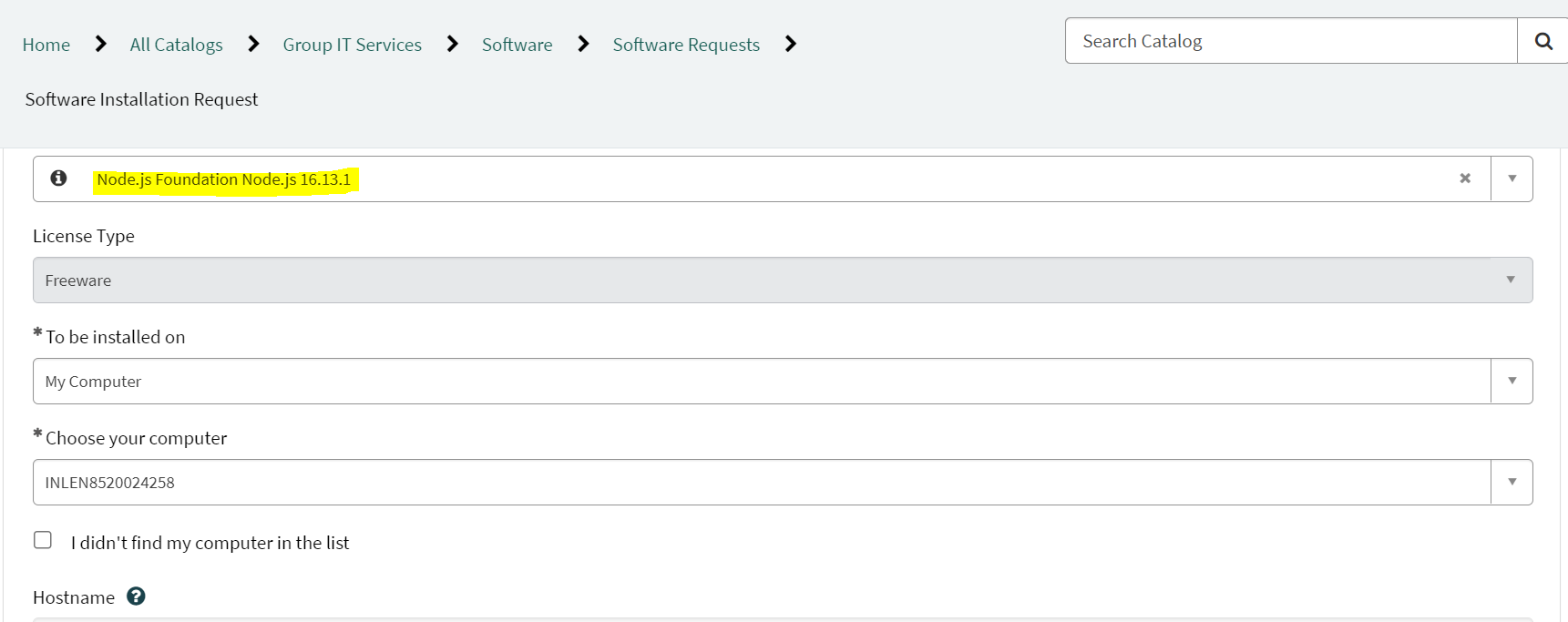
Angular Training Material

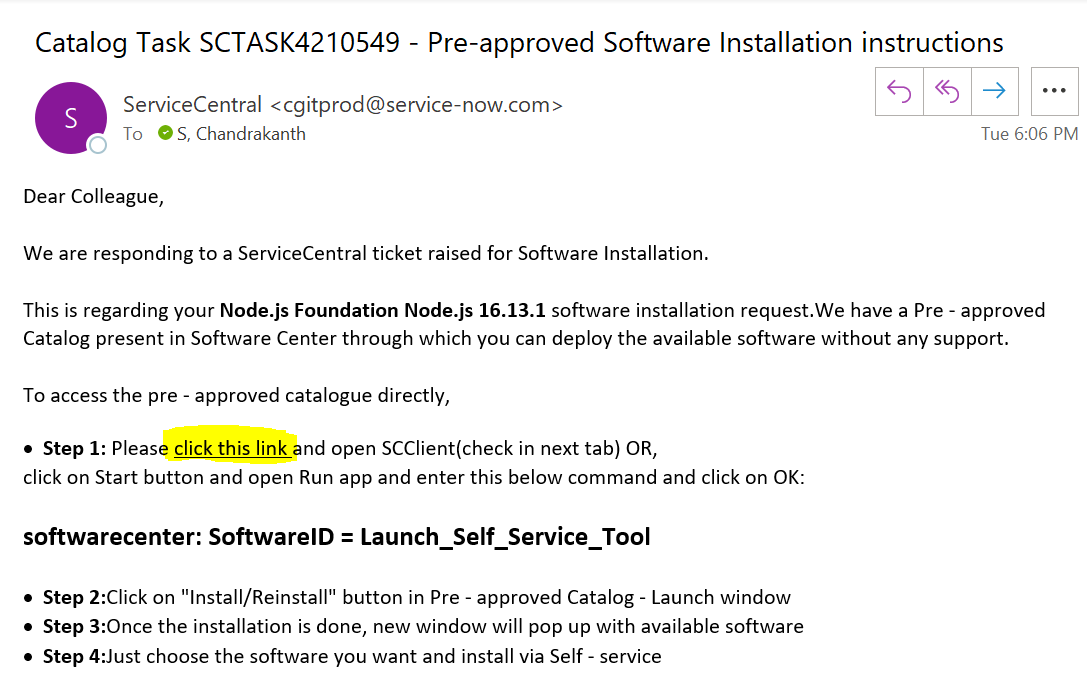
**Installation Process**:

**First** step is to Install **Node.js** in your system please follow below steps

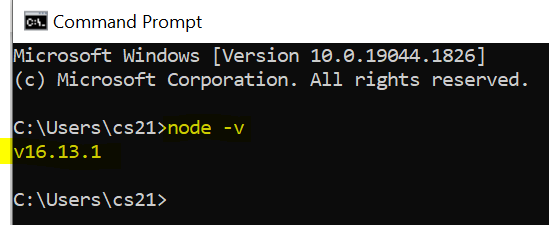
* Type Install Node in ask Adam or open Service Central raise a ticket



* Once you type in all the details and submit, You will receive a mail from Service Central please follow those steps

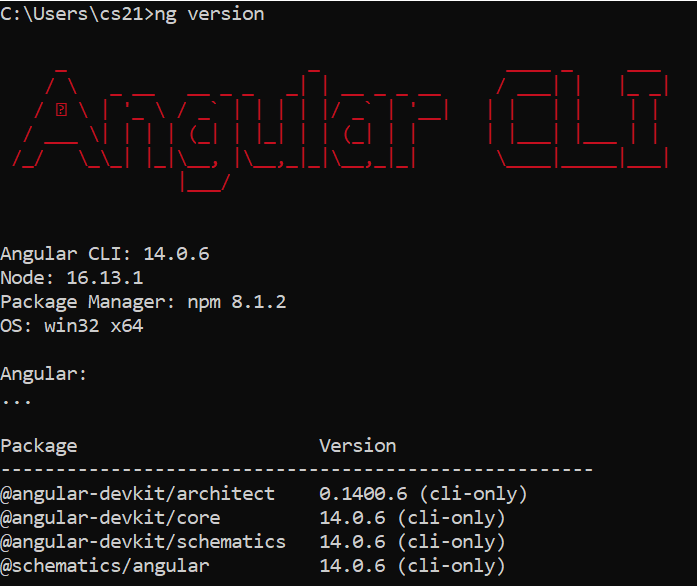


* Once you click on the link and install pop-up open with several apps type in Node in search bar and install, to confirm whether its installed or not run cmd prompt and type “**node -v**” and hit enter



**Second** step is to install **Angular CLI** – to install this you need Admin access, raise a request from Service Central requesting Admin access

* Run “**npm install -g @angular/cli**” or “**npm install -g @angular/cli@latest**” in cmd prompt/powershell once installation is completed type “**ng version**” and below should be shown

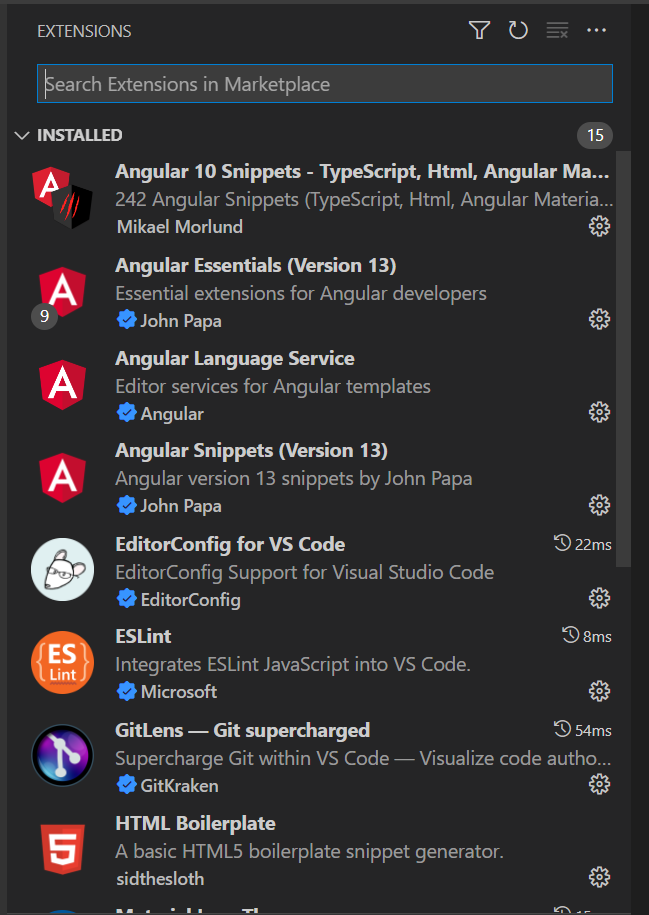


**Third** step is to install **Visual Studio Code** from web which is open source please click on below link to download visual studio

* <https://code.visualstudio.com/docs/?dv=win>
* Once Downloaded run the application and follow below steps:

1. On the left side section click on last Icon which says Extensions or cntrl + shift + x will open Extensions.
2. Search these extensions and add them to studio

* Angular Essentials (Version 13)
* Angular 10 Snippets - TypeScript, Html
* Microsoft Edge Tools for VS Code
* EditorConfig for VS Code
* ESLint
* HTML Boilerplate
* GitLens (Insiders)
* Material Icon Theme
* Npm
* Nx Console
* Prettier - Code formatter
* Winter is Coming Theme (Optional)
* Peacock (Optional)



These extensions will make your coding life easy and will maintain a standard coding practise.

Some of the short cut keys to use Visual Studio code: <https://code.visualstudio.com/shortcuts/keyboard-shortcuts-windows.pdf>

**What is Angular?**

**Angular is a JavaScript UI Binding framework which binds your View i.e., HTML UI and JavaScript Model.**

M

VM

V

(VIEW) (VIEW-MODEL) (MODEL)

This View-Model is what **Angular**

Angular helps us to build SPA (Single Page Application) by using the concept of routing.

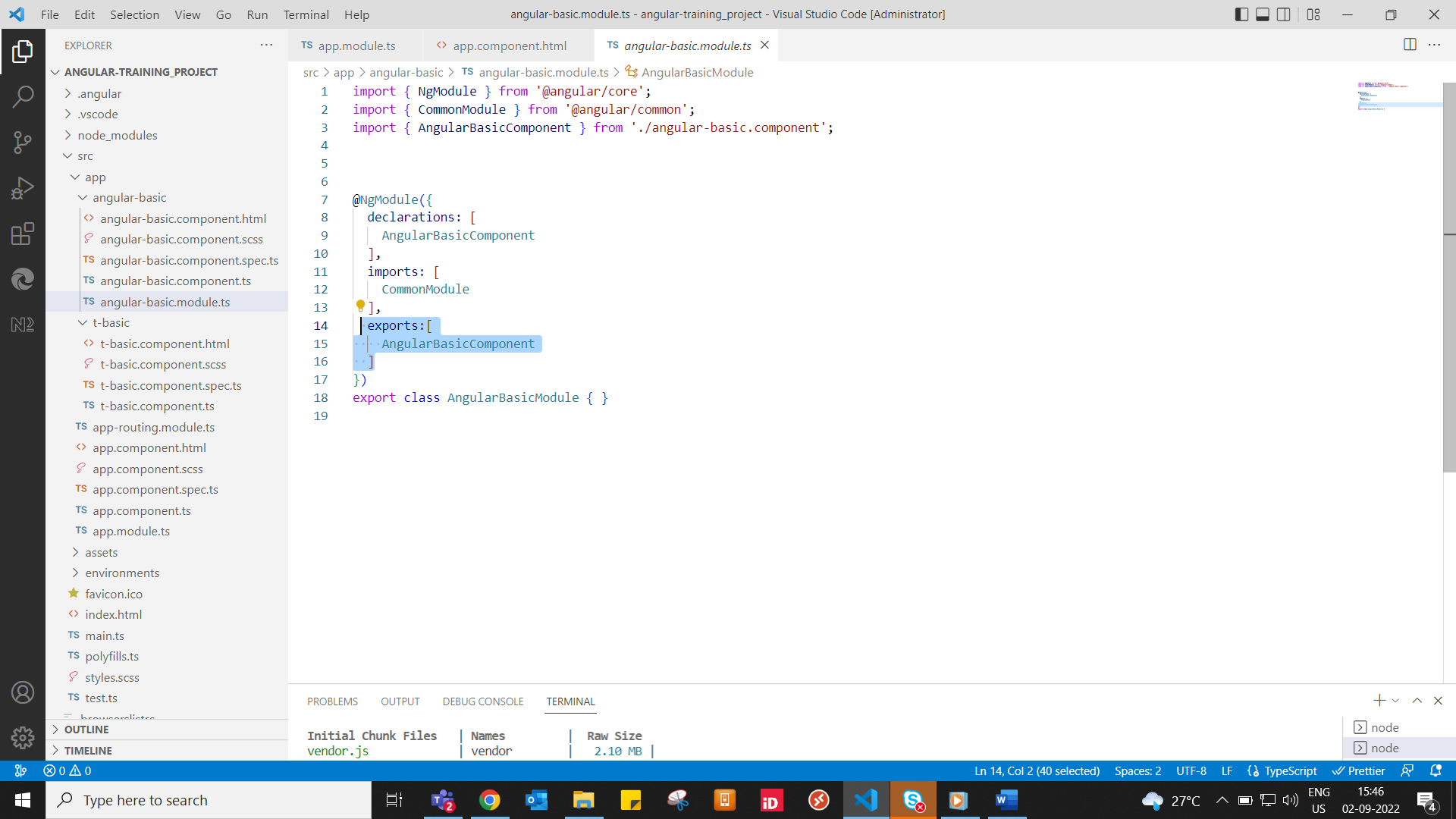
To setup new project:

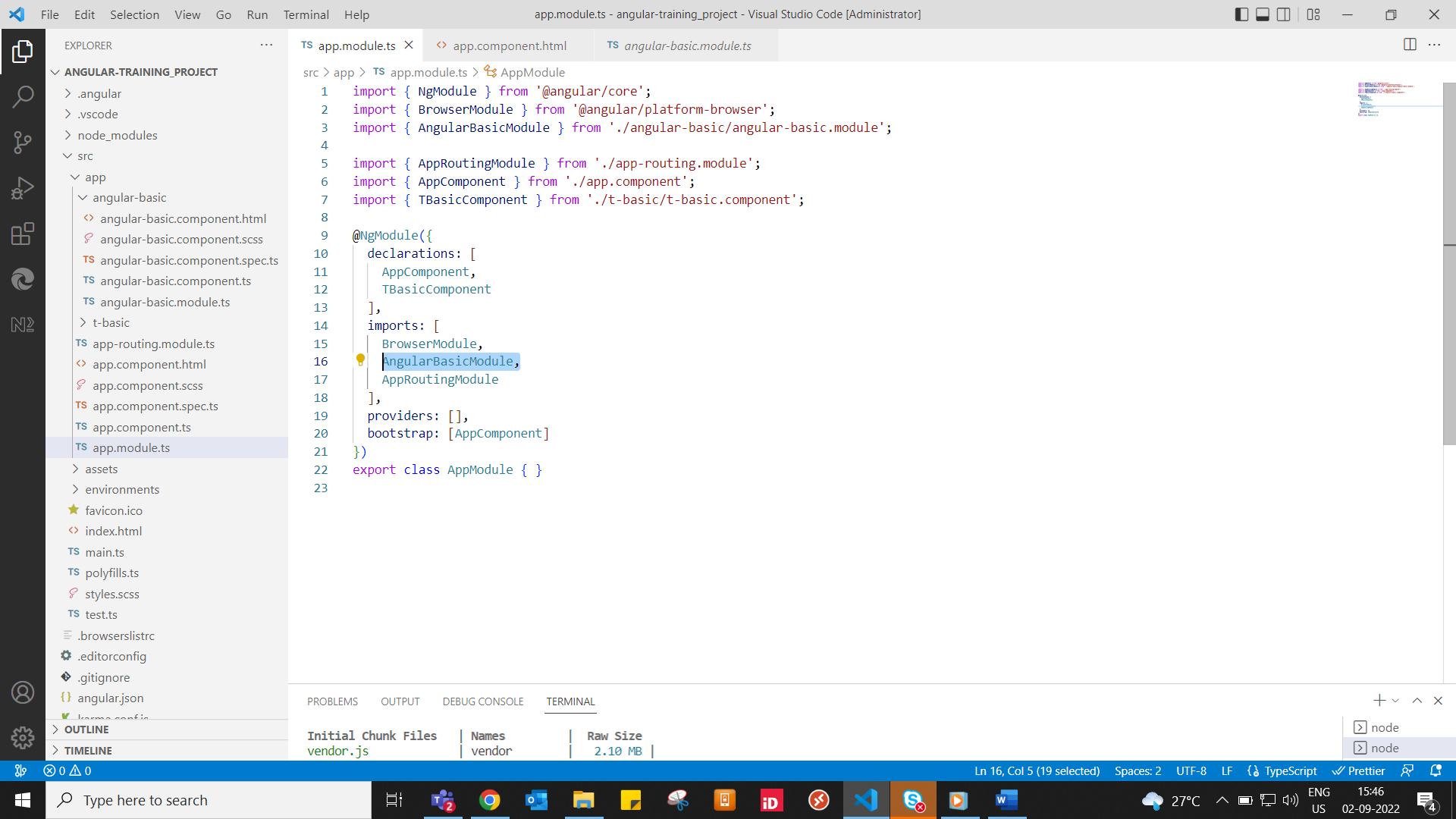
* Open cmd prompt type “**ng new my-angular-project**” my-angular-project is name of project it can be anything according to your requirement.
* To serve the project “**ng serve**” and hit <http://localhost:4200/> from browser.
* Generate new component: “**ng generate component component-name**” component-name can be any.
* Generate new module: “**ng generate module module-name**” module-name can be any.
* Generate new service: “**ng generate service service-name**” service-name can be any.

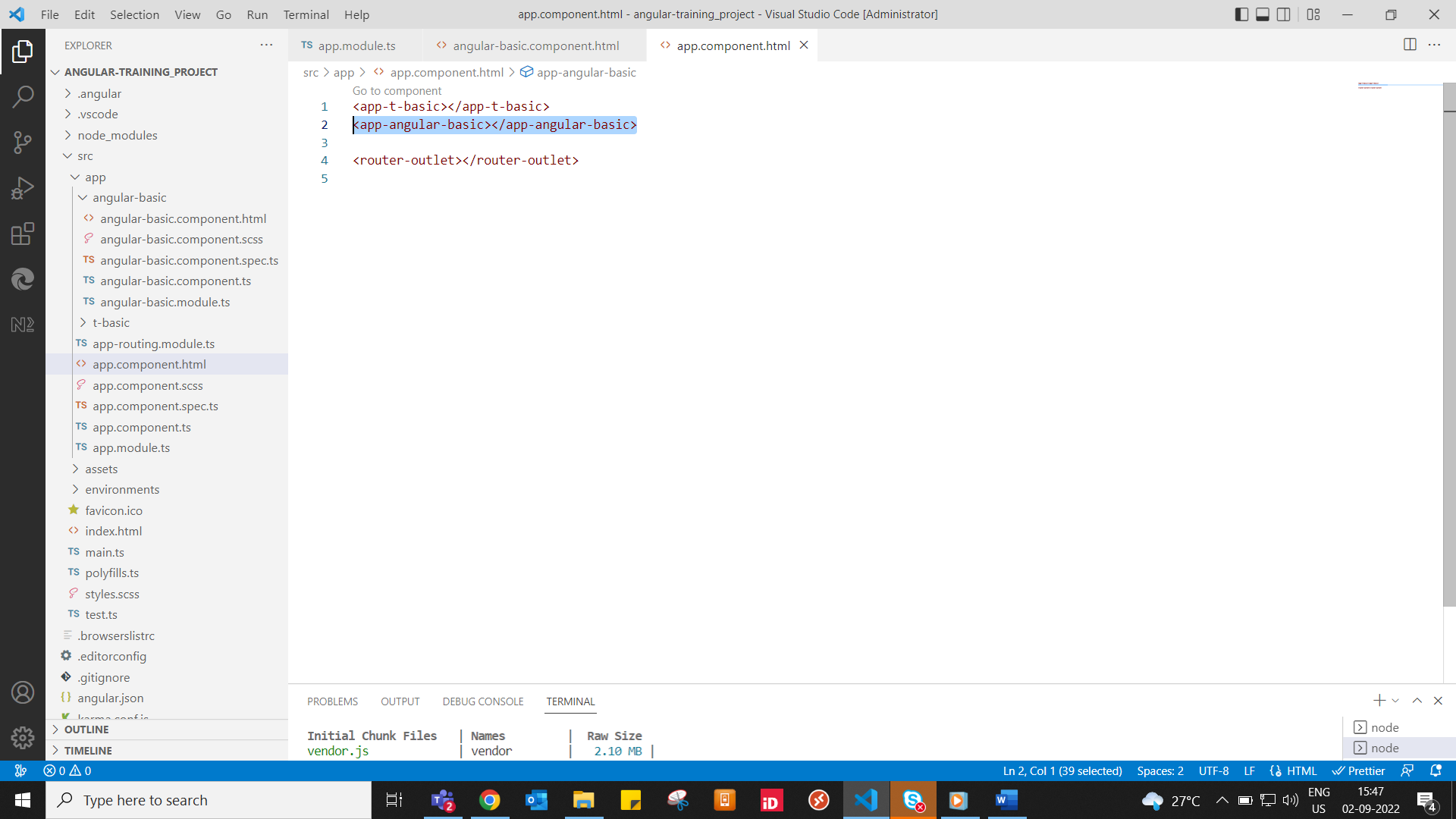
Quick commands to go thorough:

Ex path: C:\Users\PUSHKS\angular-training\_project\src\app> ng serve

* **ng new my-angular-project**
* **ng serve**
* **npm install**
* **ng generate component component-name / ng g component component-name**
* **ng generate module module-name / ng g module module-name**
* **ng generate service service-name / ng g service service-name**



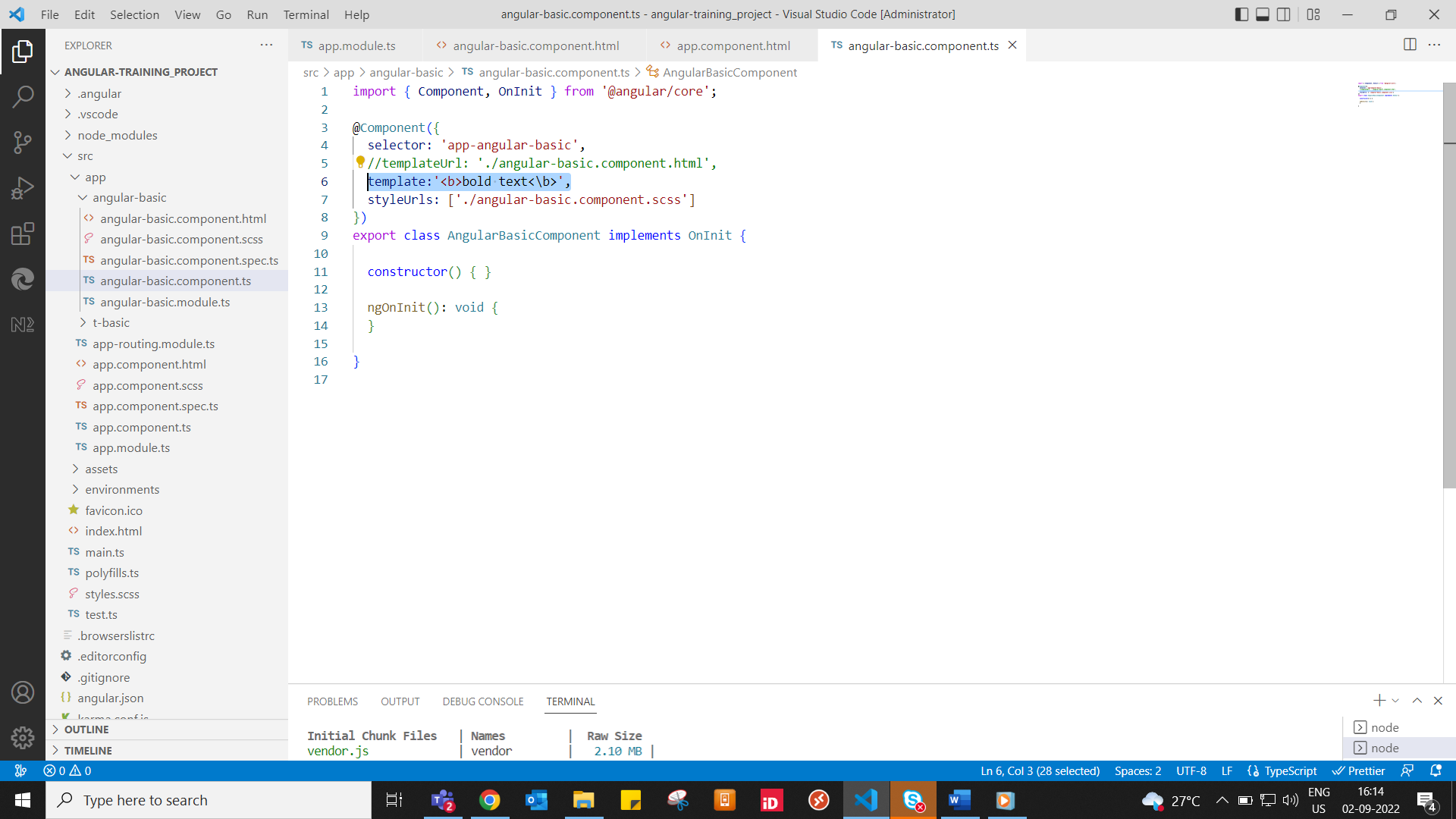


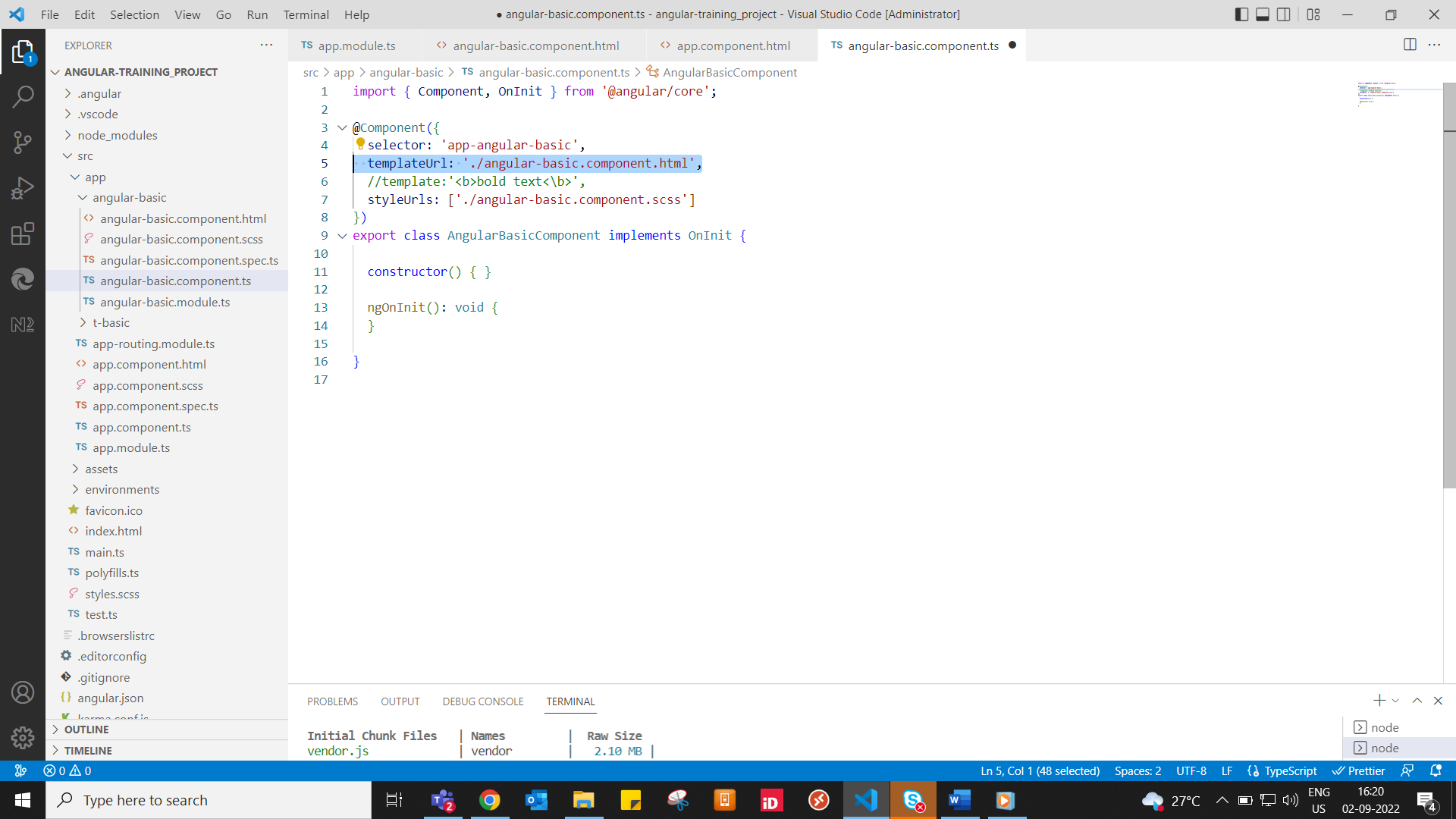


In order make connections we need to add above contents.

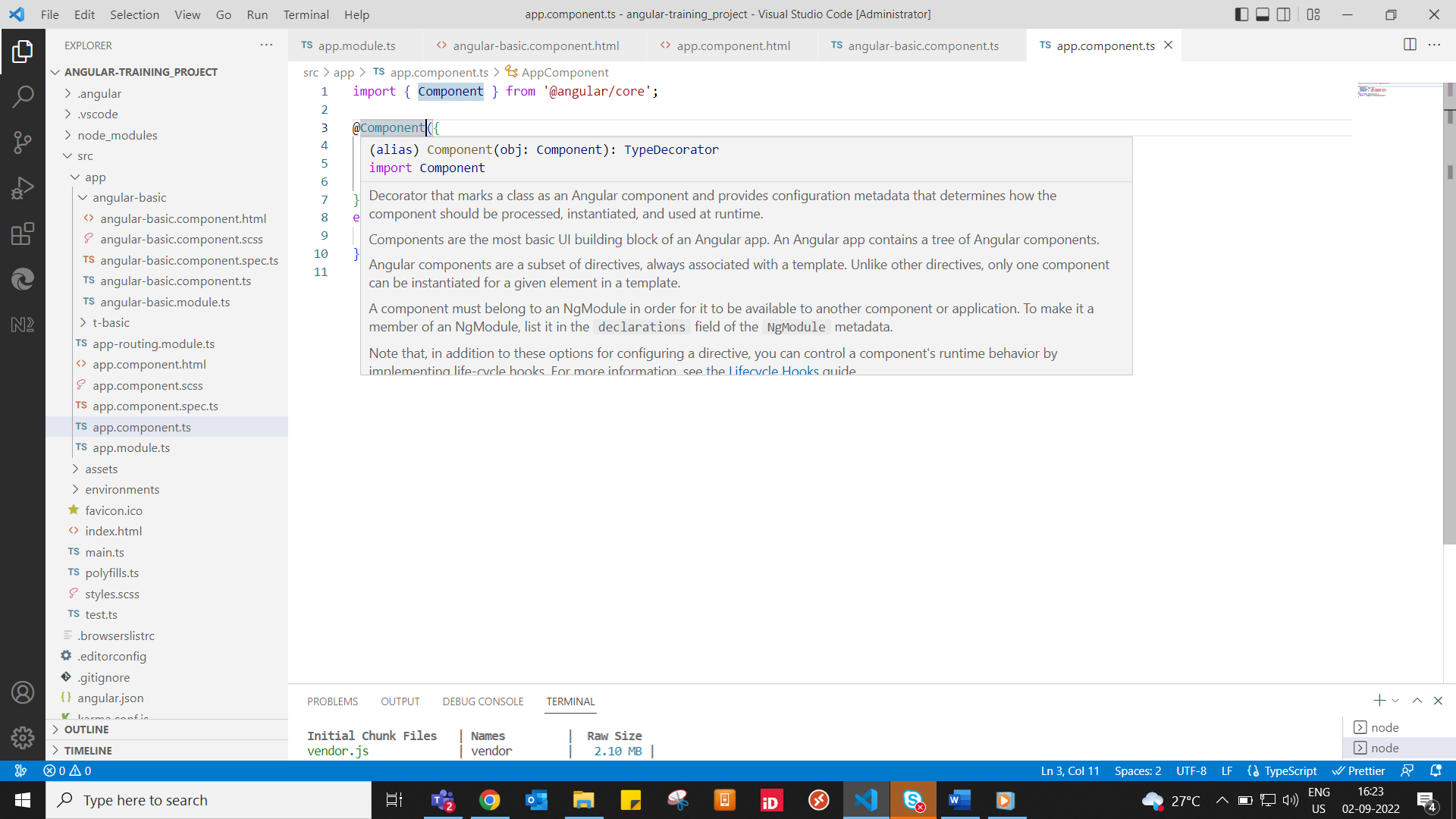
2types to access the html:

By Inline and URL

Inline



URL



@component

What are **directives?**

Directives help you to attach behaviour in HTML DOM / Directives are angular syntaxes which are inside HTML which go and change the behaviour of your HTML DOM.

Three kinds of **Directives: Structural, Attribute and Component directives.**

1. **Structural Directive:** These change the structure of HTML DOM elements; in other words, they add new element in DOM or remove one element in the DOM.

These are the some of the structural directives:

**\*ngIf , \*ngIf-else, \*ngFor, \*ngSwitch.**

1. **Attribute Directive:** These changes the appearance and behaviour of HTML elements like **NgSwitch, NgStyle, NgClass, NgModel.**
2. **Component Directives:** Directives with template, It’s like a user control. Basically components which start with @Component, for example **@Input** and **@Output** are two components and also called as child directives which communicate between the parent context.

**Data Binding:** is the connection bridge between View and the View-Model of the application/ Data Binding plays an important role in communication between a template and its component.

1. **One-way Data Binding**: - Interpolation

Property Binding

Event Binding

Interpolation: Data flows from controller to the view which you can stick or mix inside html elements. Written inside {{}}

Property Binding: Again, data flows from controller to the view but it gets attached to some user input like textbox which is written inside [].

Event Binding: In event binding happens from view to controller where event like click function happens, in case of event binding we use () were data flows from view to controller.

2.) **Two-way Data Binding:** Where the data can flow from view to the component and component to the view. It’s the combination of property binding i.e. [] and event binding i.e. (). regular brackets inside square brackets i.e. [()].

* When you need to concatenate strings, you must use interpolation instead of property binding.
* Property binding is used when you must set an element property to a non-string data value.
* If you use interpolation instead of property binding, the button will always be disabled regardless property value is true or false.

**Dependency Injection (DI):**

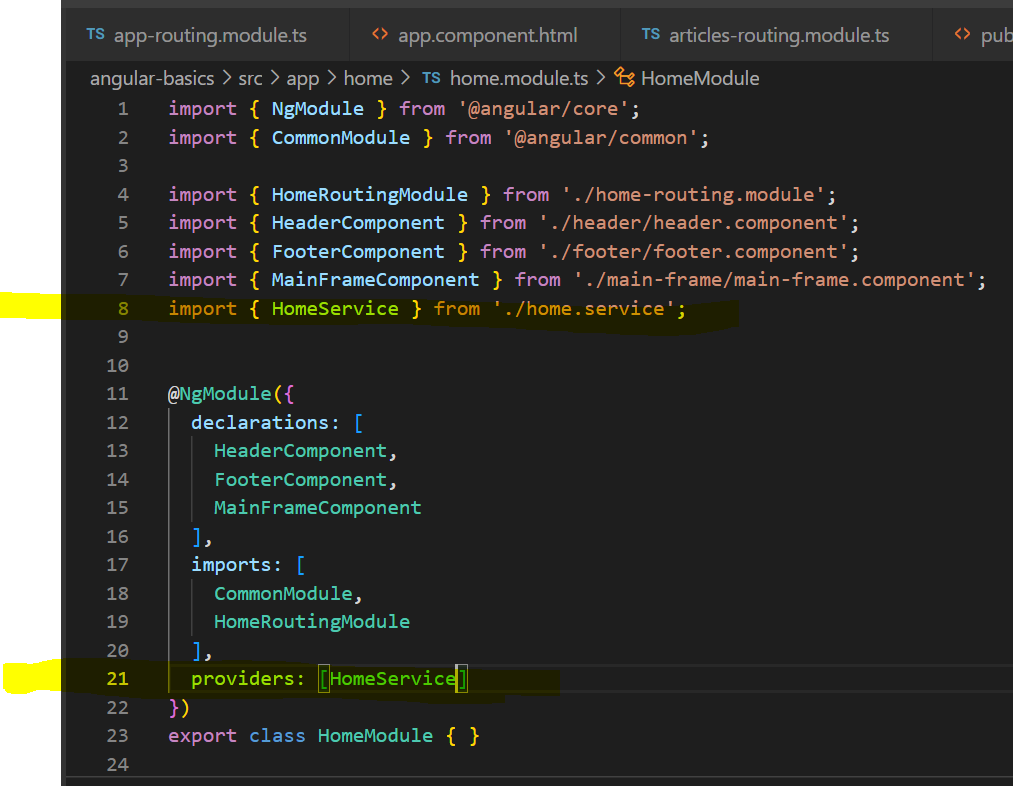
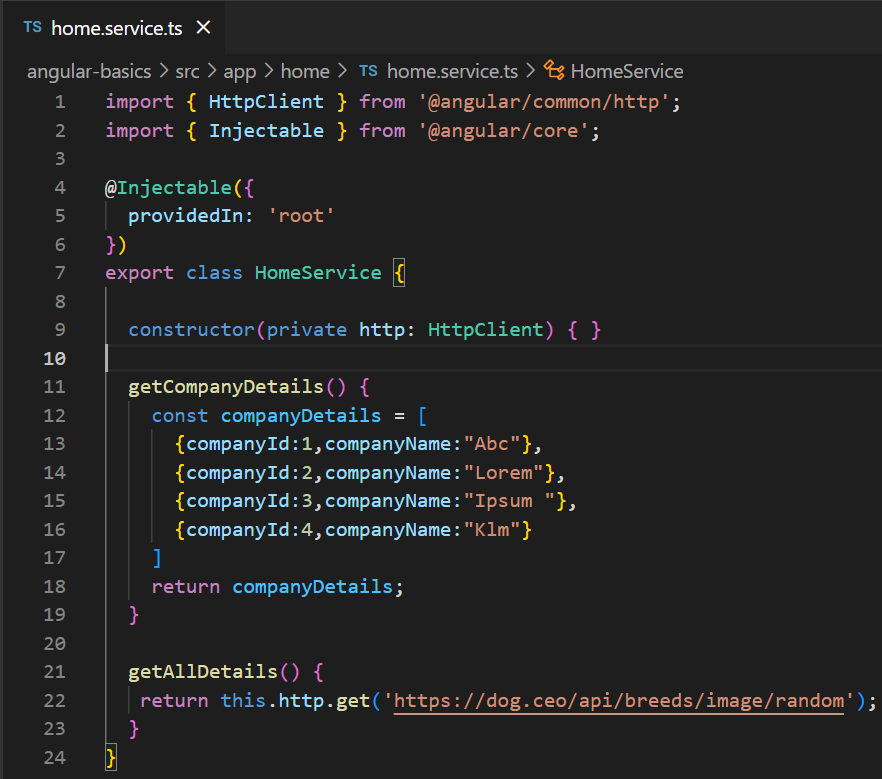
* Dependency Injection is an important application design pattern. DI is the ability to add the functionality of the components at runtime.
* DI framework lets you supply data to a component from a injectable service class, defined in its own file.
* Angular has its own DI framework, which is typically used in the design of angular applications to increase their efficiency & modularity.
* Di is a coding pattern in which a class asks for dependencies from external services rather than creating them itself.
* DI is a design pattern which is used for dynamically providing the dependencies.
* A class which we call as “Services” which are reusable classes which can be shared between Components.
* We will use a decorator called “@Injectable” to let components know that this is a dependency.

**Services:**

* Services allows us to create reusable common shared functionality between various modules and components.
* Services are singleton. Services are injected into application using Dependency Injection mechanism
* We need to create and inject services in components where we want to use them. Services are an abstract layer or process layer which consists of our application business logic.
* Services are commonly used for making HTTP requests to our endpoints APIs to request and receive the response.
* A service can have a value, method or a combination of both.

How to generate a service using CLI:

* **“ng generate service service\_name”:** service\_name is the name of service which you need to provide.
* Once the services are created those need to be imported in module in order to use that i.e. inside module.ts

* **@Injectable({ providedIn: 'root' }):**  @Injectable this decorator informs angular that we can inject it into components that its service class.

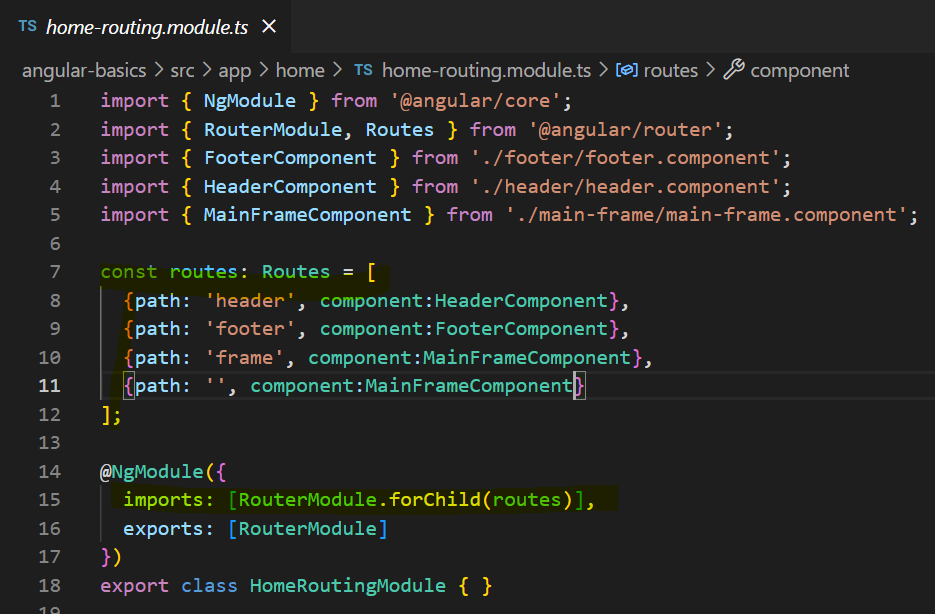
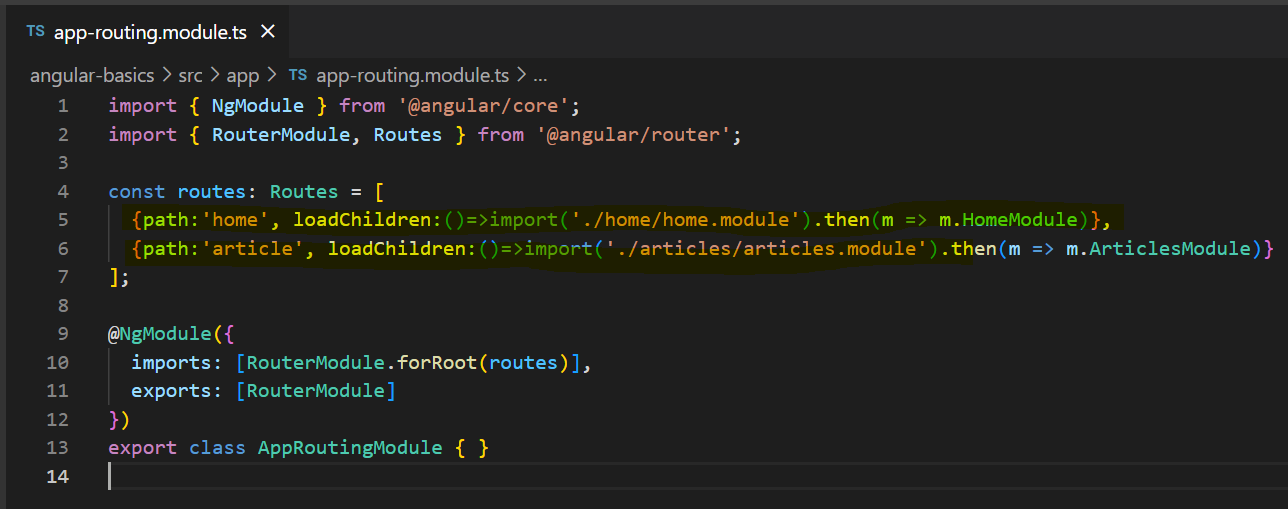
providedIn: 'root': The service is provided in ‘root’ which means its available across the application.

**Routing – Lazy Loading**

* By default, **NgModules** are eagerly loaded, which means that as soon as the app load, so do all the **NgModules**, whether or not they are immediately necessary.
* For large apps with lots of routes, consider lazy loading a design pattern that loads NgModules as needed.
* Lazy loading helps keep initial bundle sizes smaller, which in turn helps decrease load times.
* **loadChildren** – this is a function that uses the dynamic import syntax to import your lazy-loaded module only when its needed.
* Command to generate routing module:

**ng generate module “module-name” –routing**

**Main routing module Newly created routing module**



**Lifecycle Hooks: -**

* Every Angular component has 8 lifecycle hooks, that means each component can be tracked with 8 different lifecycle hooks. These 8 things together are called as component lifecycle hooks
* These are the 8 stages that a component goes through every time we create, we change, or we can use that to track individual component.

**ngOnChanges() :**

* Used in pretty much any component that has an input.
* Called whenever an input value changes.
* Is called the first time before ngOnint().

**ngOnInit():**

* Used to initialize data in a component.
* Called after input values are set when a component is initialized.
* Added to every component by default by the Angular CLI.

**ngDoCheck():**

* Called during all change’s detection runs.
* A run through the view by Angular to update/detect changes.

**ngAfterContentInit():**

* Called only once after first ngDocheck().
* Called after the first run through of initializing content.

**ngAfterContentChecked():**

* Called after every ngDoCheck().
* Waits till after ngAfterContectInit() on first run through.

**ngAfterViewInit():**

* Called after Angular initializes component and child component content.
* Called only once after view is initialized.

**ngAfterViewChecked():**

* Called after all the content is initialized and checked. (Component and child components).
* First call is after ngAfterViewInit().
* Called after every ngAfterContentChecked() call is completed.

**ngOnDestroy():**

* Used to clean up any necessary code when component is removed from the DOM.
* Fairly often used to unsubscribe from things like services.
* Called only once just before components is removed from the DOM.

Some of the most used ones are **ngOnChanges() ngOnInit() ngAfterViewInit() ngOnDestroy().**

**Router Guards: -**

Router guards helps us to present users navigating to parts of an app without proper authorization/authentication. Which means there is some custom logic we write and will check that can user access this route yes/no.

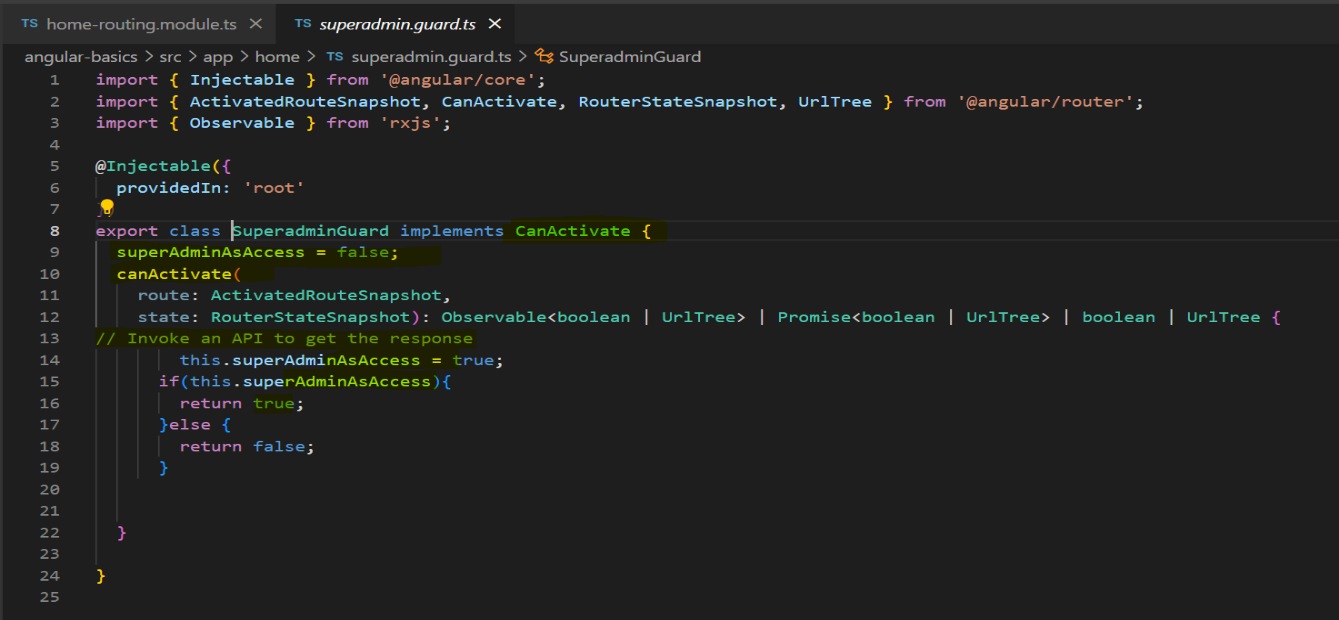
* We use Route Guards to prevent users from navigating to parts of an app without authorization.
* Route Guards are used to secure the route paths
* In most of the cases the routes and screens are protected behind a good authentication system
* The route guards resolve to true or false based on custom logic and functionality.

There are some interfaces in router guards, and they are:

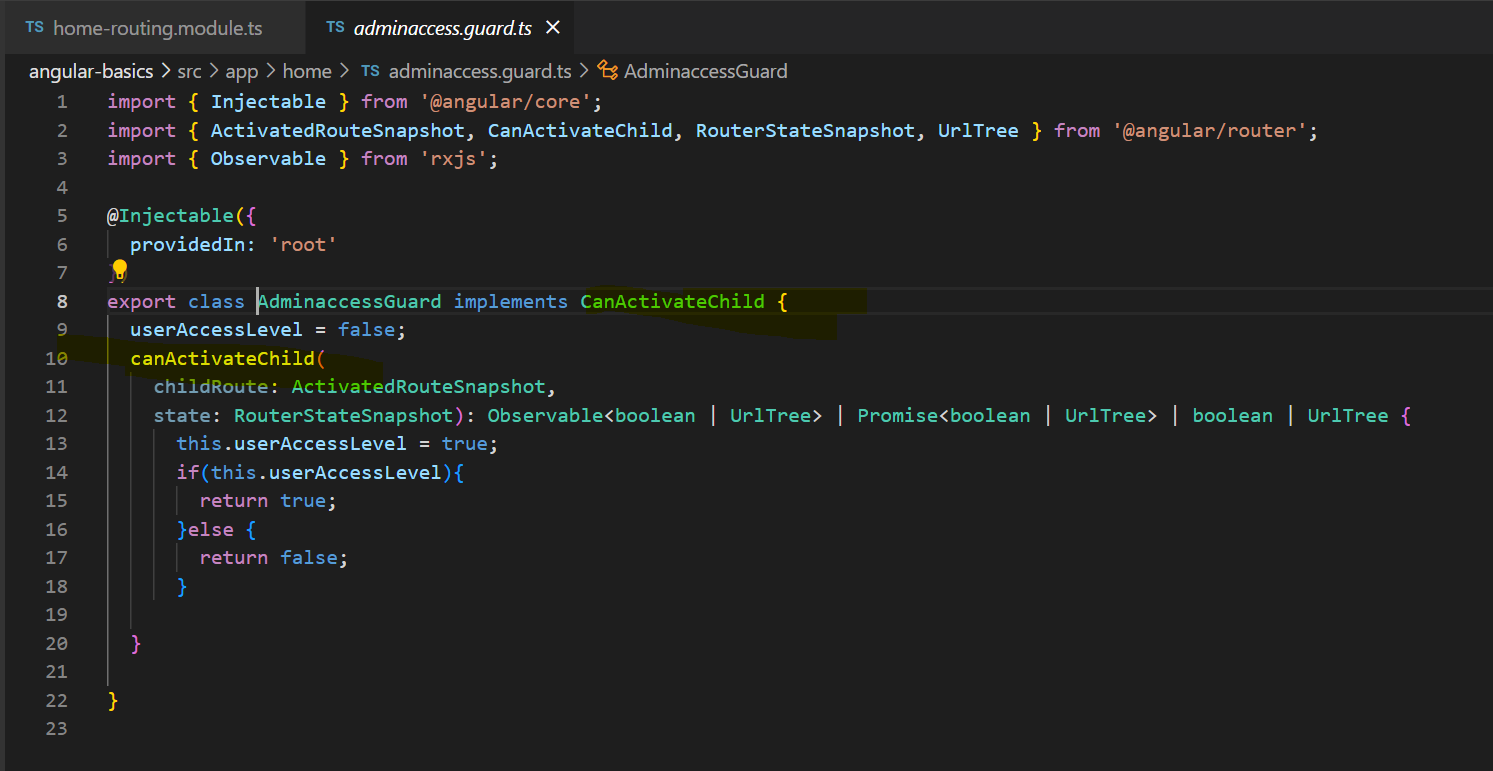
1. canActivate
2. canActivateChild
3. canDeactivate
4. canLoad
5. resolve

To generate Route Guard: **“ng generate guard Guard\_name”**

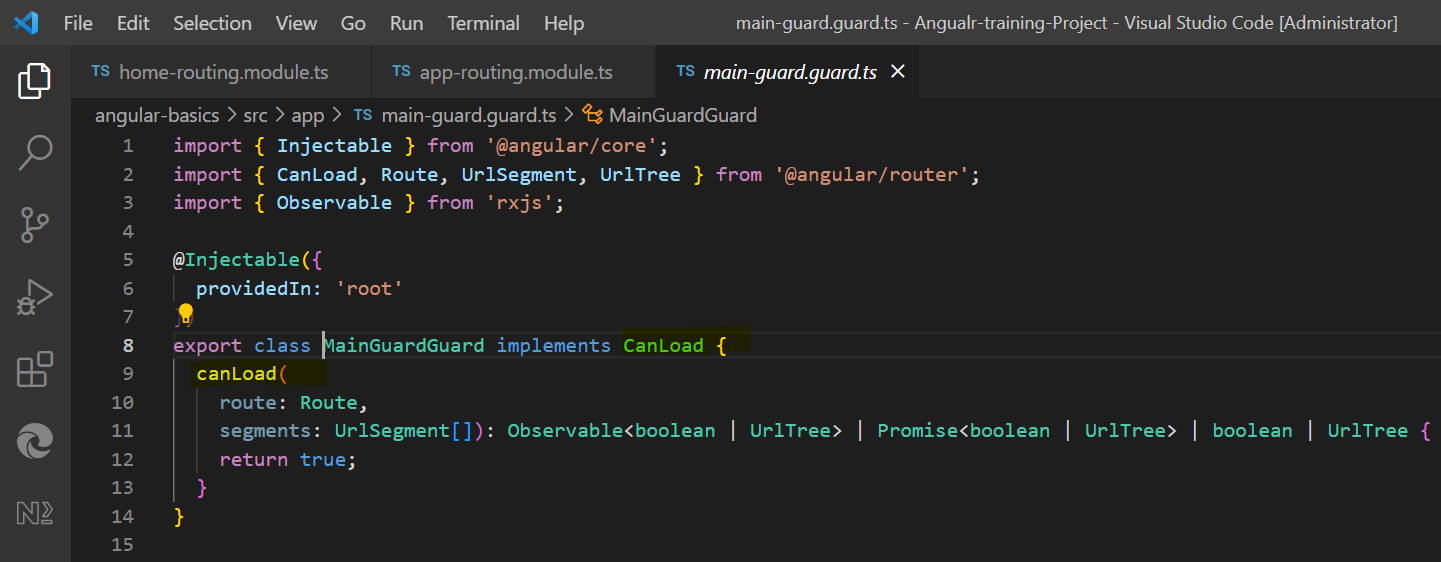
**canActivate:** Checks to see if a user can visit a route with a custom logic returning true

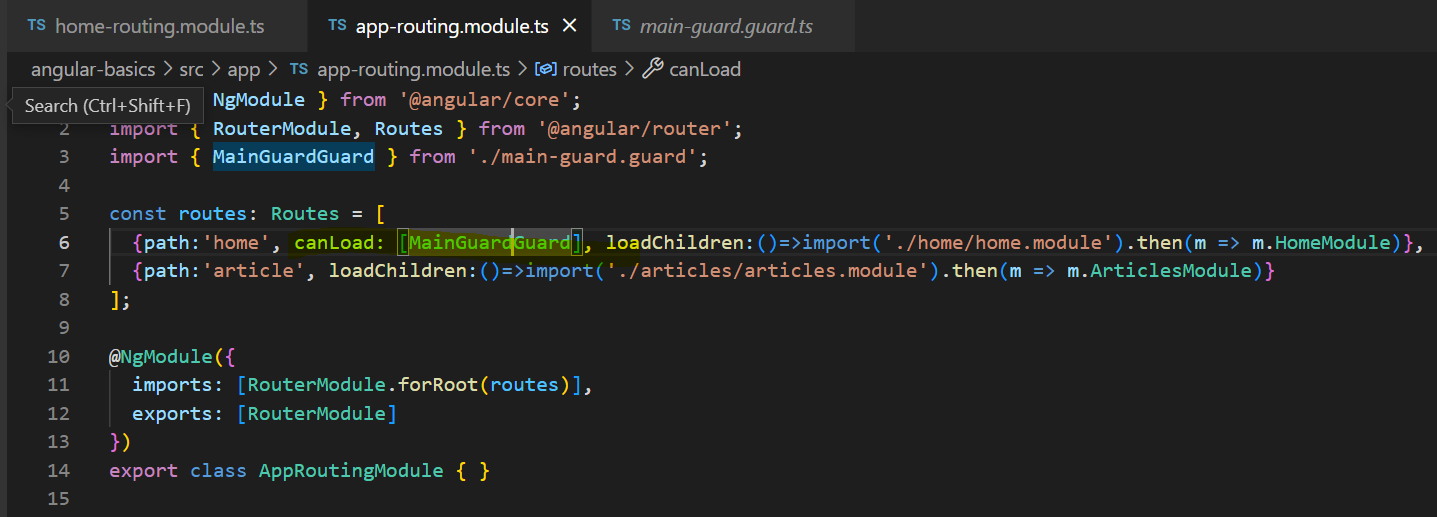


**canActivateChild:** The canActivateChild Guard works like the canActivate guard but the difference is its run before each child route is activated.



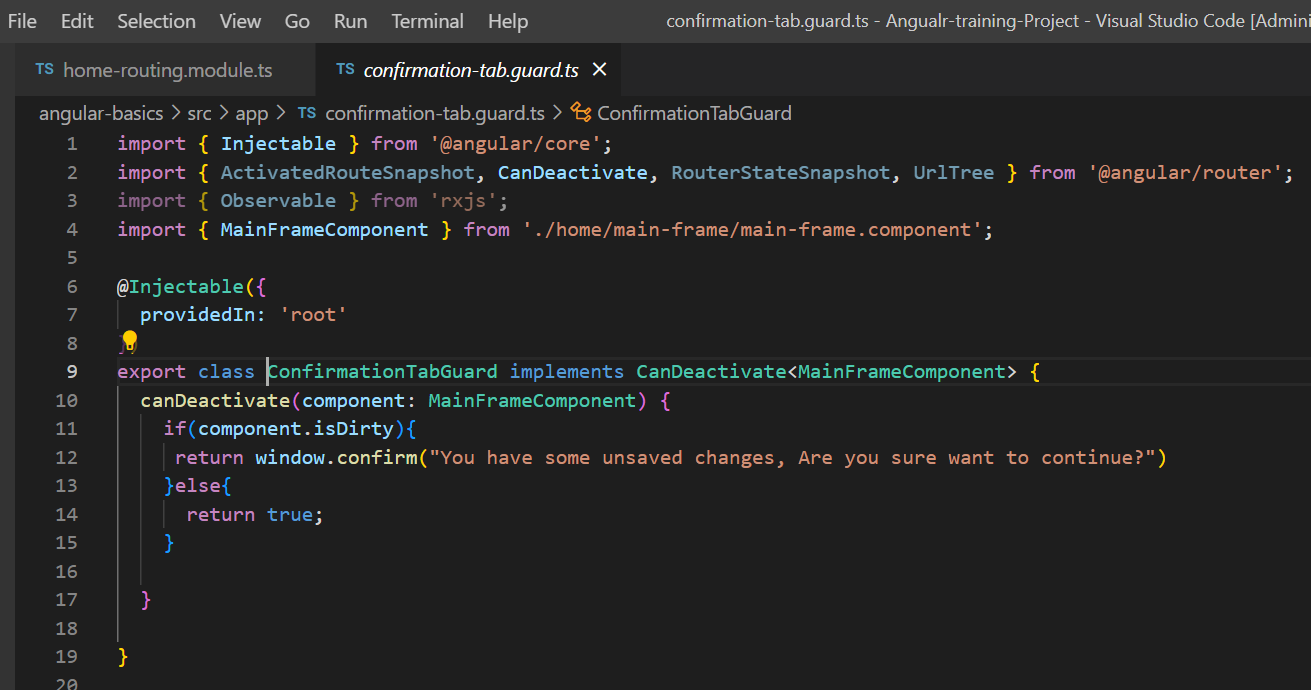
**canLoad:**  This protects the route completely, such as lazy loading the module and protects all the routes associated with that module.





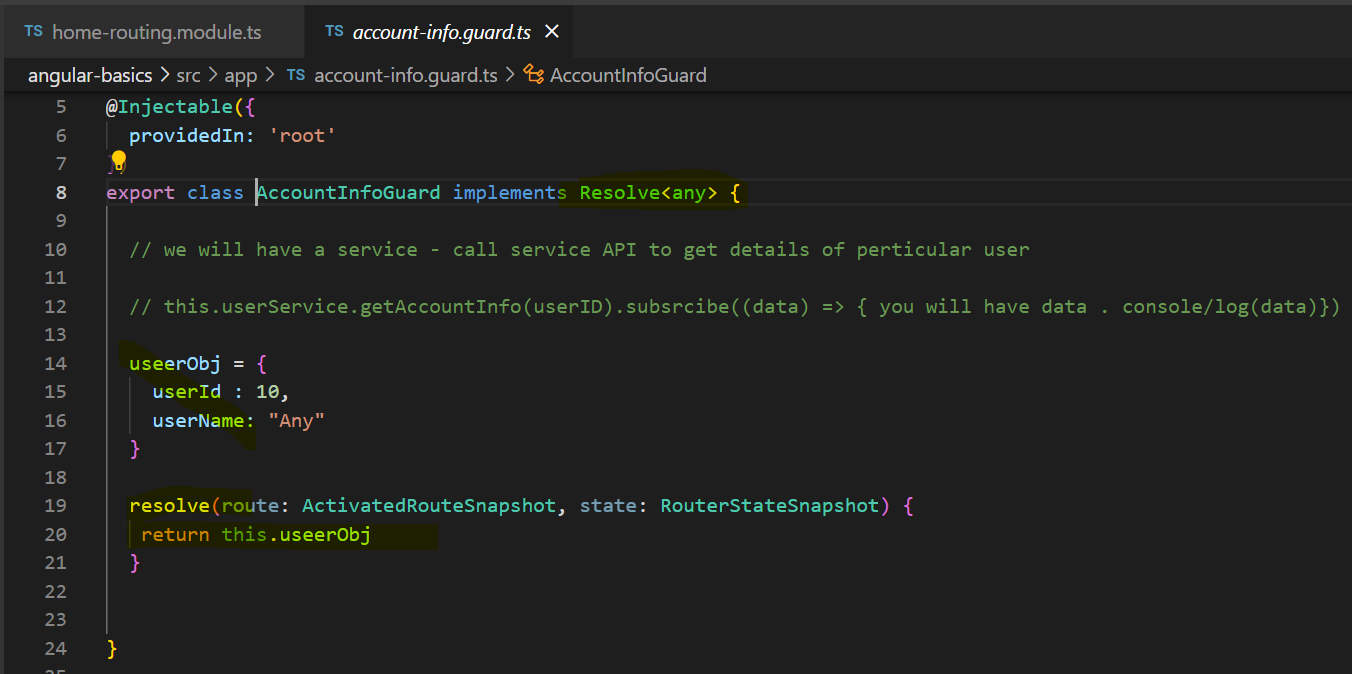
**canDeactiavte:**

* When we want to make sure that user can deactivate a particular route – we will use canDeactivate.
* Interface that a class can implement to be a guard deciding if a route can be deactivated.
* If all guards return true, navigation is called.

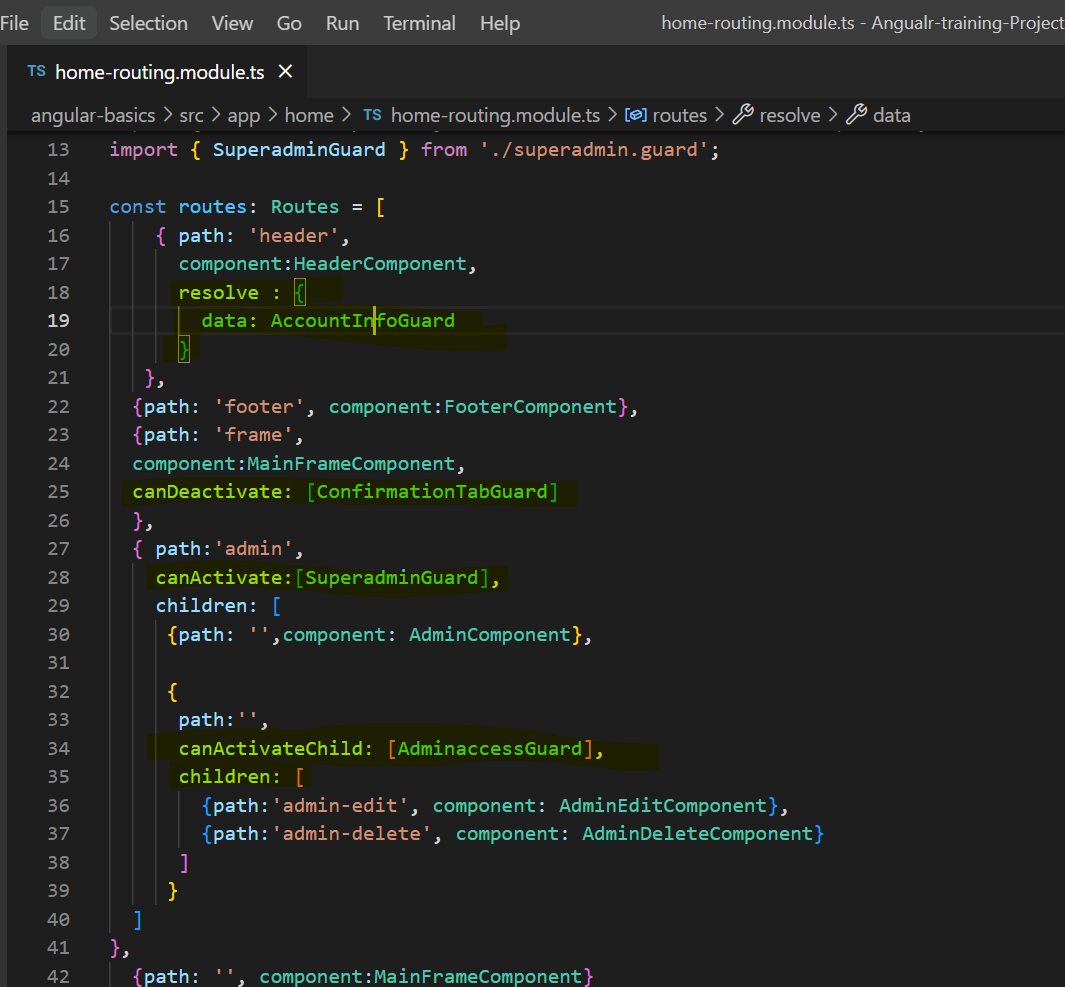


**Resolve:**

* Resolve route guard allows us to provide data needed for a route.
* If some data is mandatory for a component, try using the logic from onInit to resolve.
* Using the activatedRoute.snapshot.data we can access data and process it.



Below is the path in routing-module where every interface has been added:



**Forms:**

* Forms are very integral and essential building blocks of almost apps
* Forms allows us to gather information and data from users
* Good way to interact with the users and almost all websites will need forms in some or other way.
* We can use any CSS framework of our choice – Bootstrap or Material Design.
* Angular support for forms is: 1) Two Data Binding

2) Change Tracking

3) Validations

4) Error handling

Types of forms in Angular

* Static/Template Driven Forms:

1. Easy to use
2. Template driven forms are simple and straight forward
3. All the validations, form elements are all defined in template file
4. We will need to import **FormsModule** in app module to work with template driven forms
5. To reset form: formValue.resetForm()
6. To setvalue: formvalue.serValue()
7. Validations here are:
8. Disable
9. Required
10. minLength
11. maxLength
12. checked

* Dynamic/Reactive Forms

1. All the form elements, user interactions and validations are handled in component class
2. We will make use of Angular’s built in **formGroup** and **formControl** and **FormBuilder**
3. Can control better data binding
4. Exclusive define custom regular expression patterns of error handling
5. We will need to import **ReactiveFormsModule** in our app module
6. Very flexible and allows users to define, develop complex requirements of forms
7. More logic in the component class and less in HTML markup itself

Using of Angular Material:

To install angular material in your system, do “**ng add @angular/material**”

Once done do import all these below modules to use or you can use particular module and start working:

MatAutocompleteModule,

    MatCheckboxModule,

    MatDatepickerModule,

    MatFormFieldModule,

    MatInputModule,

    MatRadioModule,

    MatSelectModule,

    MatSliderModule,

    MatSlideToggleModule,

    MatMenuModule,

    MatSidenavModule,

    MatToolbarModule,

    MatCardModule,

    MatDividerModule,

    MatExpansionModule,

    MatGridListModule,

    MatListModule,

    MatStepperModule,

    MatTabsModule,

    MatTreeModule,

    MatButtonModule,

    MatButtonToggleModule,

    MatBadgeModule,

    MatChipsModule,

    MatIconModule,

    MatProgressSpinnerModule,

    MatProgressBarModule,

    MatRippleModule,

    MatBottomSheetModule,

    MatDialogModule,

    MatSnackBarModule,

    MatTooltipModule,

    MatPaginatorModule,

    MatSortModule,

    MatTableModule

----------------------------------------------------X---------------------------------------------------

If you have reached to end of this, please try to complete the below assessment:

* Create a project for students which shall show list of student’s data
* Create a routing module which will be main for students’ data, load this module lazy in app-routing-module
* Create two modules inside routing module and lazy load them
* First module should show data of students in a table/ using div elements should be displayed in format way in UI
* Second module should contain two components for both add and edit students’ data.
* Create a service which shall contain array of data which you should access that as a student’s data and use it across the two-modules.
* For add-component create form using reactive forms and try adding all the validations and enable submit and store the data.
* For edit-component should have a table which should be inline editable and use reactive forms to inline edit data and submit and add validations for submit.
* Create navigation links in app-html file so that you can access each modules/component, use auth-guard to navigate from one component to other.
* You can follow the below structure to create:

**Students-routing-module.ts**

**==> students-data.module.ts**

**==> students-lists.component.ts**

**==> add-edit-students-data.module.ts**

**==> add-studentsdata.component.ts**

**==> edit-studentsdata.component.ts**

**==> students-data.service.ts**

**Create auth-guard inside module to navigate from one component to other.**

**Please go through the KT videos if your stuck or you please let me know in case of any quires.**