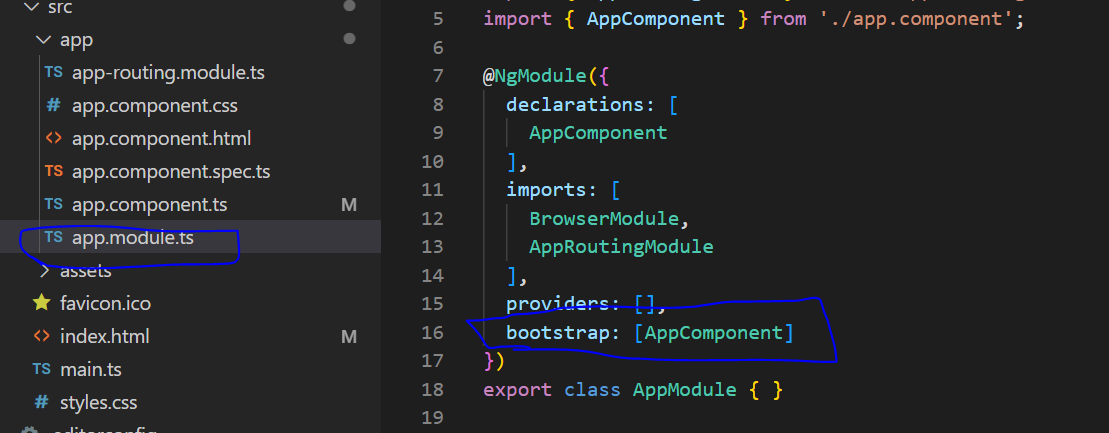
1. **Component:**

* Key feature in angular
* Whole application is built by composing it from a couple of components, which we create.
* App-component is the root component, which holds the entire application
* Root component is the component where we add other components.
* So the html file of the app component, is where we add our other components
* Each component has its own template: html file, own styling – css, and also has own business logic
* Thus component helps to split complex webpage into reusable parts.

1. **App Component:**

* App component on one hand is a normal angular component and on the other hand ,it is kind of special, since it is our root component
* It is listed in app.module.ts in bootstrap array.



* By specifying AppComponent in bootstrap array, we say to Angular that this is a special component and we should bootstrap the whole application using this component.
* For the other components we created, their selectors will not be added to index.html . Instead their selectors will be added to the app component html file

1. **Creating new Component:**

* We are building a backend for our server management application and we want to output some server information.
* We shall store that in a new folder but under the app folder, because generally in a angular cli project, all your app related content goes into app folder.
* Lets name it server, because it will hold the server component.
* Having folder name= component name is a good practice
* Each component should typically have its own folder, though it is not hard rule.

**SERVER.COMPONENT.TS**

* To name files within server folder, first we will create server.component.ts file

1. **Creating Component in server.component.ts:**

* Component is just a typescript class
* Angular will be able to instantiate it, to create objects based on the blueprint we define here.

1. **Exporting the class**

* Lets export this class, so that we can use it outside of this file too.
* Since we are going to use our server component in app component, we will add export
* The class will be named: ServerComponent

Eg: export class ServerComponent{

1. **Adding Component Decorator**

* To inform Angular that this is not just a Typescript class, but it is a Component,. We do this by adding a special decorator.
* Decorators are TypeScript feature, which allows you to enhance your classes or enhance elements in our code.
* Here for component we will use Class Decorator. Ie here it is Component Decorator
* Decorator is specified by using @ symbol

Eg: @Component

* @Component(
* {}
* )

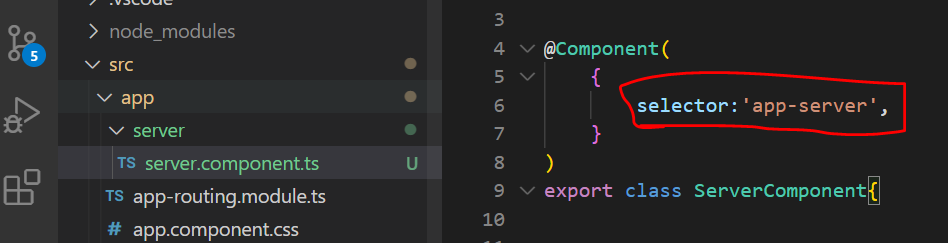
1. **Importing Component Decorator**

* Component decorator is not something TypeScript knows from the start. So we have to import it.
* Here we import from @angular/core.
* Core package gives access to core functionalities of Angular.
* After importing, javascript will be able to parse it .

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

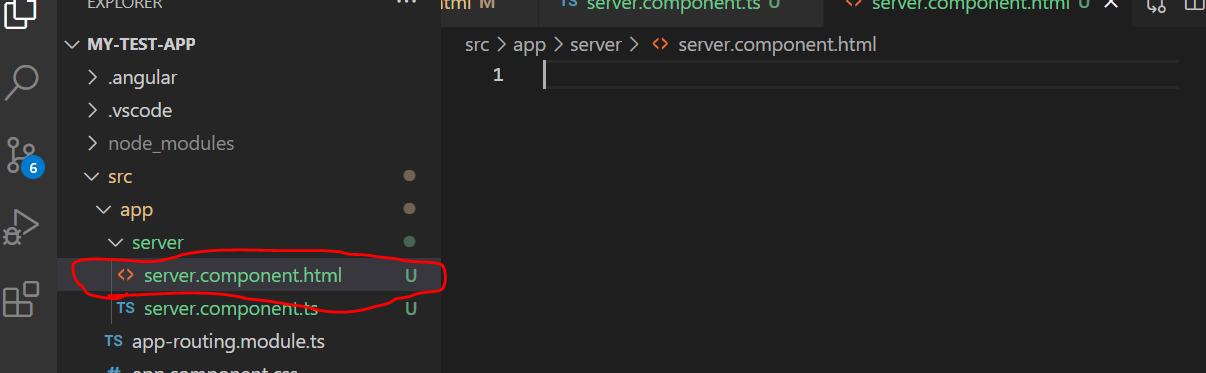
1. **Passing Javascript Object to Component decorator**

* We pass Javascript object to component decorator to configure it.
* Because without any configuration, it is not valuable to angular.
* Here we can setup some important information, which will be used as metadata for this class, which tells angular what to do with this class.
* **Selector**
* It is the HTML tag by which you will be able to use this component later in other component’s templates.
* Selector should be a string and can be given any name, but name should be unique. Ie we should not override a default html element.

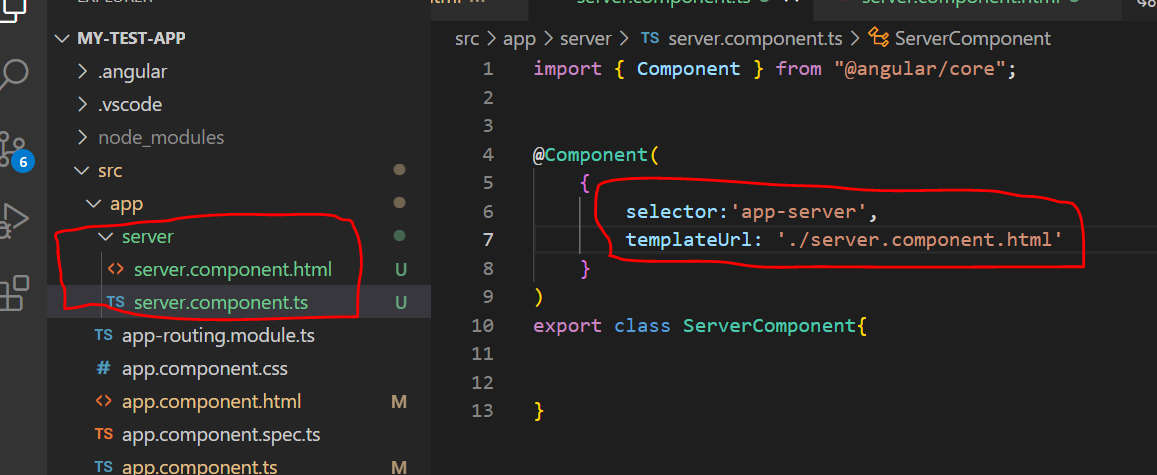


* **Template:**
* Here we reference another external file.
* This external file needs to be created.

Eg: server.component.html



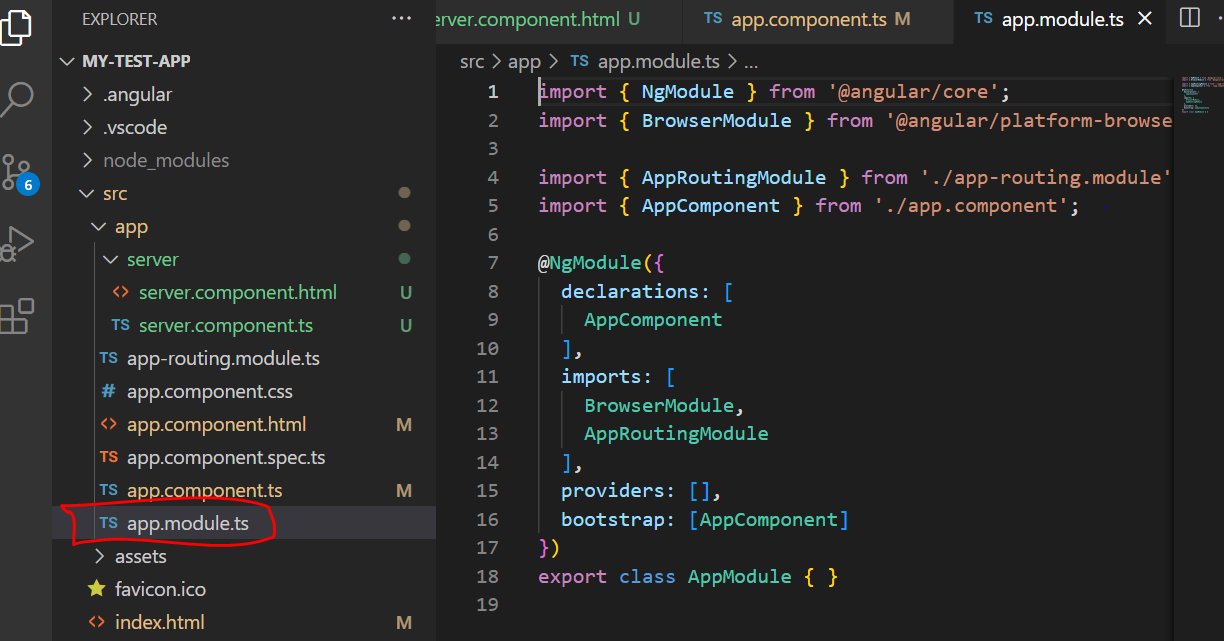
* This file holds the template of the component created.
* In template url, we should give relative path, so that it gets bundled by webpack.



Thus, with this step we have completed creation of the first component.

1. **Registering the created Component:**

* To use this component we need to go deep into app module(***app.module.ts***) and understand what happens here, because we need to change something here to use our own component



**Understanding Role of App Module and Component Declaration:**

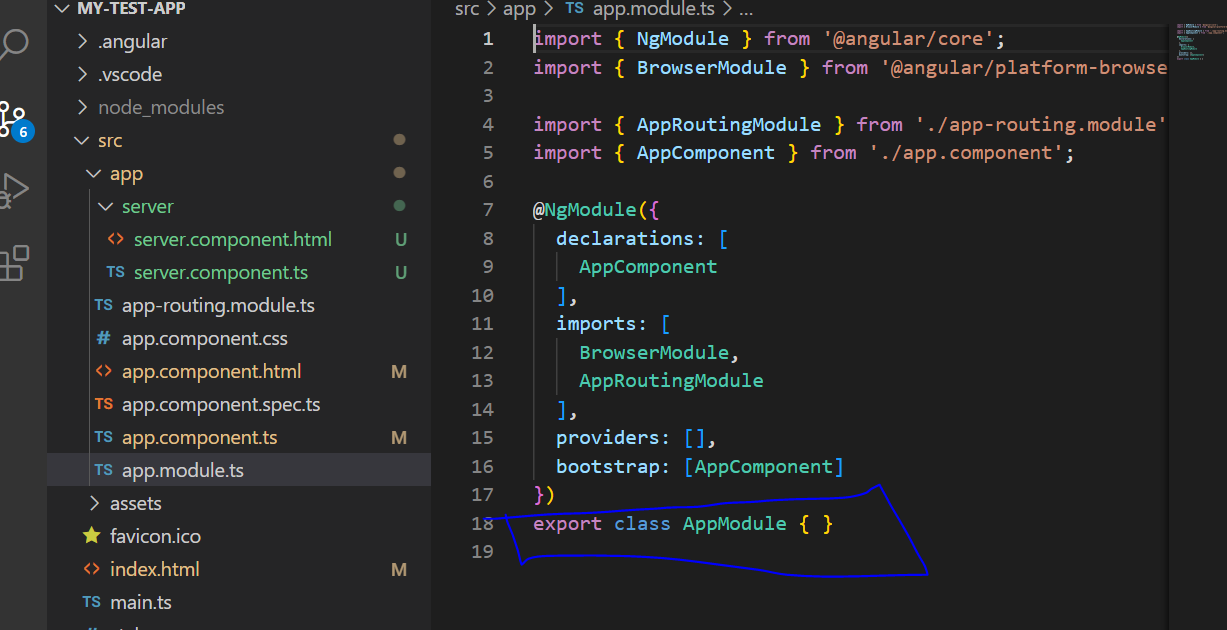
* To use the newly created server component, we need to change something about app module.

**What is app Module:**

* Angular uses components to build webpages and uses modules to bundle different pieces. Eg Components of our app into packages.
* We will be using app modules for the majority of this course.
* In big projects we will split our app into multiple app modules.

**What is a Module:**

* Module is a bundle of functionalities.
* Modules gives angular framework the information, which features does my app have and what it uses.
* Also we can observe that app module is a empty TypeScript class like our component



* Though it is a component we transform it into a module using a decorator: @NgModule Decorator which is imported from @Angular/Core

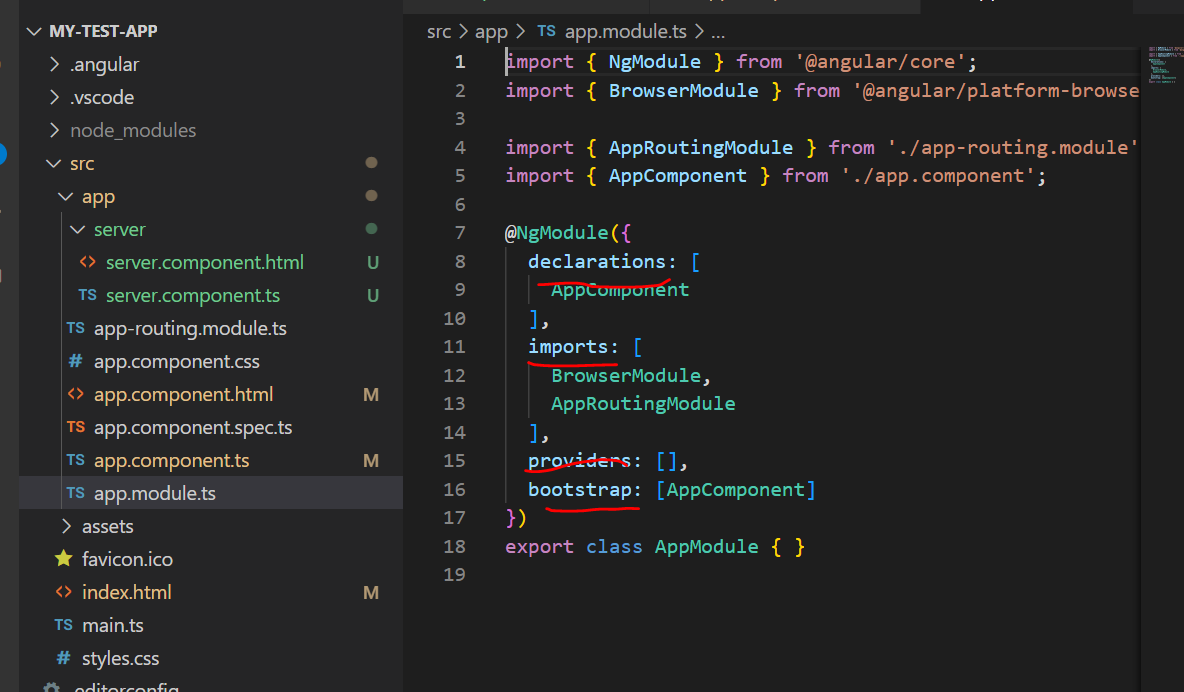
Text

Description automatically generated

* In there, we see four properties, which we setup on the object that we passed to Ng Module

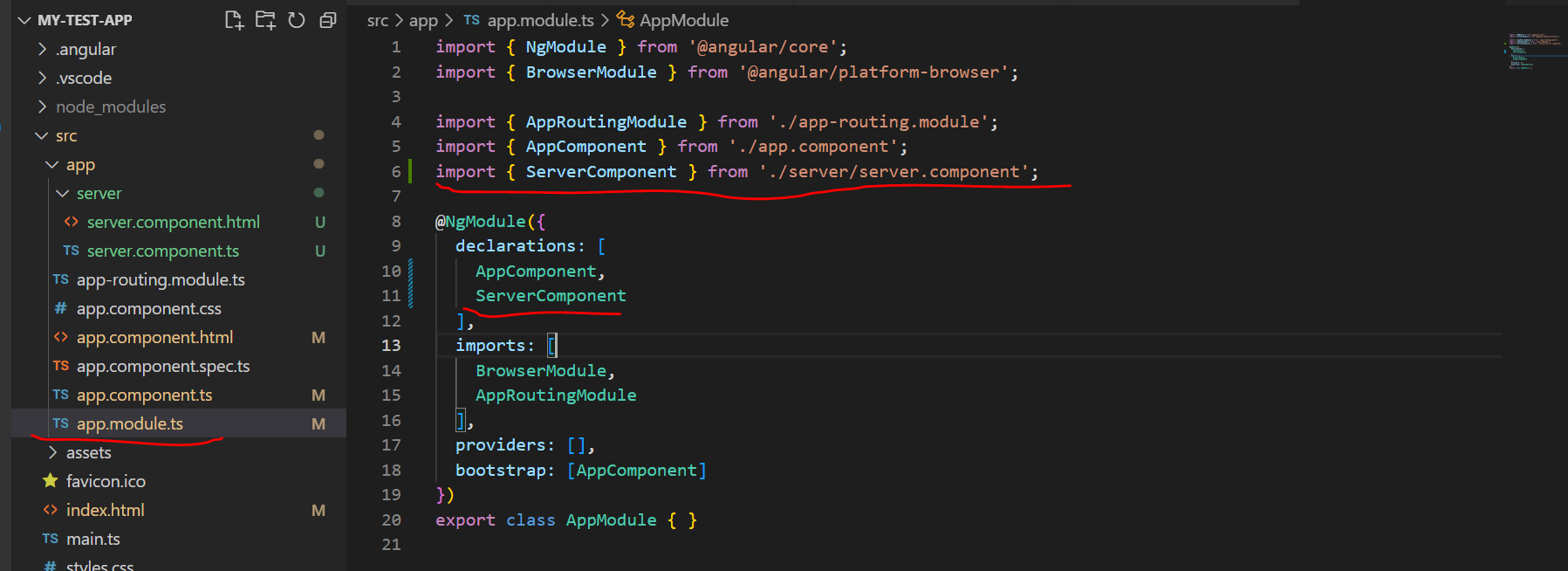
1. Declarations
2. Imports
3. Providers
4. **Bootstrap** -> Tells angular which component you should be aware of when the whole application starts. Ie which component should you basically recognize in the index html file and in our case, that was the app component

* Also we know that we wont add any more component selectors to the index html file. Hence we wont touch the bootstrap array.



**Registering New Component in NgModule declarations array:**

* By Default Angular will not scan all your files here
* Hence we need to specify that the component exists. Thus just creating new server component file is not enough
* Hence we need to register the newly created component in NgModule , to tell angular that it is part of this module
* New components are registered in the declarations array. We can see that we have already registered app component here.
* Now we will add the server component also here. Also we need to specify the import for the same .
* Note that we don’t specify the .ts extension while we import, because webpack adds it automatically while compiling

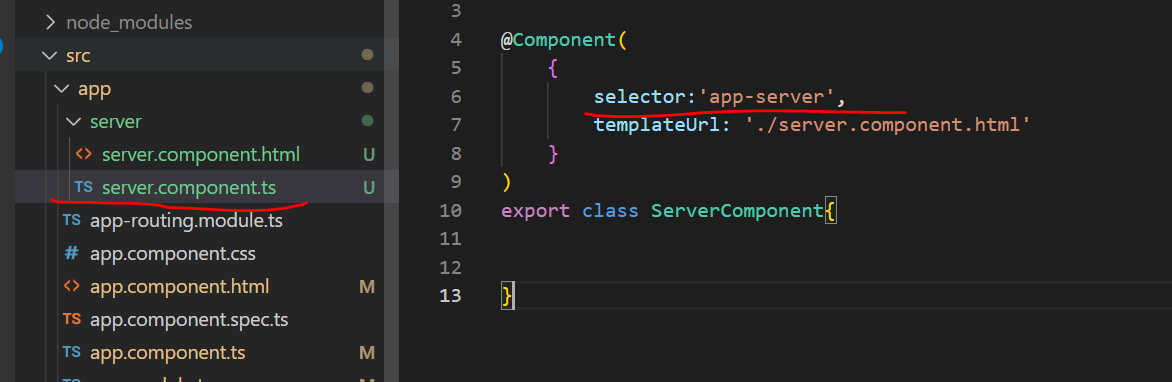


**Imports array in NgModule:**

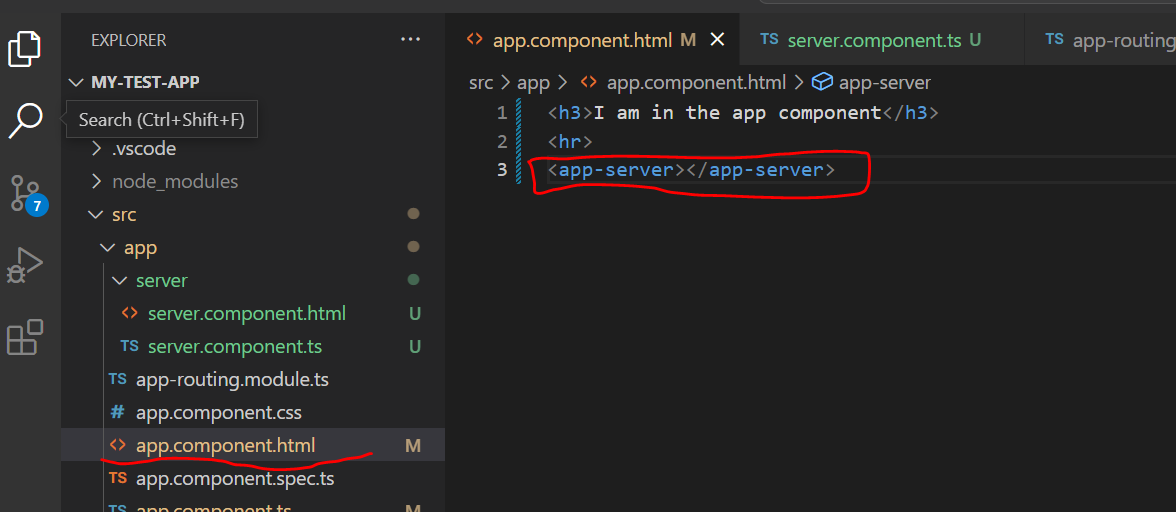
* Imports allows us to add some other modules to this module.
* We know that we can split our app into multiple modules and we can import these modules to basically make this module a bit leaner, thus outsourcing some stuff into another module
* These modules specified in app section are those inbuilt modules of angular

1. **Using the Newly Created Custom Component:**

* Our selector is present in server.component.ts which is named as app-server



* We wont be adding it to the index.html file where it wont work.
* Instead lets go to app component html file(app.component.html) – add the app server element there
* **Emmet server plugin used for writing html**

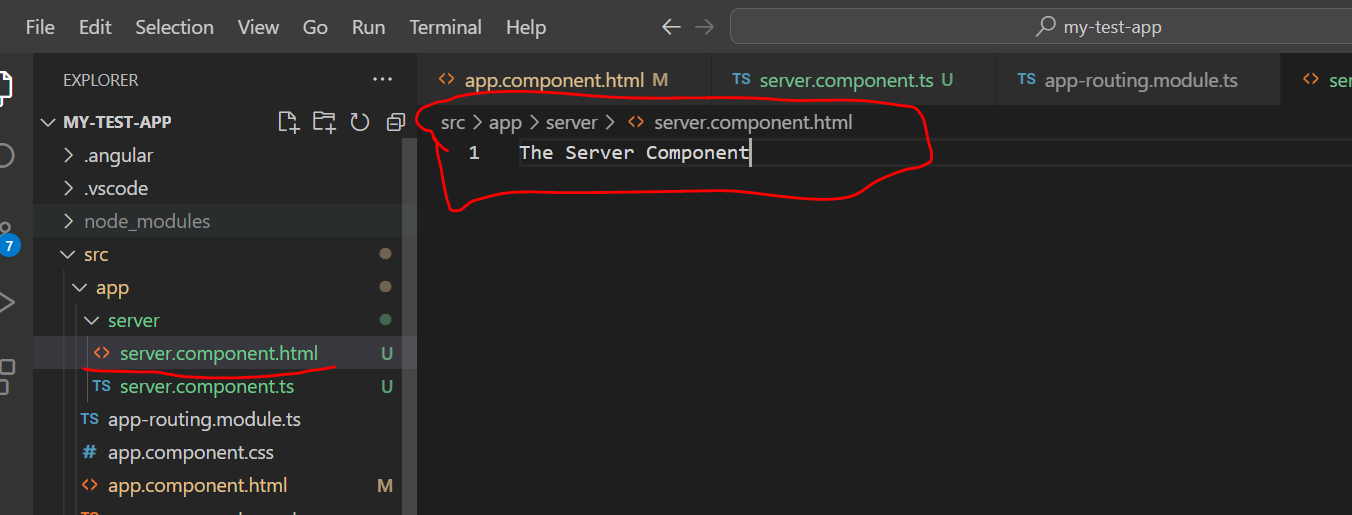


* Now we can see the below output in browser:

Graphical user interface, text, application, email

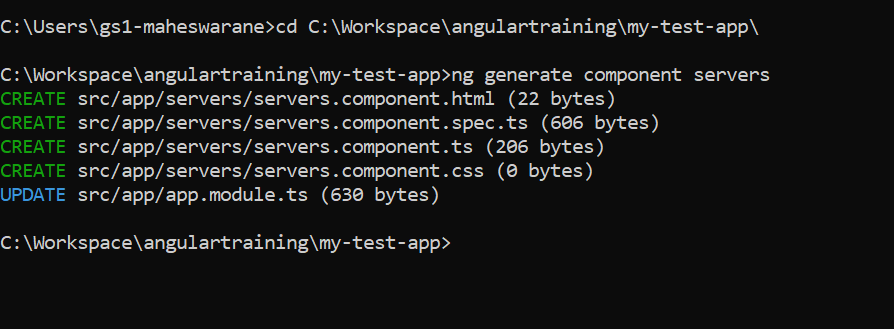
Description automatically generated

* This content : The server Component comes from file : server.component.htm file

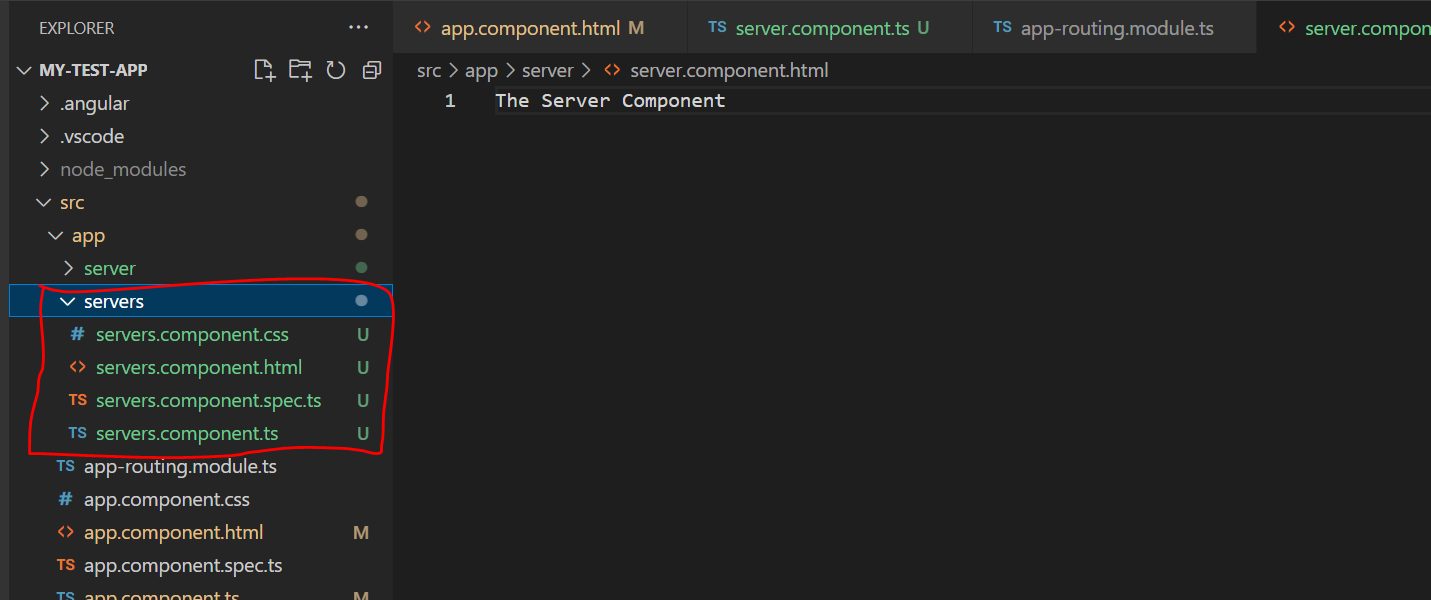


1. **Creating Components with CLI**

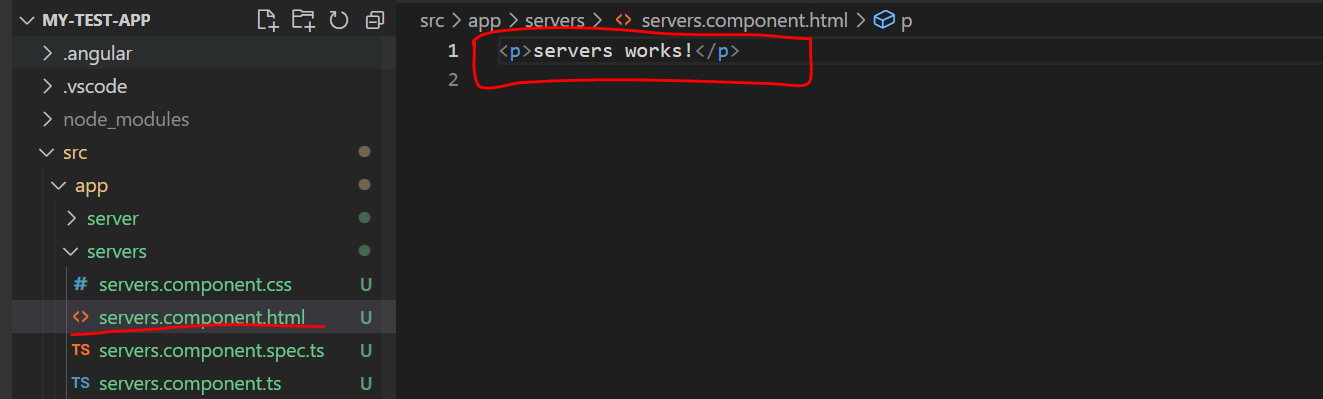
* We can create component manually by adding a servers folder with out component or use the CLI with ng generate command, which helps us to generate some elements
* Elements are those that are supported by Angular. One such element is the Component.
* Cmd: ng generate component componentName or it shortcut: ng g c componentName



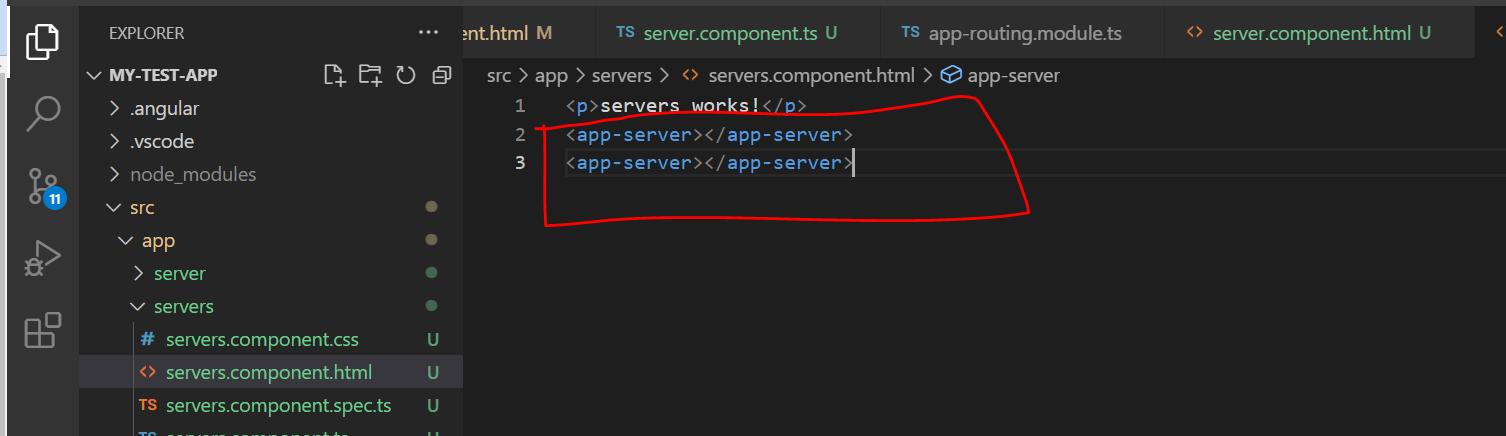
* After this, new folder servers will be created for component and four files will be created. Css,html, spec.ts and ts



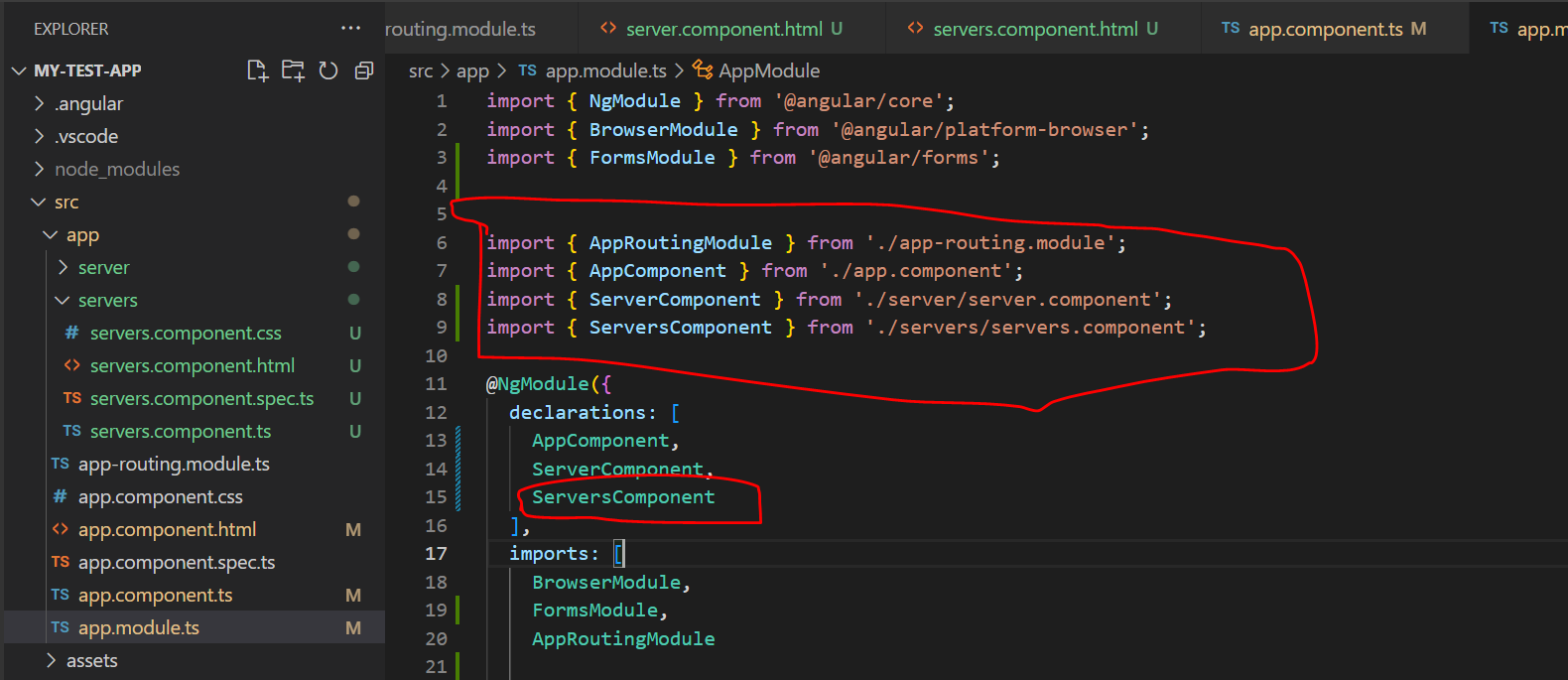
* Suppose if we want to output server from the servers component
* We already have selector: app-server for the same.
* In the template of servers component: servers.component.html, we can overwrite the default text with the app server
* Default text:



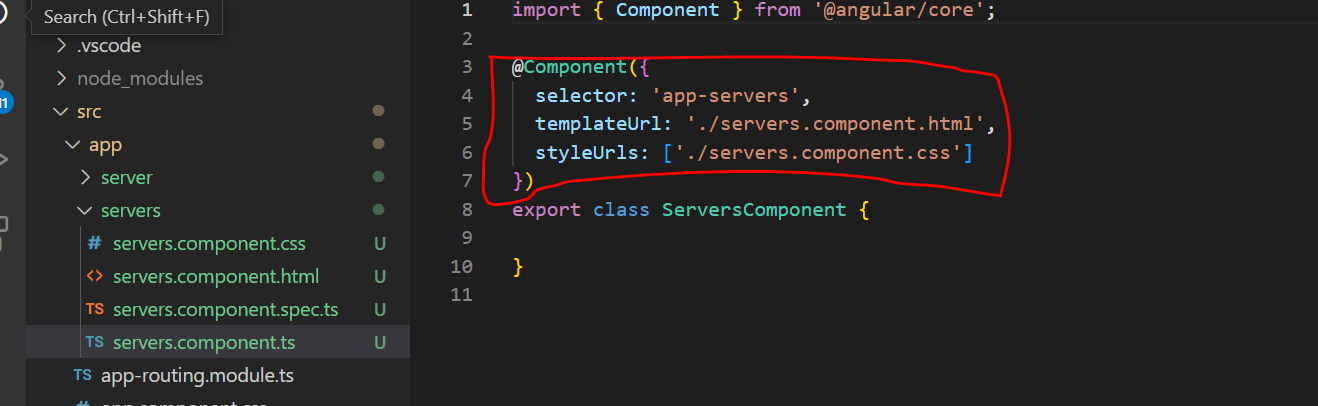
* Replaced Text:



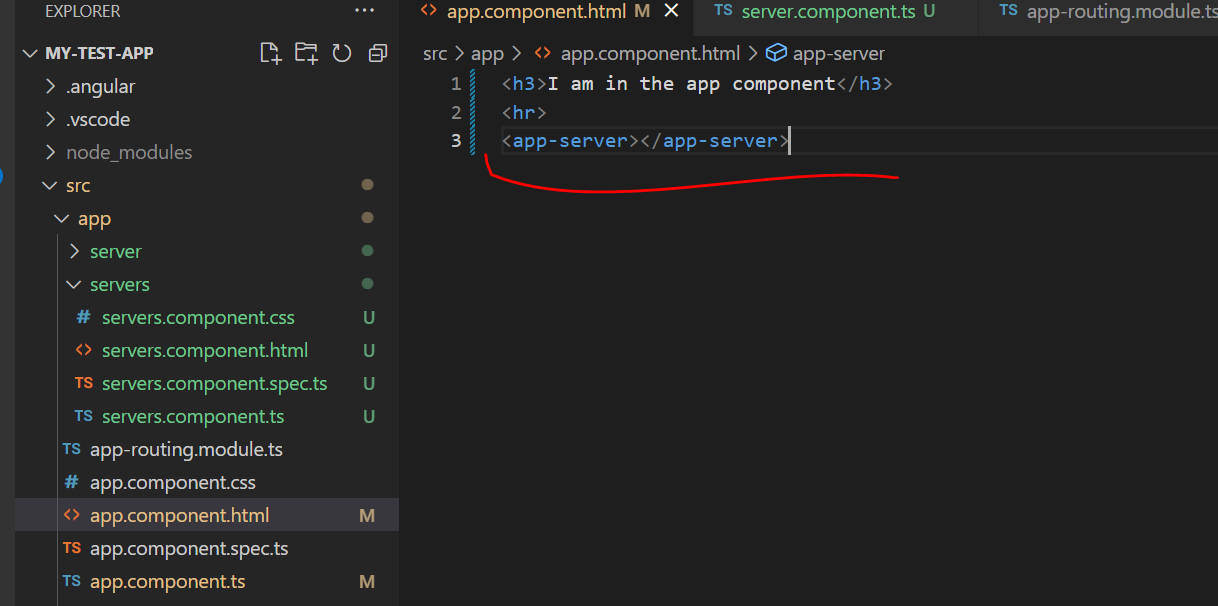
* Now we also need to update the app module . but that is done automatically by the CLI
* We can see that CLI has already automatically added import and declarations in NgModule for ServersComponent



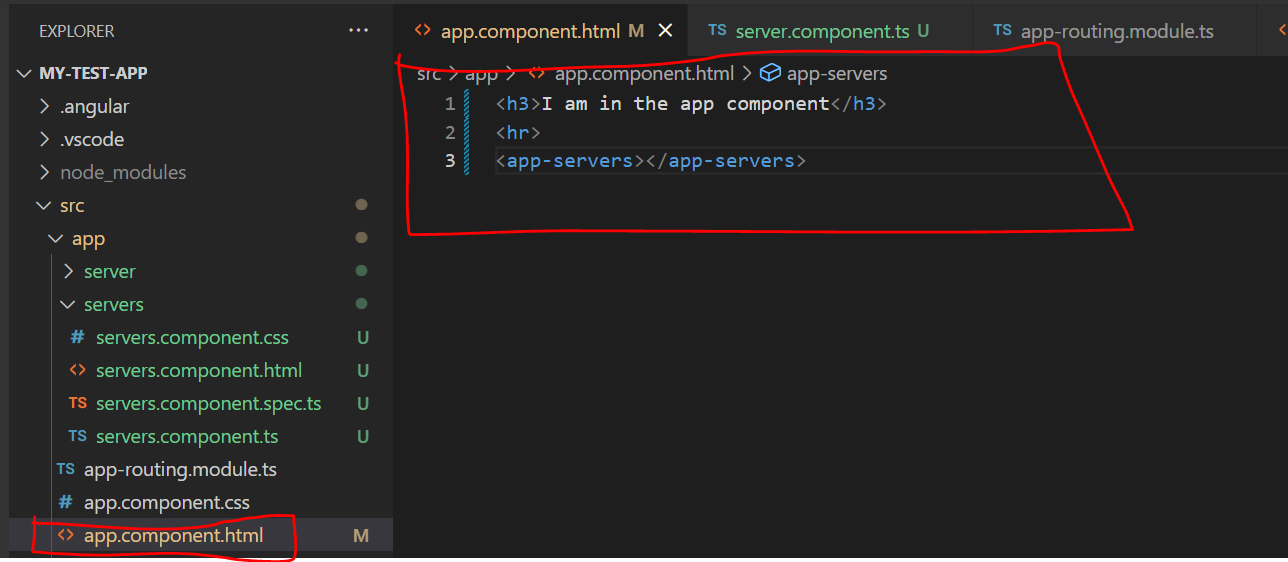
* Now we can go to our servers.component.ts file to check the selector which is app-servers



* Next we can go to app.component.html and update with <app-servers> instead of the earlier one
* Earlier one:



* Replaced one:



* We can see that browser got updated:

Graphical user interface, application

Description automatically generated

* Now if we inspect the code we can see, a) one Outer app servers (<app-servers>)component b) and in between two single <app-server> components



* Thus we learnt

1. How to create components manually or with CLI
2. How to nest them into each other. Ie Usin component selectors in any other template of any other component
3. How to replicate components by simply using the corresponding selectors multiple times.

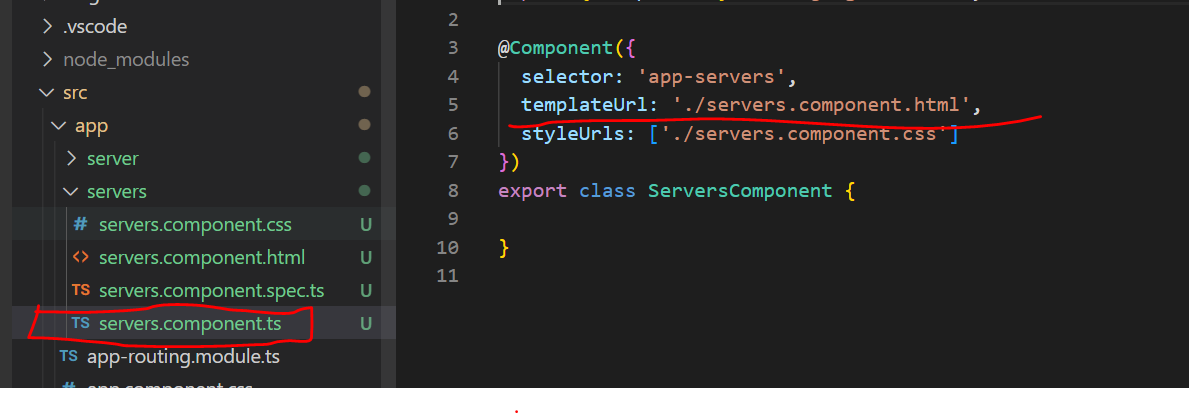
* Next lets check on how to style them

1. **Working with component Templates:**

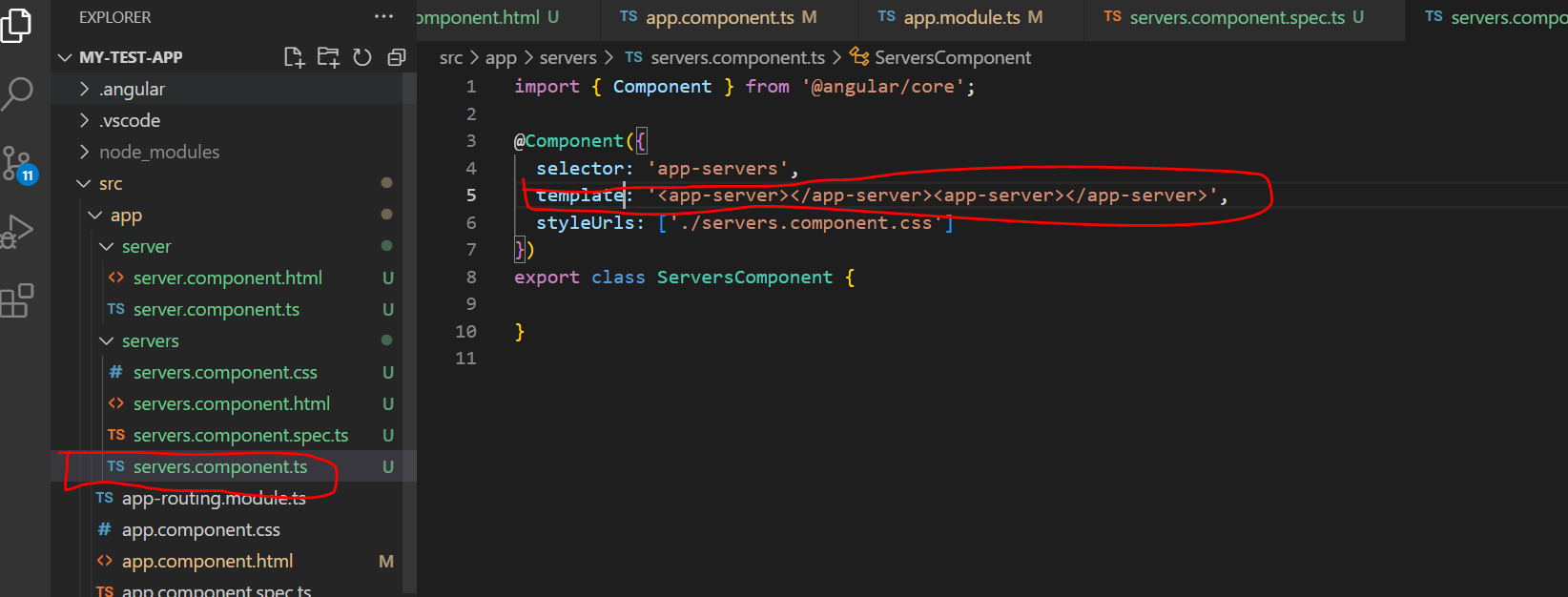
* So far we always used the external template file and not done anything regarding styling of components.
* Now its time to change both

**Inline Template:**

* Instead of using an external template file, we can also use a inline template
* It means, We define the HTML code in the Typescript code
* We can do this by simply going to our service component : servers.component.ts file



* Right now in the file we point to html file using templateUrl.
* We need to change it to template.
* One of the two needs to be present. Either a link to external template or just template to define it in this file itself
* Hence each component needs to have a template mandatorily
* This is the one property that we need to have at all times.
* Now have changed templateUrl to template as shown below and we will get the same output



* If we want to add more html code to it, we can switch from a normal string i.e with single quotation marks: ‘’ to using backticks `` for writing multiple line strings in between like below:

Text

Description automatically generated

* If we don’t have that much code to write in template, using inline template approach may be fine – where we will have all the logic in the typescript code.
* But when we have more than 3 lines of code in the html part, then using an external file is a good idea.

**Important Point:**

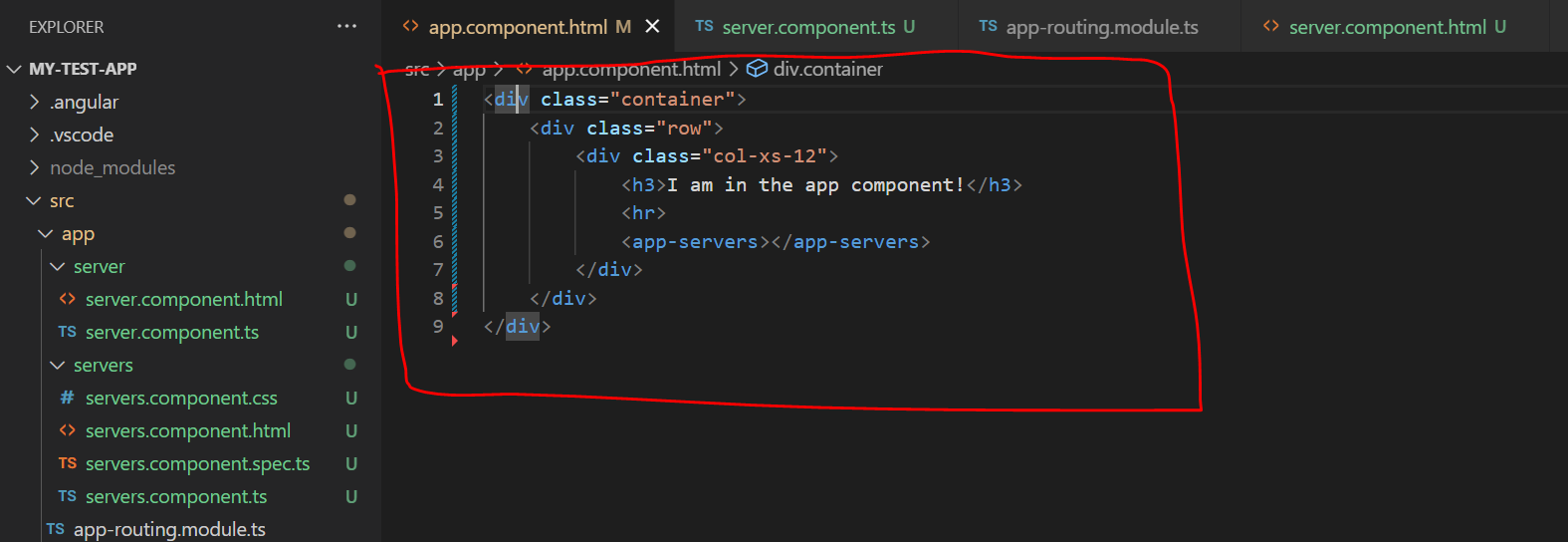
* You can avoid Selector and Styles in component typescript file, but we should have either Template or TemplateUrl mandatorily

1. **Working with Component Styles:**

* We understood that templates are important and we can use them either a) in line b) external file
* Now let’s check on styles
* For that lets go back to AppComponent for the same.

**Bootstrap classes for Enhancement:**

* The page doesnot look pretty. We can use bootstrap classes to enhance this.
* we will a div with a container class around it, then add a row -> this is just using CSS framework – Bootstrap. Container, row are all Css classes provided by bootstrap framework. They are not angular specific.
* We also add a column with class : col-x2-12 to have one column on all device widths.



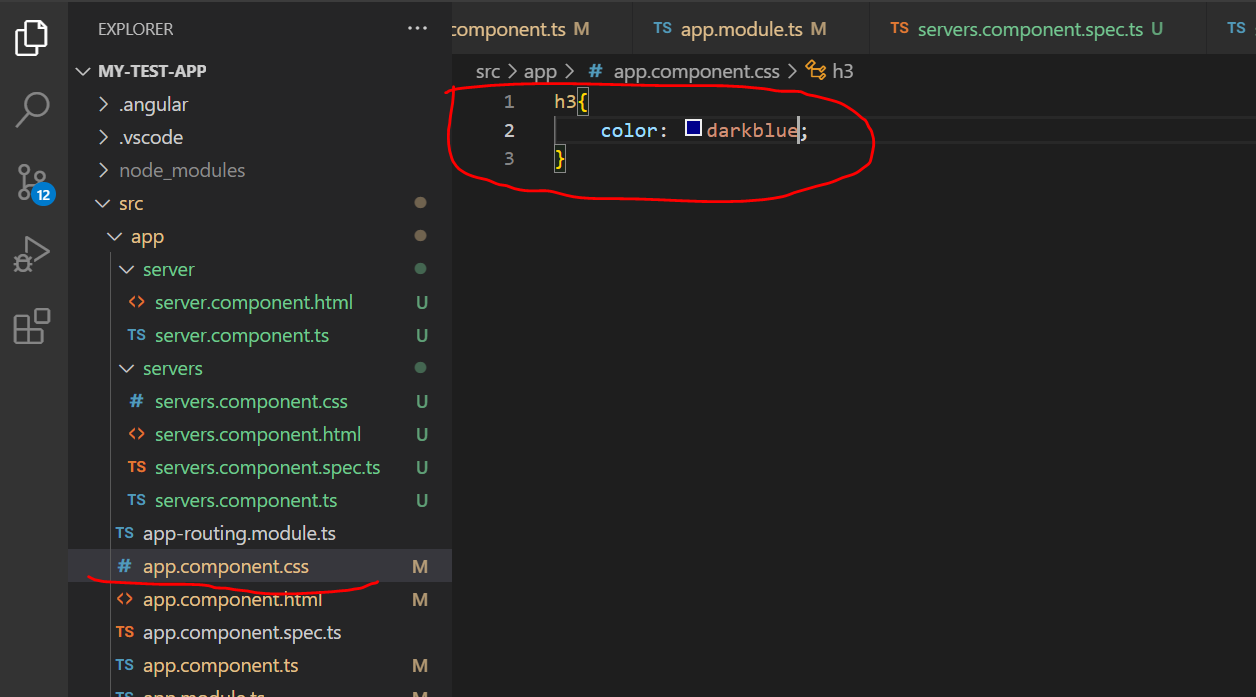
* Now we can see that our app looks nicer with the above changes:

Graphical user interface, text, application, email

Description automatically generated

**Using Angular:**

* Now we can use some tools provided by angular. Lets say I am in the AppComponent heading here
* We want to change its style. May be giving it a blue colour.
* We can do this by going to app.component.css file, which defines the styling for this component.
* Now we can write normal CSS code.
* Here we override the style of h3 tag and set its colour to blue.



* Now if we save and check the browser we can see the updated page.

Graphical user interface, text, application, email

Description automatically generated

* App.component.css defined in app.component.ts file is used to define the styles for this component. Hence it gets picked in our scenario.

Text

Description automatically generated

* Just like we are able to choose between external file and inline code for the template, we can do the same for the styling

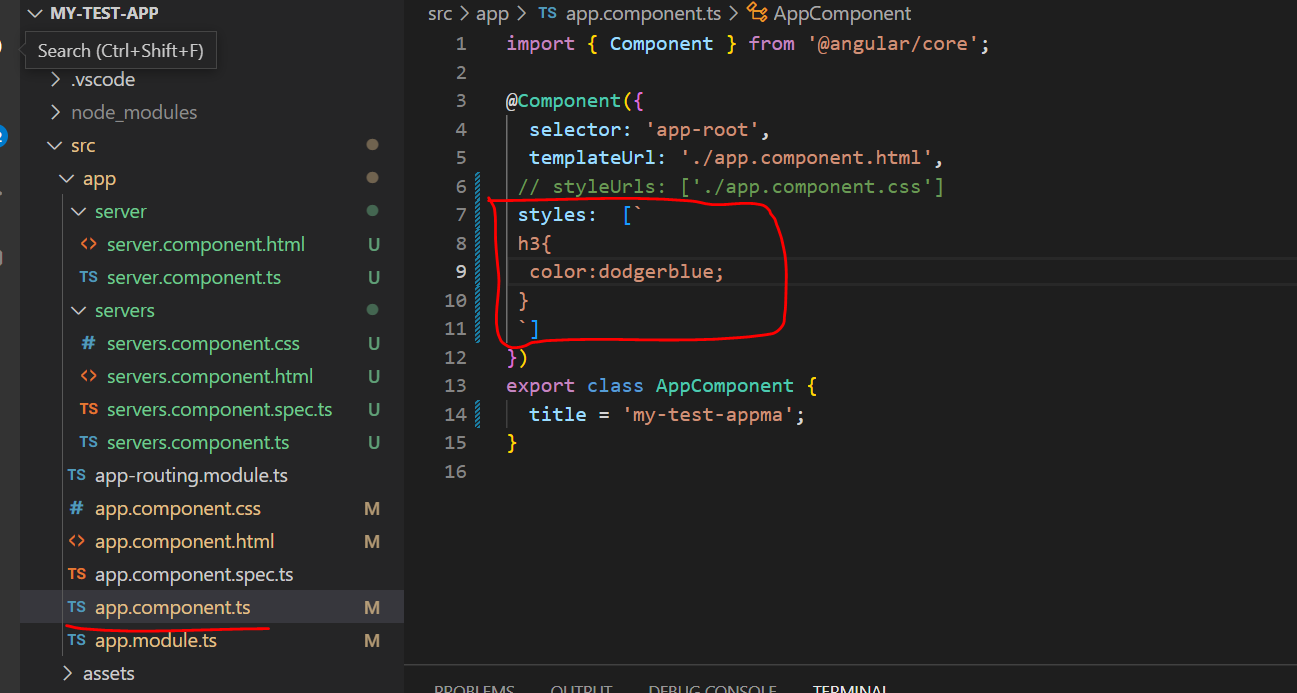
**Choosing between external file and inline code for styling:**

* In app.component.ts file we can see that styleUrls is an array here. Whereas templateUrl was not an array
* It is because we could refer multiple external style sheets here. Hence we could add pointers to other stylesheets

Text

Description automatically generated

* Besides that lets comment styleUrls here and add another property which is just styles , which also takes an array.
* It takes an array of strings. We defined the style in this string and we will be using backticks for the same – so that we can write multi-line expression here and write our h3 overwrite here.



* Now if we save and see the application in browser, we can see that the blue is updated from dark blue to dodger blue.

Graphical user interface, text, application, email

Description automatically generated

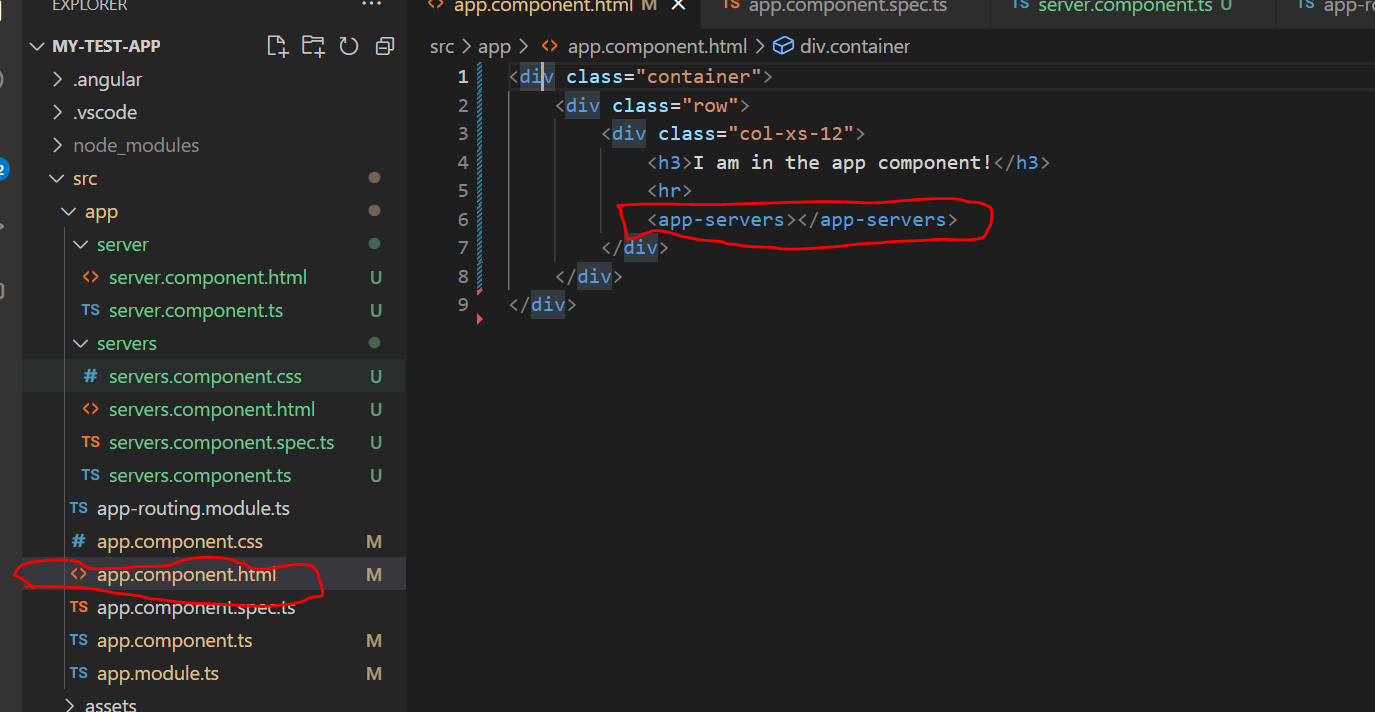
* Now we can see that the inline style is taking over.

**Deciding Inline vs External Styles:**

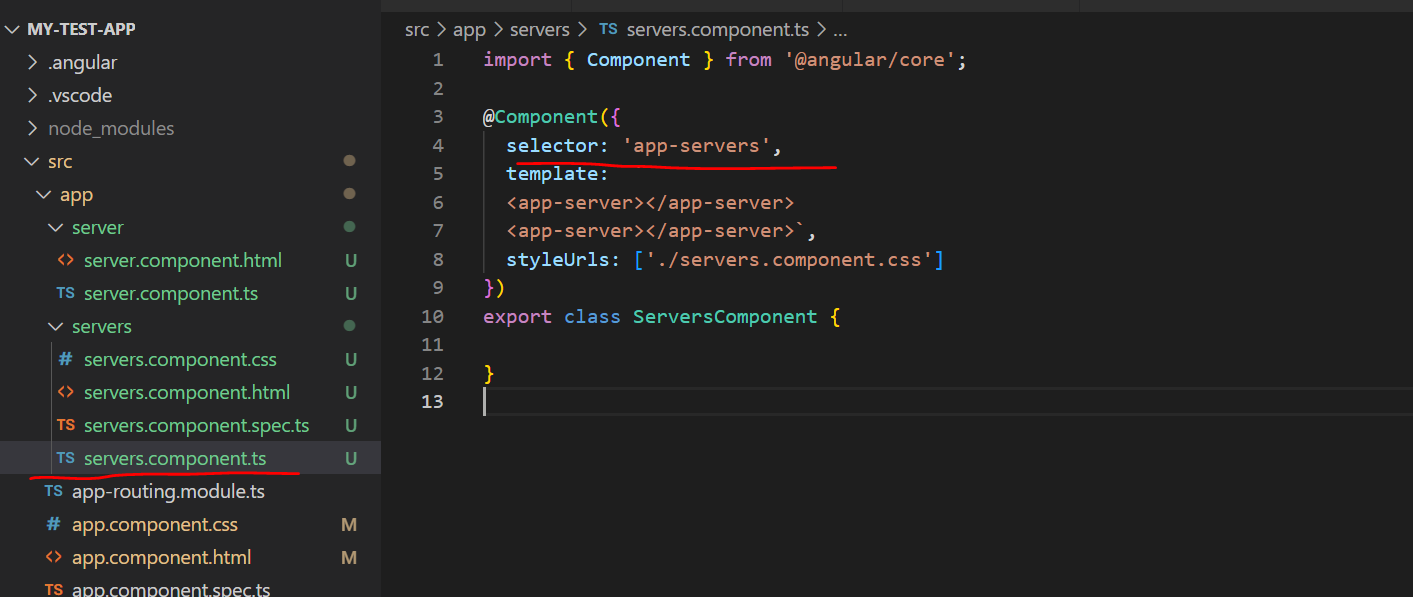
* Just like template, we need to decided which type of style we have to use. We cannot combine both url and inline style.
* But both styles has to be array here.
* In template we will have only one template. But with regards to styles we will have multiple style files or style definitions
* Rules for choosing external or inline style is basically the same like Template.
* If we have more code then we will go for external file and if it is short style definition, then we will go with inline style definition

1. **Fully understanding Component selector:**

* We already know that the selector defined in ts file( should be unique)
* We don’t have to use this type of selector
* Right now our selector is the same selector we use it in css for selecting a element.
* So we have app servers here as a element



* This is recognized by angular, because the selector in servers.component.ts is mentioned as app servers



* Hence it works like a css selector

**Attribute Selector:**

* In servers.component.ts we can also put the selector in square brackets, to use the attribute selector.
* In css, we can select elements by attribute, by enclosing that attribute in square brackets

Text

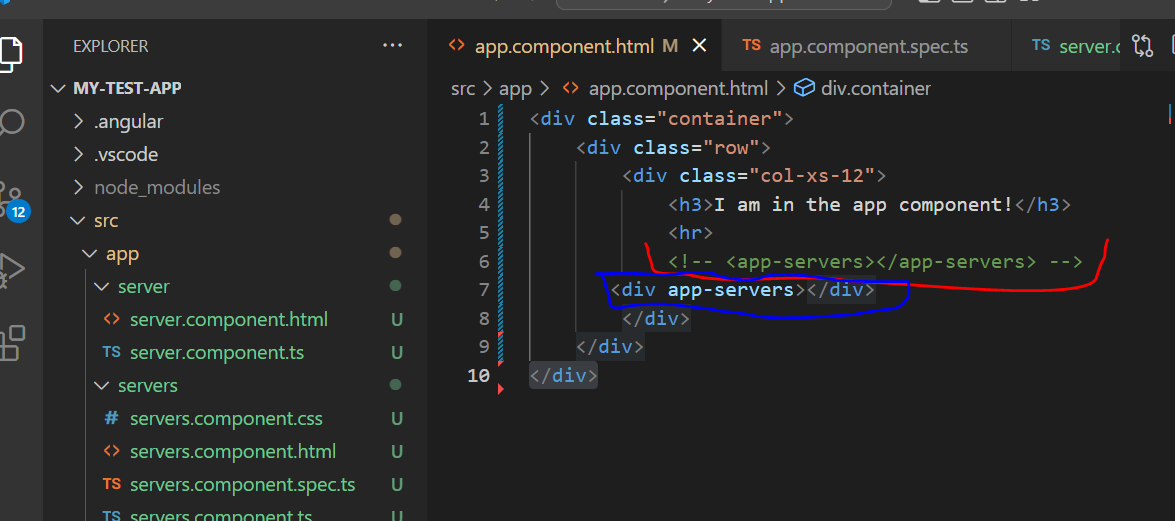
Description automatically generated

* Now if you save this and check in browser we can see that our app is broken

Graphical user interface, text, application, email

Description automatically generated

* Also if we look at the error message, it say ‘app-servers’ is not a known element.
* This is because Angular does not recognize app servers anymore. Because we changed the selector to be an attribute.
* To make this work again, we need to comment the below line in app.component.html
* And then add a div or any other element, or a normal html element, which has app-servers attribute.
* Now with this custom attribute added now, we will be able to see the server component again, because angular selects the element



* Now on checking the browser, we can see that server component appearing again without any issue.

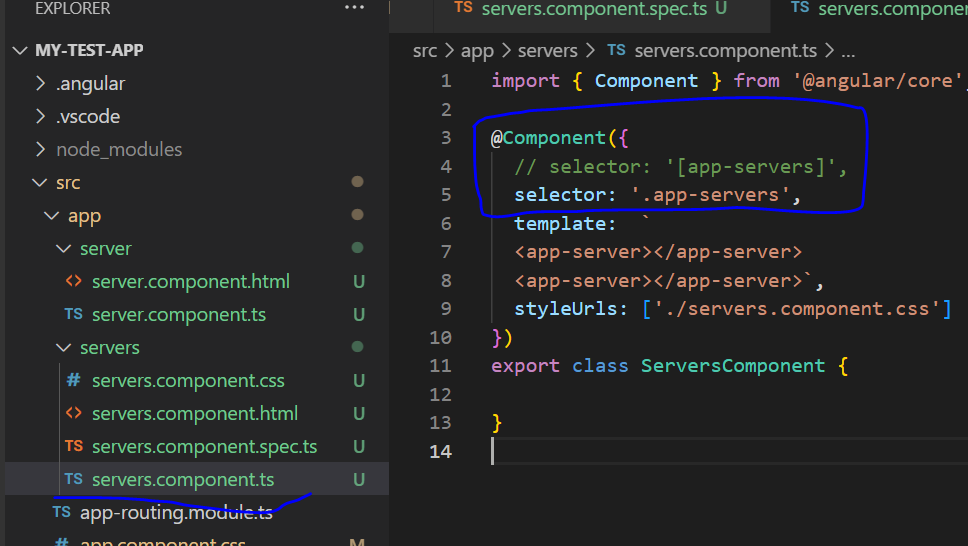
**Graphical user interface, text, application

Description automatically generated**

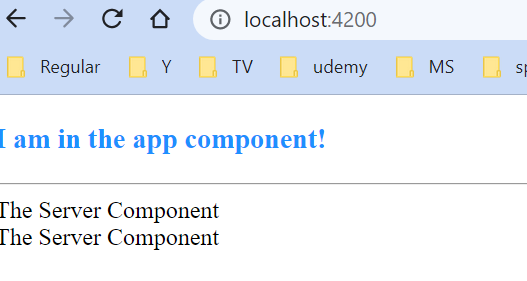
* Now with this custom attribute added now, the app works again. This is because angular selects the element by attribute and not by the element itself – because we changed the selector

**Another Alternative:**

* Another Alternative to custom attribute in selector is to select by class.
* If a dot at the beginning appservers(eg: .app-servers), just like in css.



* Here now we can select by class
* To add that we add a new div here in app.component.html file , which has a css class of app-servers
* By adding this, angular recognizes this as a selector which is why we still see the app in the browser



* So the options we have are a) Selections by attribute b) Selection by class
* On the side note, selection by id wont work, which is not supported by angular.
* Pseudo selectors like hover and others too don’t work