Angular Notes

=> to install Angular using npm run command npm install -g @angular/cli

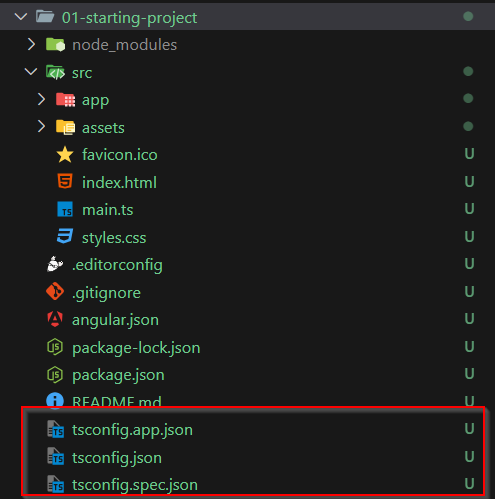
=> To create new project hit command npm new {project name}

-> install Angular Essentials extension, that is pack of more than 5 extensions that usefull for development in VS code.

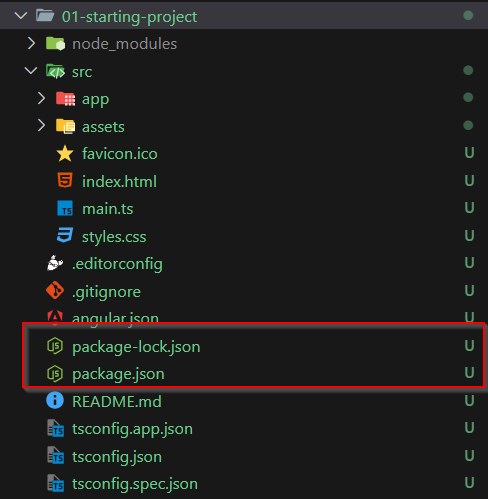
-> TO run your Angular project hit npm start command

-> ACADEMIND\_JETBRAINS is the code for free Jetbrains edditor for six months.

* Below are the Configuration files for TypeScript, generally no need to change. They are generally for the in built use to convert TypeScript code to JavaScript when software runs in browser.



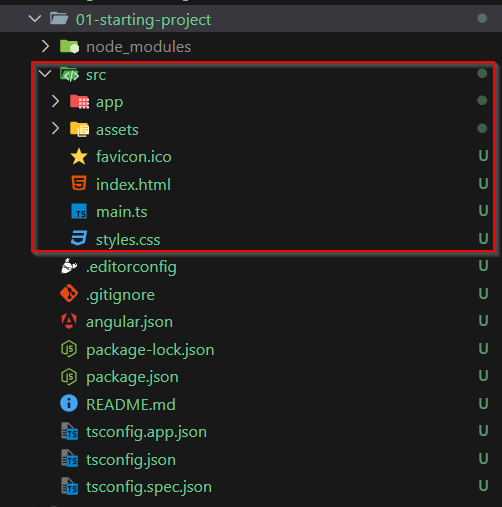
* Below files are main files about project introduction and for identify the project dependency for various npm pakages.



* Below file is generally use of the configuration of Angular and it’s various tools.

File name is angular.json.

* This src folder is main and important folder for your development code. Generally all files and folders comes under this folder.

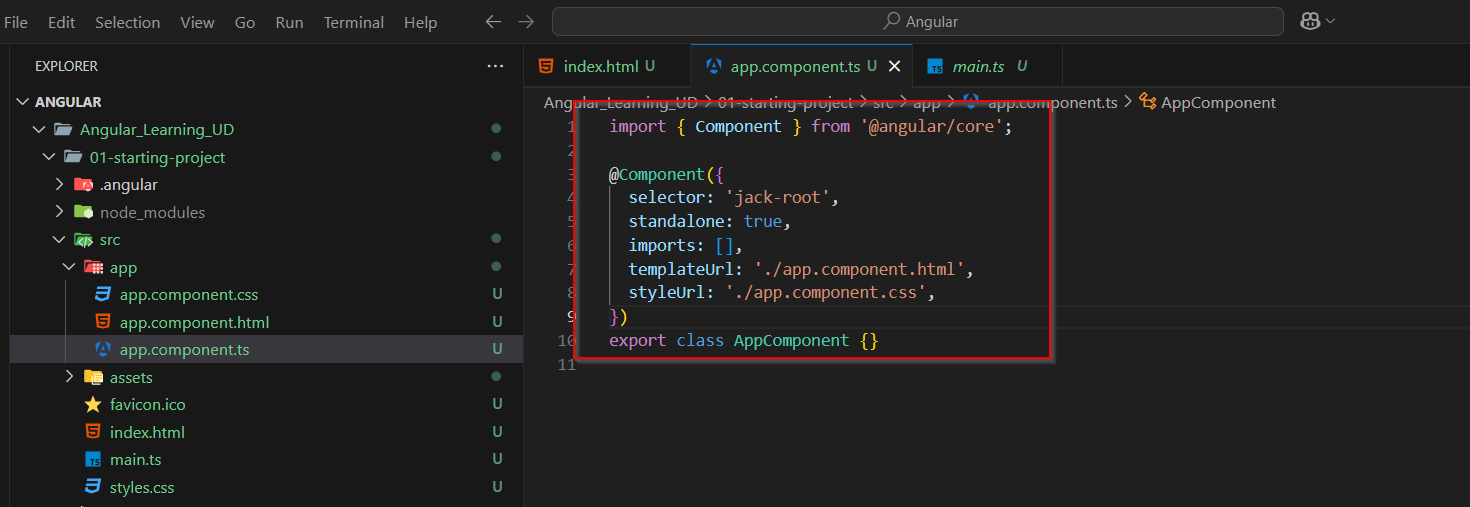


==== How Angular project runs.====

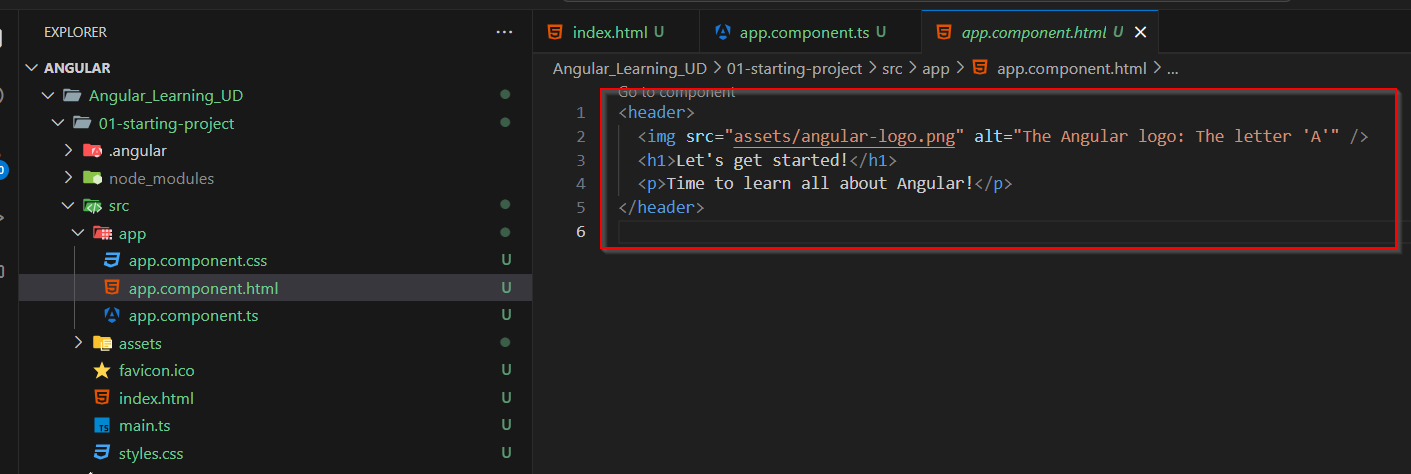
* First code file that execute after all configuration files is main.ts file.



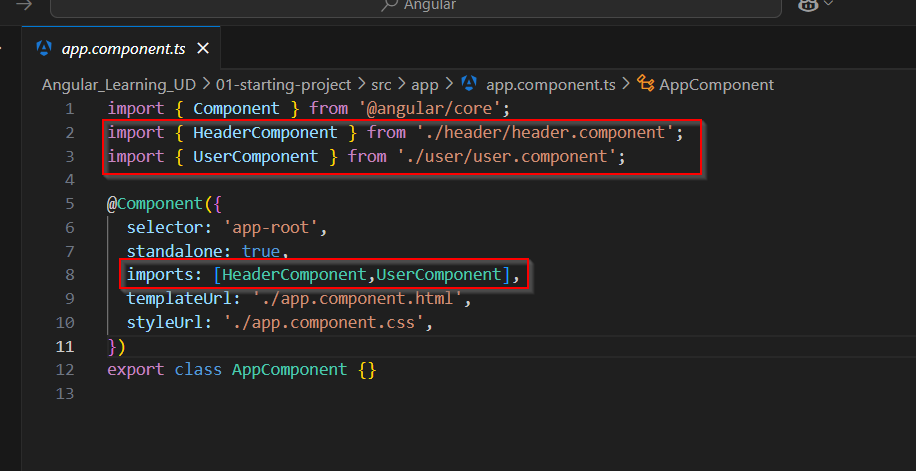
* Here we imported bootstrapApplication object from @angular/plarform-browser pakage.
* And another object imported is AppComponent that is from in built folder app/app.component.ts file.
* So When bootstrapApplication try to load AppComponent it will go to that app/app.component.ts file as per settings it loads it.



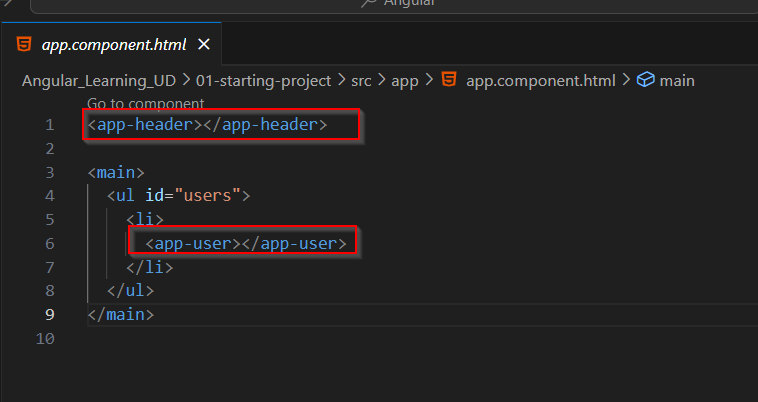
* Here Component object is imported from @angular/core pakage.
* @Component that is a decorative. This is TypeScript feature.
* Once this decorative is implemented with certain options, it will pass this all options or settings to below Javascript class that exported in same file AppComponent.
* Here one setting is “Selector” , it means browser will check for element in index.html file that is mentioned here in “selector” settings.
* And now in place of that element, for example here element is ‘jack-root’ , so browser finds the element ‘jack-root’ and replace it will what ever code return in file that is mentioned in “templateUrl” settings. Here in this case it is app.component.html.



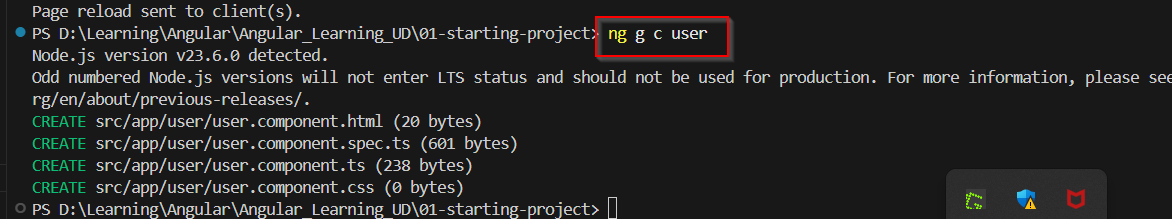
* While rendering this content it applies the style that is define in “styleUrl” settings file.
* In Angular it is recommended that App component should be the only root component and all other component should be render inside the root component.
* So for that we can register or we can say import other components in app component and then define it in “imports” array in @Compoent decorative as shown below in image.



* Now to use those custom component modify app.component.html file accordingly.



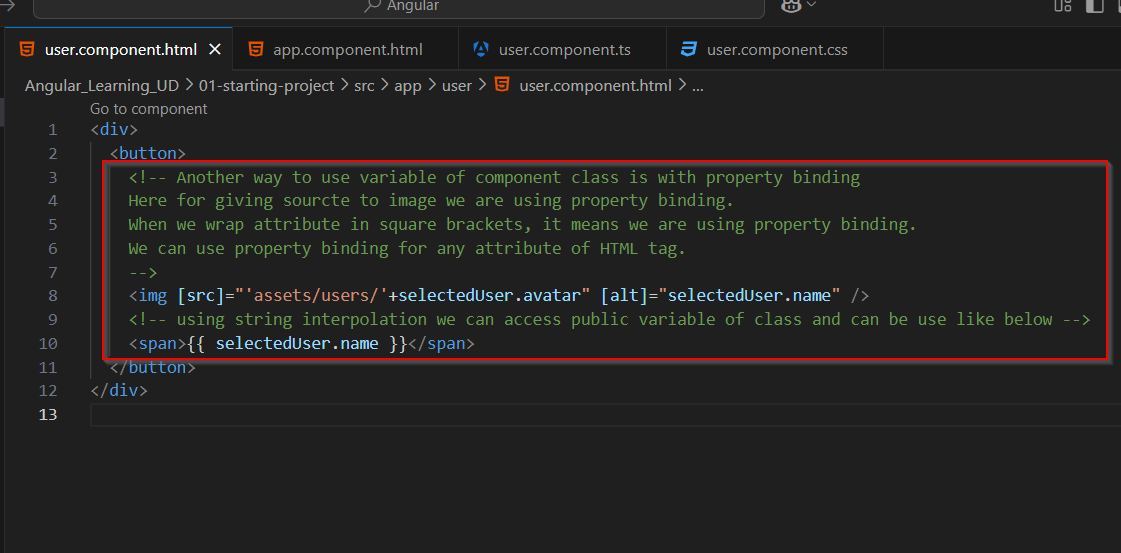
* Using ng g c command also components file can be generate. Seen as below in image.



* Another important concept of Angular is we can define variable into the component class and that can be accessible in that component html file by default.
* Here selectedUser is available for user.component.html file.

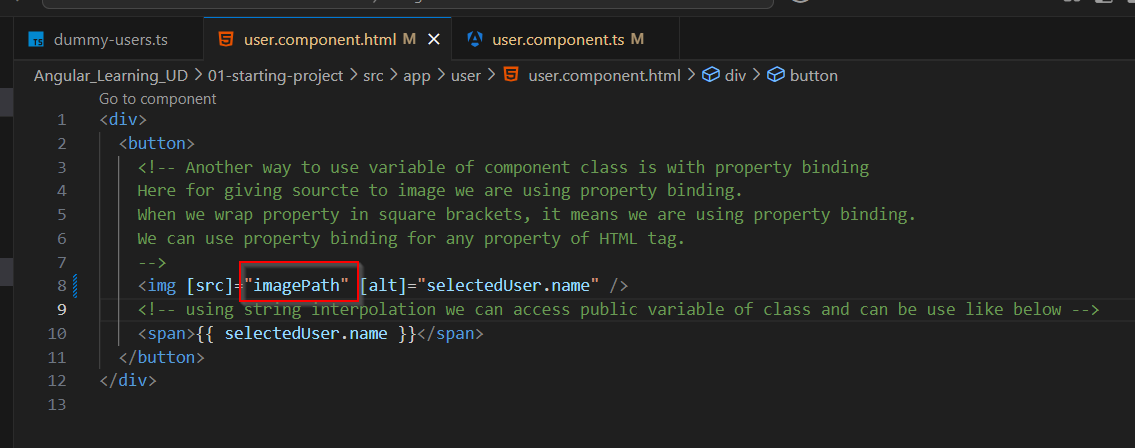


* We learn how we can use that class variable in html file
* As shown in below image we can use that variable via property binding or string interpolation.

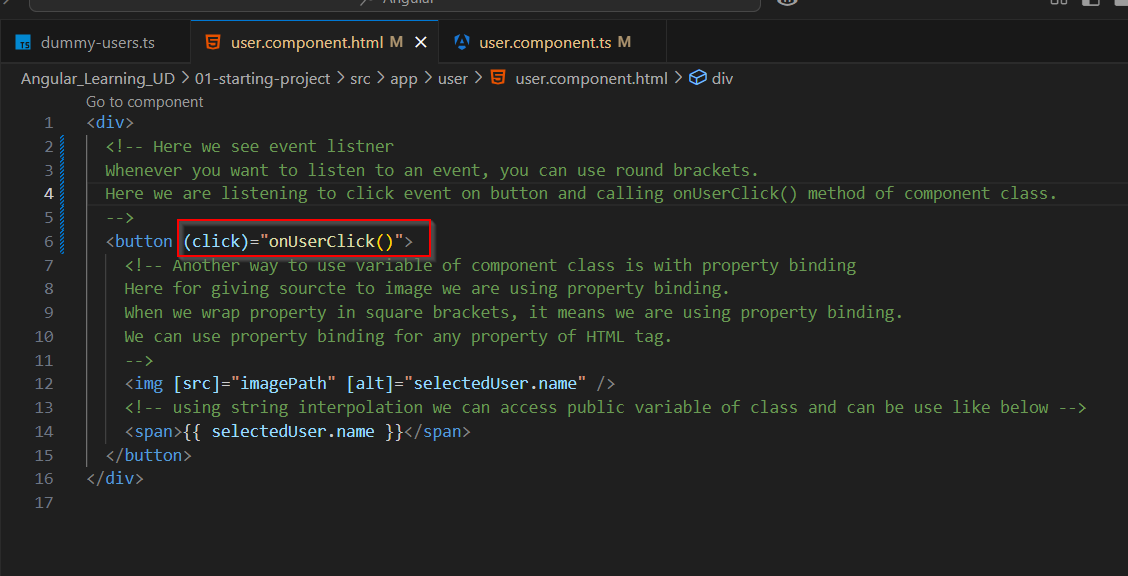


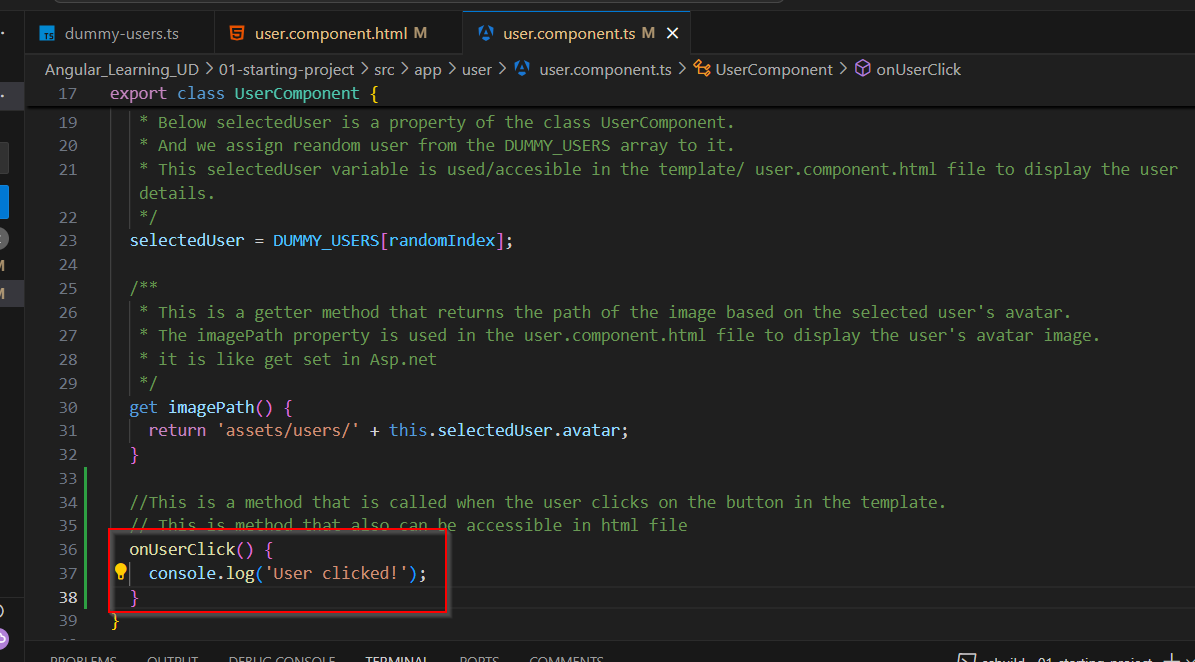
* Anther feature of TypeScript or JavaSCript we have inside class is getter or setter. We can simply use getter to get value of that property after assigned it. As shown in below image, now this “imagePath” variable is accessible in user.component.html file.



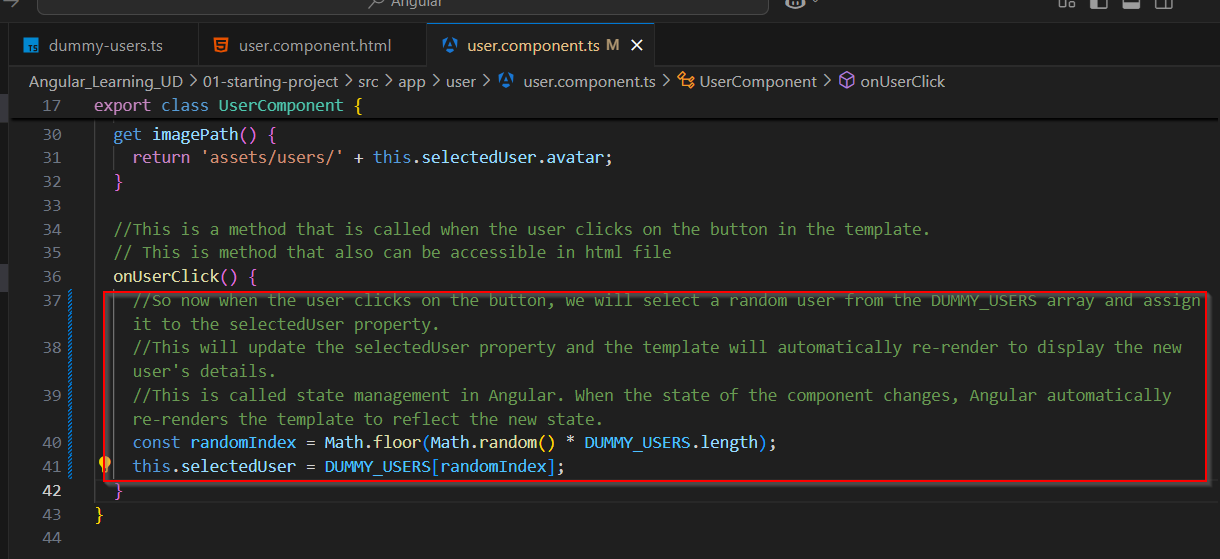


* We can define event listener in html element using round bracket, as shown in below image, Here we calling function that is written in component class file when user click on button and that’s why it is (click) event listener.

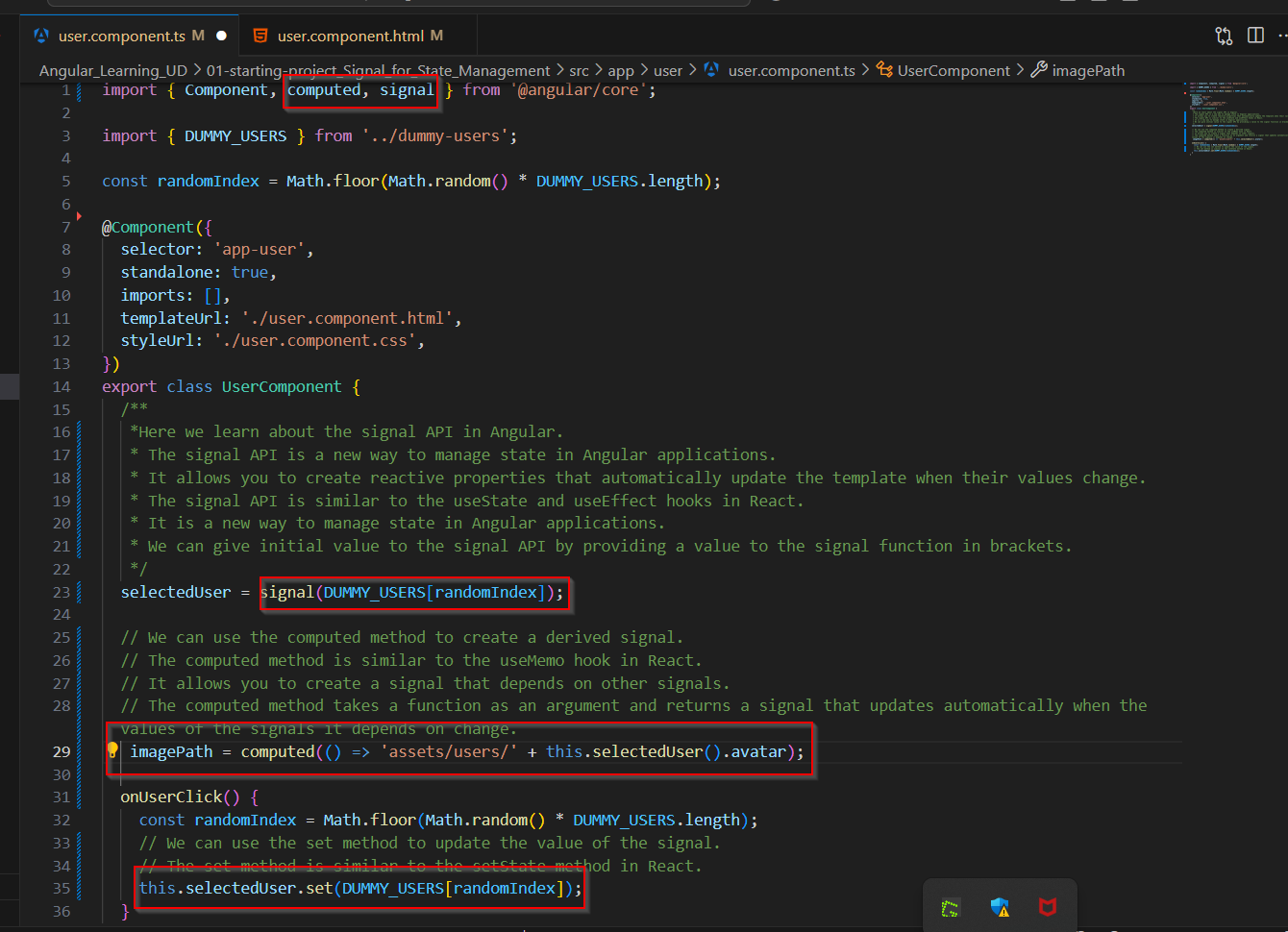




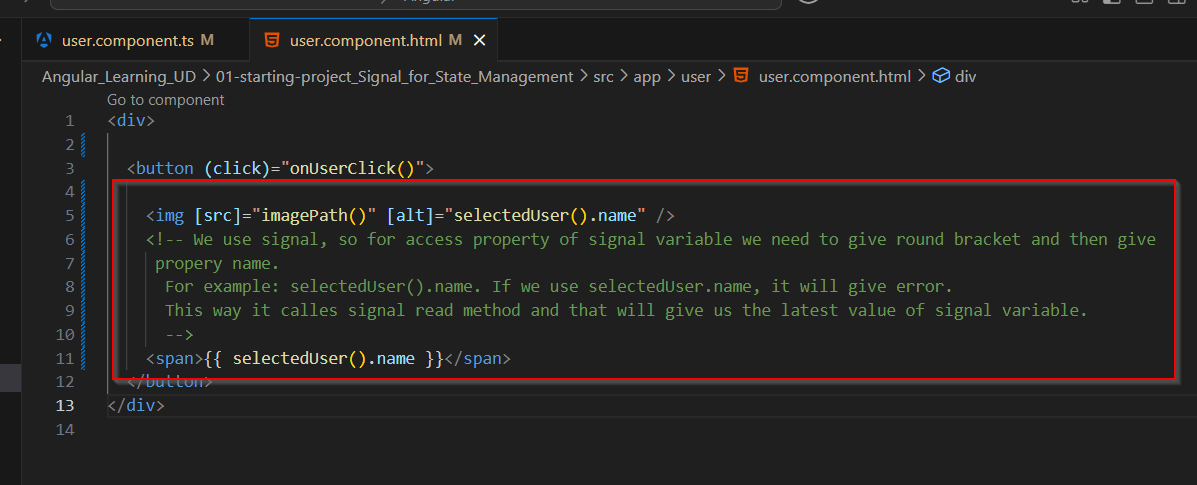
* Another concept in Angular is state management. As mentioned below in image.



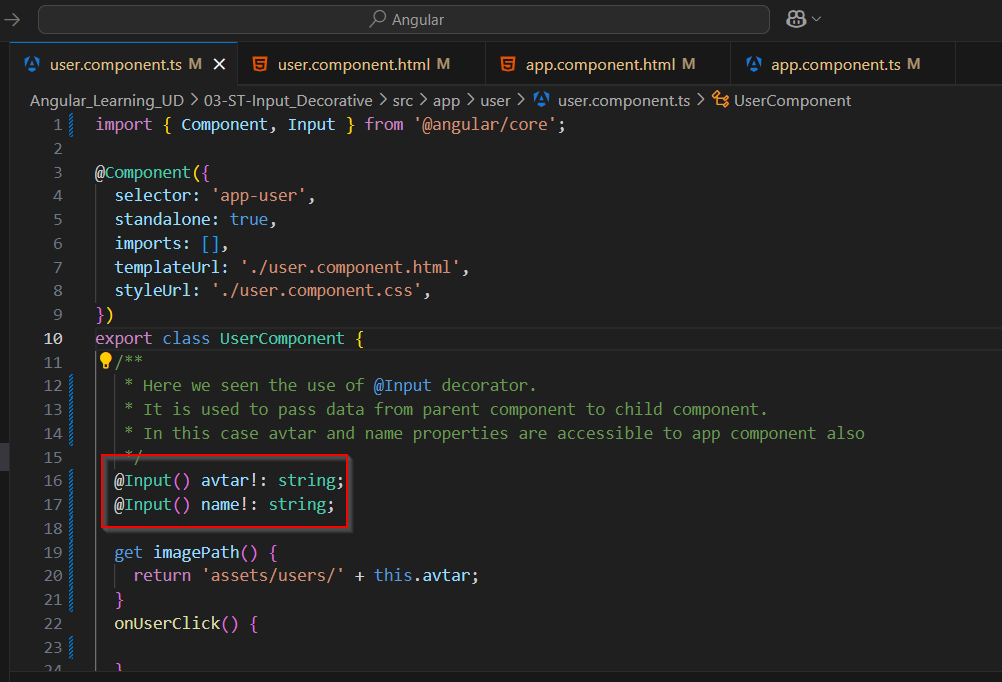
* Internally zone.js does this state management thing. It check which components property is changed and re-render only that component.
* Another concept of state management is using signal that is available in Angular version 16 and higher. This is like useState hook in react. All comments mentioned below in image.

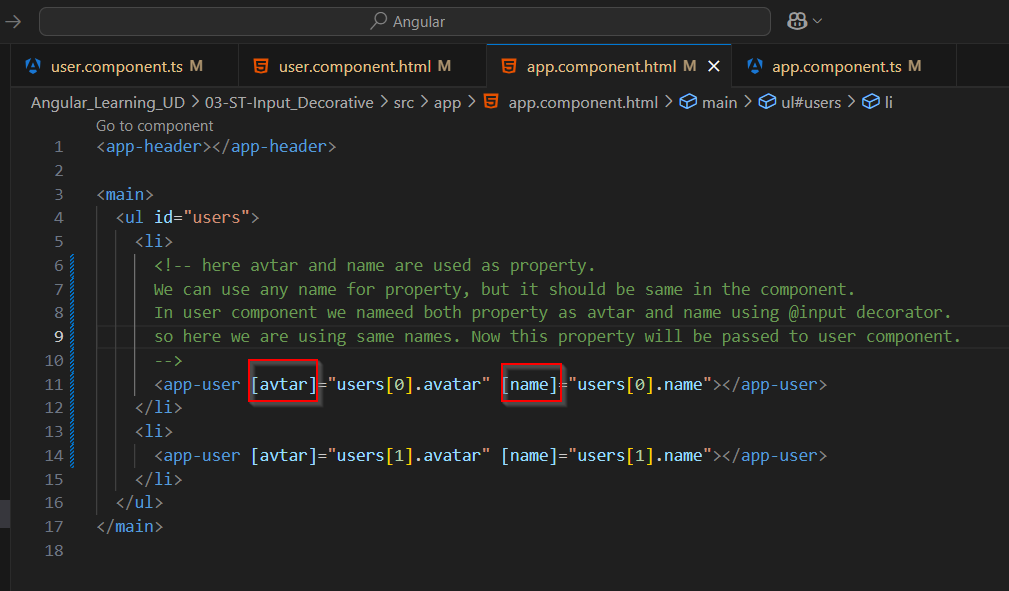


* With signal use for accessing property in html file need to use bracket as mentioned below in image.

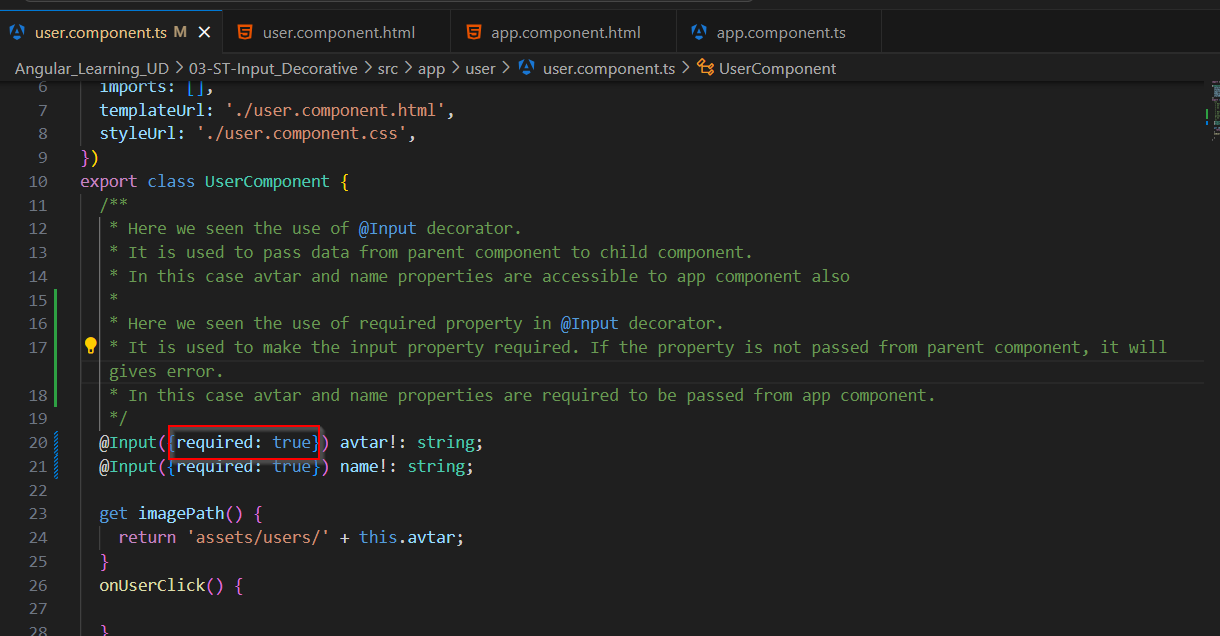


* We seen the use of @input decorator. With this decorator once we define property in component class that can be accessible in html element of it’s own as property and with that we can pass value to it’s actual/ replicable html file.



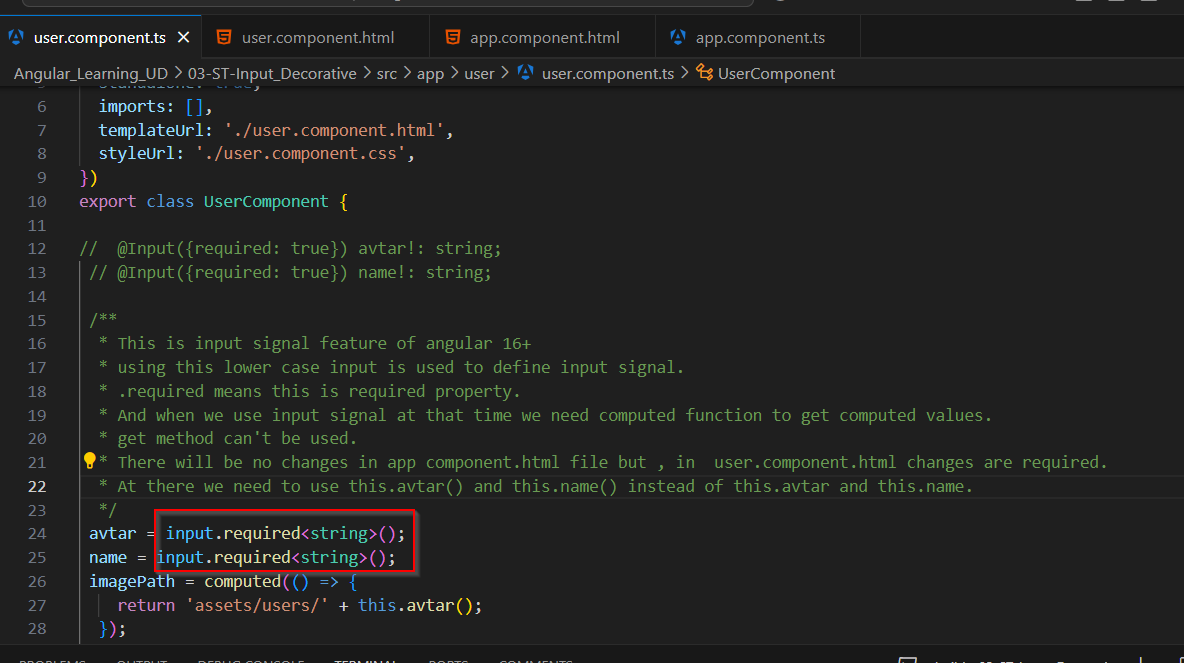


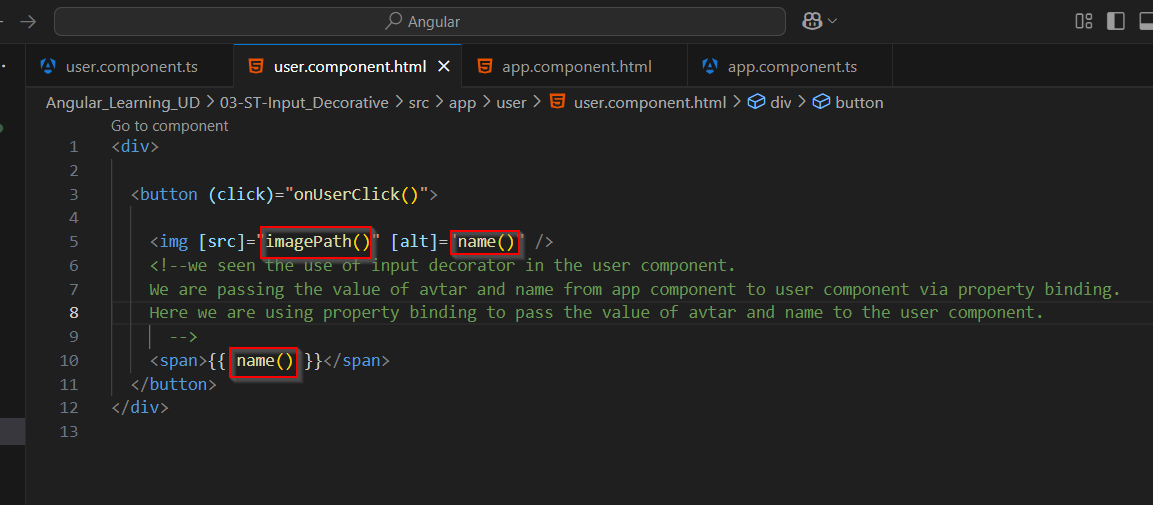




**Input Signal**

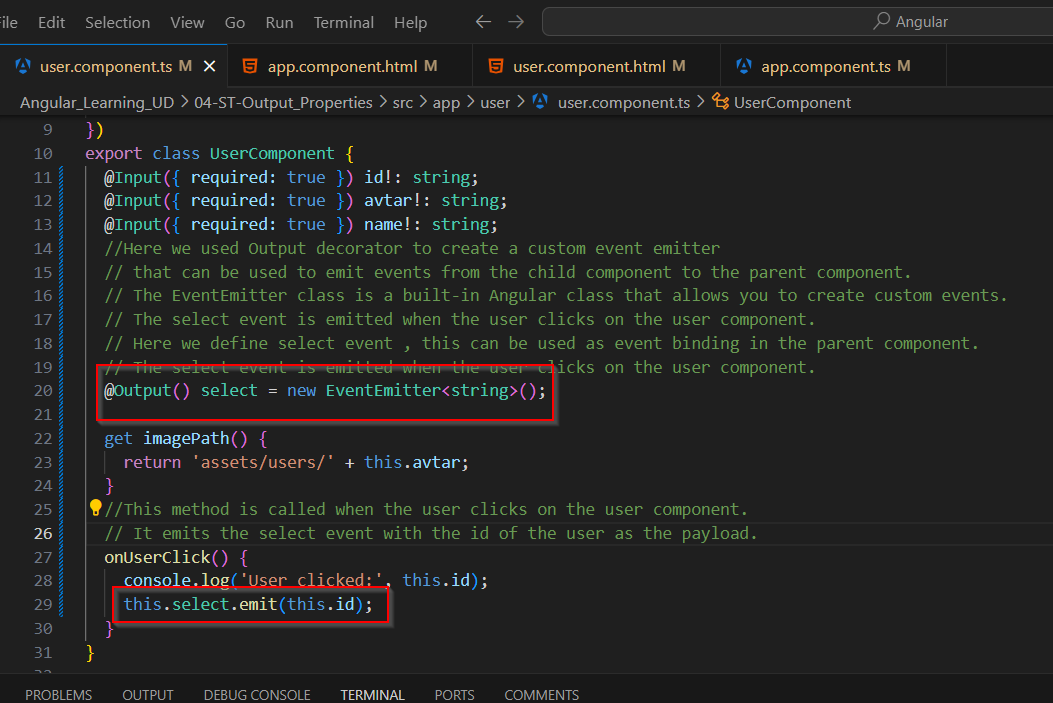
* There is another option to access property value from class to compoenent html file is using input signal. This feature available in Angular 16 and onwards version.

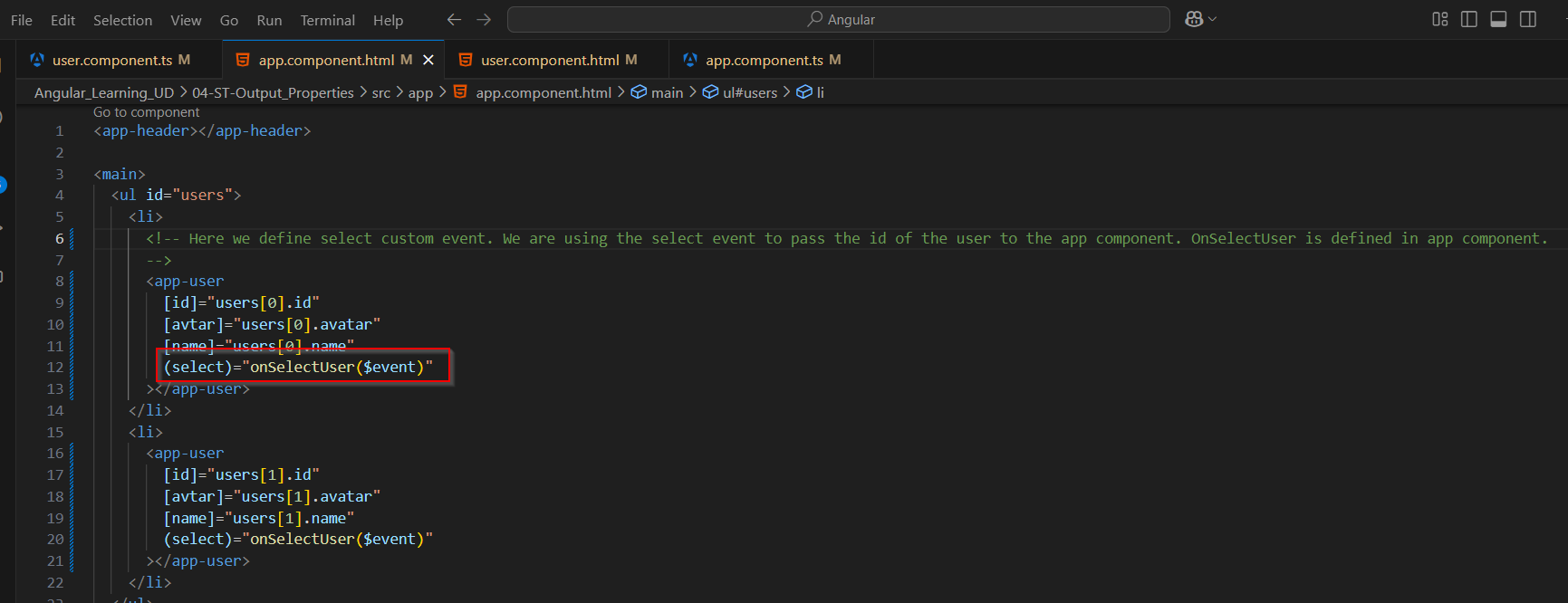


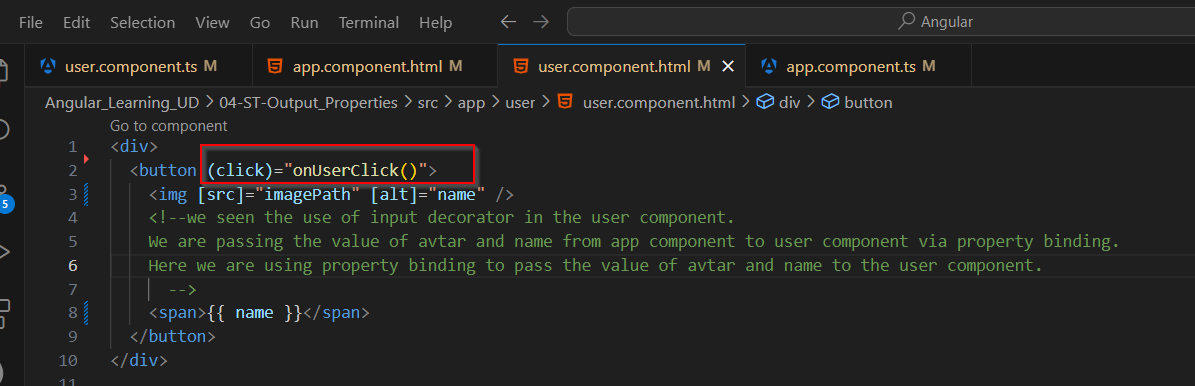


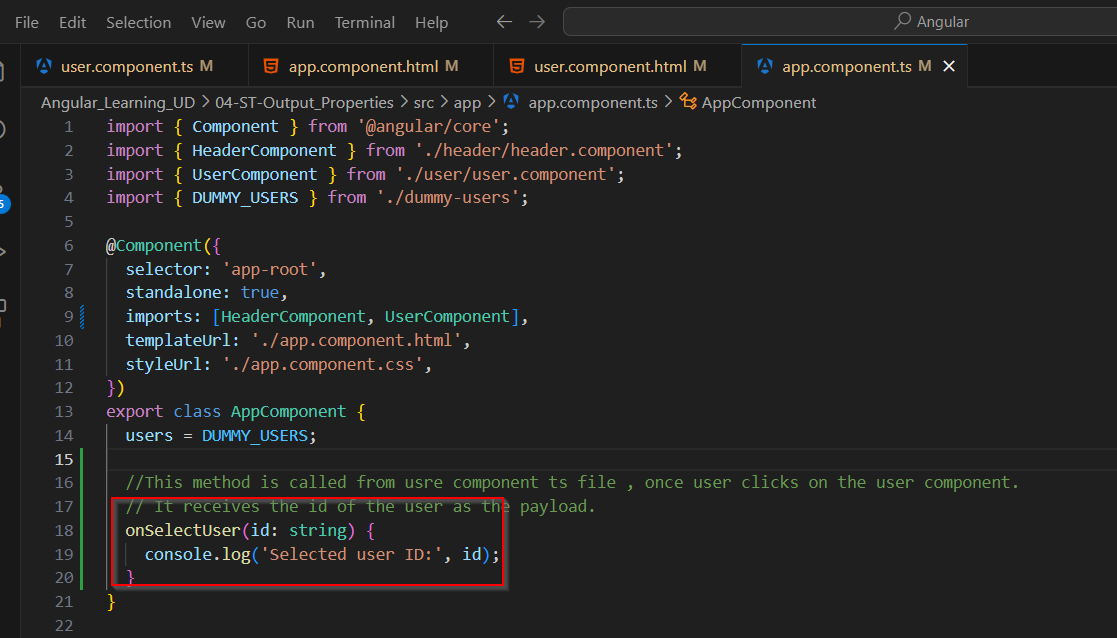
**Output decorator for custom events**

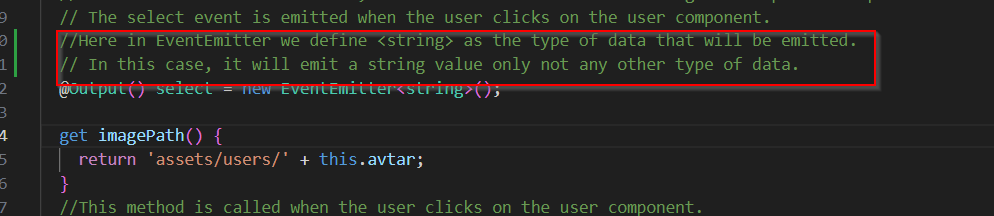
* We learned new concept @Output decorative to define custom events.





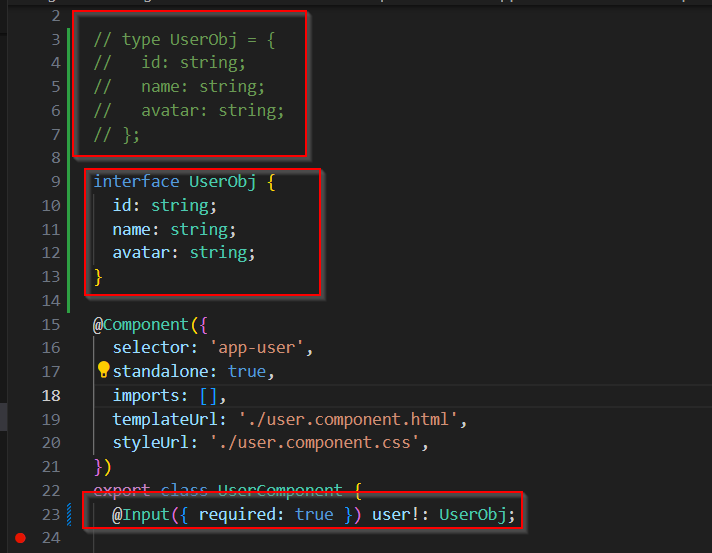






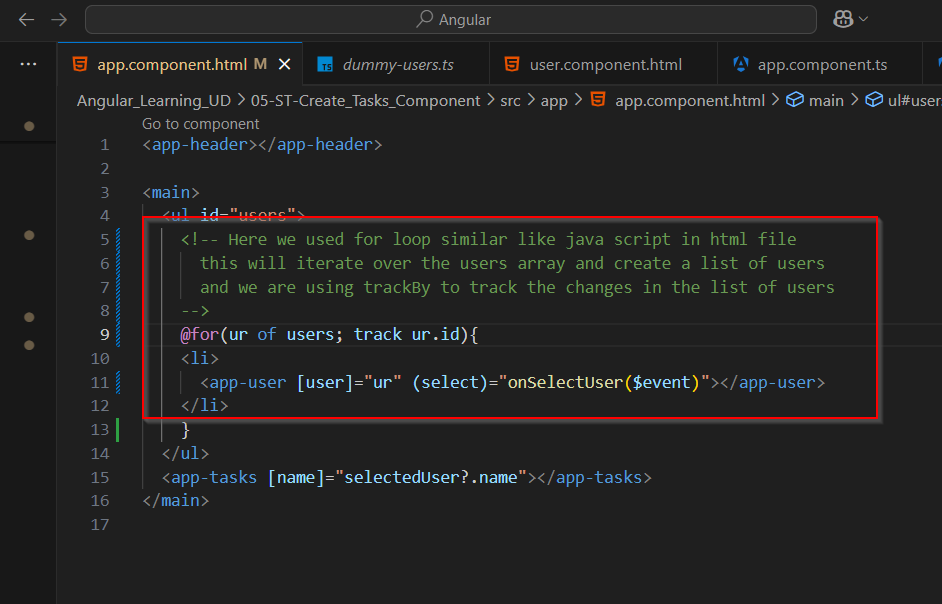
**Object Type**

* Instead of individual property we can define object type as well , as shown below. We can directly define object properties while define object variable.
* But type or interface is another typescript feature to define types of inside properties of object.



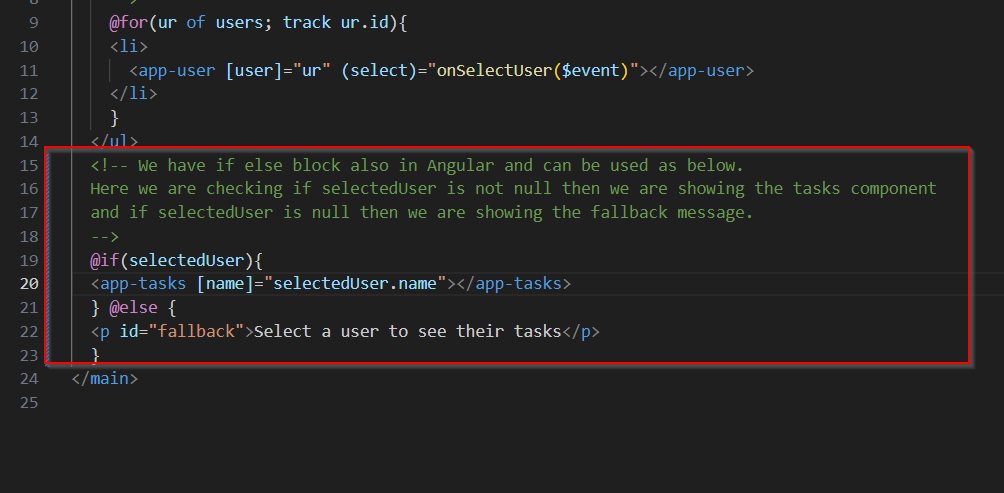
**For loop inside html file**

* We can use the for loop inside html file similar like java script syntax and use is as below.



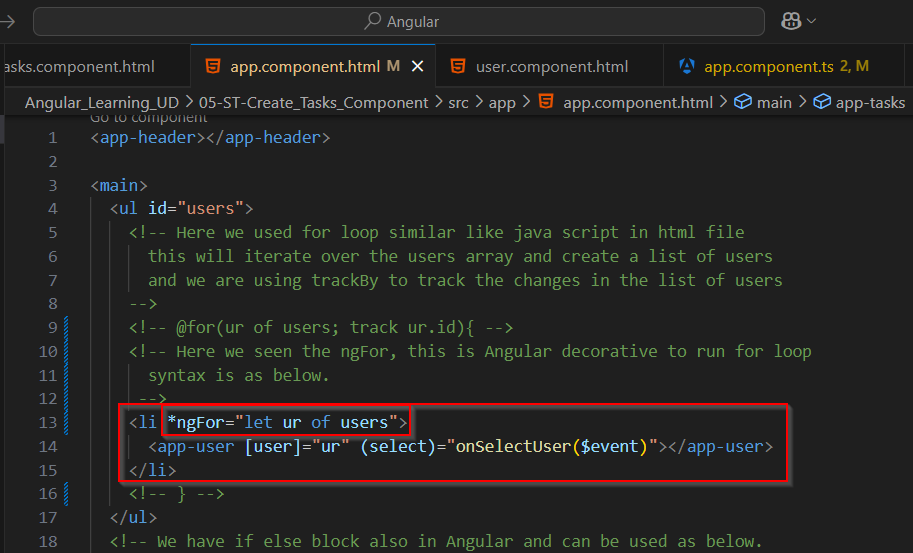
**If else block**

* For conditionally rendering data we have if else block also in angular. We can use it as below.



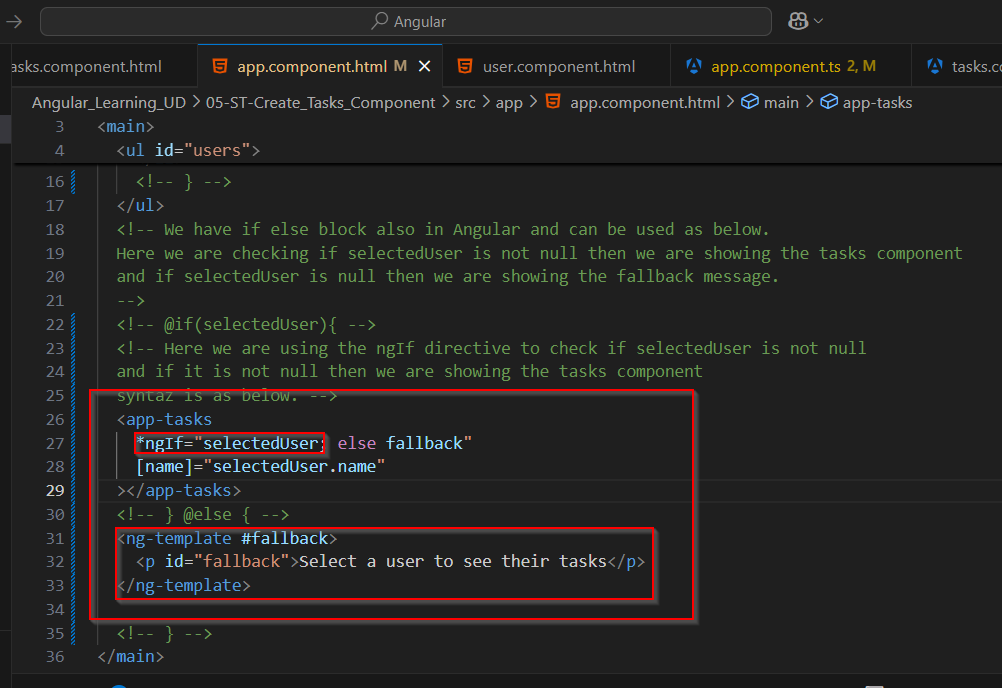
**ngFor**

* Normal above for we used is for Angular 17+ version. In older Angular projects we find ngFor directives for use of For Loop. Syntax is as below.



**ngIf**

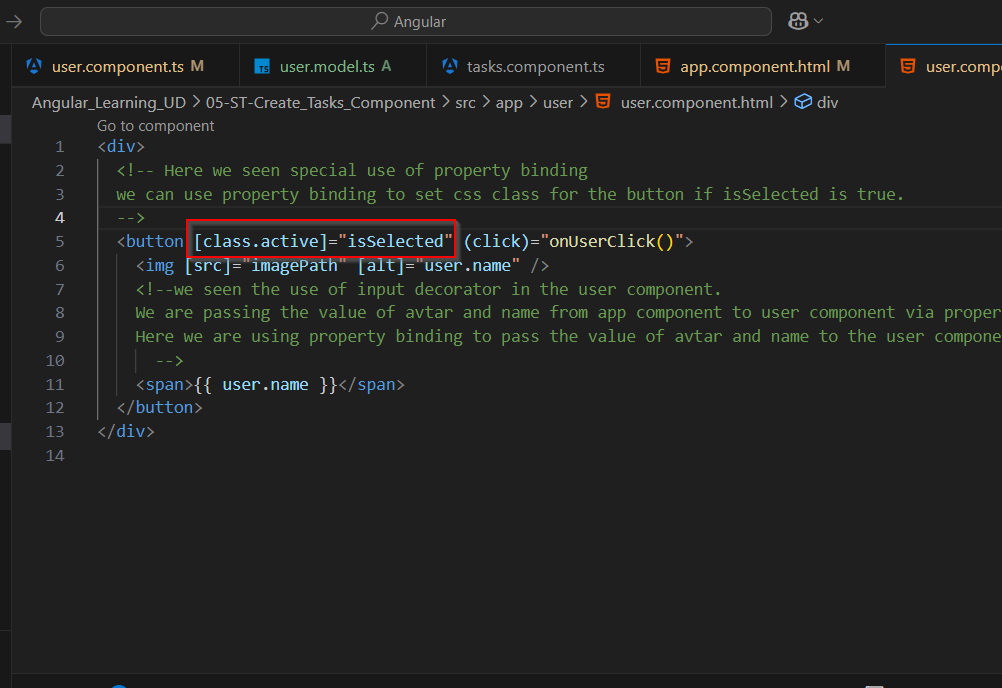
* Above if is usefull for Angular 17+ version. In older Angular projects this feature is not available and for this, ngIf directive is used. Syntax is as below.



**Conditionally set css class**

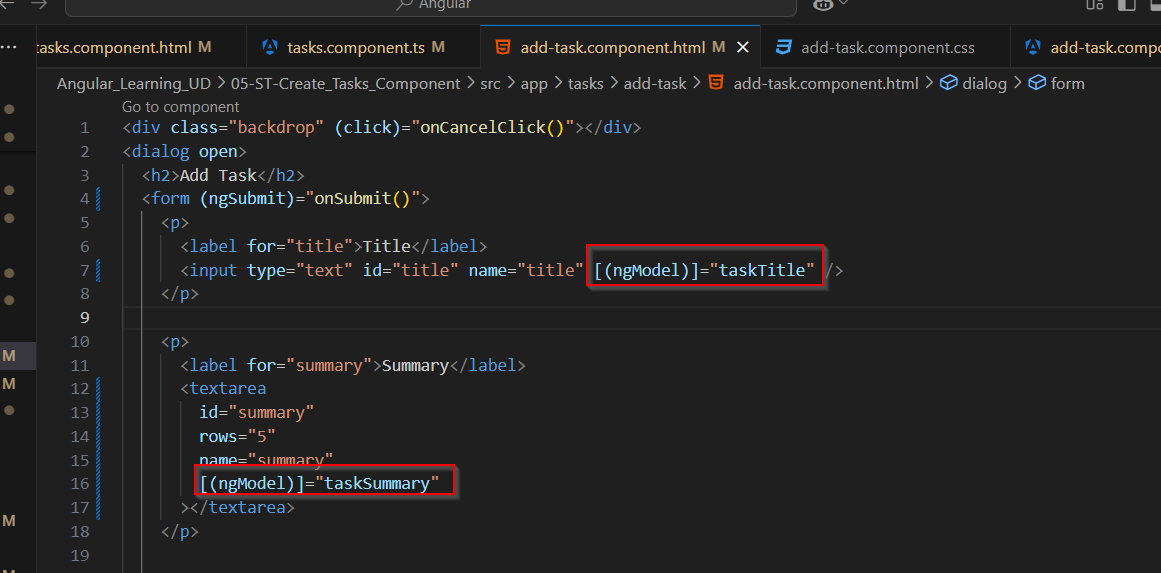
* We can set logic true , false according to our code. Ans set class as below

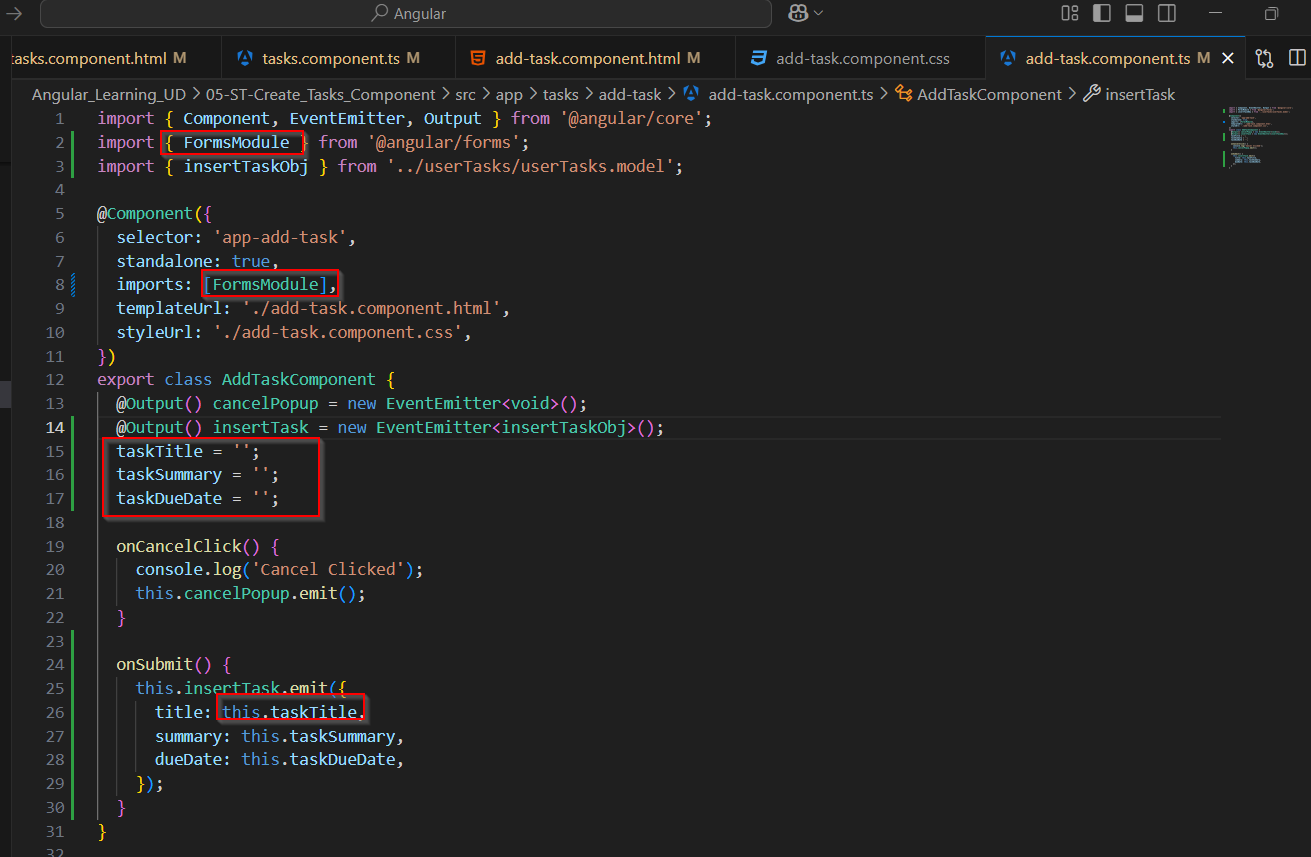
Here if isSelected is true active class will be set to button.



**ngModel directive**

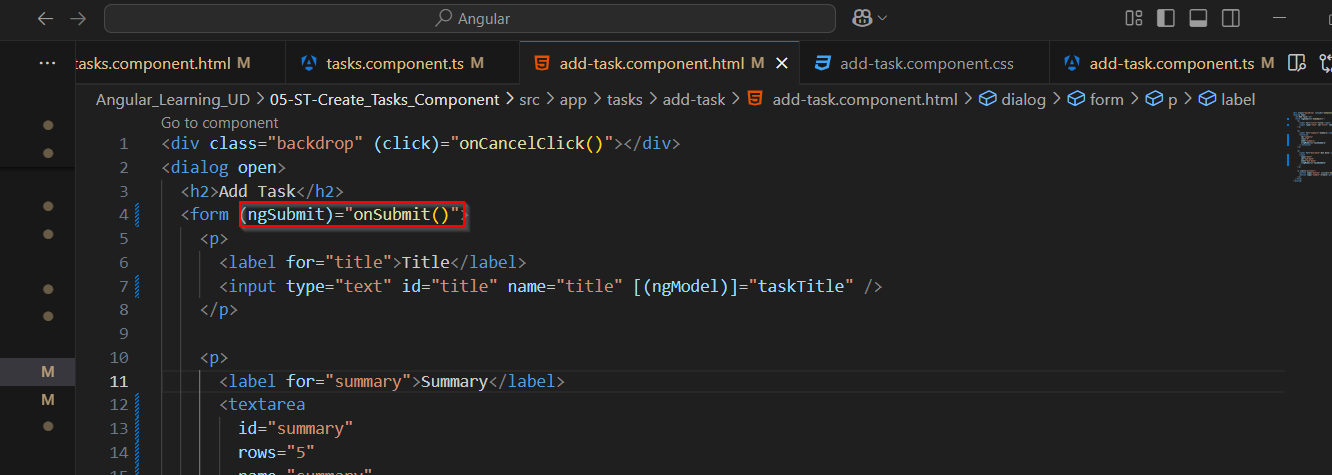
* We have ngModel directive in Angular, this directive is use for two way property binding. This is used with any input element and need to import FormsModule in component class file.
* This is useful like when if user type anything in input element that also store in defined property, if remove any thing from input element it automatically remove from property define with this.
* Here in this case when user enter anything in task title input , this will get store to taskTitle property also.
* Later on this variables taskTile, taskSummary use for further actions, just like submit forms data.





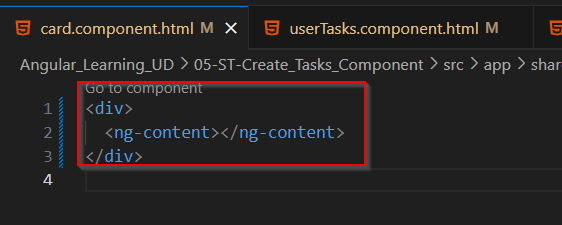
**ngSubmit event listener**

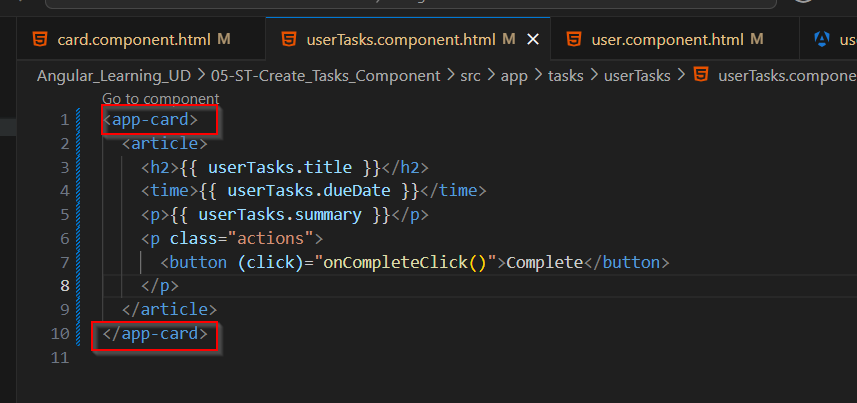
* This ngSubmit event listener is angular event listener , this is used with form element.
* So when any user clicks on submit button , it prevent default behaviour of forms submission to send request to server.
* And then we can call any method and handle it here.

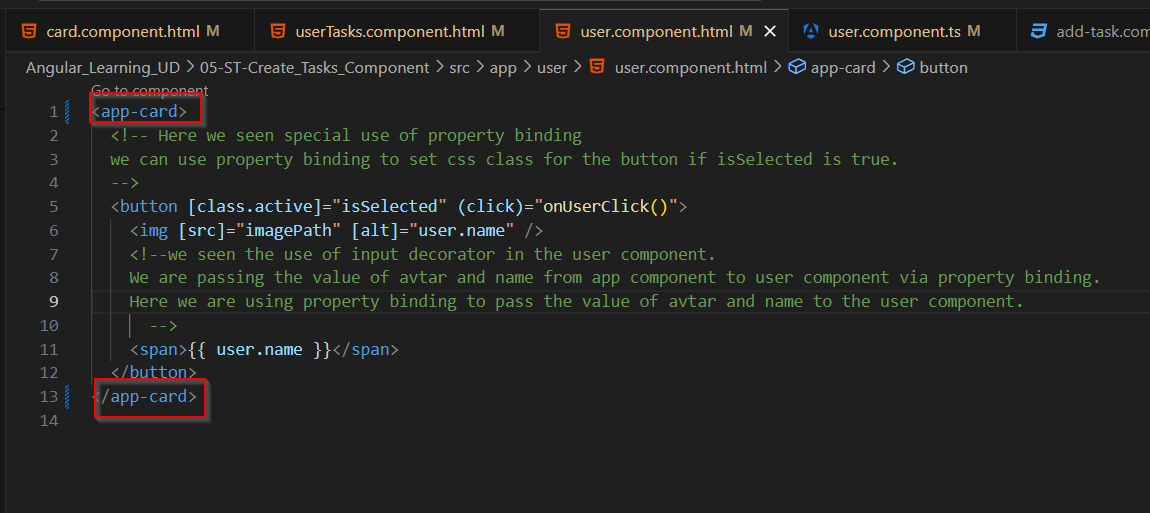


**ngContent**

* ngContent is a Angular element/tag. That can replace the child component content.
* This used for content projection.
* In this case ngContent replace the user component data and userTasks html elements.
* So inside that div element of card component, user component and userTasks component data will show.
* This can be used like container.

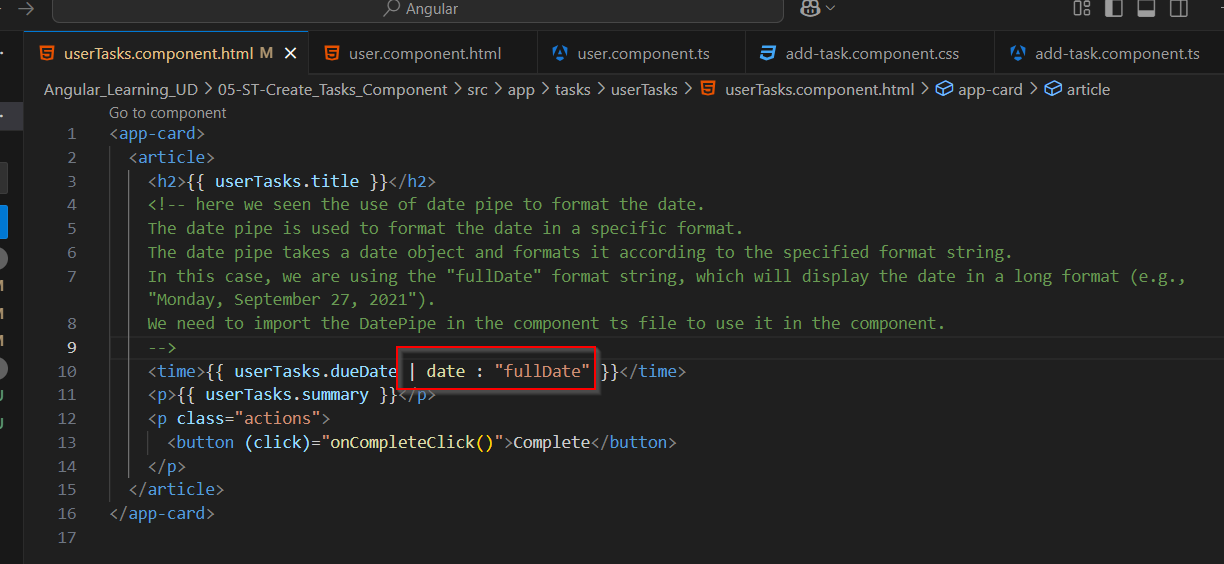


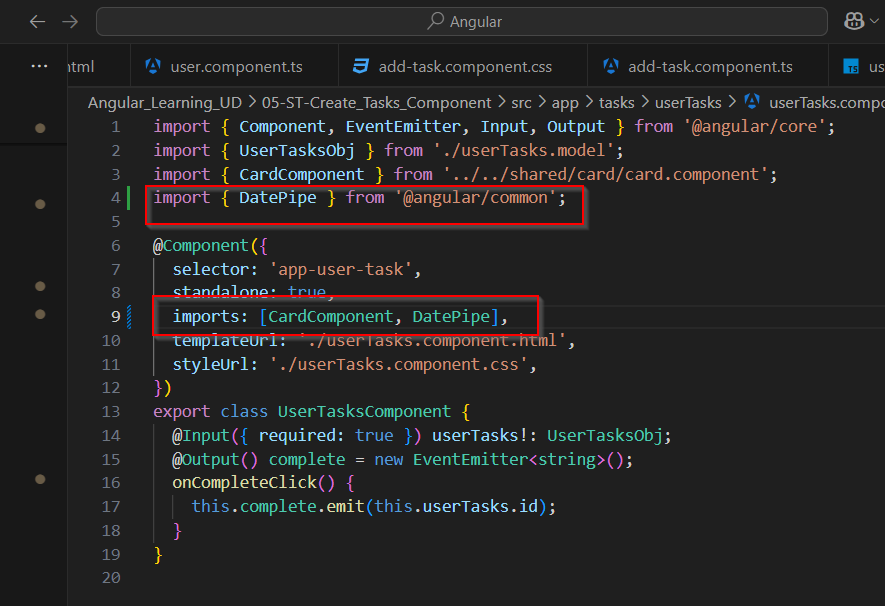




**Date pipe**

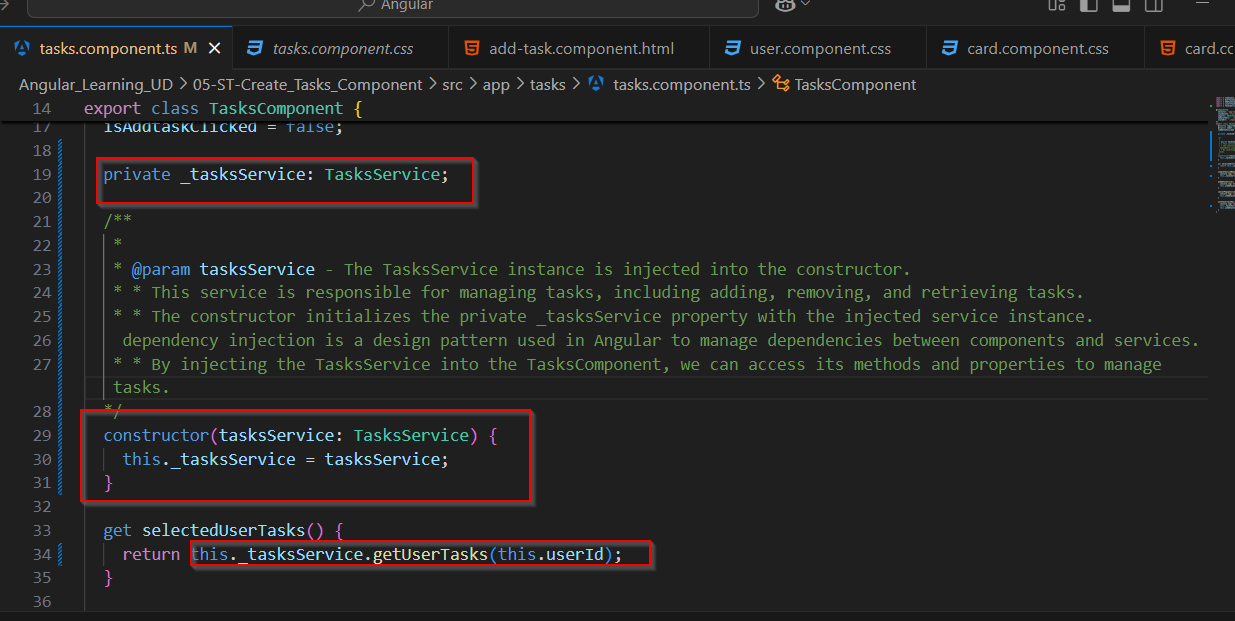
* In Angular there are many pipes are defined. One of the pipe is date pipe.
* With this date pipe we can format the date. Syntax and use is as below. We need to import that in component ts file.





**Dependancy Injection and Service**

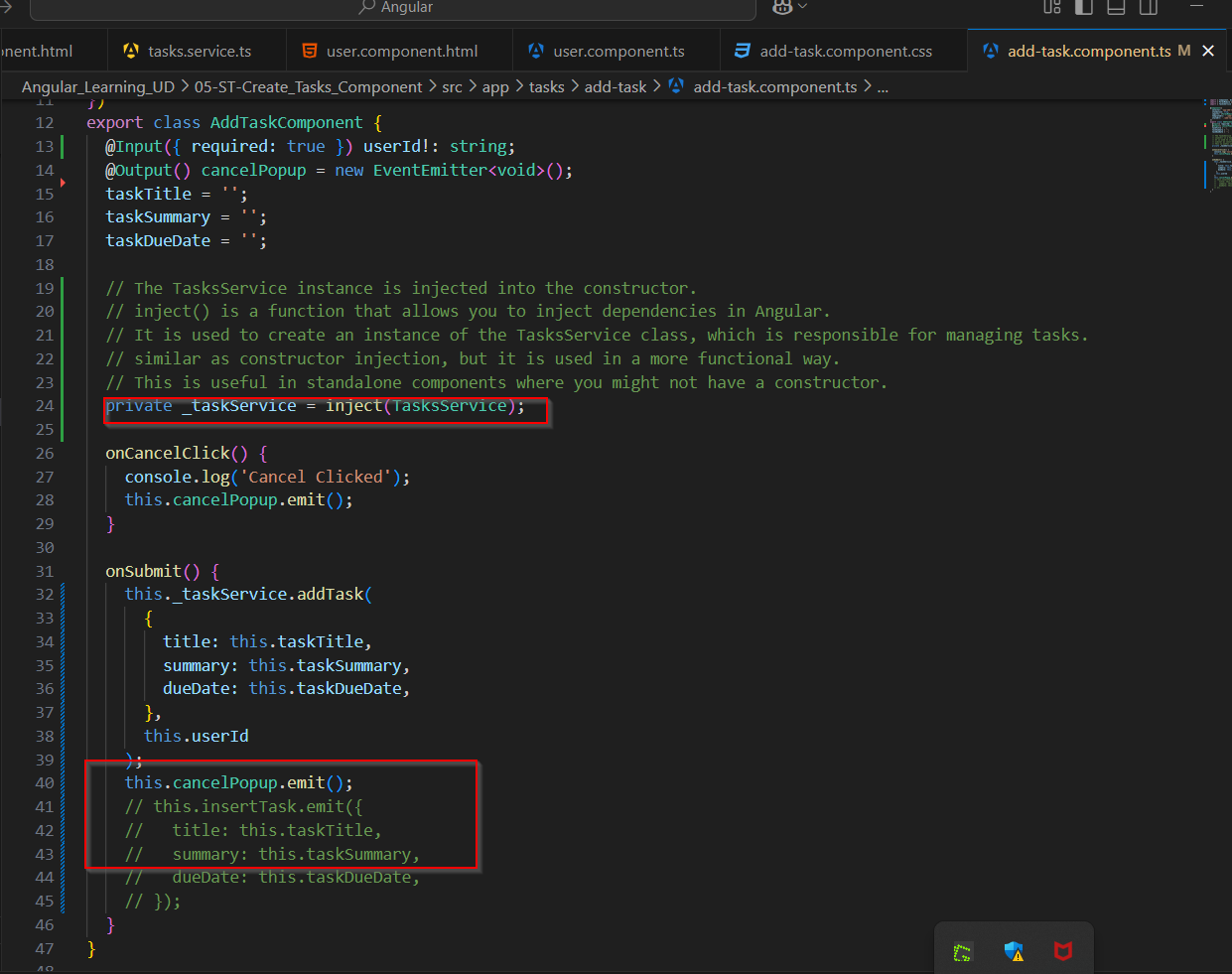
* Here we seen another two concept of Angular, first one is Service and Second is dependency injection.
* Service is simply a class to separate the business logic related to data, such as save, delete, get.
* Dependancy injection is , if we initiate object of class/service and if that service/class used more than one place , it creates multiple copy of that class object.
* And all these copies are different , means one object made some changes and have updated property value the other object completely unaware with this.
* Due to this instead of initiare manully object of that class/service use dependency injection to initiate the class/service.
* This way all place were this initiate using DI , have one common object.
* Syntax and use is as below.





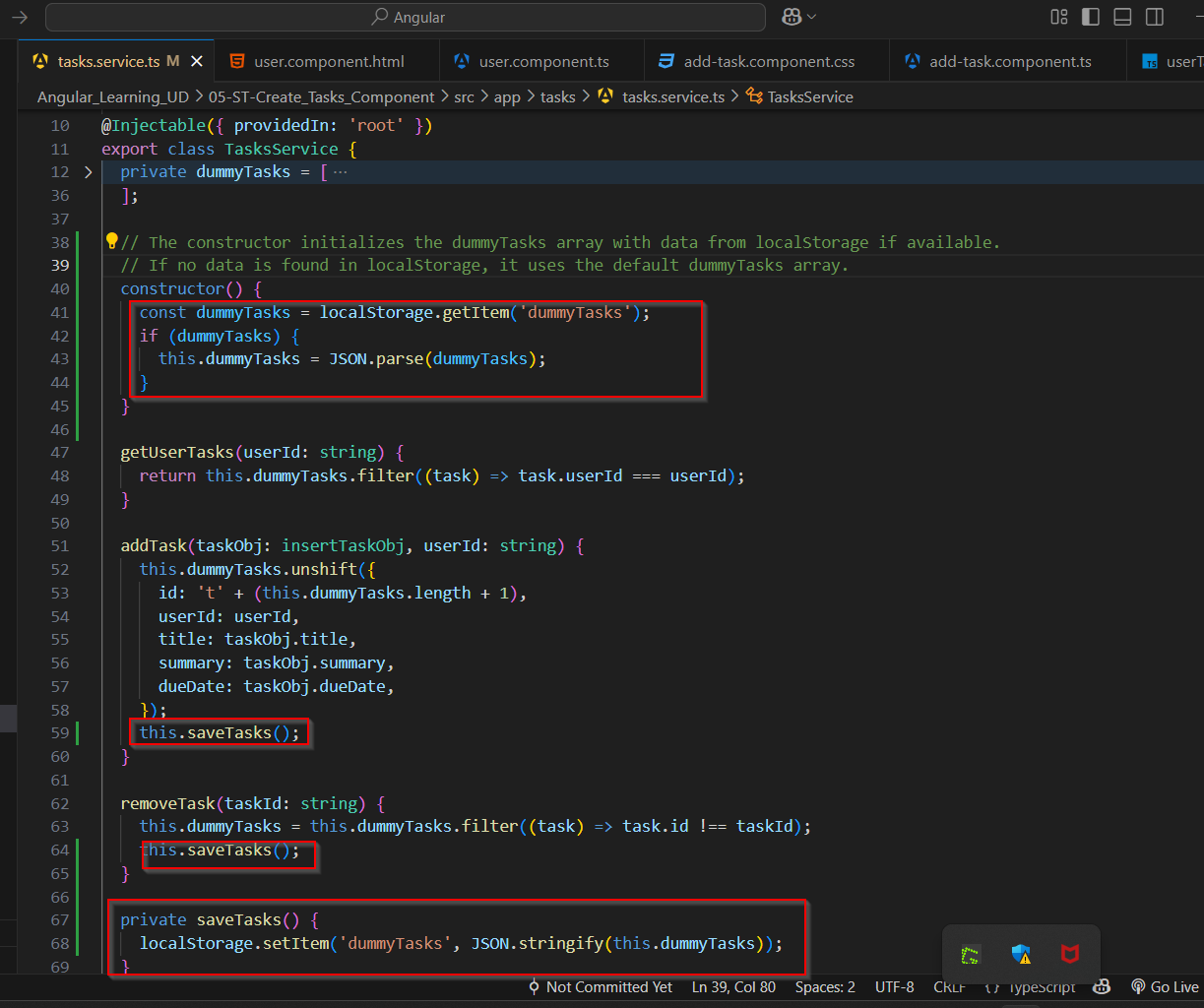
**Inject() function for DI**

* Another way to do DI is with inject() function of Angular, this behaves same as constructor DI.
* This time we used our service in Add task component ts file and due to this removed addTask event emitter. Diretly called service saveTask method and save data. Made other changes related to this.



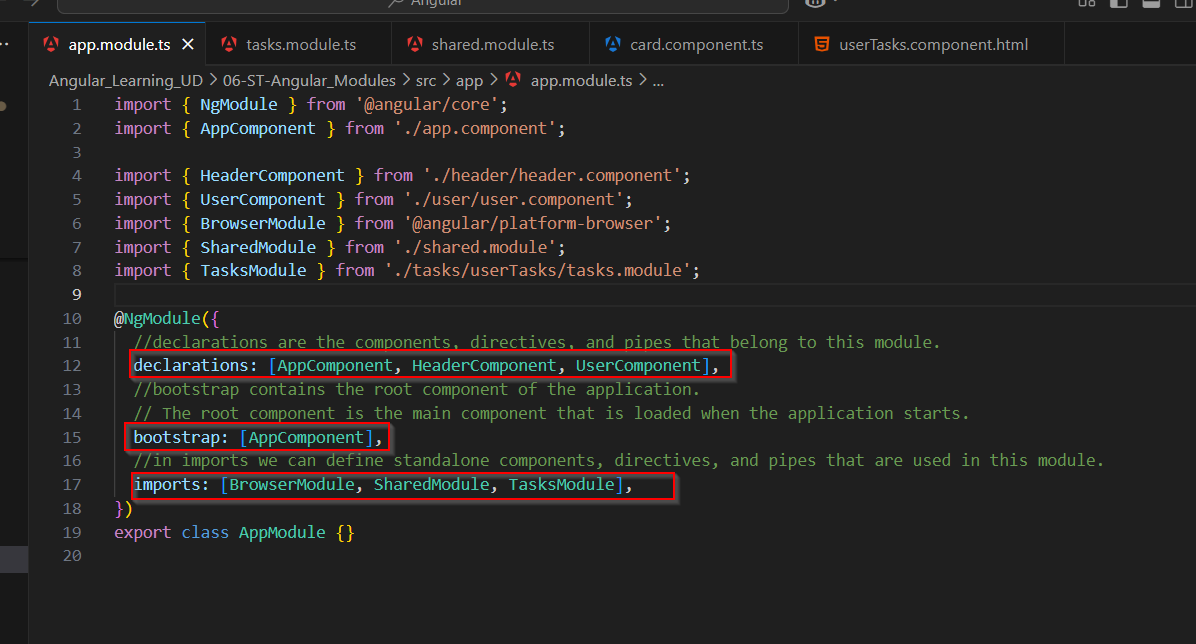
**Localstorage to save data**

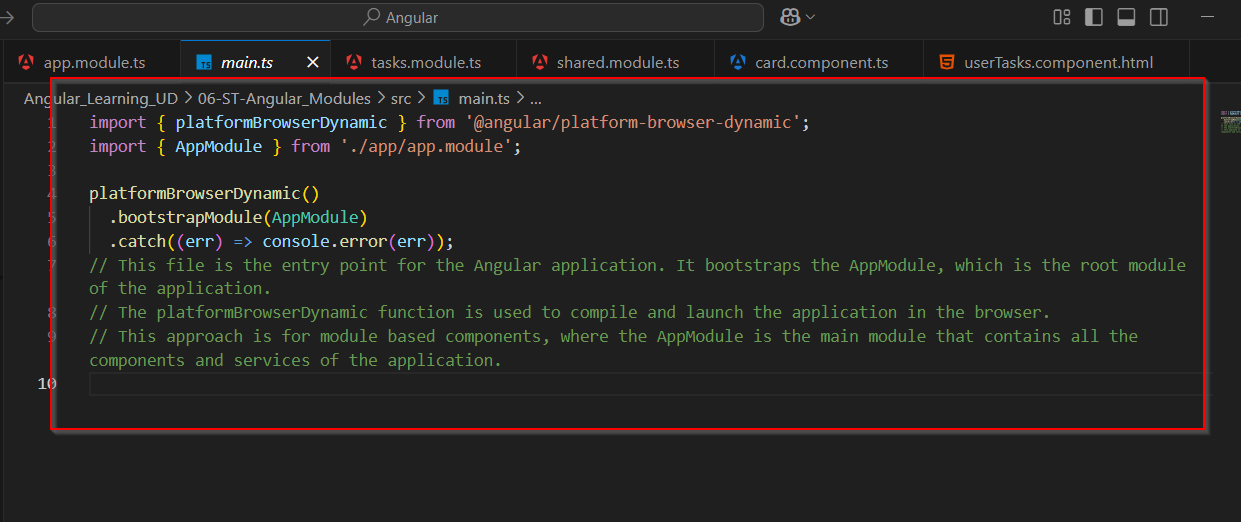
* Another concept we learned is store data to local storage. Using getItem and setItem we can set data and get data.
* Here in this we store data in local storage on adding a task or removing a task.
* And if data exists in local storage with that particular key, we populate that data using constructor.

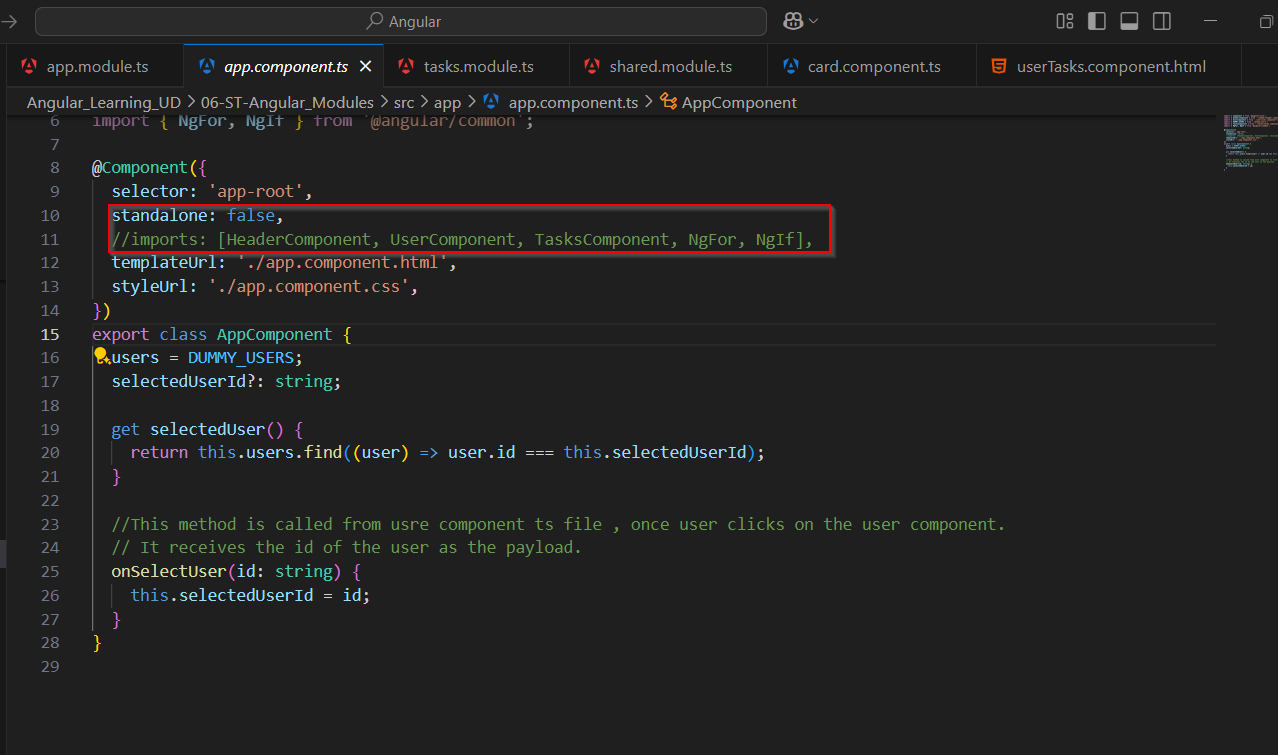


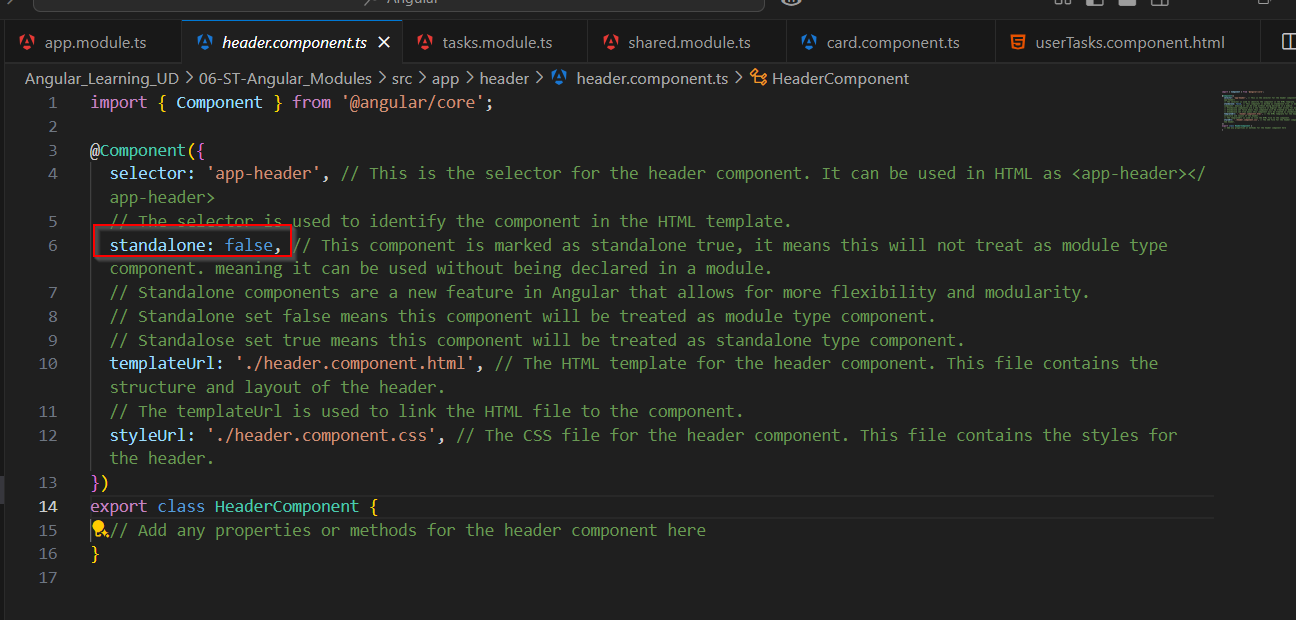
**ngModule**

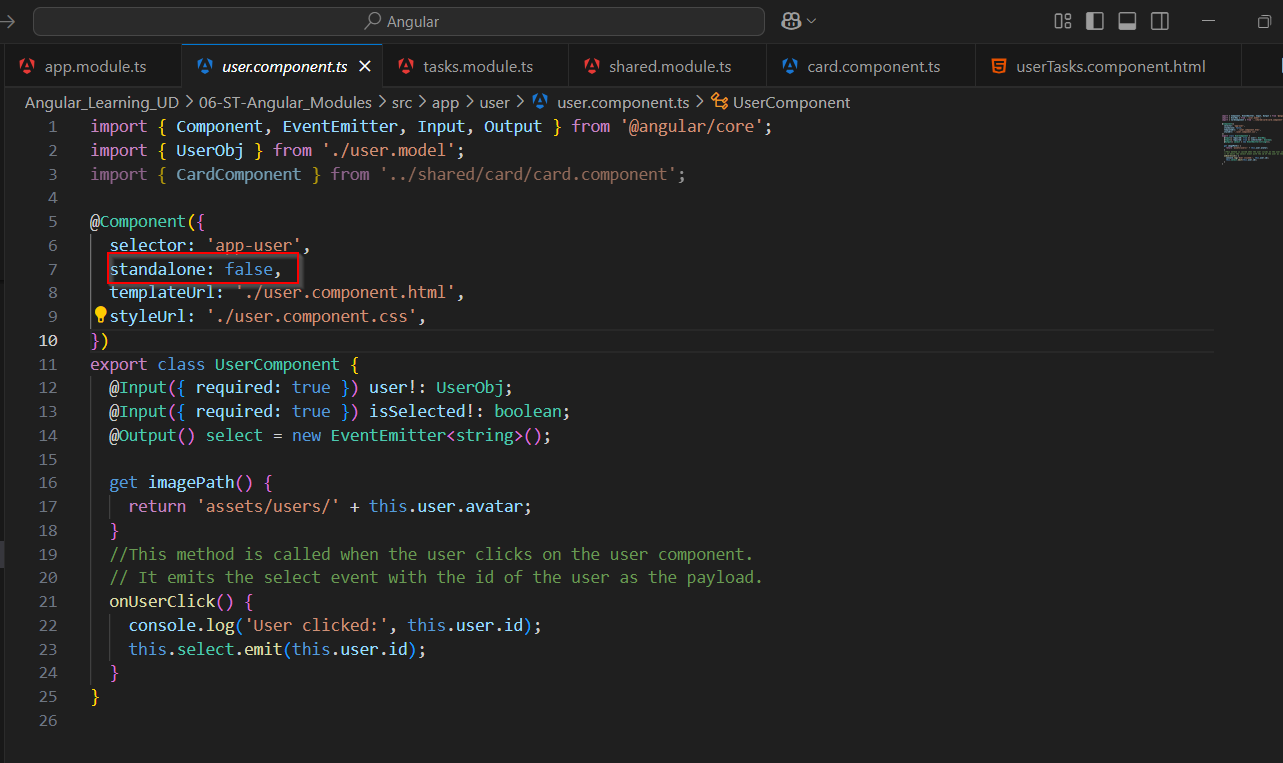
* ngModule is another concept of Angular.
* Generally nowdays in morden version of Angular, stand alone component is used.
* But in older projects or still module based components are used.
* In this module based component first we have one root module, e.g app.module.ts
* In which we can declare all components that are talk together, or entire application components also.
* Or we can define shared modules also, for example, for Card component we have one shared module , that will get imported in app module.
* For all task related component we have one tasksmodule that imported to app module and so on.
* For achive this module based concept we need to make change in main.ts file also.
* This is the file from where angular project run/render. So now instead of first render App component , we need to render app module.
* Also in app module we need to define one root component also. That can be done using bootstrap configuration.
* All examples are as below, we converted our starting project entire from stand alone component to module based component.

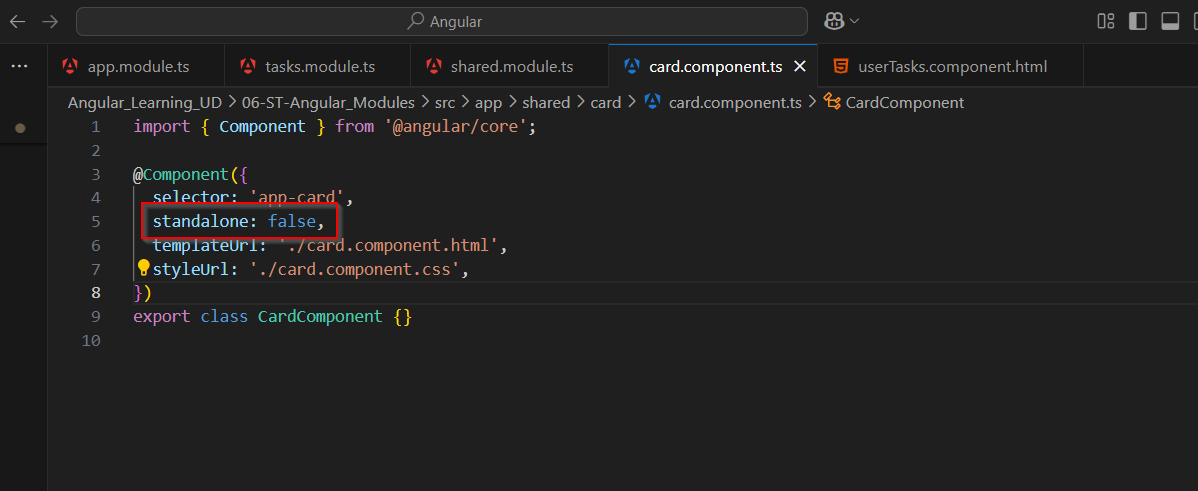


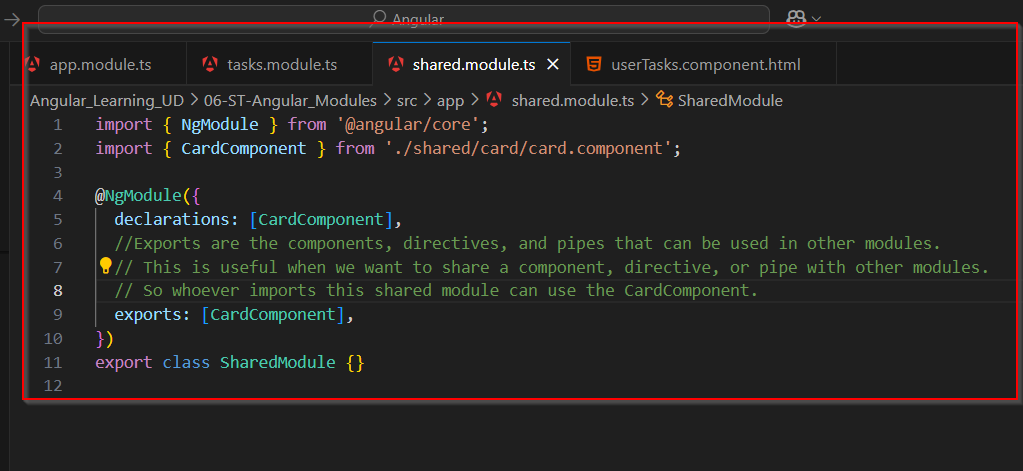


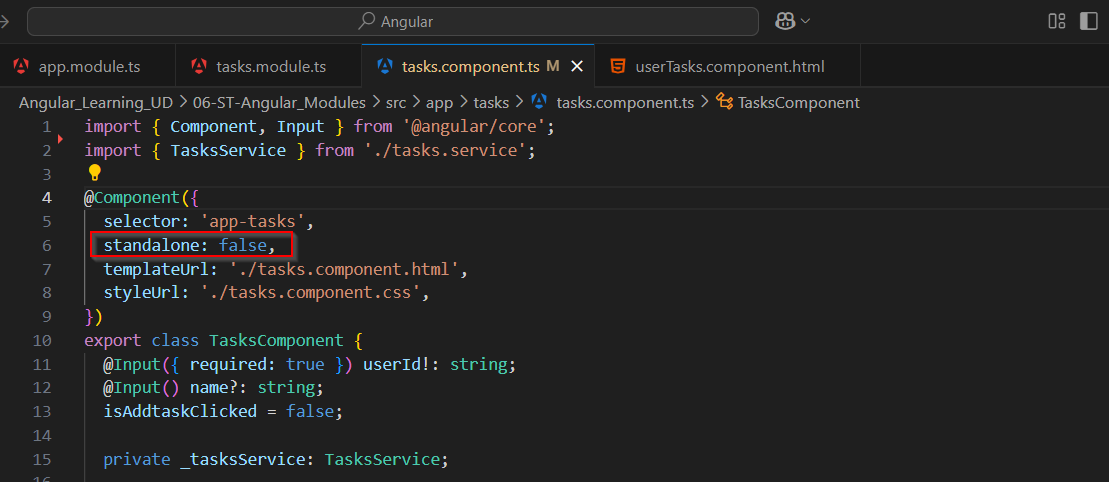


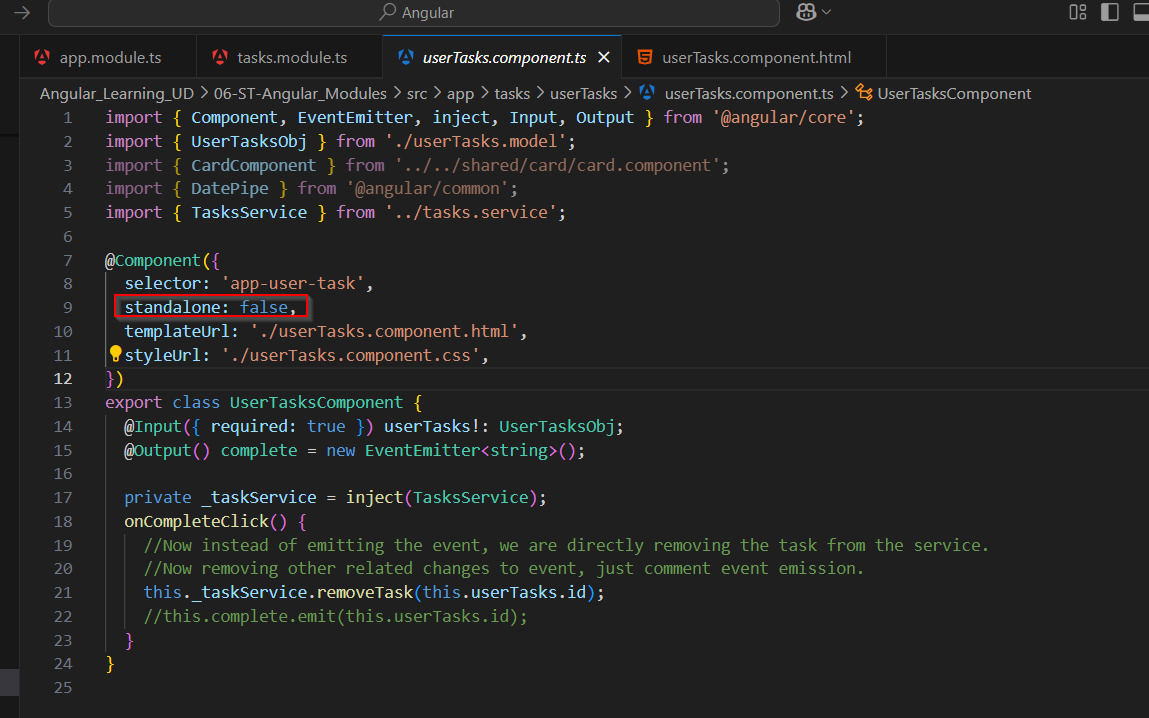


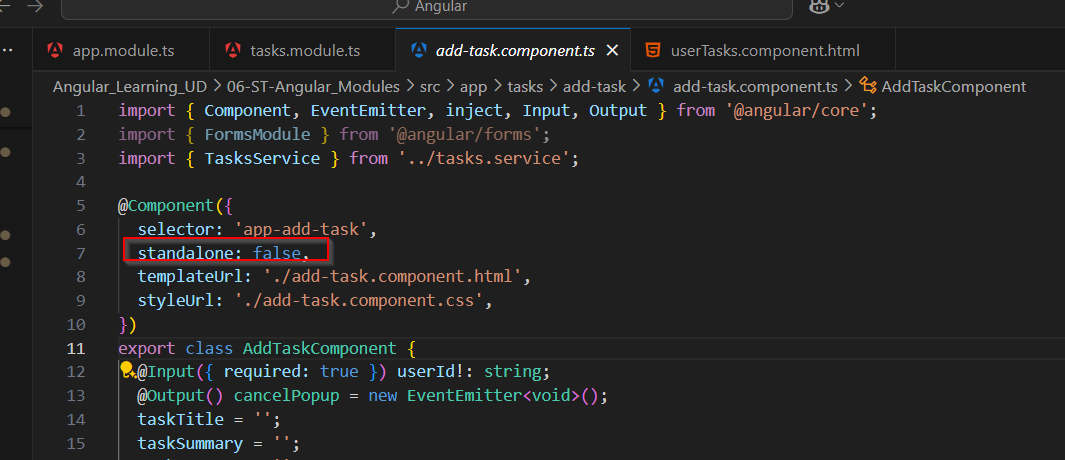


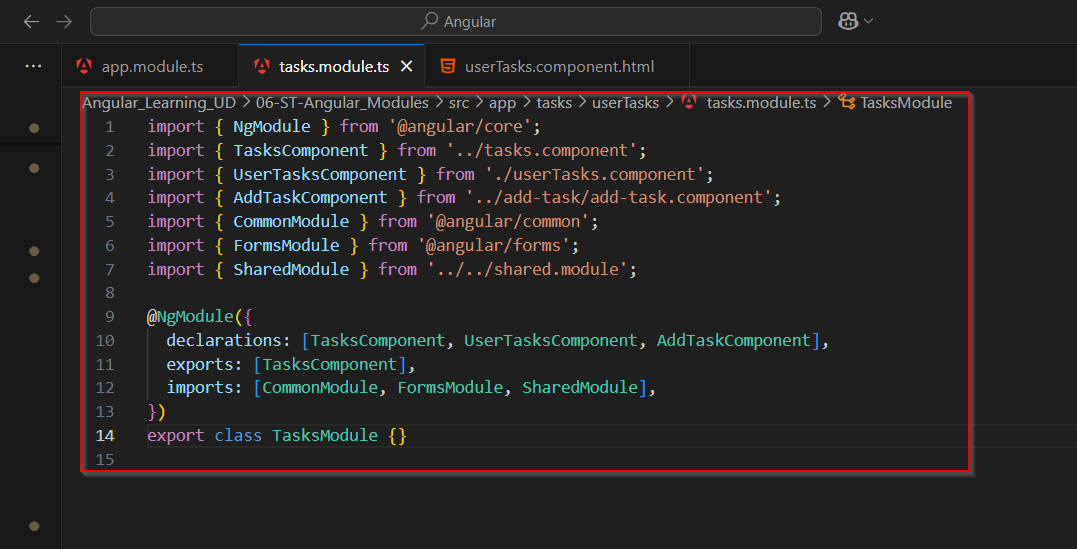






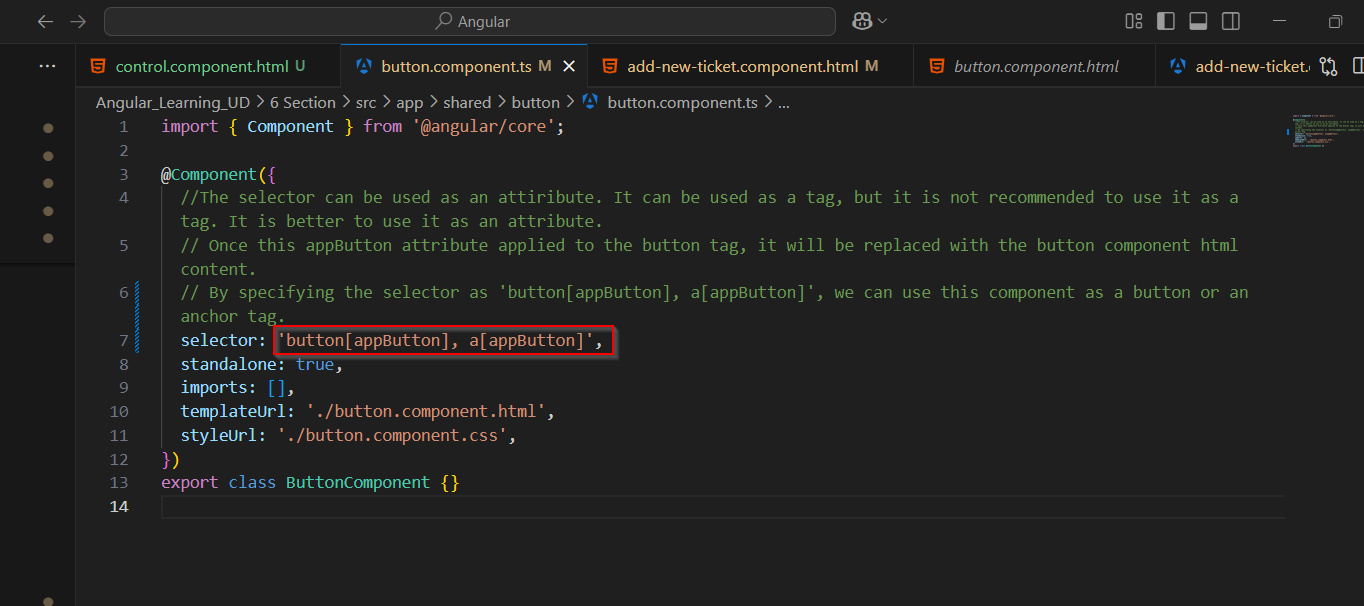


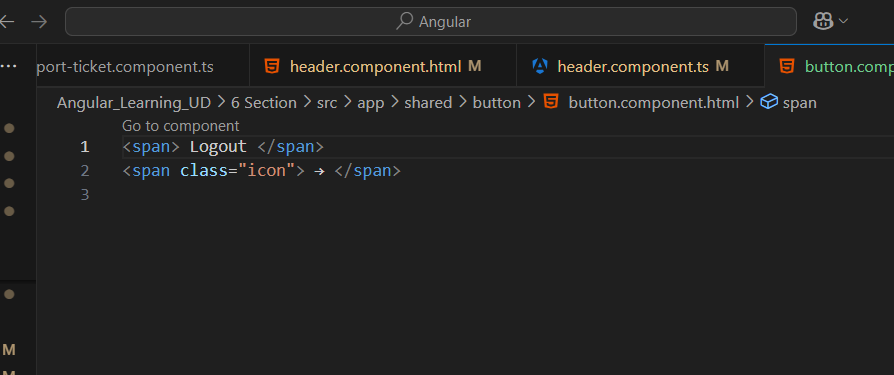




**attributeSelector**

* We seen here another concept of attribute in selector property of @component directive.
* When select define as an attribute , as did below way, whereever we want to use it we need to define it as attribute and not as an html element.
* Once it used as attribute it will replace content inside that tag with this custom component html thig.
* Here in example in button tag once appButton attribute used, all html thing will replaced in between button tag with html content of this custom button component.

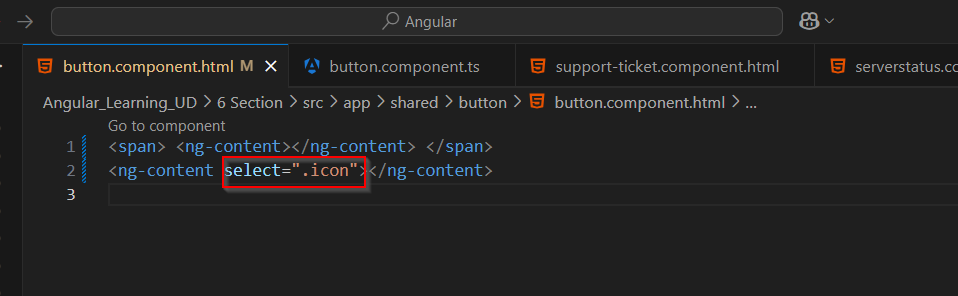


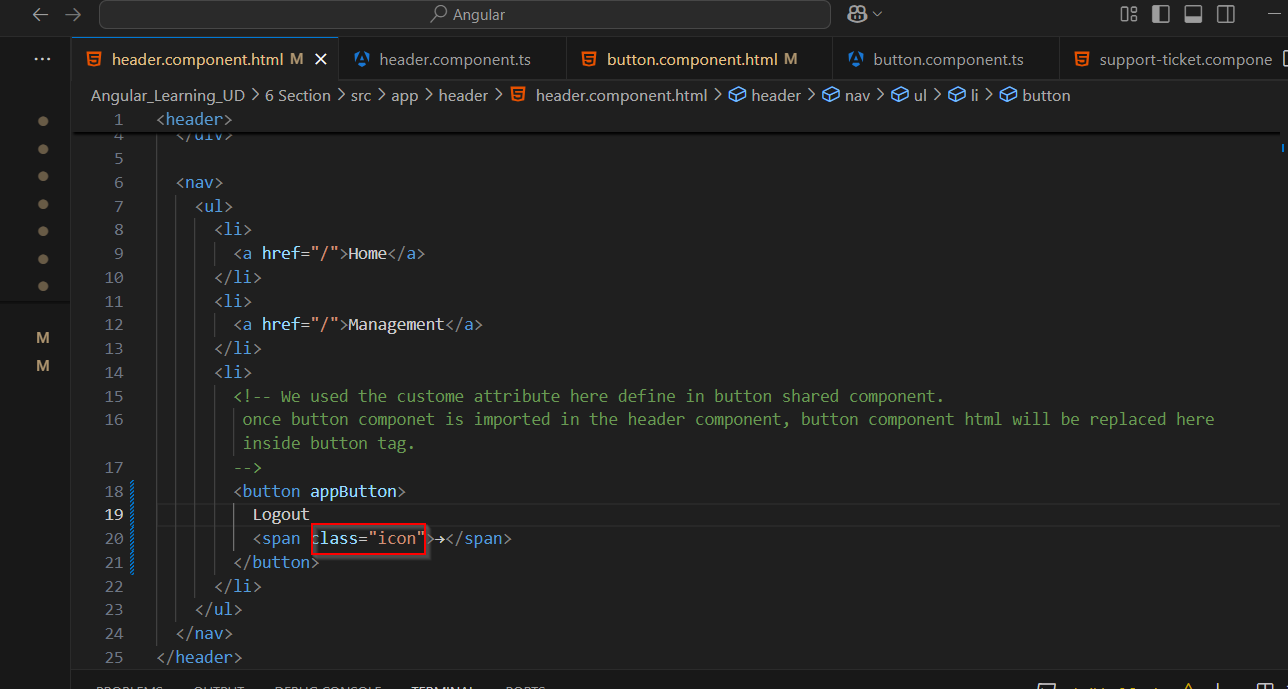


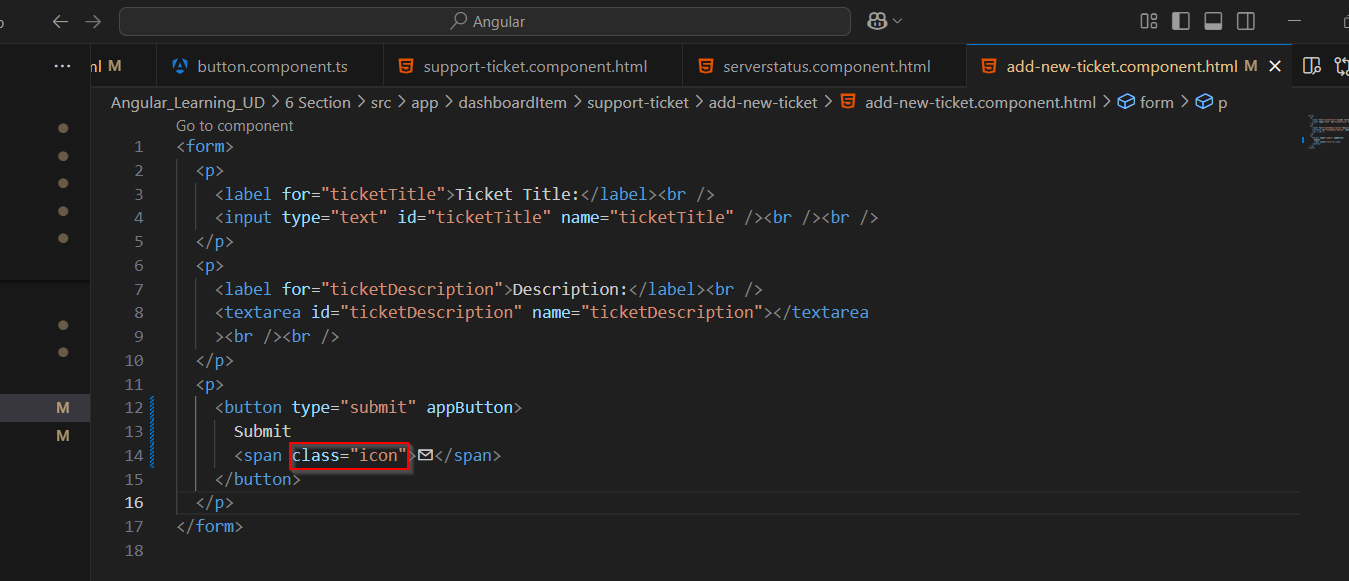


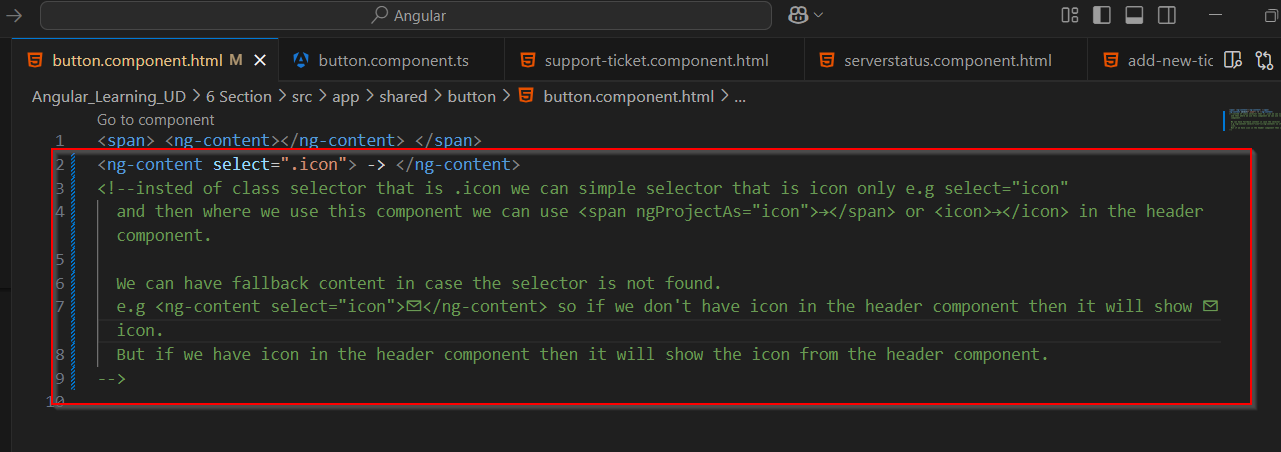
**ngContent with select**

* Ng Content element have select attribute or property.
* When this is used with this, angular check for exact match with that selector.
* Here in example we put .icon in select attribute , it means it check for element that have icon class and that will be replaced at the point of ng-content place.

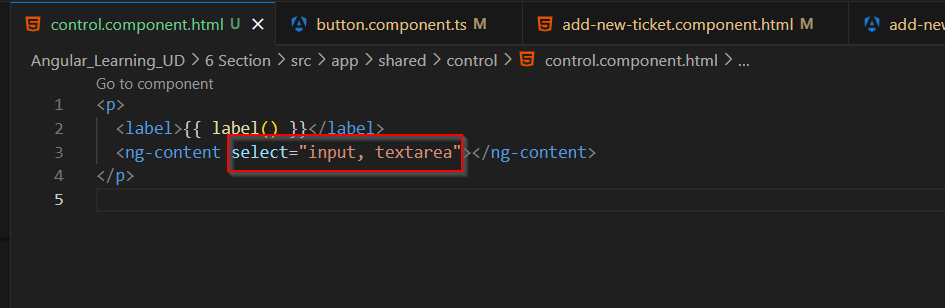


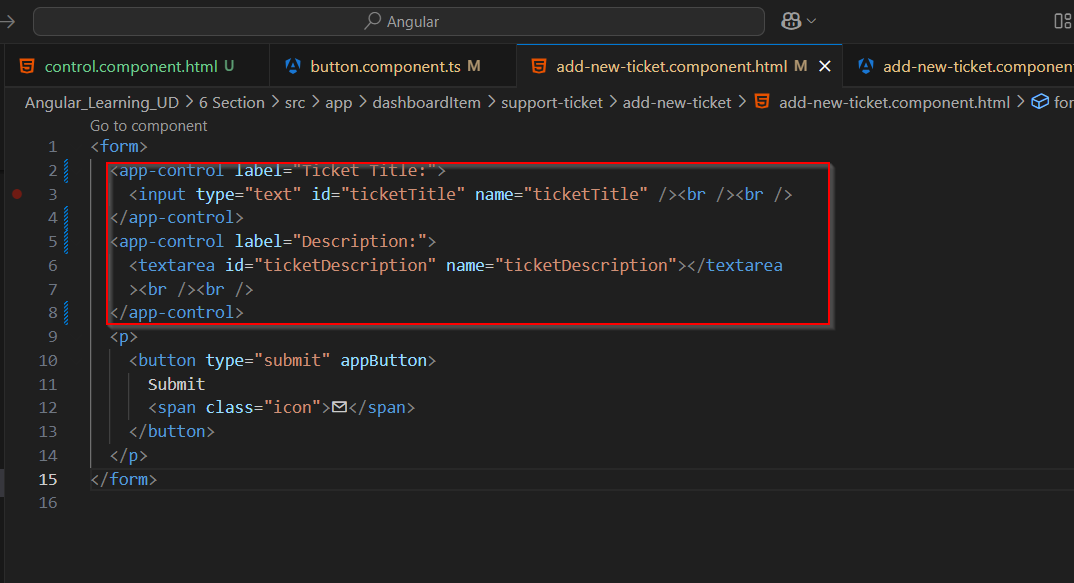






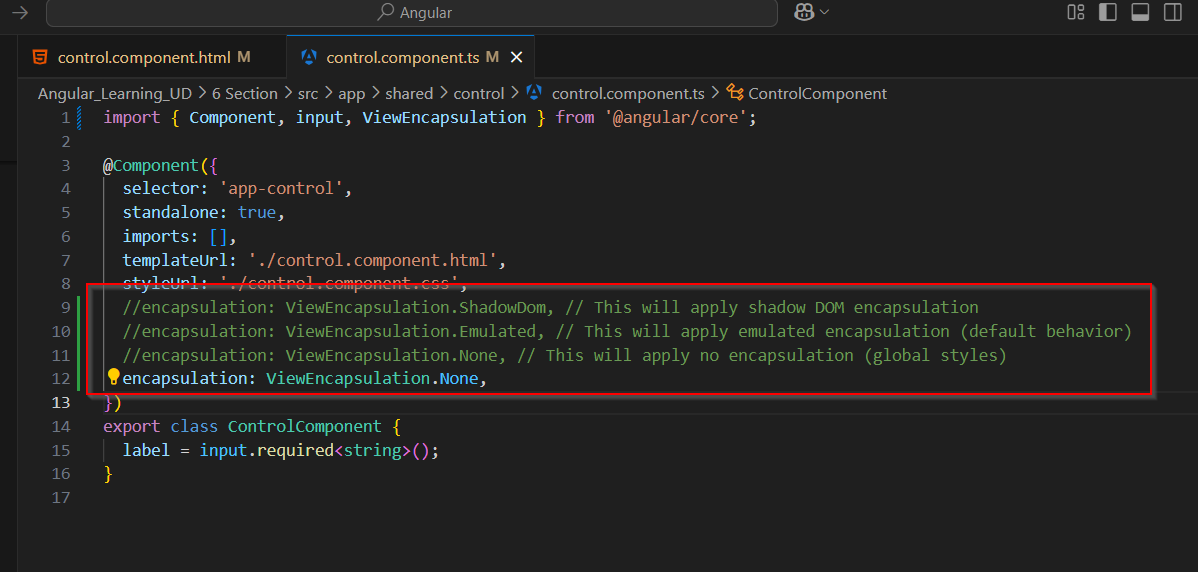
* Select property can be used for multiple html element also. As seen in example below if define two elements with comma separated both elements will be replaced by ng-content.



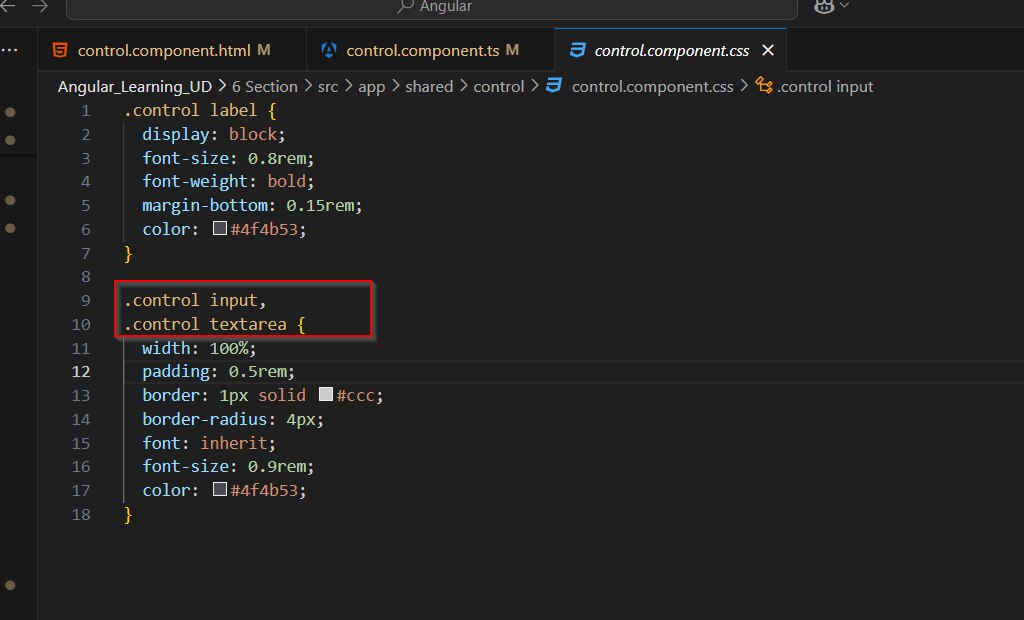


**Encapsulation**

* Encapsulation is another setting in @component decorator.
* When css define as component based and particular component have ng-content element at that time of applying css, angular not replaced actual tag.
* It sees only ng-content tag while applying css and due to this case in control component , input and textarea will not get css
* For this we can use encapsulation setting as ViewEncapsulation.none to make control component css global and then it will apply to input and textarea.

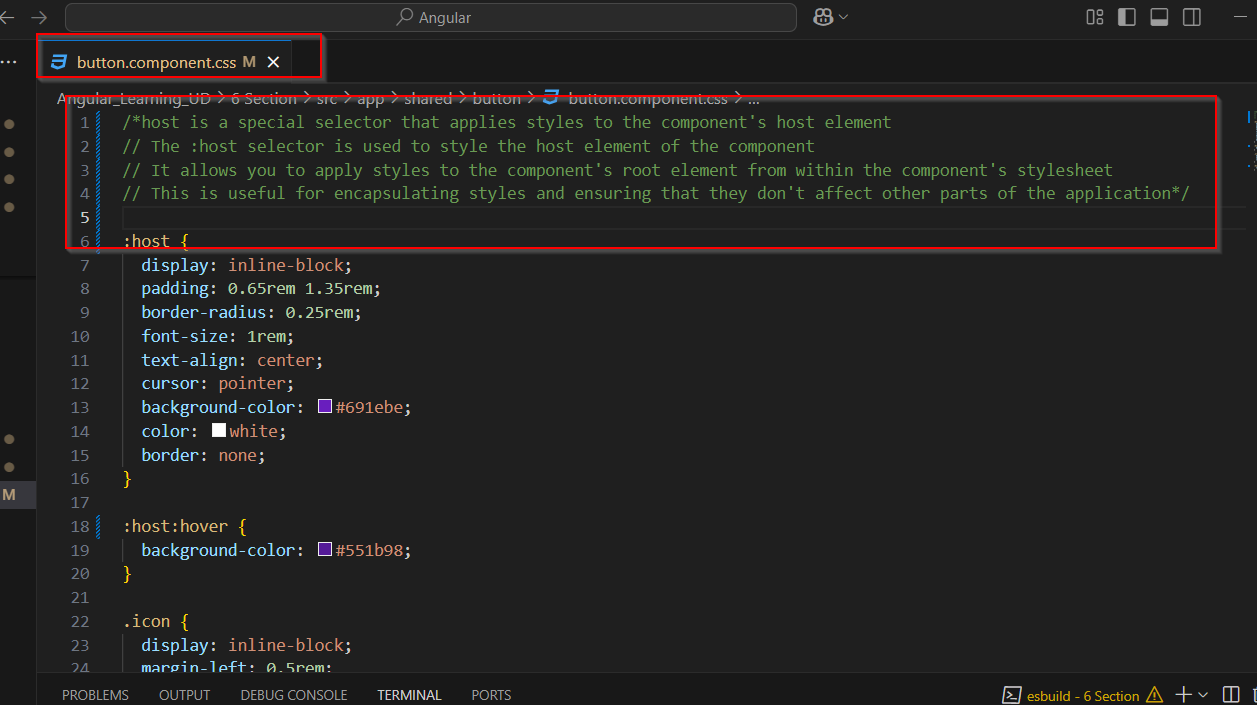






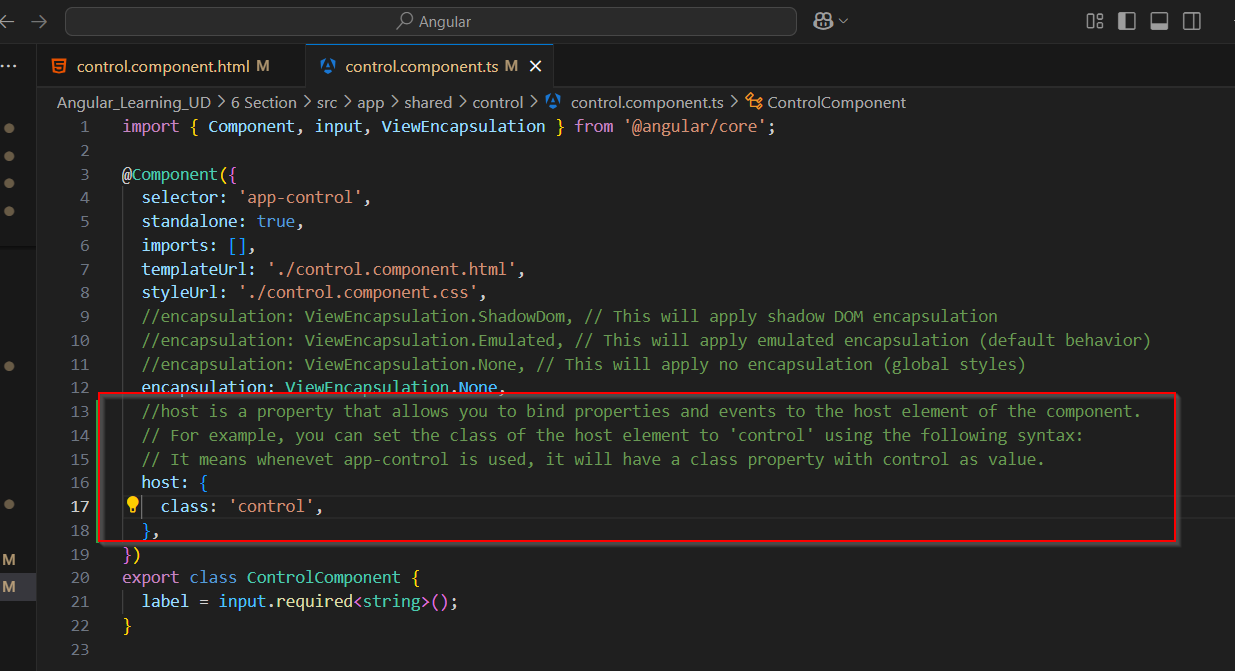
**Host selector in css**

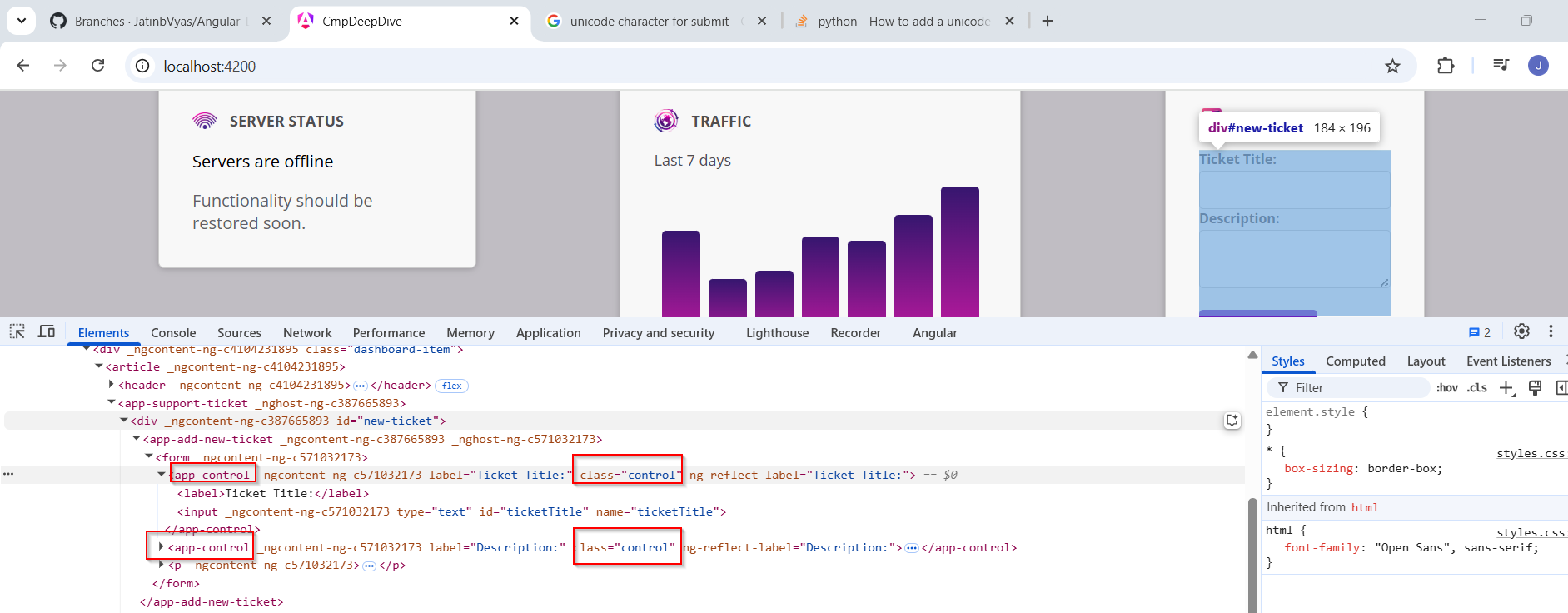
* Host selector in css is usefull feature of css
* Here in our case button is the host element of button component, so :host replace with button and apply style to button element.

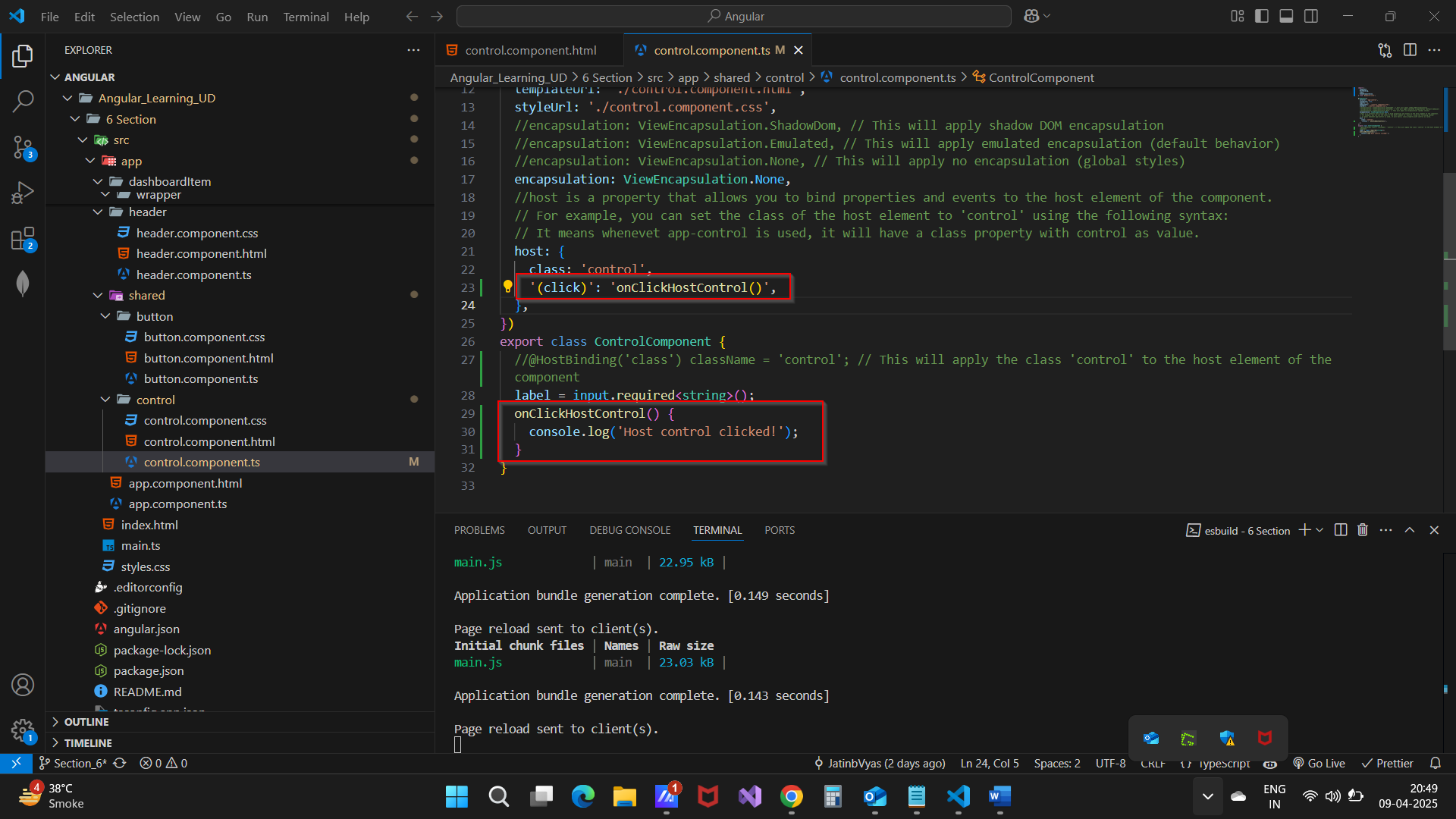


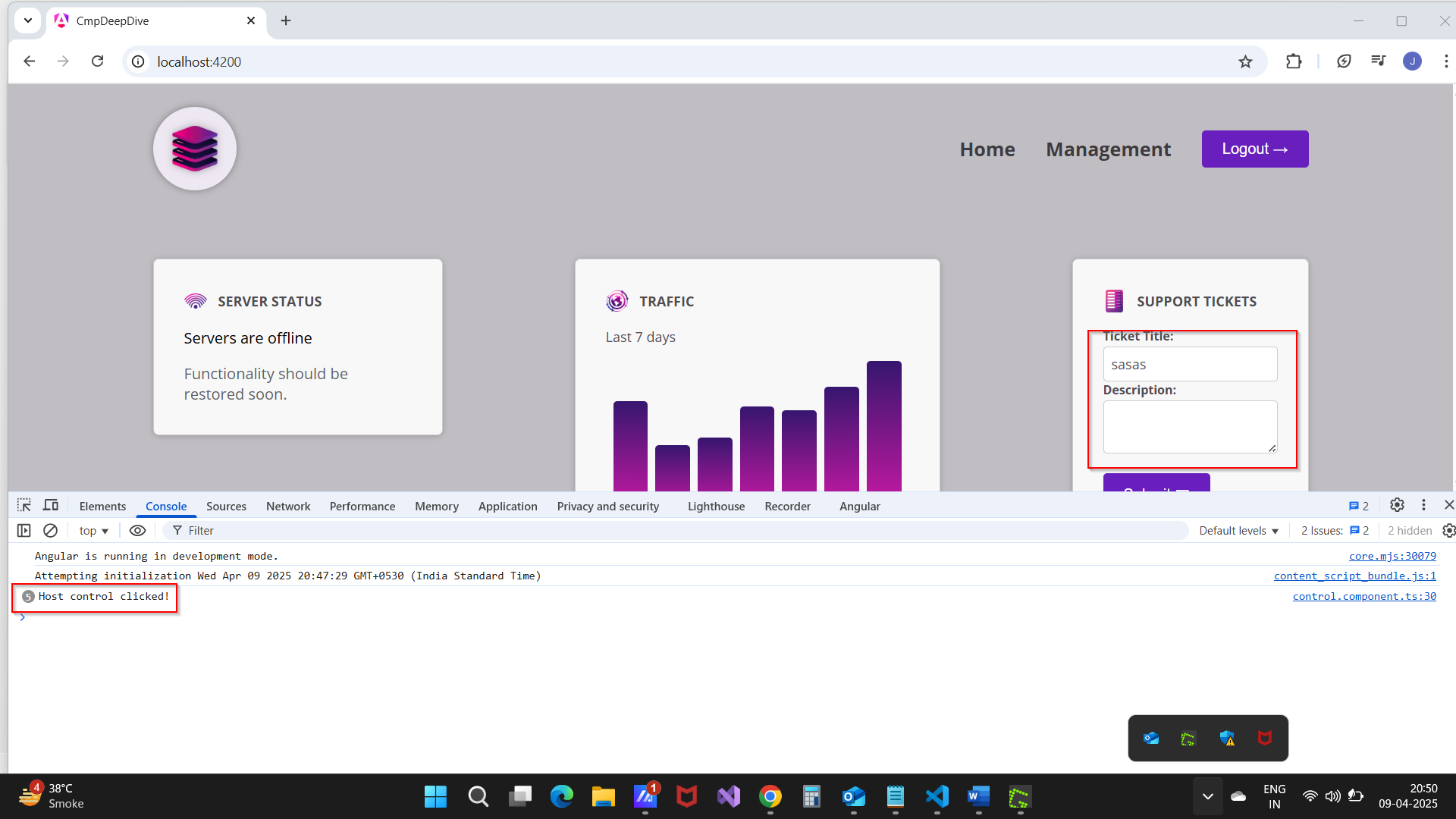
**Host setting**

* In @component decorator we have one more setting called host,
* Using this setting we can set any property and it’s value for that particular component.
* Here in this case whenever app-control is used it will added class as property with control as value.
* We can set the click event also to host control using event binding as shown in below example. So when control is clicked event will occure.









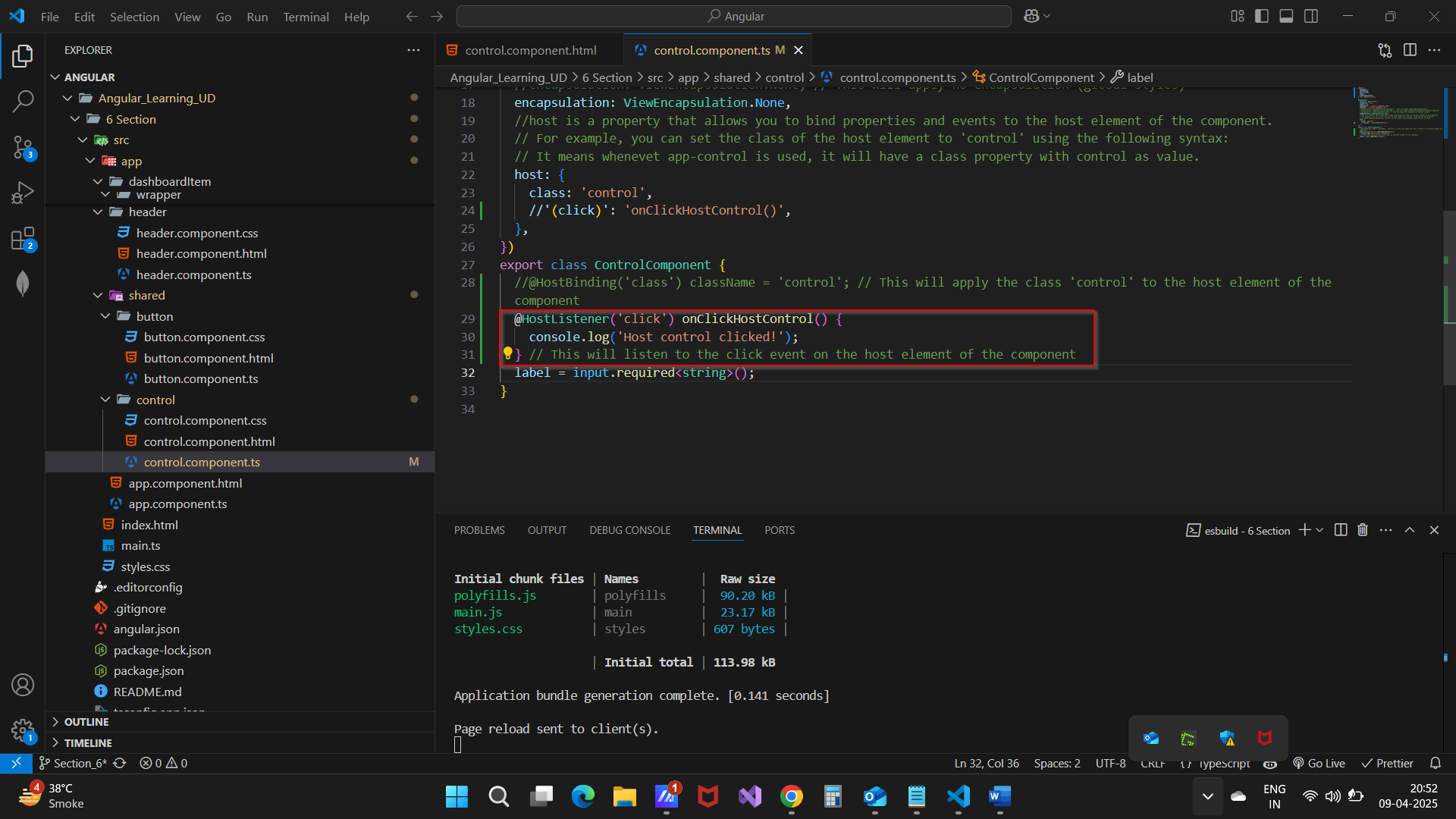
**@hostBinding**

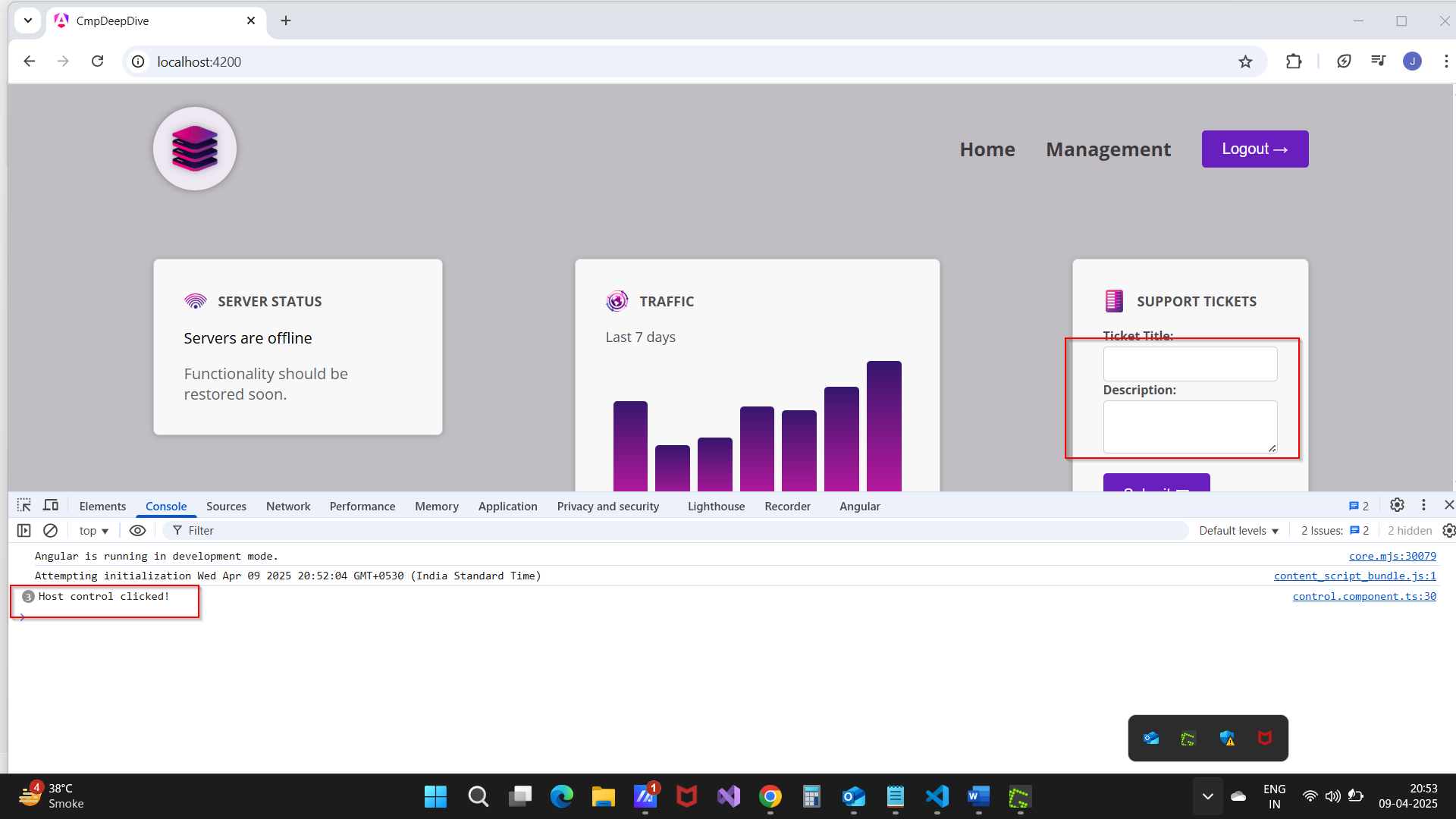
* We have @HostBinding() concept that works similarly to host settings in @component decorator.
* It will set property with given value to host controller. As shown below.



**@hostlistener**

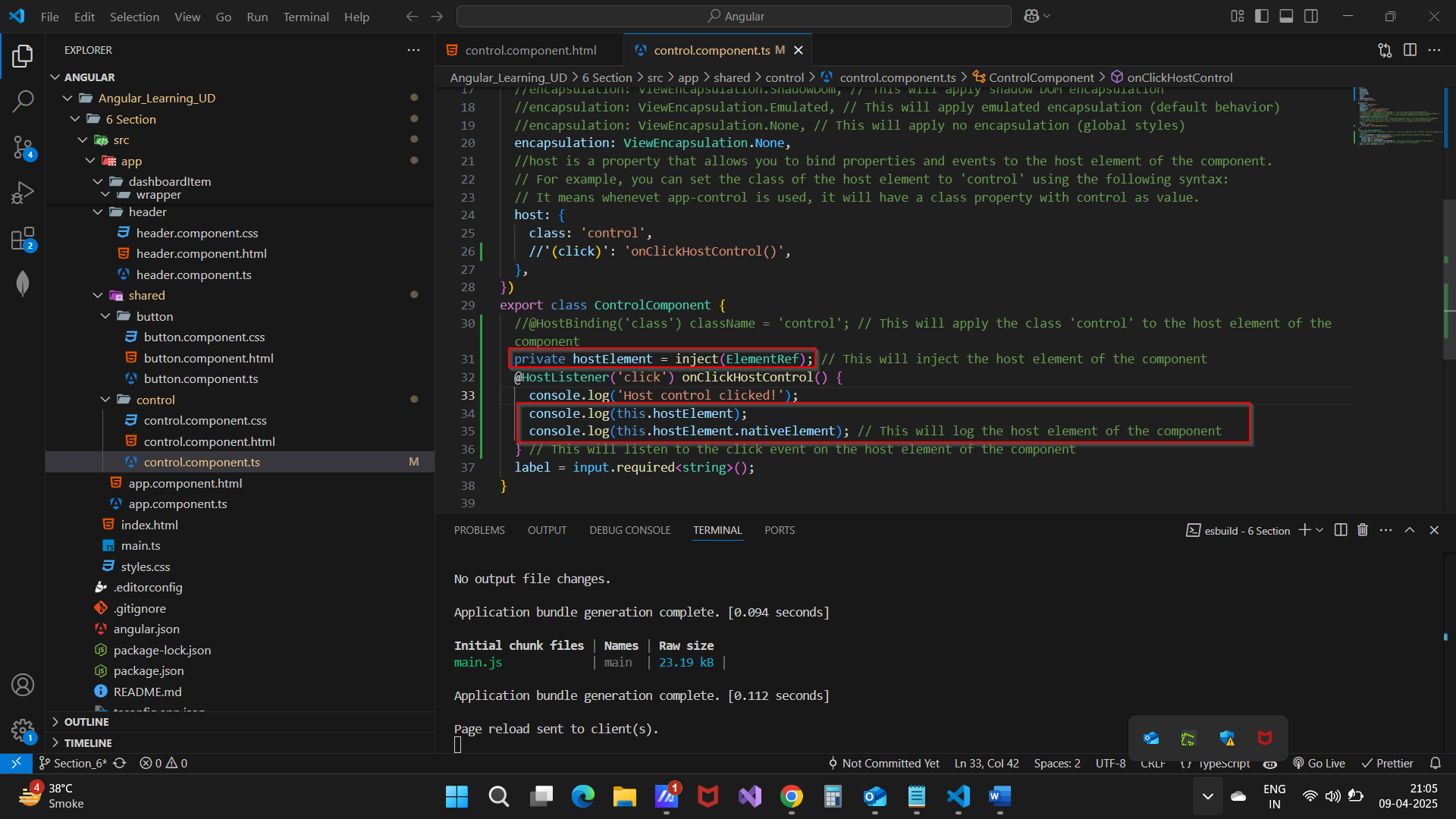
* We have @hostlistener() to bind and event to host controller same host setting with event binding.





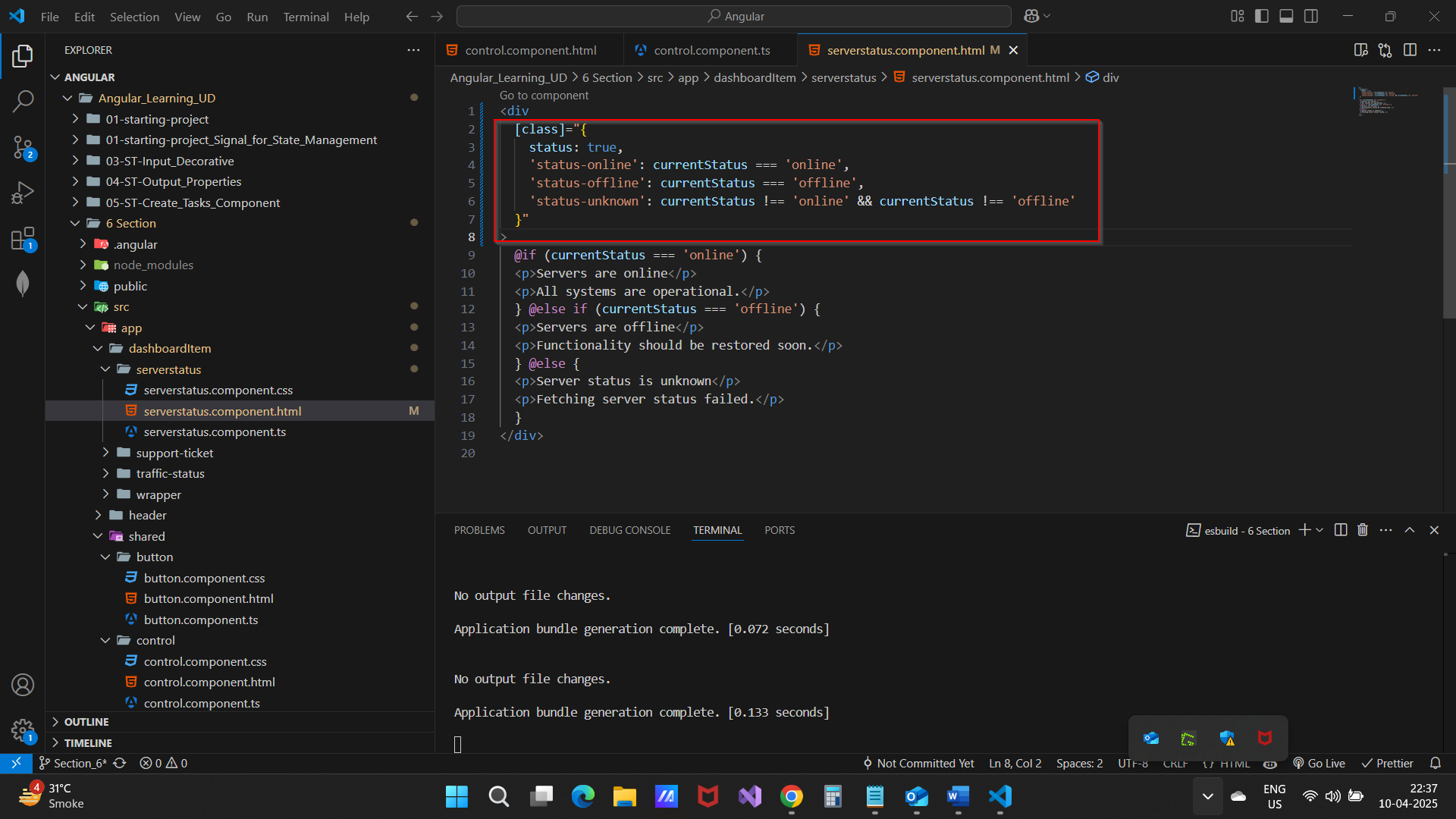
**Access to host element**

* We can access to host element using Elementref class, we need to inject that class using inject().
* Syntax and use is as below.



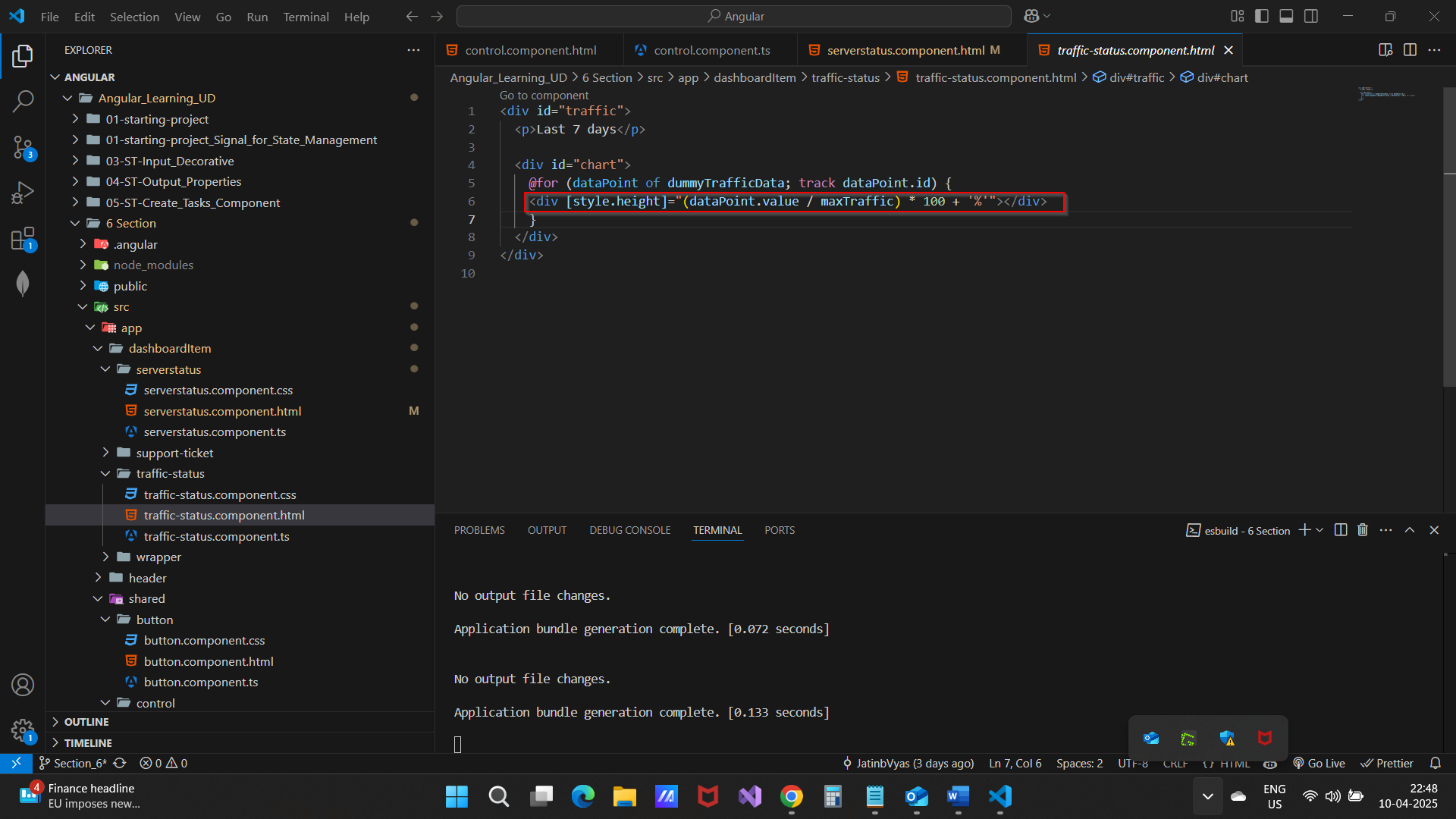
**Conditionally bind class**

* We seen earlier for one condition how we can bind css class dynamically.
* Now we saw here how set css class dynamically if there is multiple condition and based on that different type of css class is there.
* Syntax and use is as below. Class name is used as key in object and right hand side conditions are there.



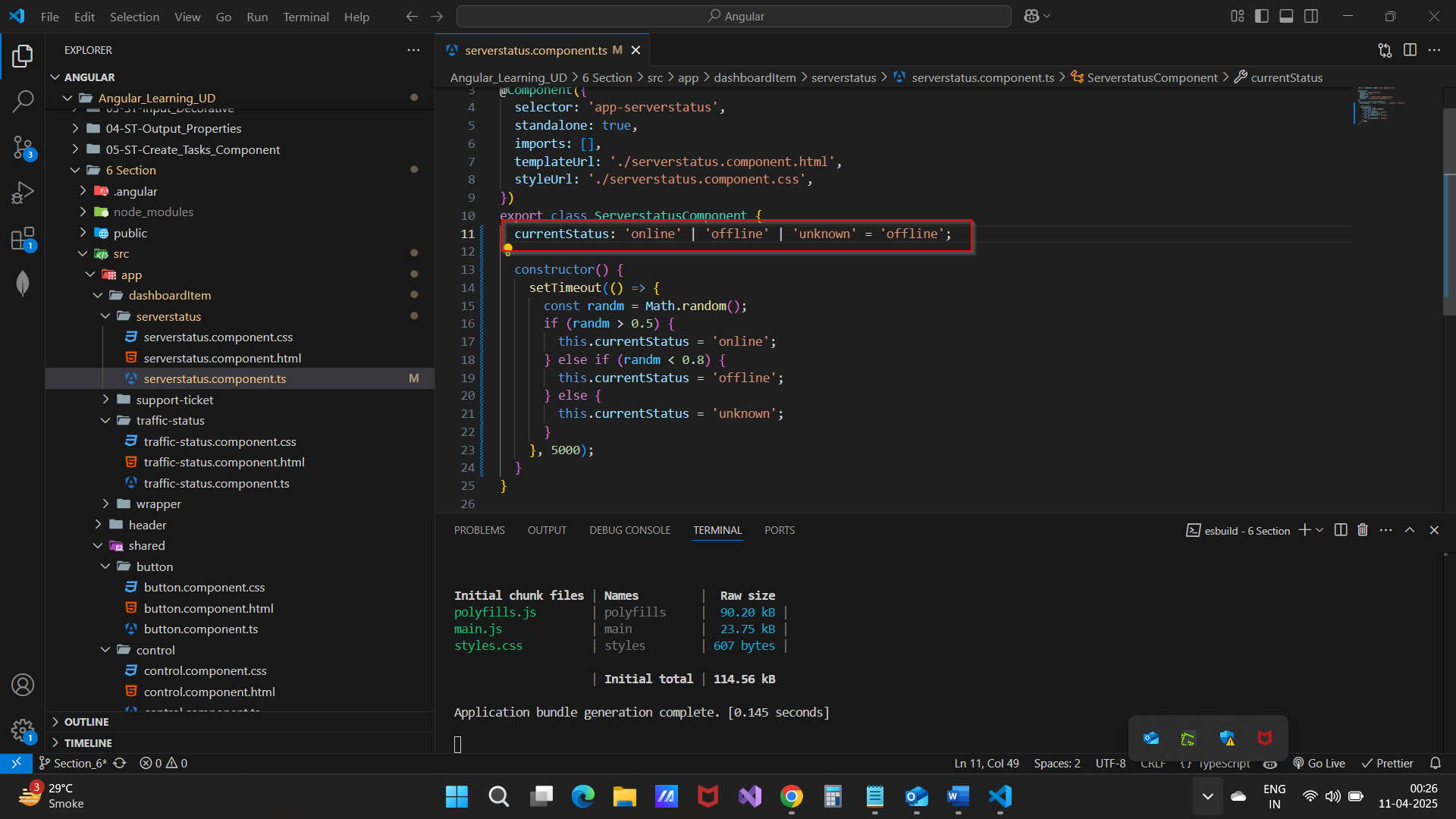
**Dynamically bind style**

* Same as class we can bind style dynamically as shown below.



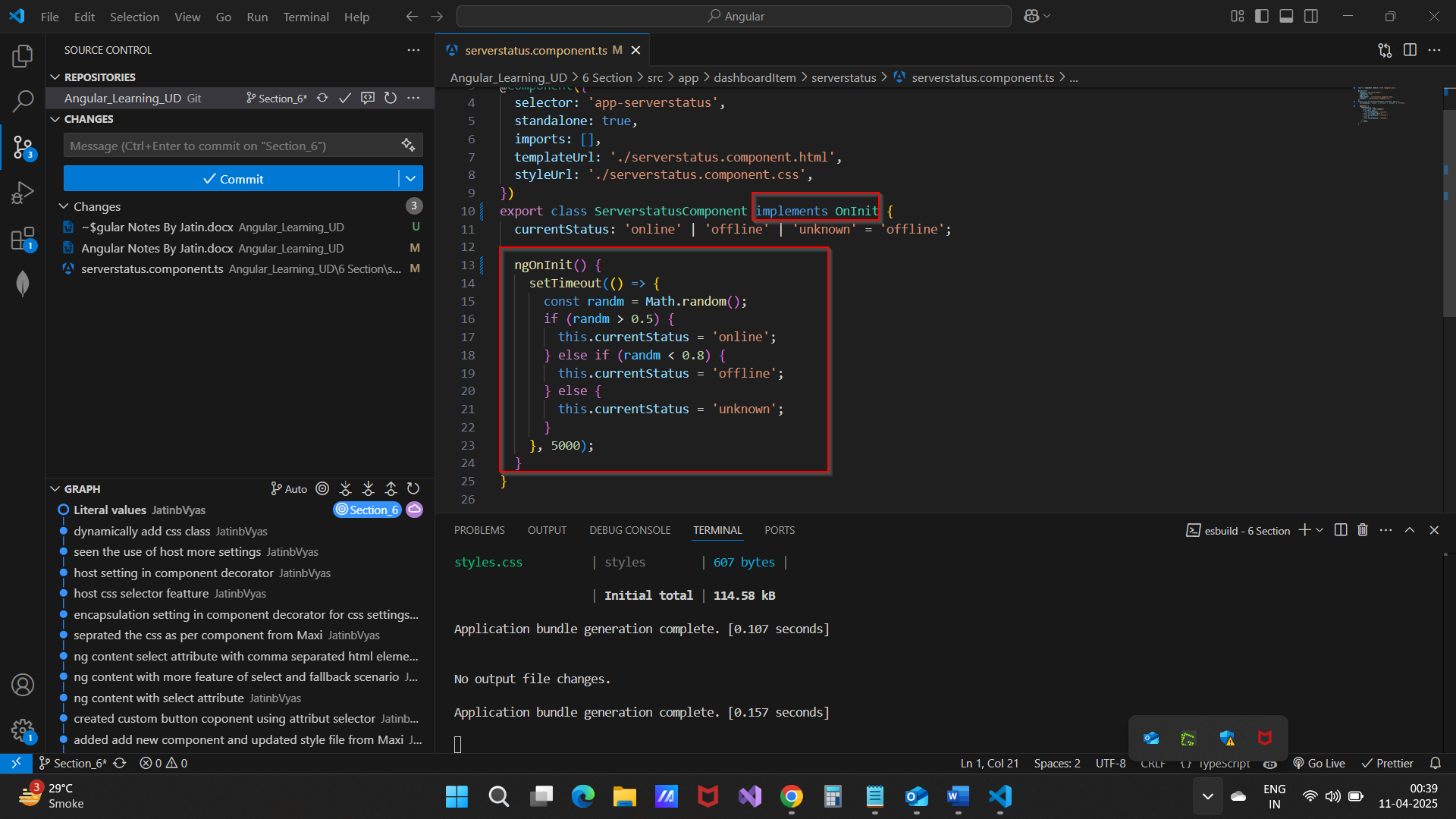
**Literal Values**

* We have one type script feature is Literal values.
* It means you can define the string value to property while declaring so only those string will get accepted while assigning.
* Syntax is as below. So in this case only these three value can be assign to property currentStatus nothing else.



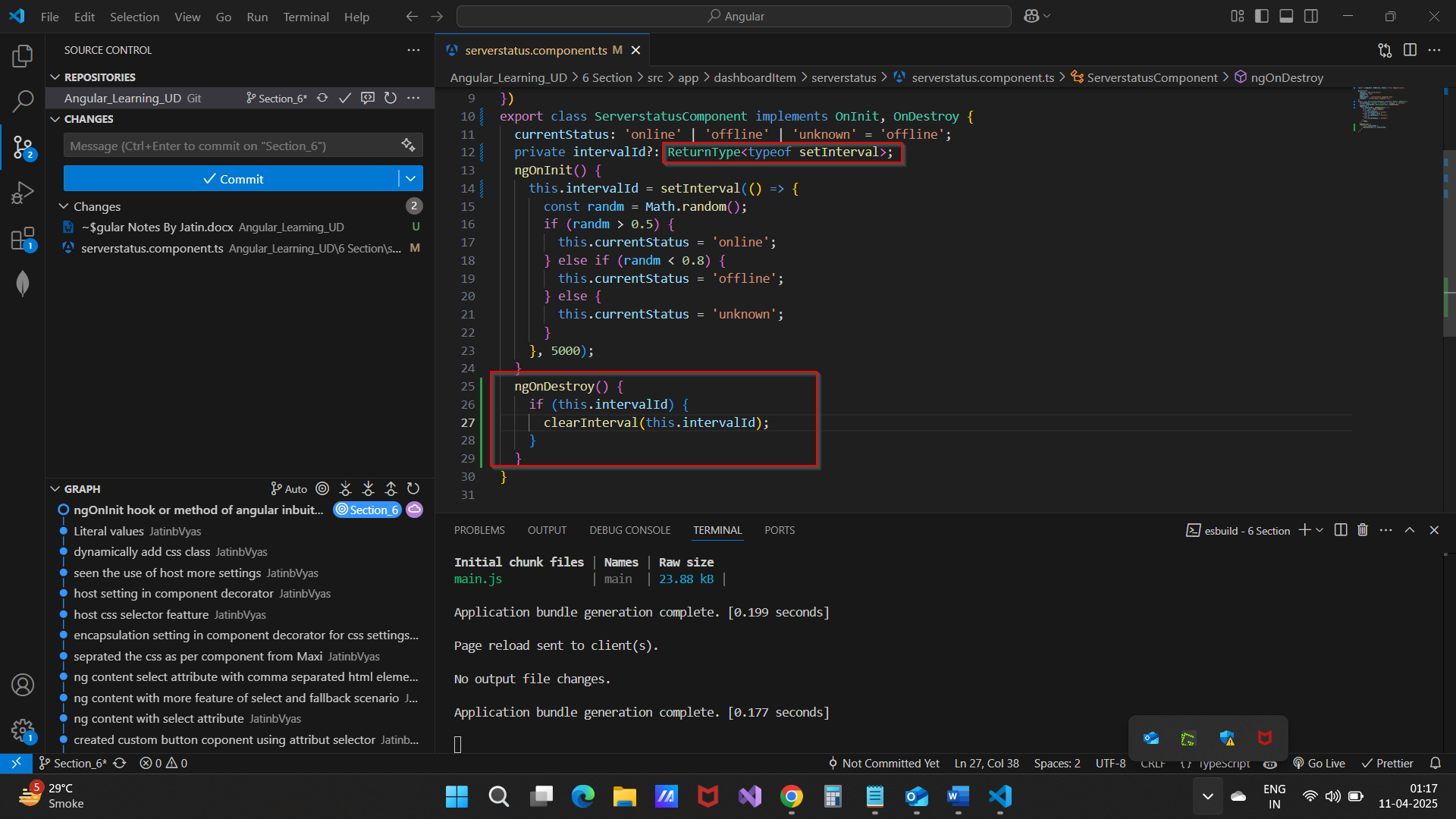
**ngOnInit**

* ngOnInit is inbuilt hook or method of Angular.
* It runs once when component is initialized. So it is recommended to do all thing here that is related to initialization of properties.
* Syntax is as below.
* Implement interface ngOnit while using it, this will avoid typo mistake.



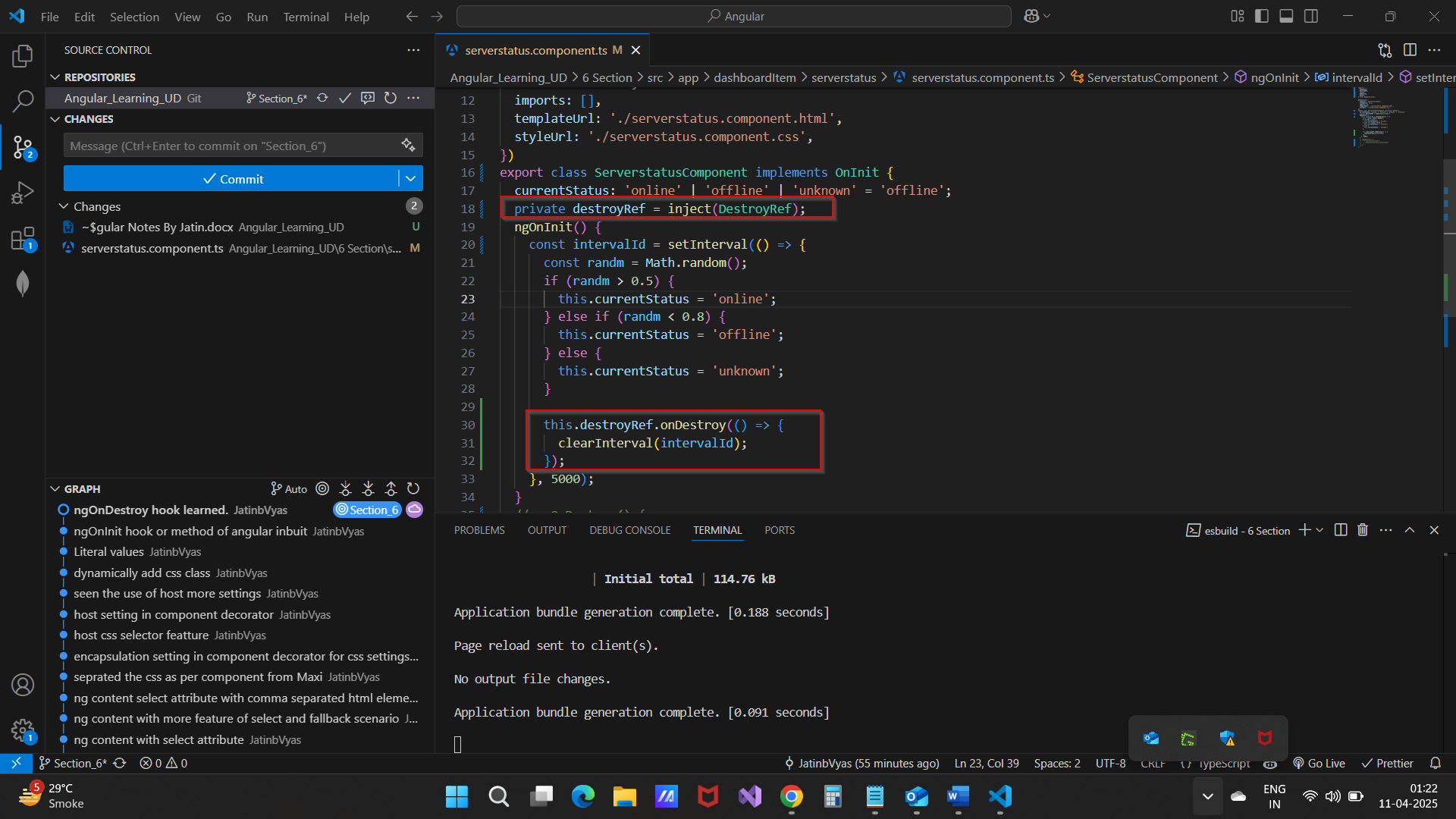
**onDestroy()**

* We have onDestroy inbuilt method of Angular.
* This method is called when component is destroyed or we can say removed from screen.
* So this hook can be useful for release of memory or cancel intervals that get set.
* Syntax is as below.



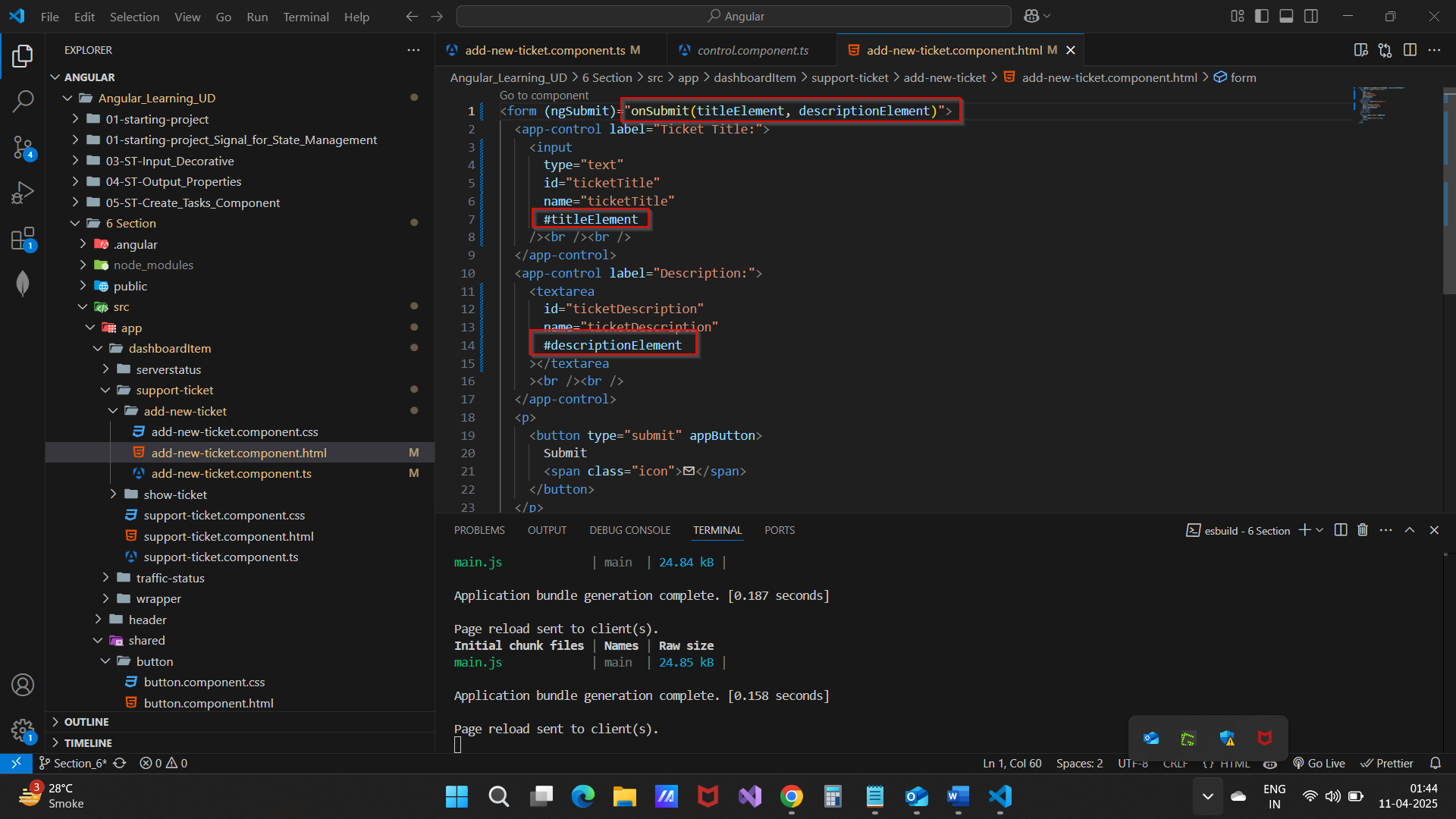
**DestroyRef**

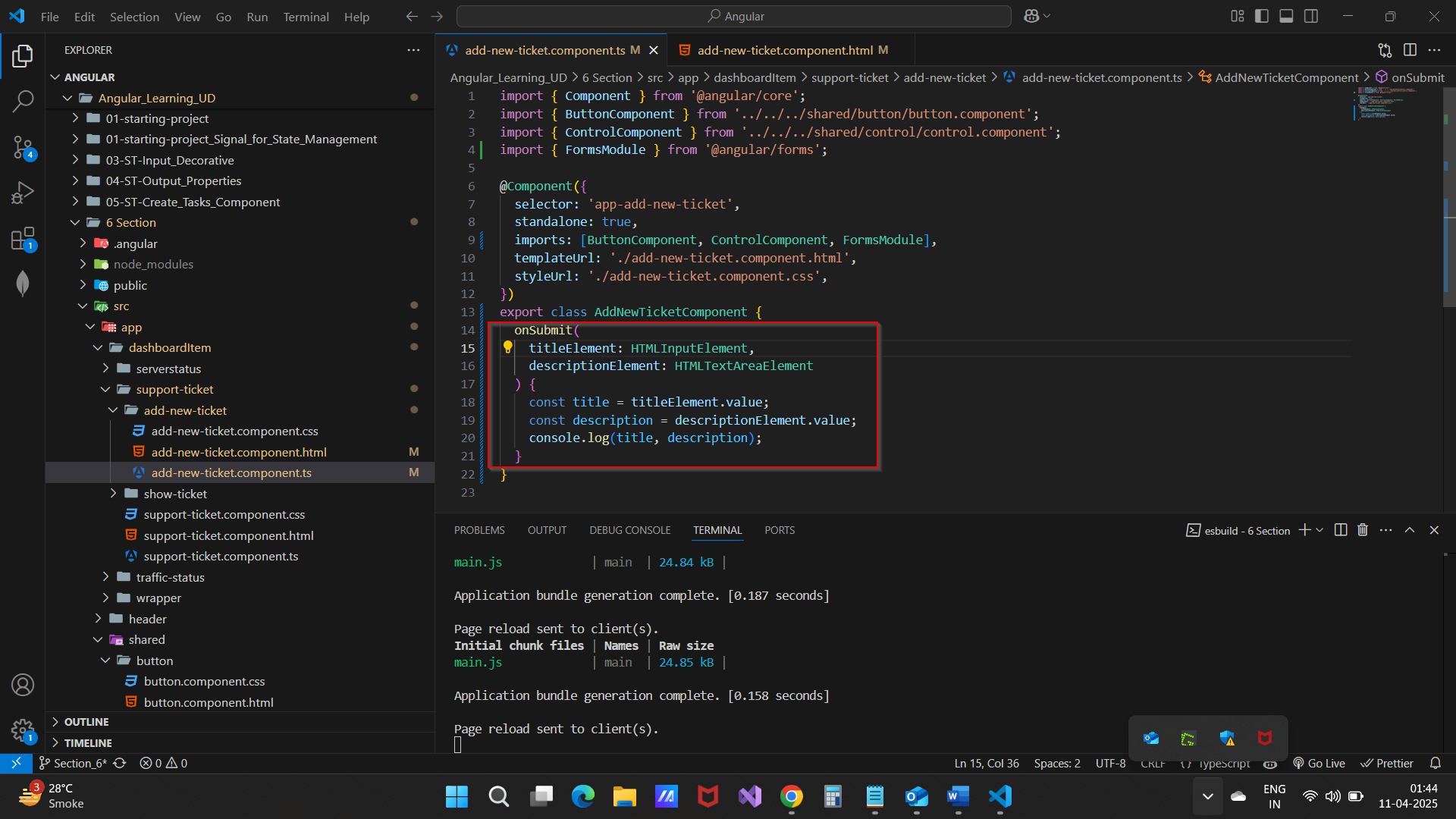
* Instead of ngOnDestroy we have DestroyRef in Angular 16 or above.
* This works same as ngOnDesctory to release memory, or we can say execute at last when component is expired.
* This takes one function inside onDestroy and where we can write our business logic related to that particular method.
* This can be define as much as we want.



**Template Variable**

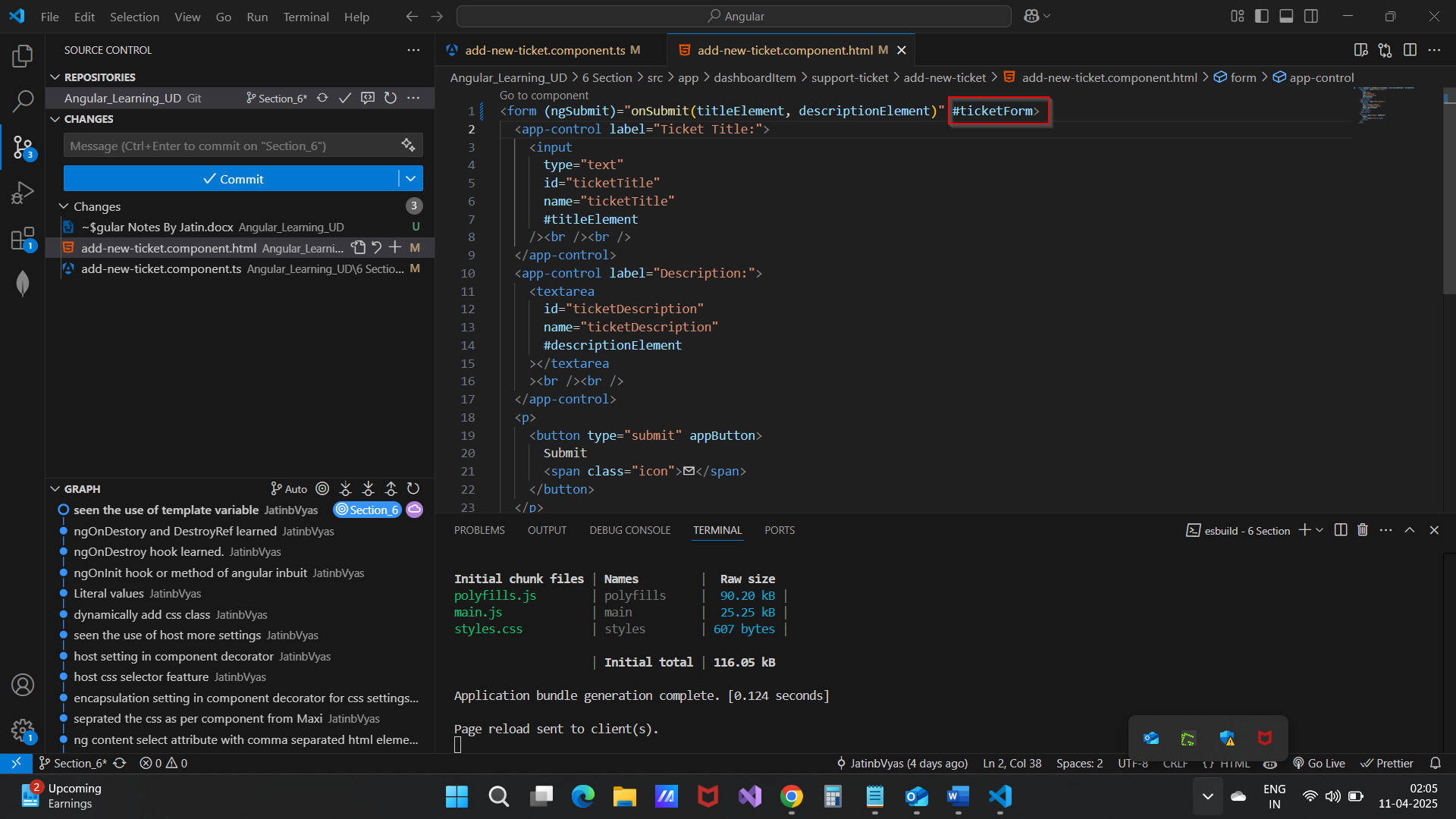
* We seen the [(ngModule)] for two way binding and fetching control values to class properties.
* There is another way to fetch control value is using Template variable.
* This is Angular feature that you can define template variable using #{nameofvariable} in particular tab.
* Now this variable can be use anywhere in that html.
* We can pass that in onSubmit function with proper type and this way we can fetch the value of controls.
* Syntax and use is as below.

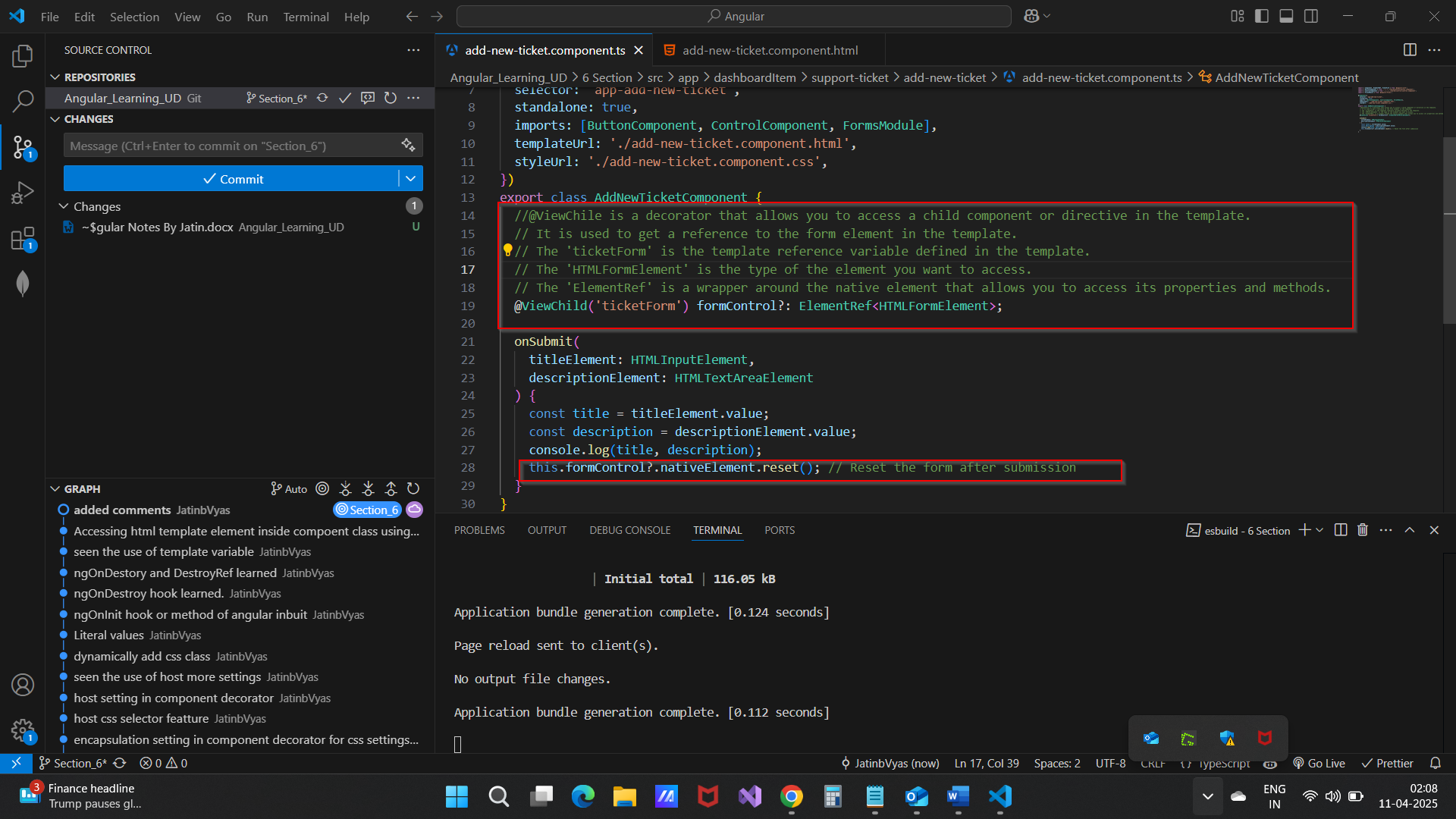




**@ViewChild decorator**

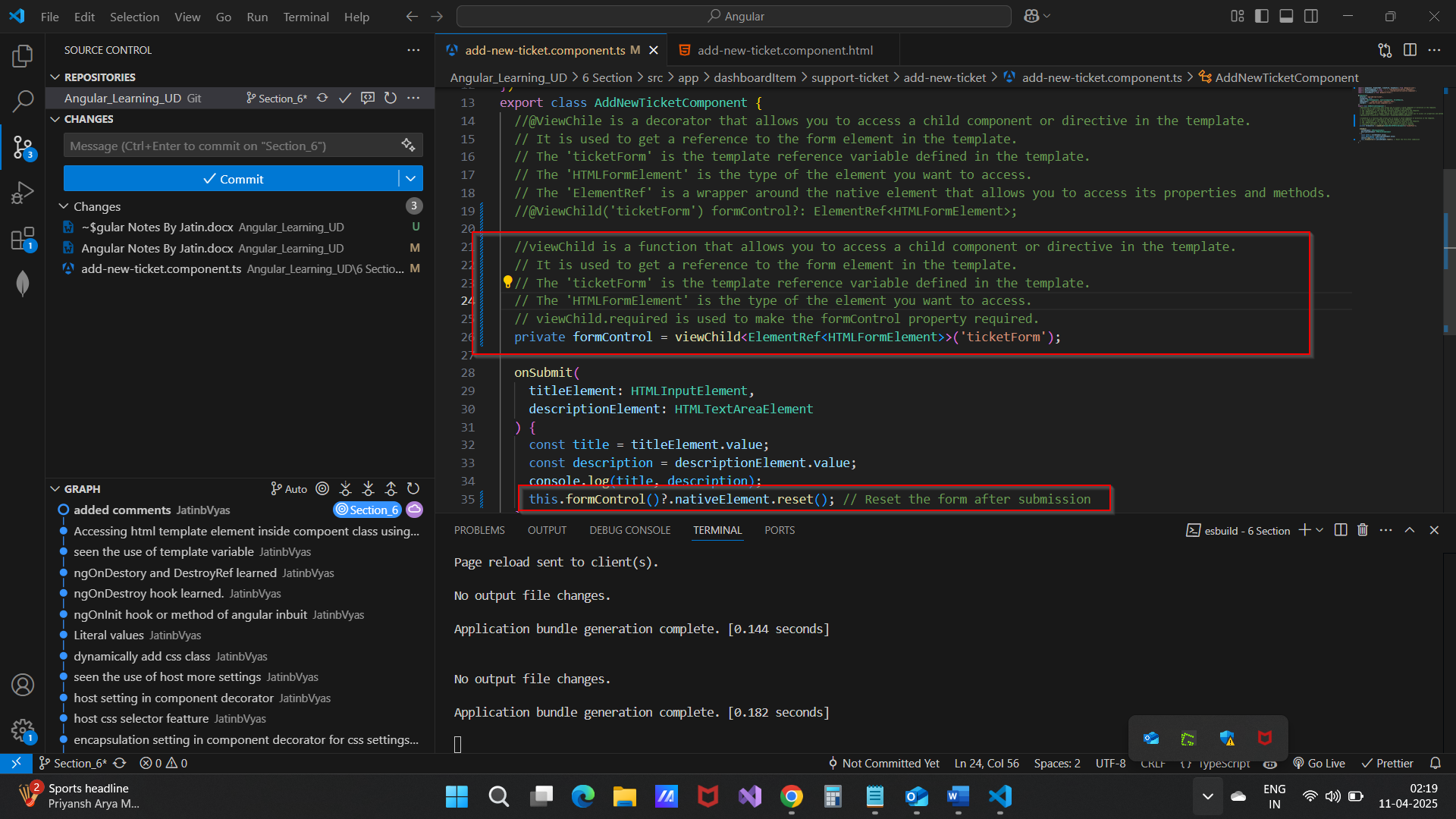
* @ViewChild decorator is a way to access html template element that is defined with template variable.
* For example here in this case we declare template variable for Form tag and then this variable new used inside @ViewChild decorator and give access of Form tag to define property , here in this case to formControl.
* To access we need to define this as Element Ref and then HTML type, here in this case is HTMLFormElement.
* And due to elementref we need to access using nativElement property at last.
* ViewChildren is also there that works same as ViewChild but it can give access to multiple element with give name or type like ButtonComponent in this case.
* @ViewChild and viewChild main difference is when element access by @ViewChild decorator that is not available in ngOnInit hook or method, only available in ngAfterViewIniti hook.
* When you access element using viewChild , if element found that can be accessible in both ngOnInit and ngAfterViewIniti





**viewChild signal**

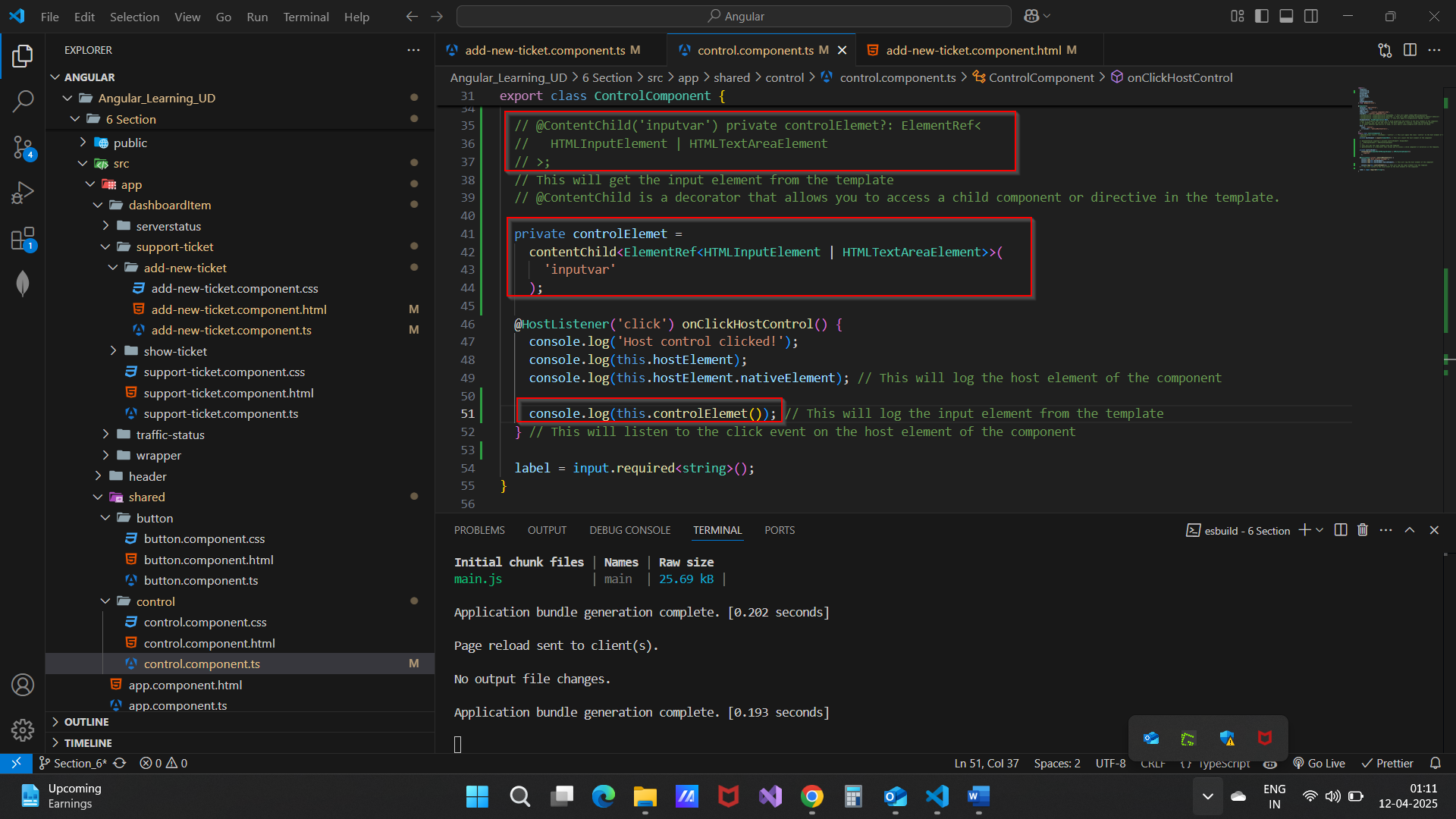
* This works same as @ViewChild decorator.
* But this is a signal .
* Syntax and use is as below.

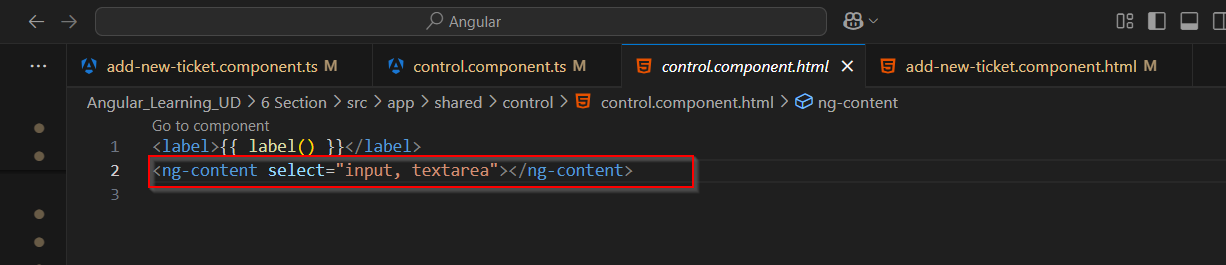


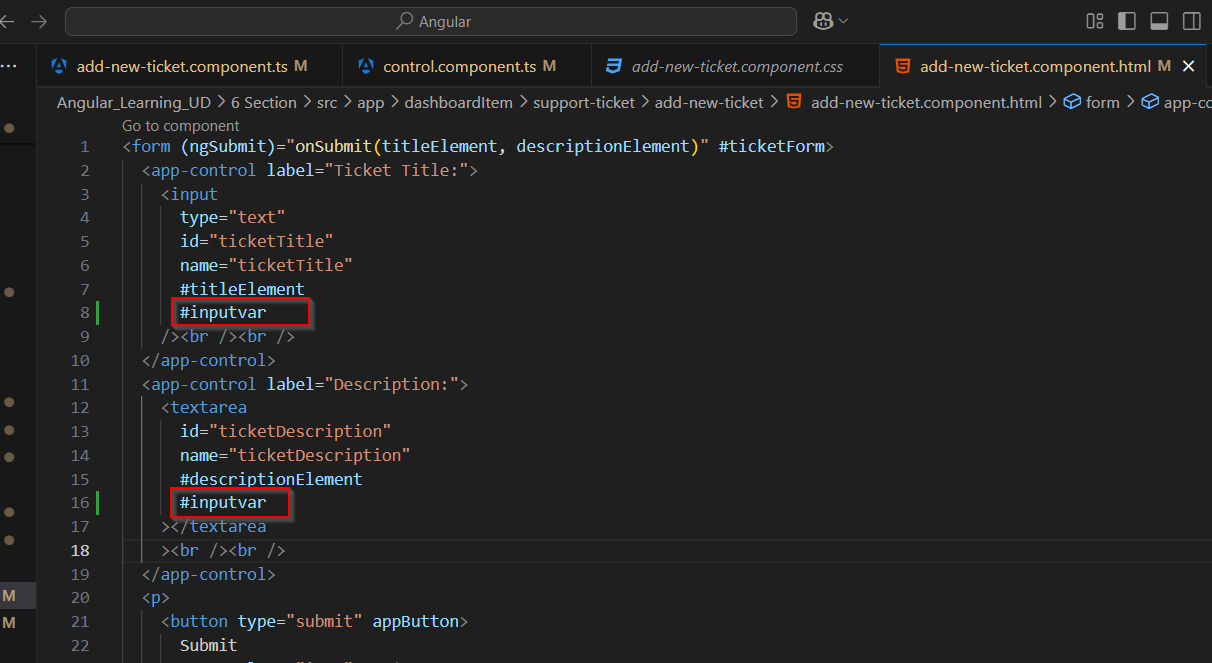
**@ContentChild decorator**

**contentChild signal**

* We have @ContentChild decorator and contentChild signal both for access html template element that is projected by ng-Content tag.
* Means the controls that is replaced by ng-Content and show in html template that can be access by this decorator.
* Syntax is as below.
* @ContentChild and contentChild main difference is when element access by @ ContentChild decorator that is not available in ngOnInit hook or method, only available in ngAfterContentInit hook.
* When you access element using contentChild , if element found that can be accessible in both ngOnInit and ngAfterContentInit

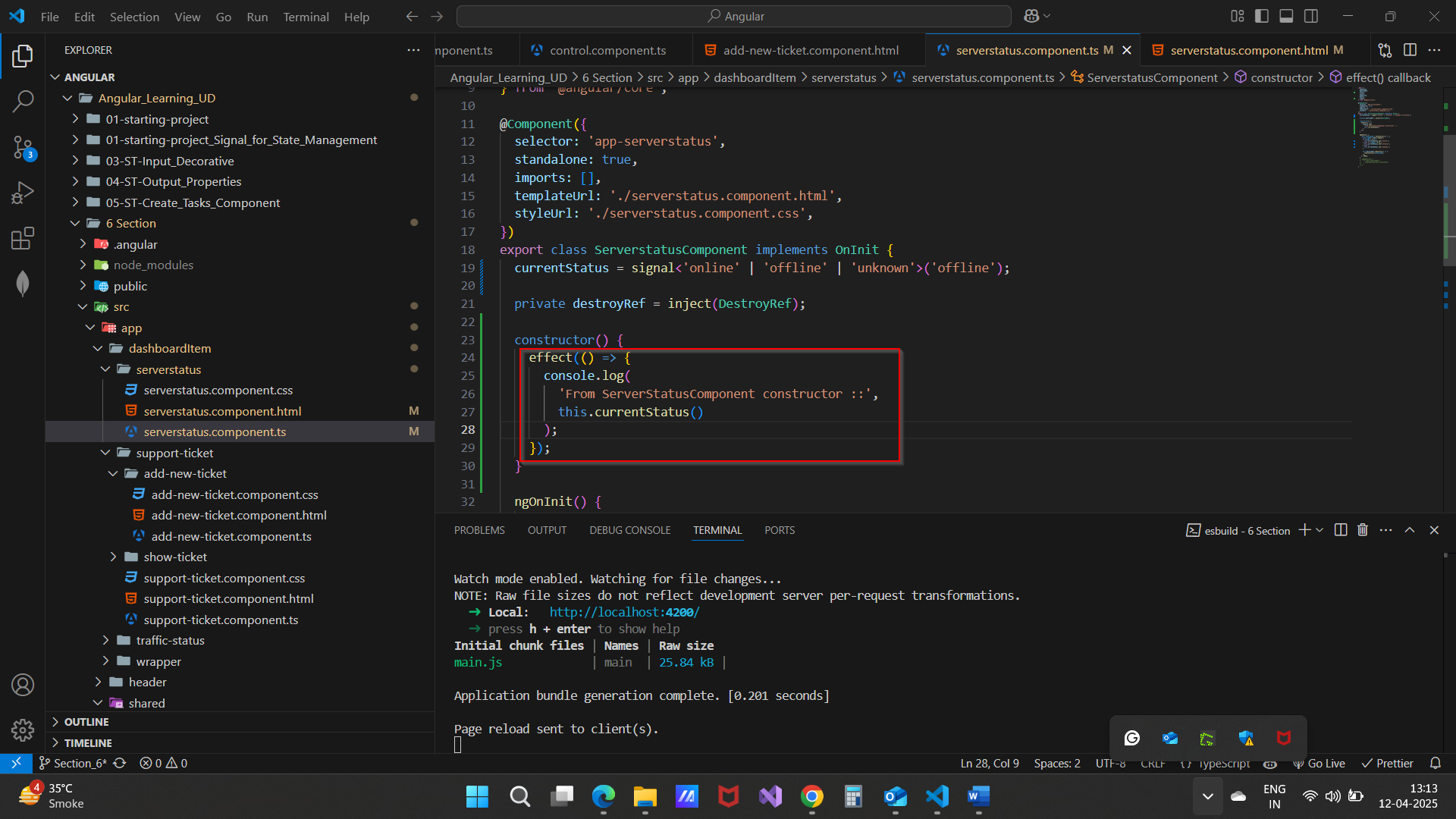






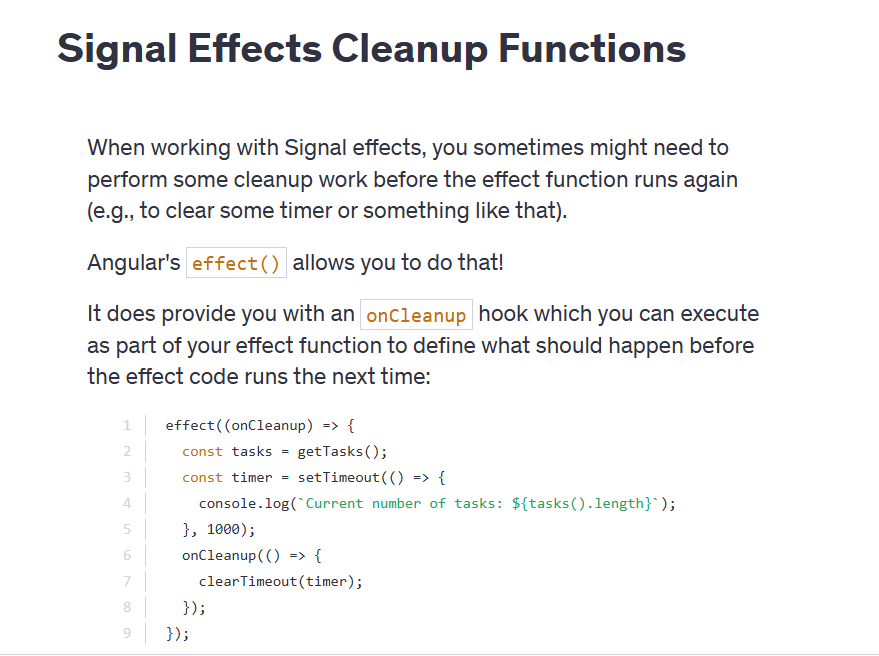
**Effect signal method**

* In Angular, we have the Effect signal method.
* Normally when property is defined with singnal, Angular automatically set the subscription so whenever property value change it effect every place when it used.
* But in case property is used in constructor Angular not set subscription over there.
* So you will not get updated value everytime.
* For this solution effect method is useful.
* Effect method set subscription for property in constructor also.
* Use and syntax is as below.



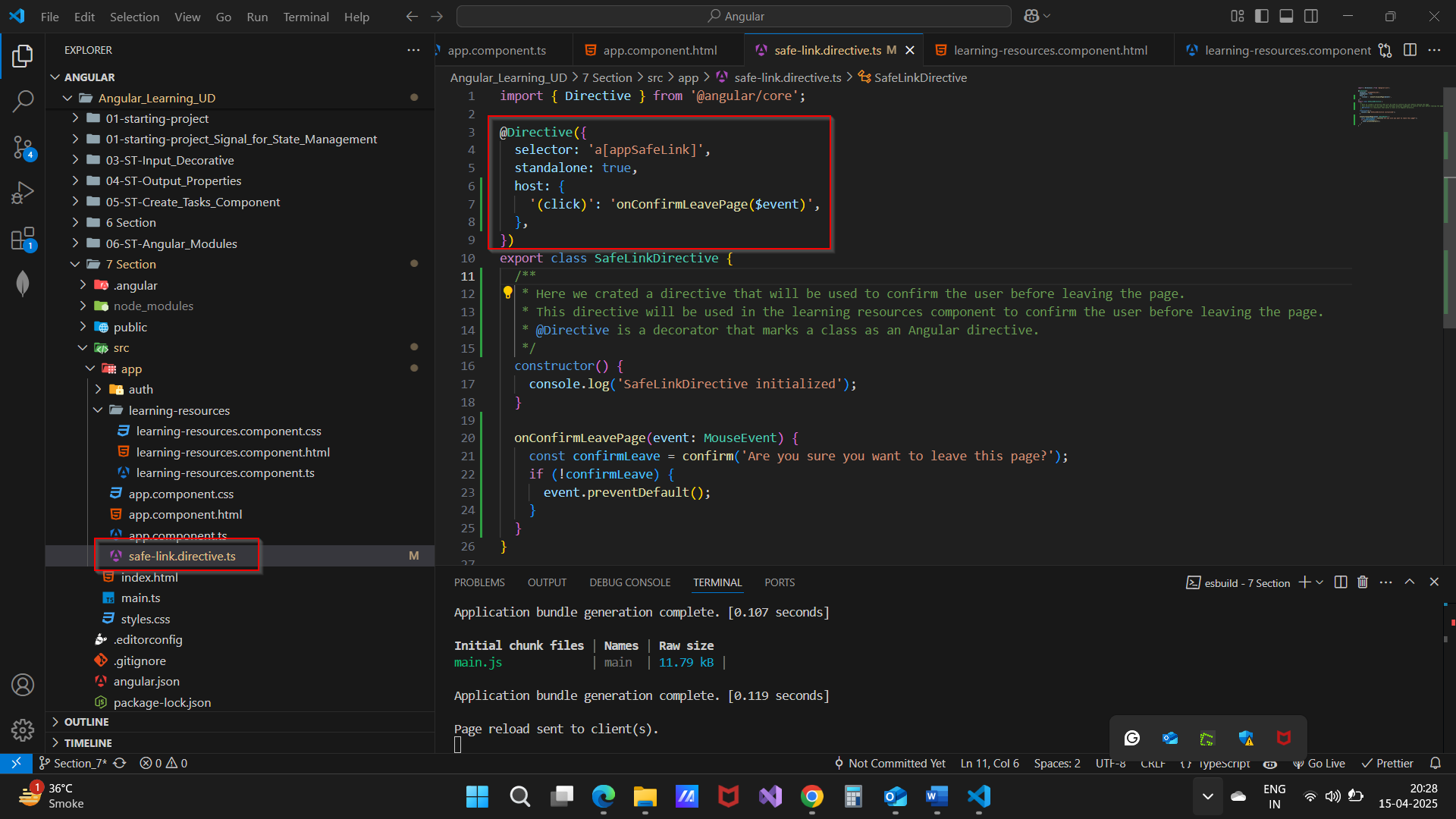
**Cleanup()**

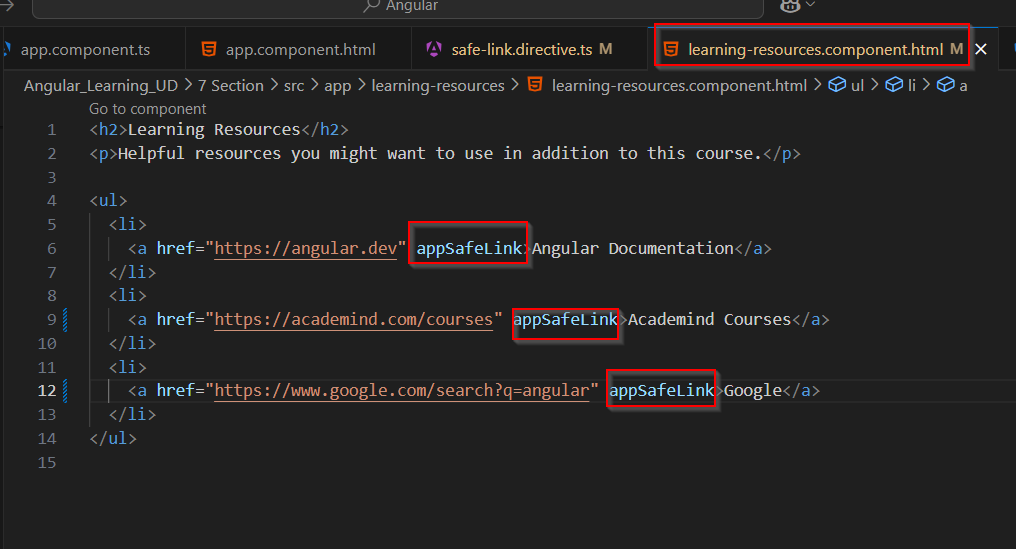
* We have cleanup function also that can be used with effect().
* Syntax and use is as below.

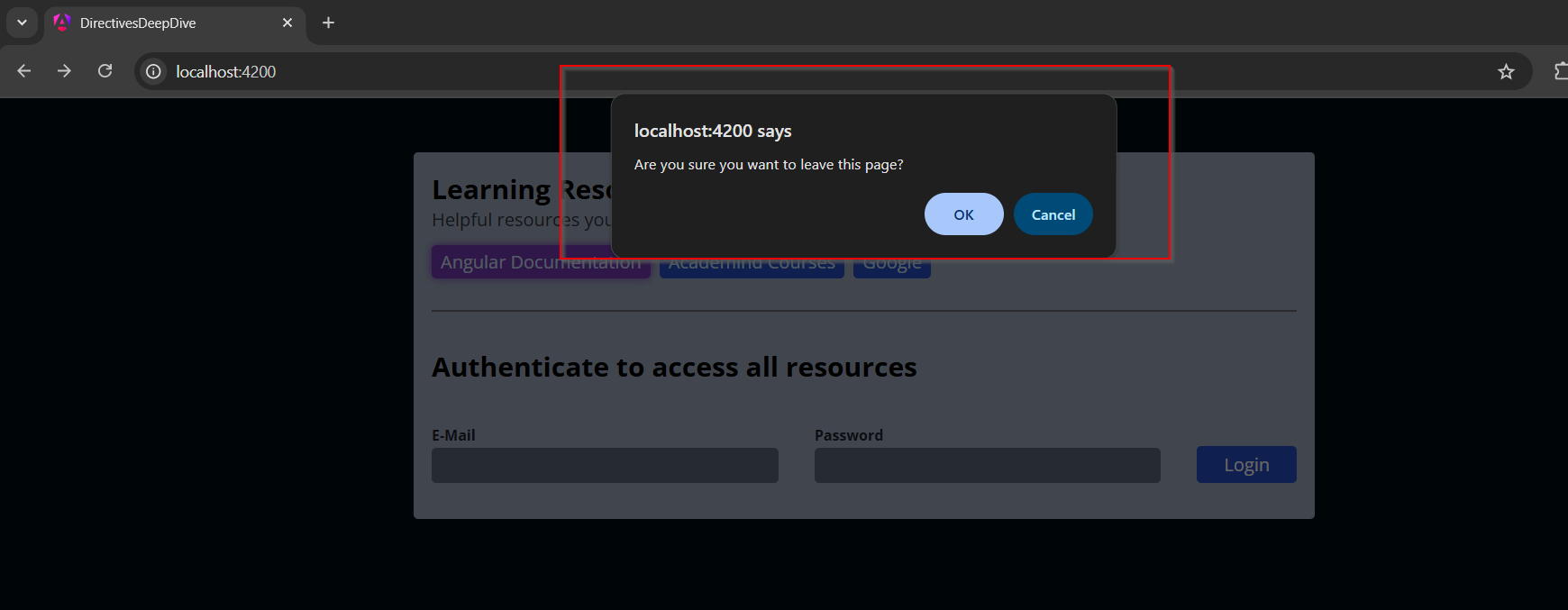


**Custom Directive**

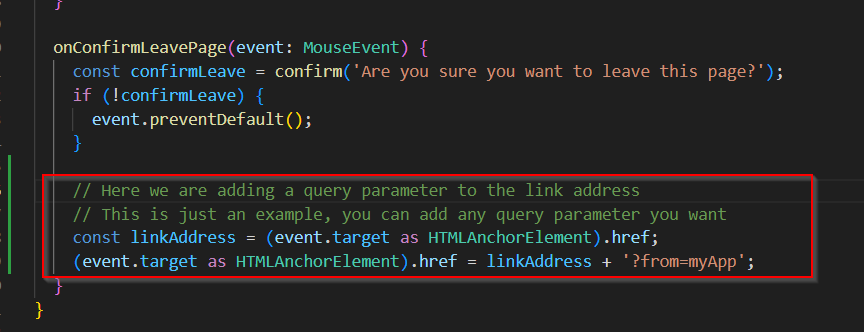
* In Angular, we can define our custom directive.
* Normally, we have an Attribute and a structural directive. For example, ngModule is an attribute directive while \*ngFor and \*ngIf are structural directives.
* To create a custom directive, we first need to create the directive file and need to create a class decorated with the @Directive decorator that will convert that class into a directive.
* Here, in this case, we created a SafeLink directive that will be used in the learning-resource template page with an anchor tag to confirm before leaving the site.
* Syntax and usage are as below.



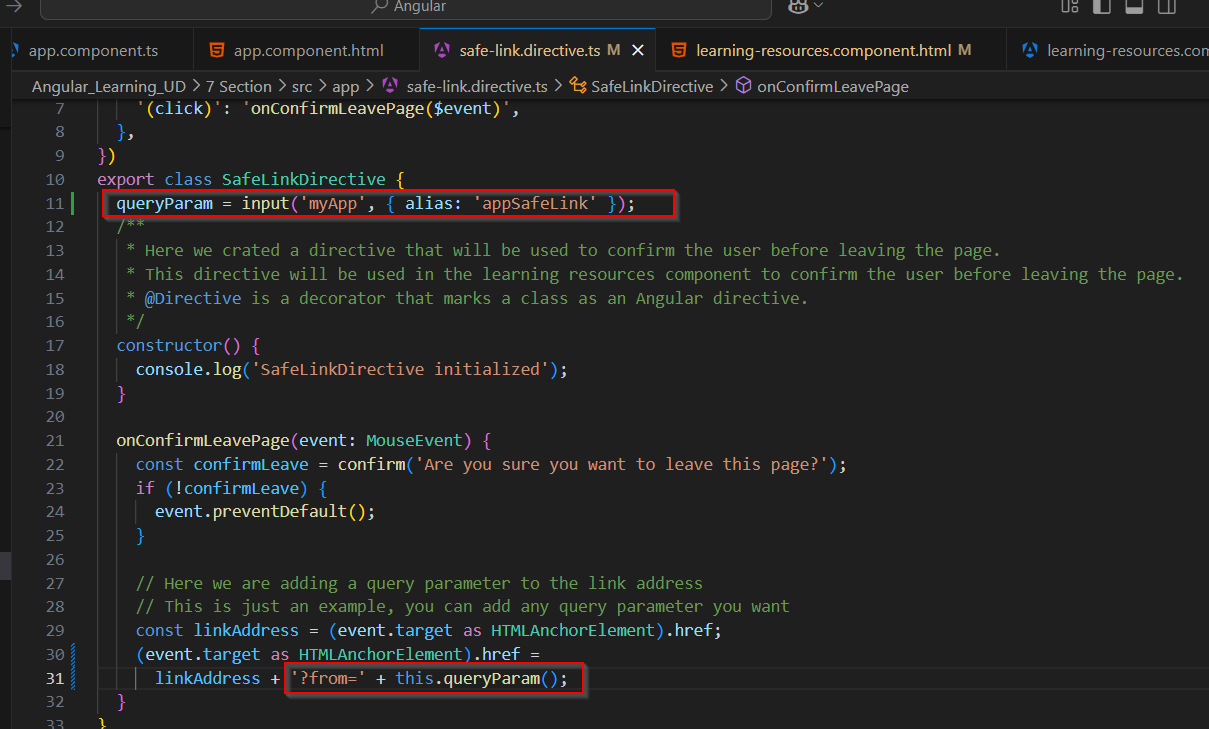


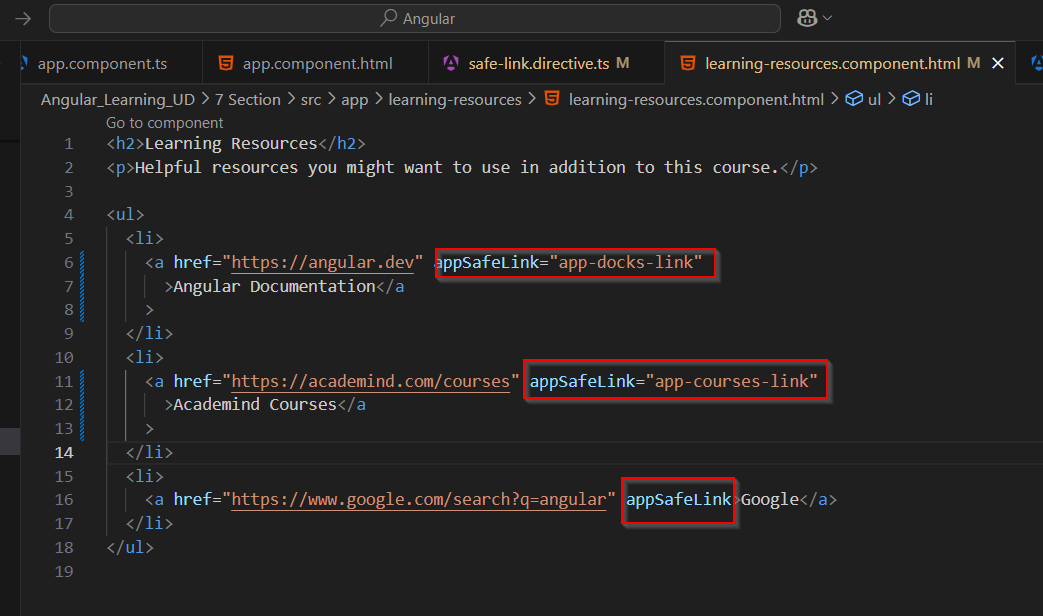


* Changing link address from custom directive.

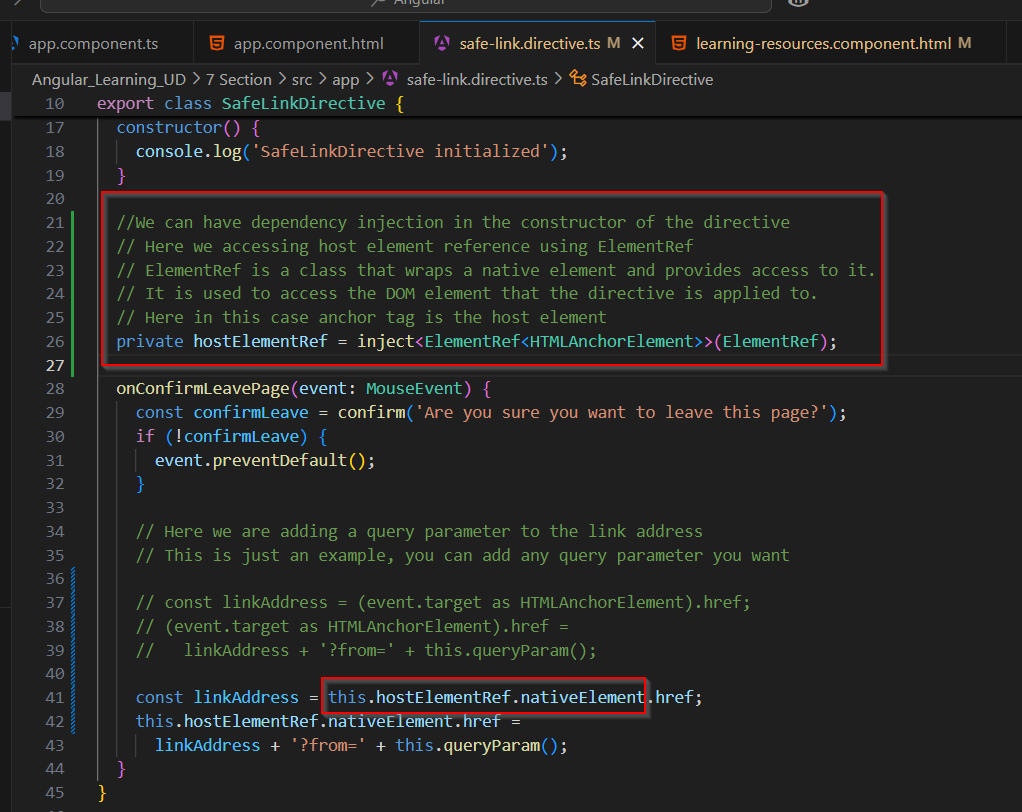


* We can have custom query params also. And for this, we can use inputs in directives.
* Syntax and usage are as below.



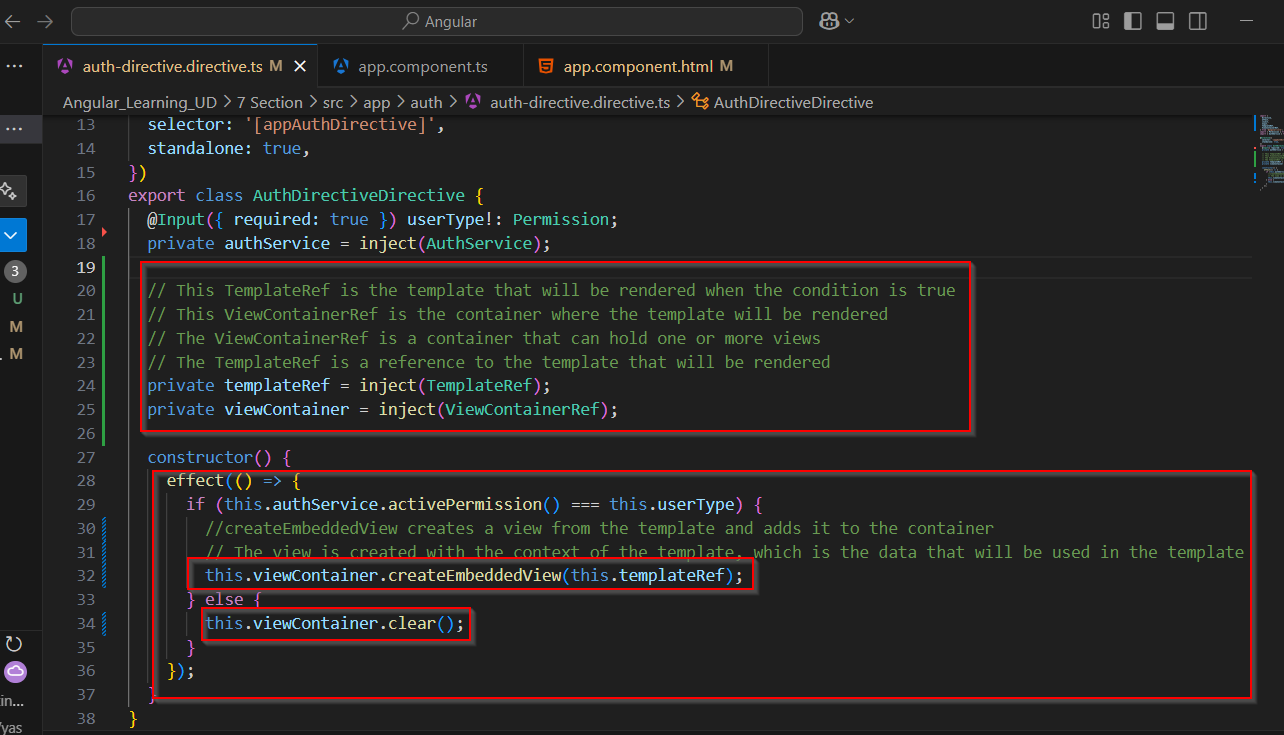


* We can have dependency injection also in directives same as components.
* One of the usages is shown below.



**Structural custom directive**

* In Angular, we can have our structural directive. Just like ngIf.
* For that, we need to create the directive class and use TemplateRef and ViewContainerRef to determine whether to render the element or not.
* Syntax and usages are as below.



* When using a custom structural directive at that time, our elements should be wrapped inside the ng-template built-in component. If \* is not used.



* Instead of using ng-template, we can also use \* for a structural directive, but it’s recommended to use an alias for the input property.
* The usage and syntax are as below.

