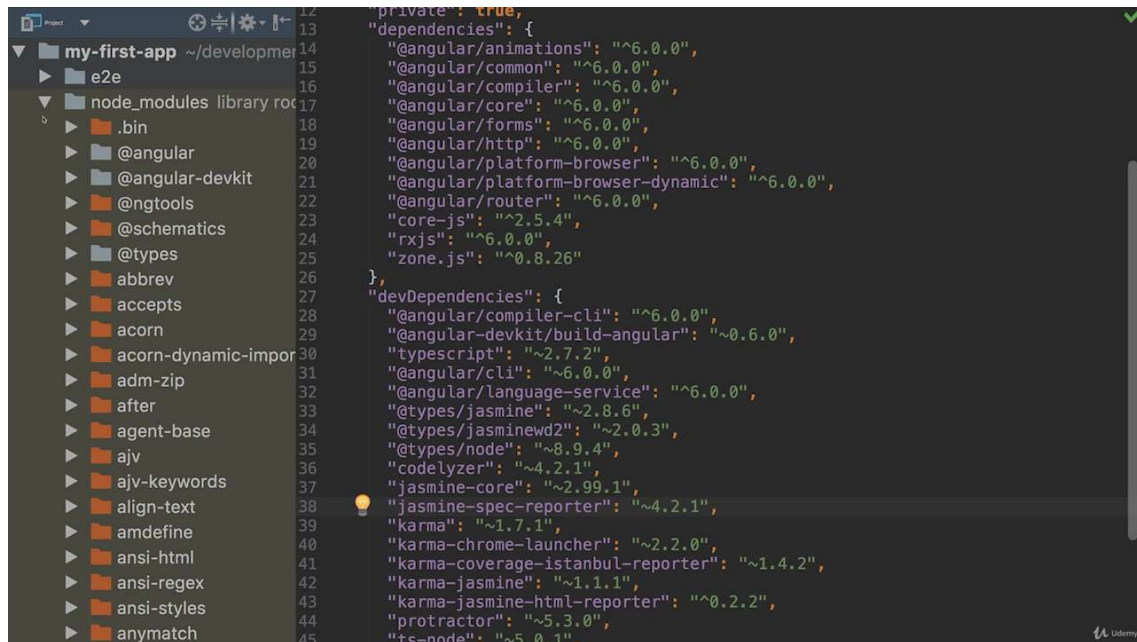
Generate components, routes, services and pipes with a simple command. The CLI will also create simple test shells for all of these.

We need node because we use npm to do our work .

Now this is your npm created project structure  
dont be afraid most of these files are  
configurations.

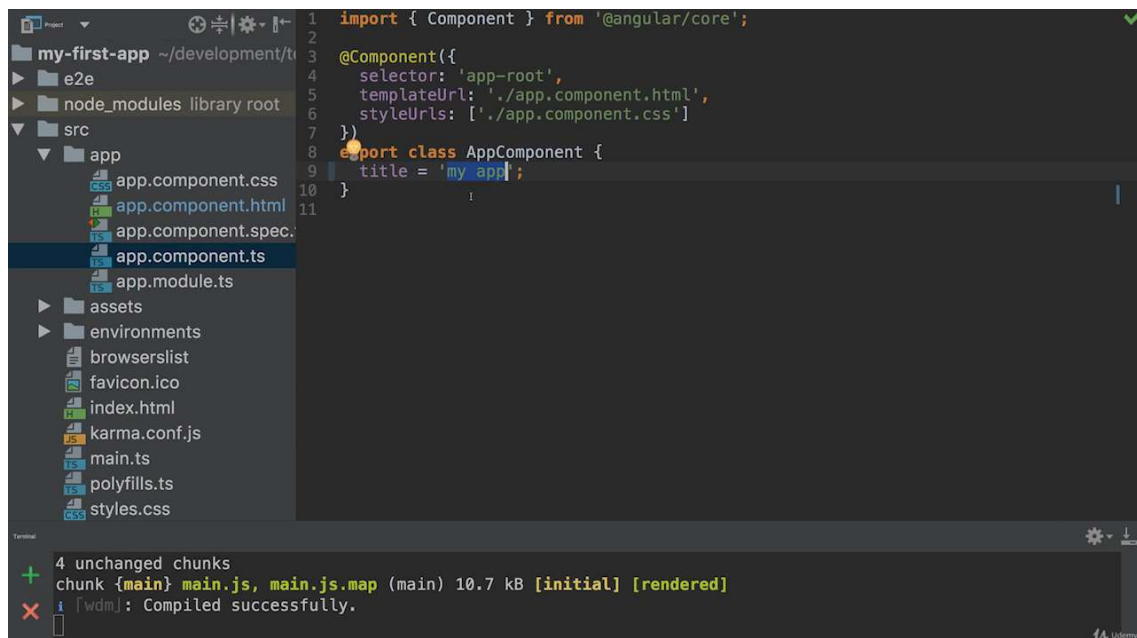


In package .json you can view all dependencies we use they are stored in node\_modules folder.



```
12  "private": true,
13  "dependencies": {
14    "@angular/animations": "^6.0.0",
15    "@angular/common": "^6.0.0",
16    "@angular/compiler": "^6.0.0",
17    "@angular/core": "^6.0.0",
18    "@angular/forms": "^6.0.0",
19    "@angular/http": "^6.0.0",
20    "@angular/platform-browser": "^6.0.0",
21    "@angular/platform-browser-dynamic": "^6.0.0",
22    "@angular/router": "^6.0.0",
23    "core-js": "^2.5.4",
24    "rxjs": "^6.0.0",
25    "zone.js": "^0.8.26"
26  },
27  "devDependencies": {
28    "@angular/compiler-cli": "^6.0.0",
29    "@angular-devkit/build-angular": "~0.6.0",
30    "typescript": "~2.7.2",
31    "@angular/cli": "^6.0.0",
32    "@angular/language-service": "^6.0.0",
33    "@types/jasmine": "~2.8.6",
34    "@types/jasminewd2": "~2.0.3",
35    "@types/node": "~8.9.4",
36    "codelyzer": "~4.2.1",
37    "jasmine-core": "~2.99.1",
38    "jasmine-spec-reporter": "~4.2.1",
39    "karma": "~1.7.1",
40    "karma-chrome-launcher": "~2.2.0",
41    "karma-coverage-istanbul-reporter": "~1.4.2",
42    "karma-jasmine": "~1.1.1",
43    "karma-jasmine-html-reporter": "^0.2.2",
44    "protractor": "~5.3.0",
45    "ts-node": "~5.0.1",
```

So src is the folder that we need to do our coding We will dive deep in to this in later sessions.



```
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 = 'my app';
10 }
11
```

Terminal

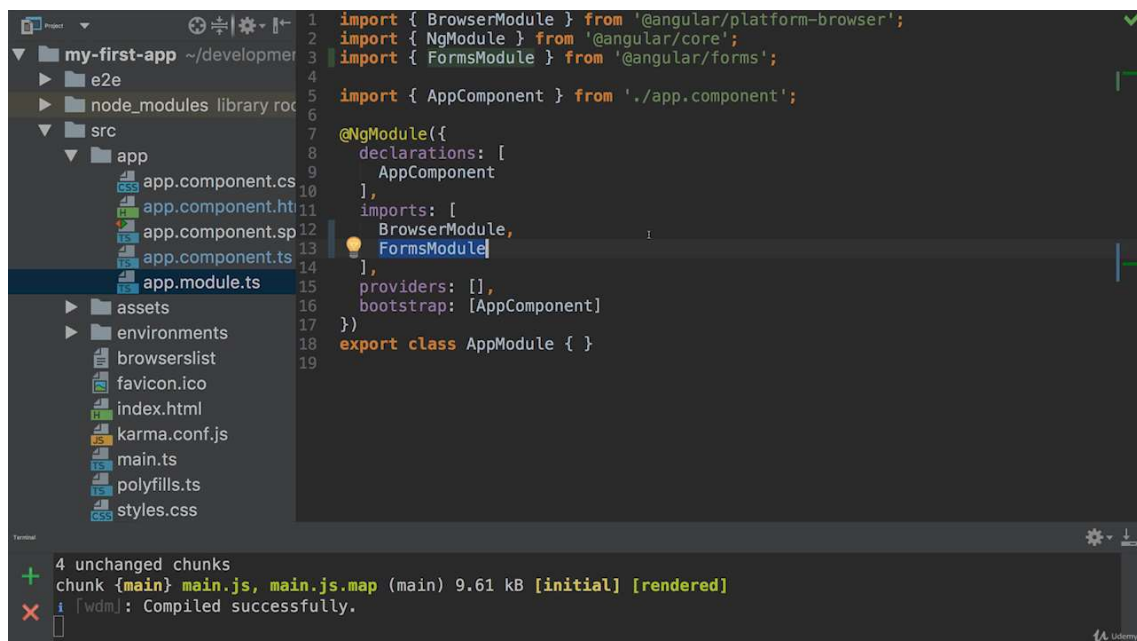
```
4 unchanged chunks
chunk {main} main.js, main.js.map (main) 10.7 kB [initial] [rendered]
i [wdm]: Compiled successfully.
```



So we will write a small programme to get a name and view it .

**To make some thing listen to input changes and view it we need to get [(ngModel)] and use it but before using it we must specify the import and tell angular that we are going to use it.**

We specify these imports in app.module.ts  
“FormsModule” is the sub package we using here in ngmodel



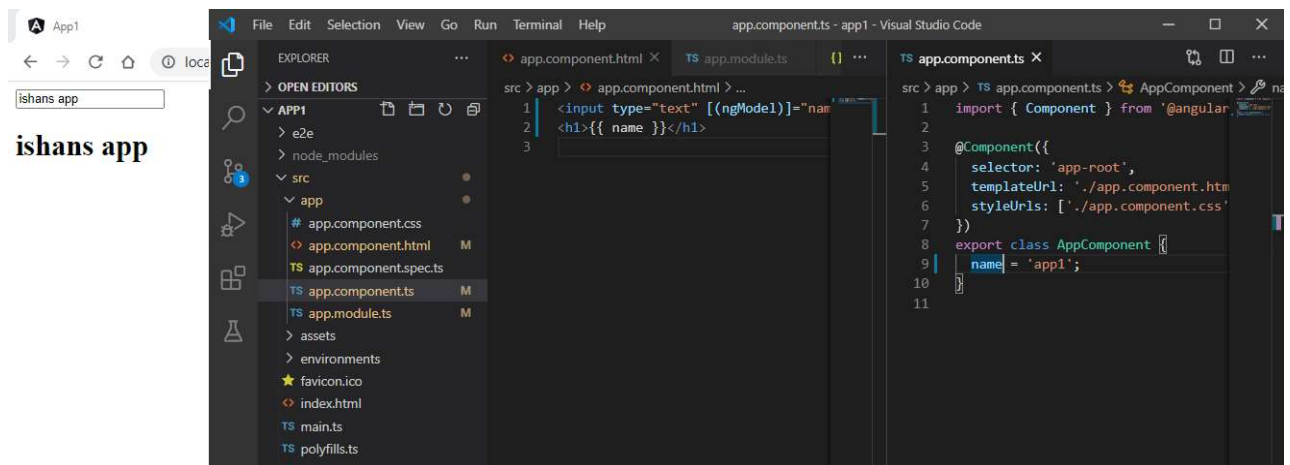
The screenshot shows a code editor with a file explorer on the left and a terminal at the bottom. The file explorer shows a project named 'my-first-app' with a file 'app.module.ts' selected. The code editor displays the following TypeScript code for 'app.module.ts':

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

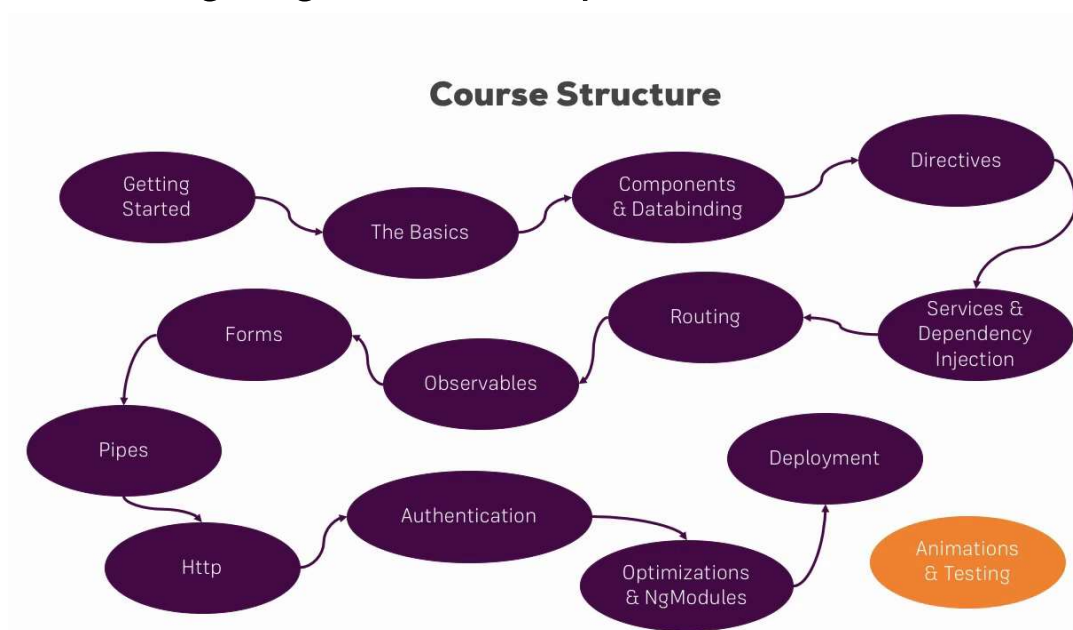
The terminal window at the bottom shows the following output:

```
4 unchanged chunks
chunk {main} main.js, main.js.map (main) 9.61 kB [initial] [rendered]
i [wdm]: Compiled successfully.
```

And you need to make a variable to store at the listener in app.component and use it in the app.component.html like below so then when u type something in input it dynamically changes the h1 tag awesome.

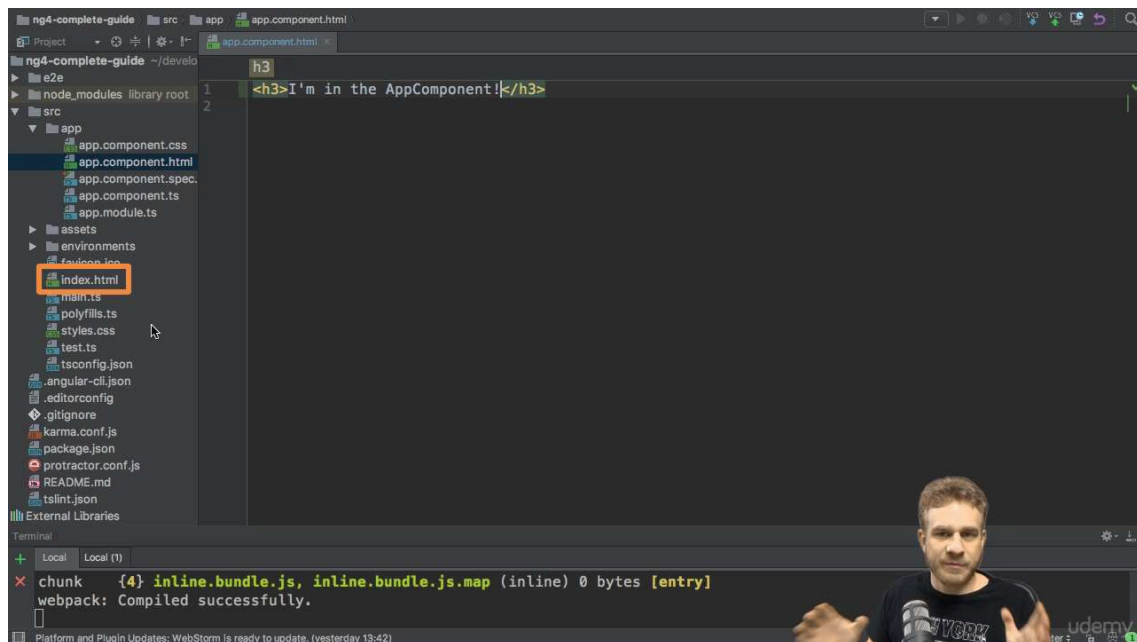


Now we going to dive deeper this is our course

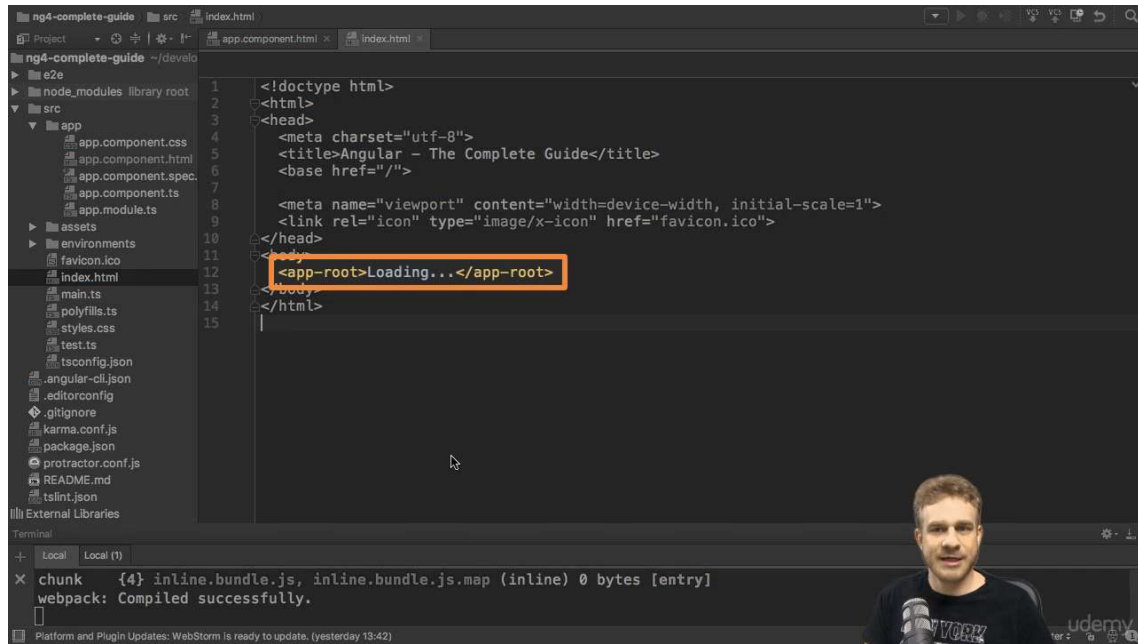


## We can dive deeper now

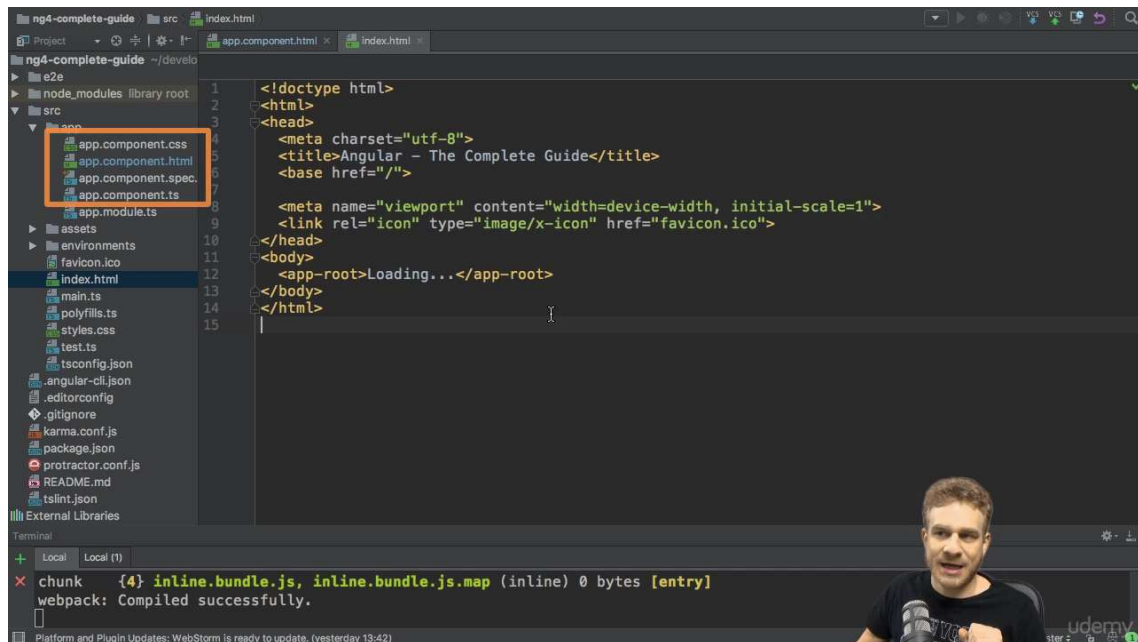
First which file do u think that server given to us is it app.component.html ? nop it is the index.html file that served by the server and most importantly index.html is the “Single page” that we all talking about when we discuss single page apps.



And another interesting thing about index.html  
It has some new html elements like a `<app-root>`  
And the “Loading...” inside it not shown in final  
view what magic is this ? what is `<app-root>` ?  
What happen to loading .... ?

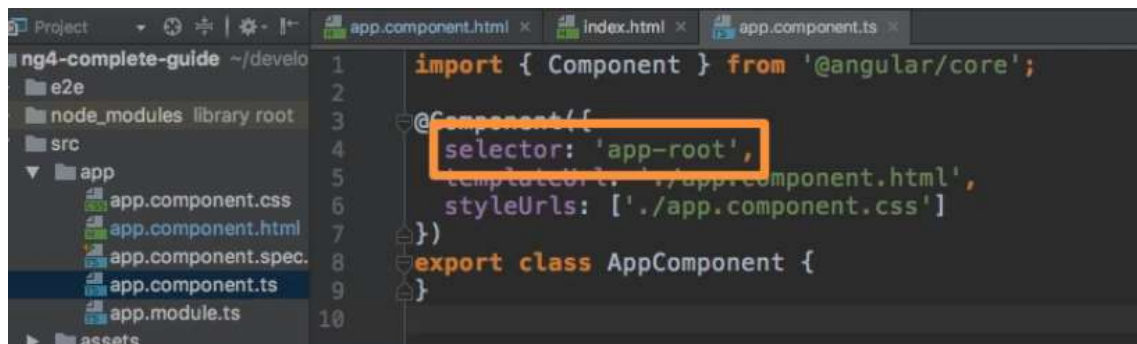


Actually this `app-root` is an angular component. We can see it under the `app` in `src`. We will dive deep into it in further sessions and create our own components but for now this component is made by angular CLI.



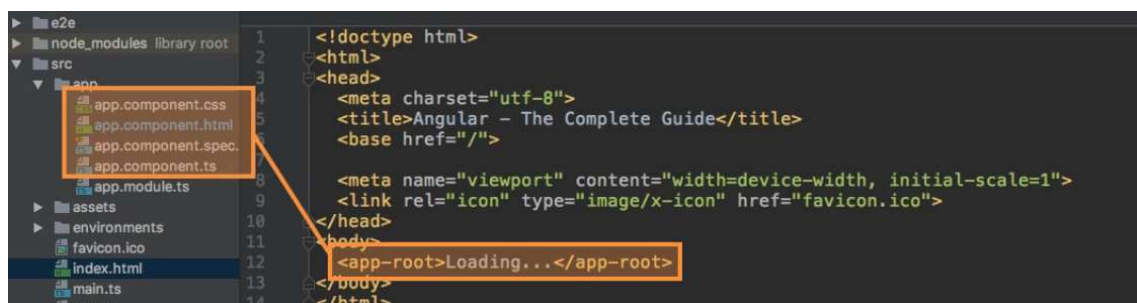
All the files in app folder with the name of app.component is related to this <app-root> component .

In app.component.ts the secret reveals it has given the name to these app folder components as **Selector:'app-root'**



```
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 }
10
```

So angular know to replace <app-root> with these app folder contents when we use it in index.html  
We can make and use our custom components in future .



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

But how angular start and run our components logic in body tag of index html ?

When angular cli start after we start using ng serve  
It bundled the index.html with js bundles so in final index.html we can view these js scripts . look below



```
<html>
<head>
  <meta charset="utf-8">
  <title>Angular - The Complete Guide</title>
  <base href="/">

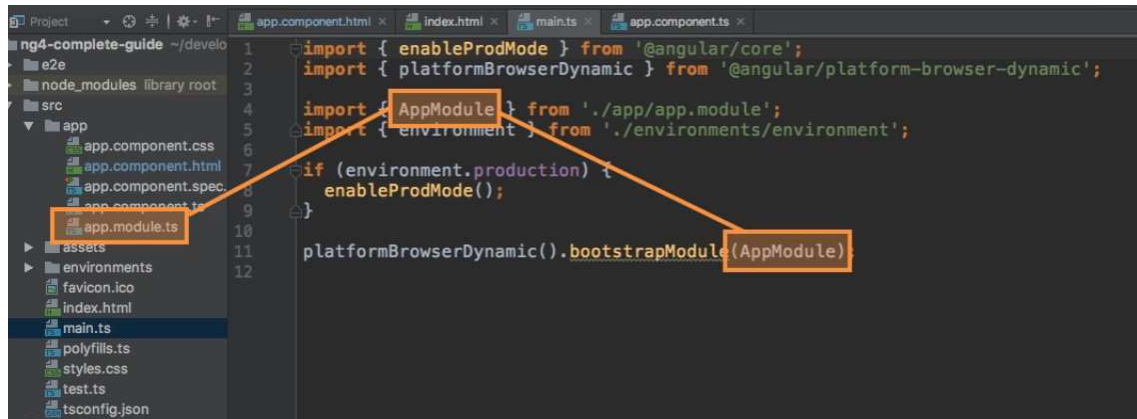
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root>Loading...</app-root>
  <script type="text/javascript" src="inline.bundle.js"></script><script type="text/javascript" src="polyfills.bundle.js"></script><script
type="text/javascript" src="styles.bundle.js"></script><script type="text/javascript" src="vendor.bundle.js"></script><script type="text/javascript"
src="main.bundle.js"></script></body>
</html>
```

SO these are the first js codes that get executed  
This flow goes like this

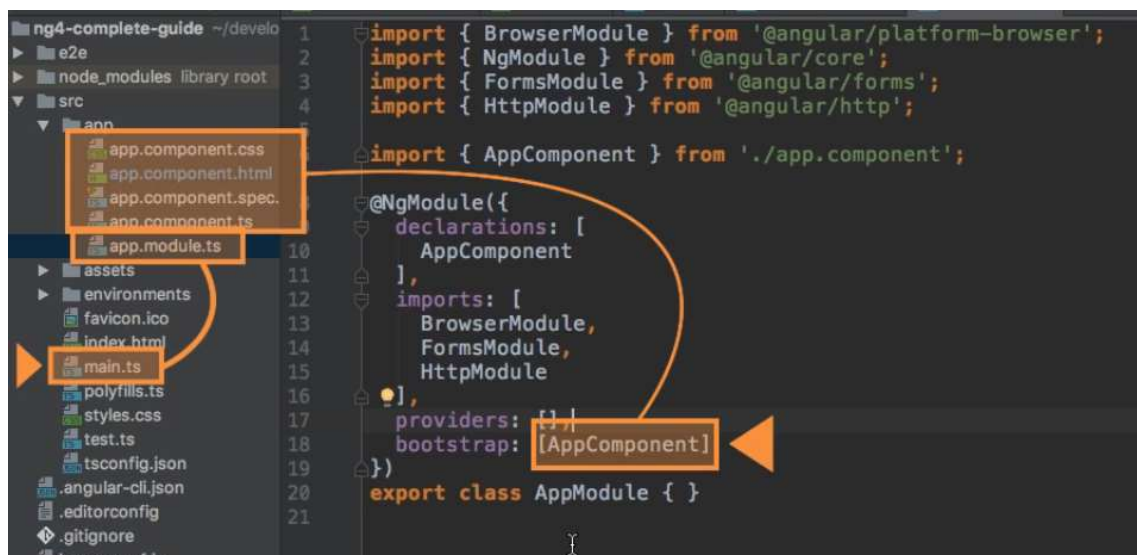
First main.ts get executed

Then it imported the AppModule and started our app by calling bootstrapModule(AppModule) method.





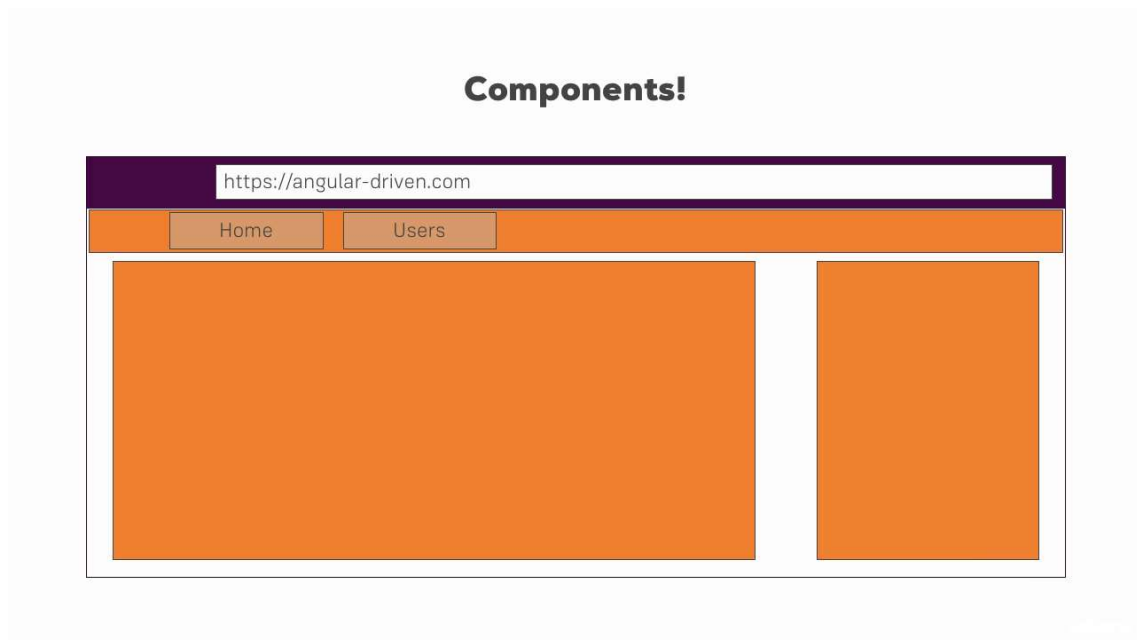
In app.module.ts we list all the components as a array. By giving this “ bootstrap:[AppComponent]” we tell to angular that these are the components that we going to use in our index.html .See below diagram



Because we told the angular to do like that in the bootstrap array of app.module.ts it reads components like app.component.ts and knows to handle <app-root> like selectors when we use it in index.html.

## **Components are important so deep dive into it**

Components are reusable and flexible we build our web page using components in angular.



## **Creating a component**

Make a folder having the component name its a good practise and clean code.

We make server folder(component that tell about server information) and make .ts files inside it.

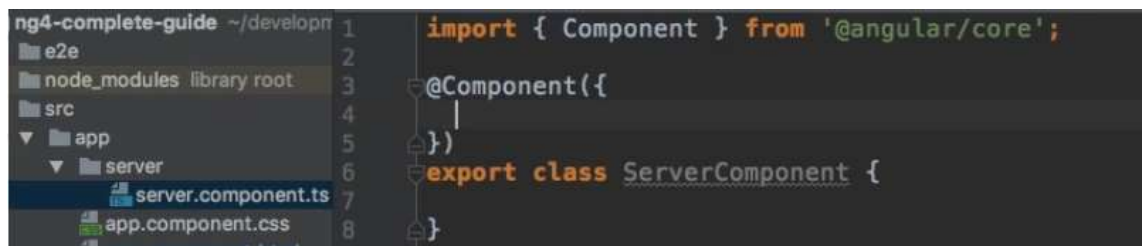


- Component is just a typescript class
- Use “export class” syntax to make this typescript class able to be export.



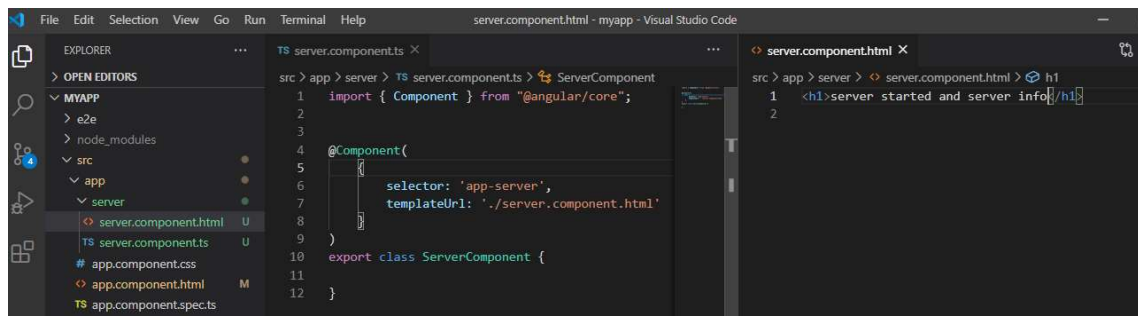
```
1
2 export class ServerComponent {
3
4
5 }
```

- Nothing special if make a normal ts class so to make it identified by angular we use java annotations like syntax in ts “decorators” but for to use it we need to import it from angular core.



```
1 import { Component } from '@angular/core';
2
3 @Component({
4 })
5
6 export class ServerComponent {
7
8 }
```

- So now we have told angular that this is officially a component and further we need to specify an element name(make it unique, don't use “h1” like names) and a html template class for this component.



So we have successfully made a component we can now use it .

# App Module

Consider your angular Application as a building.

A building can have  $N$  number of apartments in it. An apartment is considered as a module.

An Apartment can then have  $N$  number of rooms which correspond to the building blocks of an Angular application named components.

Now each apartment (Module) will have rooms (Components), lifts (Services) to enable larger movement in and out the apartments, wires (Pipes) to transform around and make it useful in the apartments.

You will also have places like swimming pool, tennis court which are being shared by all building residents. So these can be considered as components inside SharedModule.

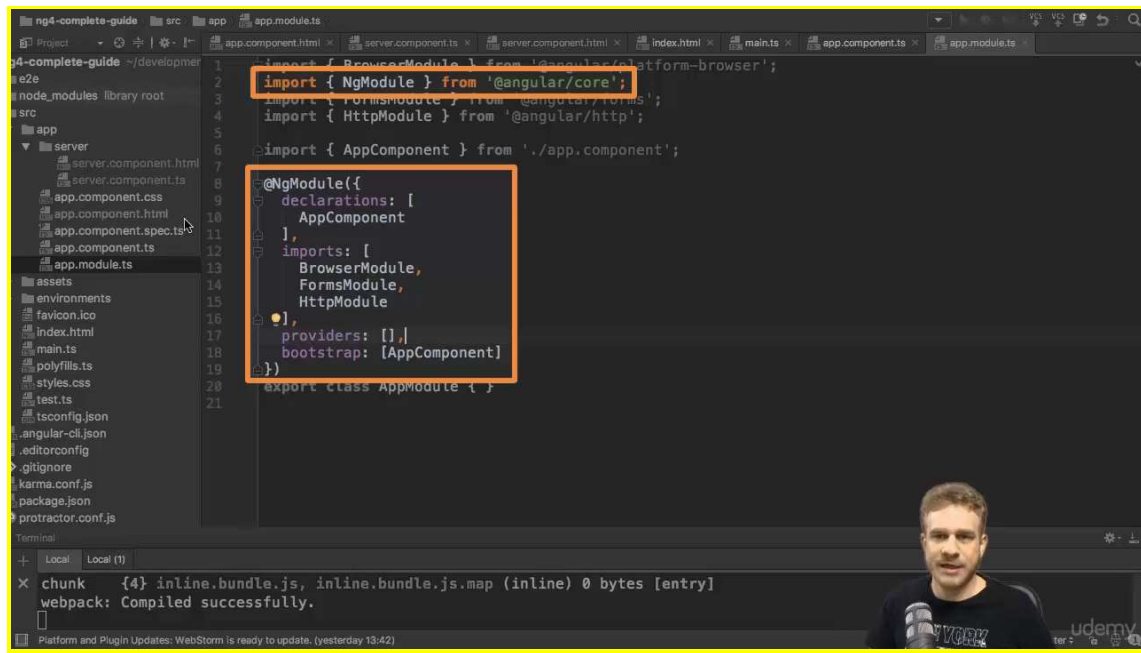
Basically, the difference is as follows,

Module	Component
A module instead is a collection of components, services, directives, pipes and so on.	A component in Angular is a building block of the Application with an associated template.
Denoted by @NgModule	Denoted by @Component
The Angular apps will contain many modules, each dedicated to the single purpose.	Each component can use other components, which are declared in the same module. To use components declared in other modules, they need to be exported from this module and the module needs to be imported.

An NgModule describes how the application parts fit together. Every application has at least one Angular module, the *root* module, which must be present for bootstrapping the application on launch. By convention and by default, this NgModule is named AppModule

We have single app module but for huge projects we can use more than one modules. So what is the app.module.ts its also a class but given module functionalities by using a decorator

## @NgModule



So as in @component decorator we defined about template class and a name for the element .

We need to define some values here in

## @NgModule

We will check one by one.

The [@NgModule](#) has several metadata properties.

## imports

We need to list all the external modules required including other Angular modules, that is used by this Angular Module

## Declarations

The Declarations array contains the list of components, directives, & pipes that belong to this Angular Module. We have only one component in our application AppComponent.

## Providers

The [Providers](#) array, is where we register the services we create. The [Angular Dependency injection](#) framework injects these [services](#) in components, directives. pipes and other services.

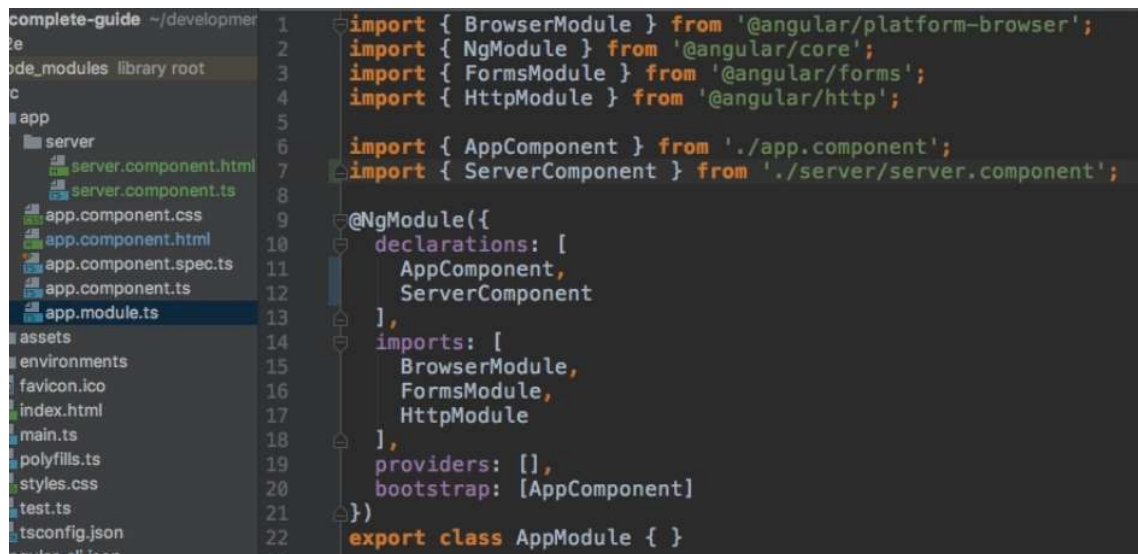
## Bootstrap

The component that angular should load, when this Angular Module loads. The component must be part of this module. We want AppComponent load when AppModule loads, hence we list it here.

The Angular reads the bootstrap metadata and loads the AppComponent

most importantly just making a component is not enough we need to register it inside @NgModule in app.module.ts.

First we need to add components to the declarations array.



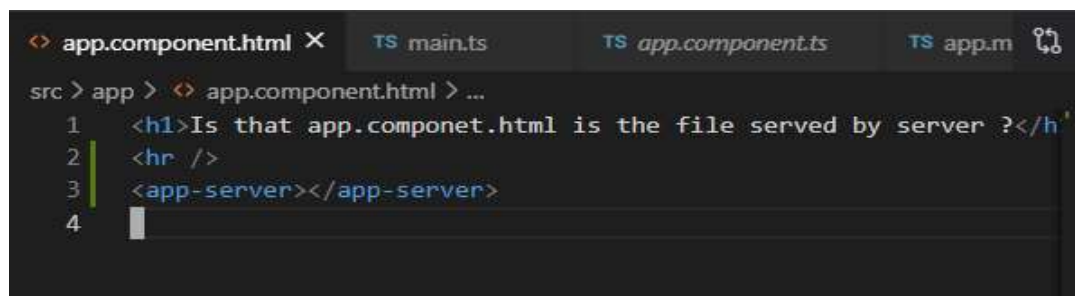
```
1 import { BrowserModule } from '@angular/platform-browser';
2 import { NgModule } from '@angular/core';
3 import { FormsModule } from '@angular/forms';
4 import { HttpClientModule } from '@angular/http';
5
6 import { AppComponent } from './app.component';
7 import { ServerComponent } from './server/server.component';
8
9 @NgModule({
10   declarations: [
11     AppComponent,
12     ServerComponent
13   ],
14   imports: [
15     BrowserModule,
16     FormsModule,
17     HttpClientModule
18   ],
19   providers: [],
20   bootstrap: [AppComponent]
21 })
22 export class AppModule { }
```

So in the imports array we specify other modules like FormsModule or HttpClientModule .

We use providers array for services.

We give our own app component to the bootstrap array because its also used as a element in index.html.

To use our created component just add it to the app.component.html as



```
1 <h1>Is that app.componet.html is the file served by server ?</h1>
2 <hr />
3 <app-server></app-server>
4
```

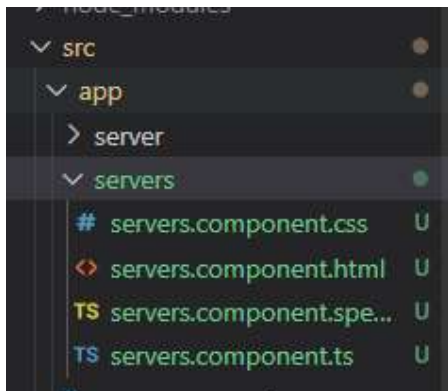
## Create component using Angular CLI

1. Use `ng generate component --name--`

Or

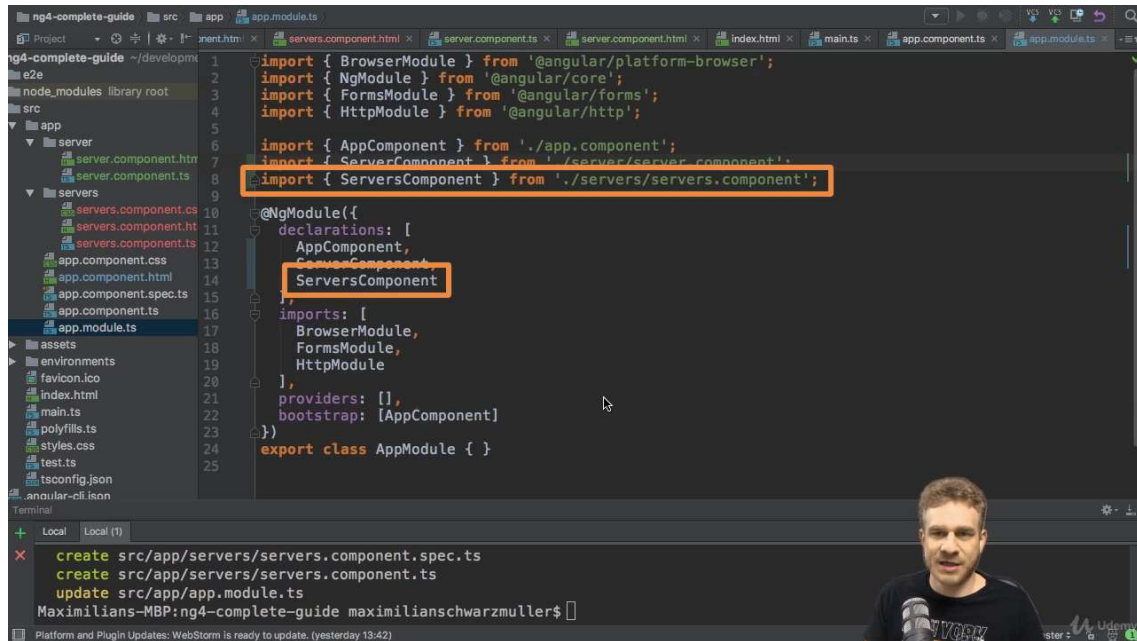
`ng g c --name--`

2. So give `ng g c servers` command.



3. We don't need test file (spec.ts) so delete it.

4. Angular has automatically add it to app.module.ts but please check it.



5. use it in the app.component.html (we gave 2 server elements to servers.html) view the change.

Sorry index.html is the file that serve by server !

Is that app.componet.html is the file served by server ?

servers works!

server started and server info

server started and server info



See the nested nature of components through dev tools.



## Styling a component

In templates u can simply write your html not in a separate file but remember if its more than 3 lines don't use this approach.

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

@Component({
  selector: 'app-servers',
  template: `
    <app-server></app-server>
    <app-server></app-server>`,
  styleUrls: ['./servers.component.css']
})
export class ServersComponent implements OnInit {

  constructor() { }

  ngOnInit() {
  }
}
```

You can use

`styleUrls: ['./servers.component.css']`

To style the component and because this is a array u can use multiple style sheets .

Or just use

```

src > app > TS app.component.ts > AppComponent
1  import { Component } from '@angular/core';
2
3  @Component({
4      selector: 'app-root',
5      templateUrl: './app.component.html',
6      // styleUrls: ['./app.component.css']
7      styles: [`
8          h1 {
9              color:red;
10         }
11     `]
12 })
13
14 export class AppComponent {
15     title = 'myapp';
16 }
17

```

## For selector name

We can use .class or attribute but no id(angular dont support it).

```

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

@Component({
    // selector: '[app-servers]',
    // selector: '.app-servers',
    selector: 'app-server',
    template: `
        <app-server></app-server>
        <app-server></app-server>`,
    styleUrls: ['./servers.component.css']
})
export class ServersComponent implements OnInit {

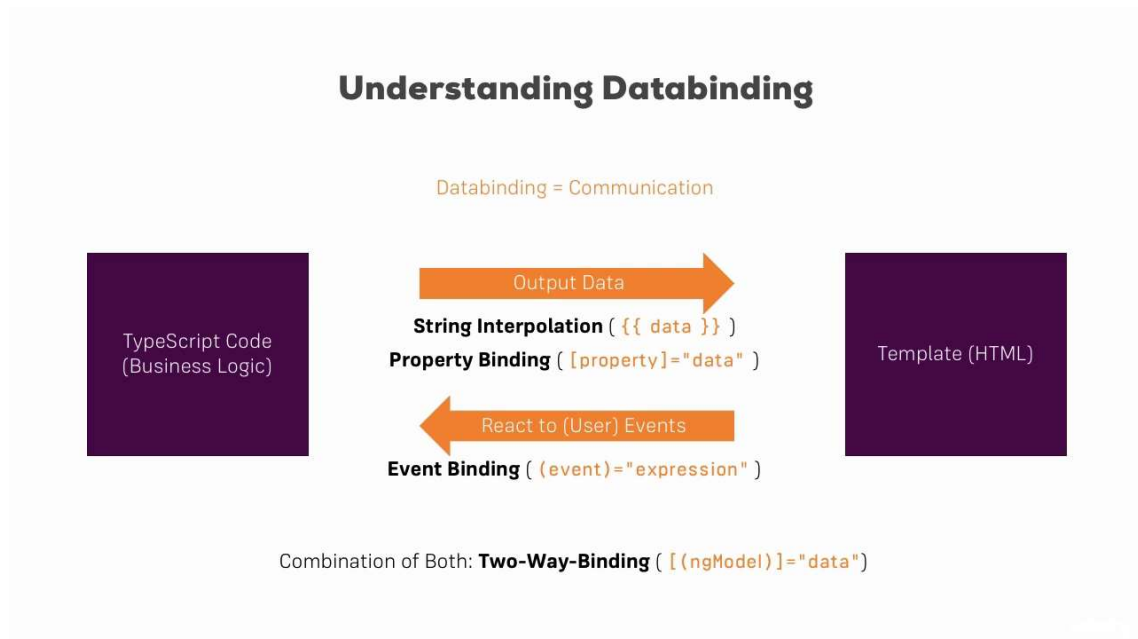
    constructor() { }

    ngOnInit() {
    }
}

```

Should be ' <app-server></app-server>'

# DataBinding



## 1.String Interpolation

Interpolation- "a remark interjected in a conversation."

"as the evening progressed their interpolations became more ridiculous"

What is a string interpolation in angular ?

Any Expression which can be resolved to a string at the end (number also ok because it can be converted into string at the end).

And u cant write multiline.

Ternary ()?true:false is also ok.

This is how to do it

```
export class ServerComponent {  
  serverId: number = Math.random();  
  serverStatus: string = "offline";  
  
  getServerStatus() {  
    return this.serverStatus + " of this server checked by a method  
and -";  
  }  
}
```

```
<h1>  
  {{ getServerStatus() }} Server {{ serverStatus }} with Id {{ serverId  
}}  
</h1>
```

## 2.Property Binding

You can bind a property to a element by this

```
export class ServersComponent implements OnInit {
  allowNewServer = false;
  constructor() {
    setInterval(() => {
      this.allowNewServer = !this.allowNewServer;
    }, 2000);
  }

  ngOnInit(): void {
  }
}
```

```
<p>servers works!</p>

<button class="btn btn-primary" [disabled]="allowNewServer">Add
server</button>
<br />
<br />
<div class="alert alert-success">Success</div>
<app-server></app-server>
<app-server></app-server>
```

So in here angular dynamically changes disable or enable 2 seconds .

# When to use string interpolation and when to use property binding ?

```
export class ServersComponent implements OnInit {
  allowNewServer = false;
  constructor() {
  }
  ngOnInit(): void {
  }
}
```

```
<button class="btn btn-primary" [disabled]="!allowNewServer">Add
server</button>
```

Technically there are 3 ways to bind properties.

1. String Interpolation - `{{ expression }}` - render the bound value from.
2. component's template and it converts this expression into a string.
3. Property Binding - `[target]="expression"` - does the same thing by rendering value from component to template, but if you want to bind the expression that is other than string (for example - boolean), then property Binding is the best option.
4. `bind-target="expression"` - This is not a regular way of using.

Its always your decision to use whichever option that fits your use case.

When rendering **data values as strings**, there is no technical reason to prefer one form to the other.

If we want **data values as boolean or other than string** then we should go for property binding

Property binding(`[]`) and interpolation(`{{ }}`) , both are similar and both supports one way data binding(supply data from component to html template).There is a minor difference between them. We must use property binding for non-string data.

## Event Binding

We use event binding if we want to talk from html template to component.

on+EventName is good for these trigger methods because easy to identify .

And tip for remember is onclick like method become click by removing “on” in angular

```
export class ServersComponent implements OnInit {
  allowNewServer = false;
  serverName: string = "not available Now";

  constructor() {
  }
  ngOnInit(): void {
  }

  onAddServer() {
    this.serverName = "Server started and running !";
  }
}
```

```
<button
  class="btn btn-primary"
  [disabled]="allowNewServer"
  (click)="onAddServer()"
>
  Add server
</button>
<div class="alert alert-success">{{ serverName }}</div>
<app-server></app-server>
<app-server></app-server>
```



Dont code too much inside these event bindings if its something simple like changing boolean value use it inline.

## Get data from Inputs

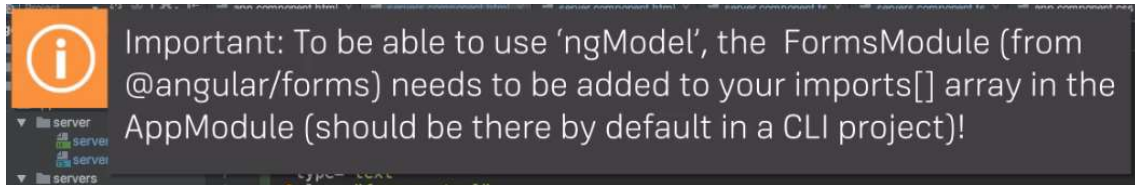
```
<input type="text" class="form-control"
(input)="onUpdateServerName($event)" />
```

```
onUpdateServerName(event: Event) {
  this.currentServerName = (<HTMLInputElement>event.target).value;
}
```



A screenshot of a web form. It features a text input field with the text "hellolshan" and a blue button labeled "Add server hellolshan".

## Two way data binding



Not like in event binding when we add 2 way data bind changes from every where affect to our selected element and we use this when we have default value.

In below event bind or any other place change to currentServerName affect to 2 way bind .

But 2 way bind change or change of it value from anywhere dosen't affect and dosent show in event bind.

Difference between the event binding and the twoWay data binding

event binding

---

---

---

---

2 way data bind

---

---

---

---

Add server

```
<div style="border: 1px solid; margin-left: 100px">
  <h5>Difference between the event binding and the twoWay data
binding</h5>
  <h4>event binding</h4>
  <input
    type="text"
    class="form-control"
    (input)="onUpdateServerName($event)"
  />
  <br />
  <br />
  <h4>2 way data bind</h4>
  <input type="text" class="form-control"
[ (ngModel) ]="currentServerName" />

  <button
    class="btn btn-primary"
    [disabled]="allowNewServer"
    (click)="onAddServer()"
  >
    Add server {{ currentServerName }}
  </button>
</div>
```

# Directive

## What are Directives?

### Directives are Instructions in the DOM!

```
<p appTurnGreen>Receives a green background!</p>
```

```
@Directive({  
  selector: '[appTurnGreen]'  
})  
export class TurnGreenDirective {  
  ...  
}
```



We doesn't have used any other directives other than component directive

## \*ngIf-(i is capital in \*ngIf)

- WE add \* to the ngIf to tell that angular this is a structural directive. Structural directives change the structure of the DOM.
- Super important fact is \*ngIf gets boolean expression may be method call or a variable .
- Another super important thing is \*ngIf don't make things hidden but make things add or remove .

```
<div *ngIf="serverCreated" class="alert alert-success">
```

## Use if else in \*ngIf

We use ng template with a local reference and add it to the place where we want to show when conditions becomes false.

```
<div *ngIf="serverCreated; else noServerMarker"
class="alert alert-success">
  Server was created server name is {{ currentServerName }}
</div>

<ng-template #noServerMarker>
  <h1>Still no server is created !</h1>
  <hr />
</ng-template>
```

## Styling Elements Dynamically with ngStyle directive (Attribute directive)

We make servers and assign online offline randomly and we want to use different colors for online and offline

```
<h1 [ngStyle]="{ backgroundColor: getColor() }">
  {{ getServerStatus() }} Server {{ serverStatus }} with Id {{ serverId
}}
</h1>
```



ngStyle is the directive.

[] to say bind its not a part of the name.

Parentheses are events of the element you are working on, like the click on a button like your example; this could also be mousedown, keyup, onselect or any action/event for that element, and what is after the = is the name of the method to call -- using the parenthesis for the call. That method should be defined on your component class, i.e.:

```
<element (event)="method()"></element>
```

Brackets works the other way. They are to get data from your class -- the opposite of the parenthesis that were sending the event -- so a common example is the usage of a style like this:

```
<element [ngStyle]="{display:someClassVariable}">
```

See? You are giving the element a style based on your model/class.

For this you could have used...

```
<element style="display:{{ModelVariable}};">
```

The recommendation is that you use double curly brackets for things that you will print on the screen like:

```
<h1>{{Title}}</h1>
```

Whatever you use, if you are consistent, it will help the readability of your code.

Lastly, for your \* question, it is a longer explanation, but it is very VERY important: It abstracts some methods' implementation that otherwise you would have to do to get an `ngFor` to work.

One **important update** is that in the `ngFor` you will no longer use hash; you need to use `let` instead as follows:

```
<tr *ngFor="let movie of movies"> <td>{{movie.title}}</td> </tr>
```

One last thing worth mentioning is that all of the above applies also for your components, e.g. if you create a method in your component, it will be called using `()`:

```
<my-owncomponent (onSearched)="MethodToCall()" "[MyInputData]="SearchParamsArray"></my-owncomponent>
```

## If u want a real time color use this approach

```
color = "";

constructor() {
  this.serverStatus = Math.random() > 0.5 ? 'online' : 'offline';
  this.realtimeColor();
}

realtimeColor() {
  setInterval(() => {
    this.color = (Math.random() > 0.5) ? 'red' : 'green'
  }, 2000);
}
```

```
<h1 [ngStyle]="{ backgroundColor: color }">
  {{ getServerStatus() }} Server {{ serverStatus }} with Id {{ serverId
}}
</h1>
```

Still no server is created !

offline of this server checked by a method and - Server offline with id 0.5551354063228995

offline of this server checked by a method and - Server offline with id 0.008921575910403323

## Adding css classes Dynamically with ngClass

```
<h1
  [ngStyle]="{ backgroundColor: color }"
  [ngClass]="{ online: color === 'red' }"
>
  {{ getServerStatus() }} Server {{ serverStatus }} with Id {{ serverId
}}
</h1>
```

This also add css class only when condition met .



## \*ngFor - (Structural directive changing the dom itself)

- ngFor start like this first we make a temp variable.
- Then we get the array from the ts .
- Syntax is like “let tempName of array”.
- Then it loop through every variable in loop and we can use the tempName variable we have given in the loop.

```
export class ServersComponent implements OnInit {
  allowNewServer = false;
  serverName: string = "not available Now";
  currentServerName = 'TestServer';
  serverCreated = false;
  servers = ['TestServer1', 'TestServer2']
  constructor() {
  }
  onAddServer() {
    this.servers.push(this.serverName);
    this.serverCreated = true;
    this.serverName = "Server started and running !";
  }
  onUpdateServerName(event: Event) {
    this.currentServerName = (<HTMLInputElement>event.target).value;
  }
}
```

```
<app-server *ngFor="let val of servers"></app-server>
```

Server Name

Testserver 2

Add Server

Server was created, server name is Testserver 2

Server with ID 10 is offline
Server with ID 10 is online
Server with ID 10 is offline
Server with ID 10 is online
Server with ID 10 is offline
Server with ID 10 is offline
Server with ID 10 is online
Server with ID 10 is online
Server with ID 10 is offline
Server with ID 10 is offline
Server with ID 10 is offline
Server with ID 10 is online
Server with ID 10 is offline
Server with ID 10 is online
Server with ID 10 is offline