

Angular TS

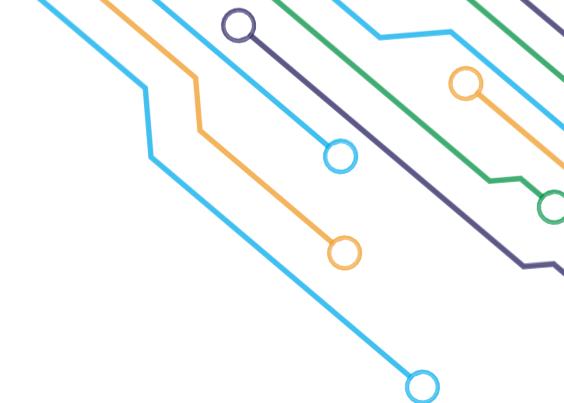
Education and Training Solutions 2023



- 1 What is Angular?
- 2 Advantages of Angular.
- 3 Angular Versions.
- 4 Single-Page Application.
- 5 Create Angular Project.
- 6 The Flow of Execution of Angular App.
- 7 Component in Angular.
- 8 Generate a new component.



What is Angular?



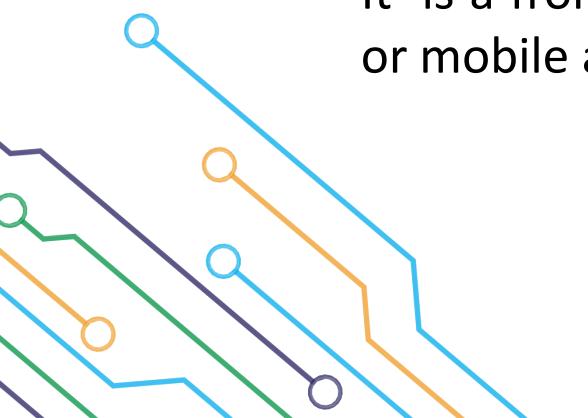
Overview of Angular

Angular is a development platform for building Single Page Applications.

It uses TypeScript & HTML to build Apps.

The Angular itself is written using TypeScript.

It is a front-end framework comes with every feature you need to build a complex web or mobile application.

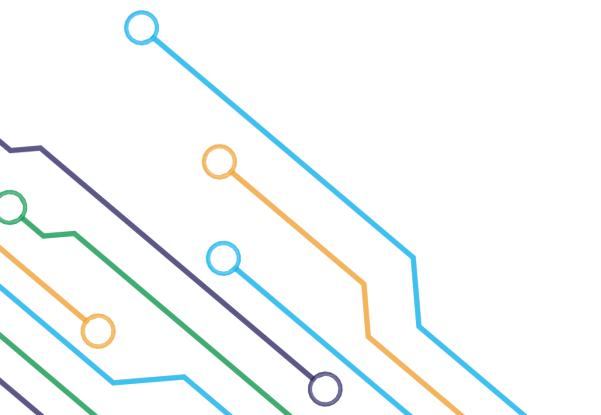




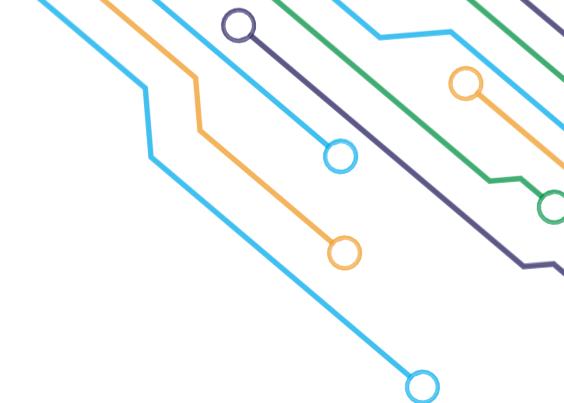
Overview of Angular

It comes with features like Component, Directives, Forms, Pipes, HTTP Services, Dependency Injection, etc.

Angular website:
<https://angular.io/>



Advantages of Angular



Advantages of Angular

1. Comprehensive:

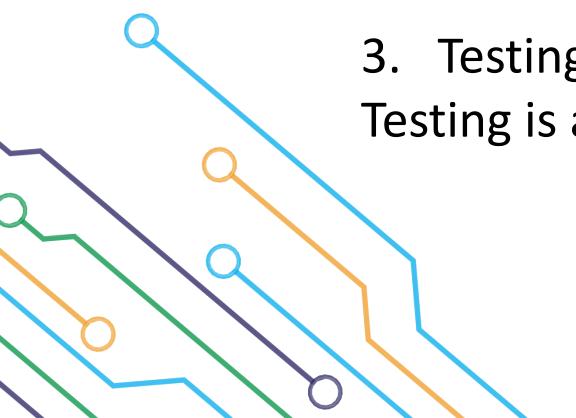
The angular framework is a full-featured framework that provides out-of-the-box solutions for server communication, routing, and more.

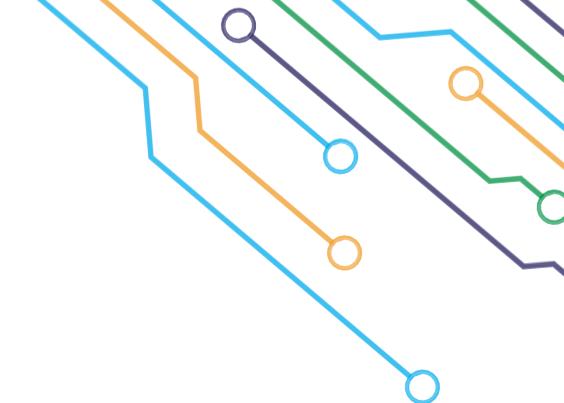
2. Browser Compatibility:

Angular is cross-platform and compatible with multiple browsers.

3. Testing:

Testing is a first-class tool, and Angular was built to be testable from the beginning.





Advantages of Angular

4. Custom Components:

Angular allows users to build their own components that can pack functionality along with rendering logic into reusable pieces.

5. Data Binding:

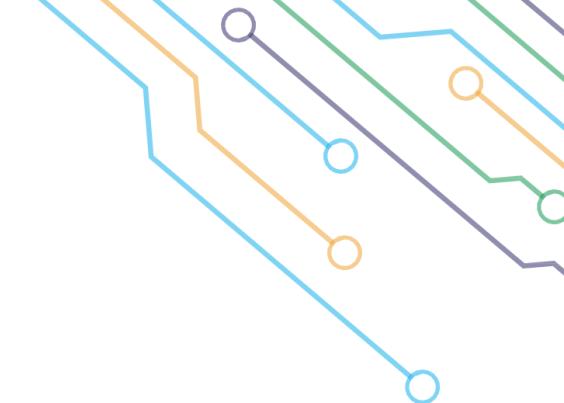
Angular allows users to effortlessly move data from JavaScript/TypeScript code to the view and react to user events without having to write any code manually.

6. Dependency Injection:

Allows users to write modular services and inject them wherever they are needed.



Angular Versions



Angular Versions

The early version of Angular was named AngularJS.
Then later it was renamed just Angular.

Angular Versions:

- AngularJS 1. X
- Angular 2
- Angular 3
- Angular 4
- Angular 5
- Angular 6
- Angular 7



Angular Versions:

- Angular 8
- Angular 9
- Angular 10
- Angular 11
- Angular 12
- Angular 13
- Angular 14
- Angular 15

Angular is constantly growing with better features and faster performance.

Single-Page Application

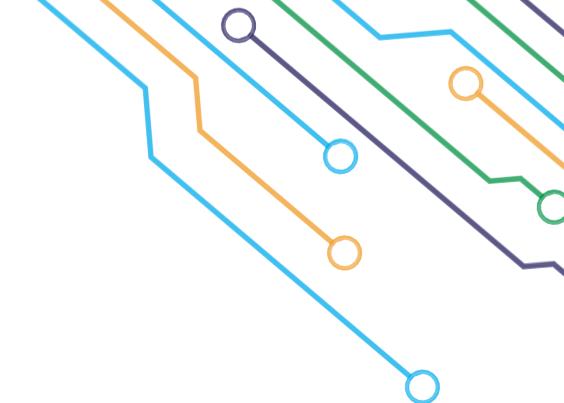
Single-Page Application

vs.

Multi-Page Application

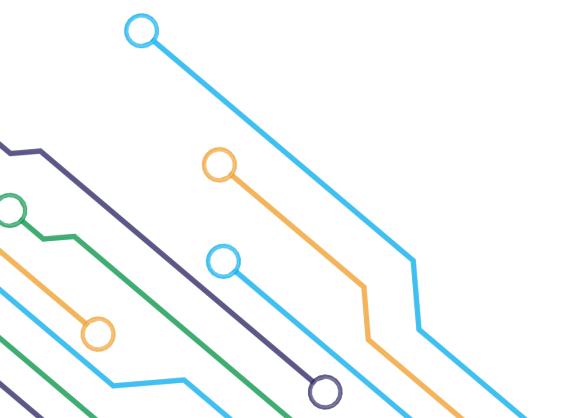
Multi-Page Applications

Multi-Page Applications (MPA) were traditionally used, where every time you clicked on a link, a new page was loaded from the server. Additionally, it was time-consuming and increased server load, which slowed the website.

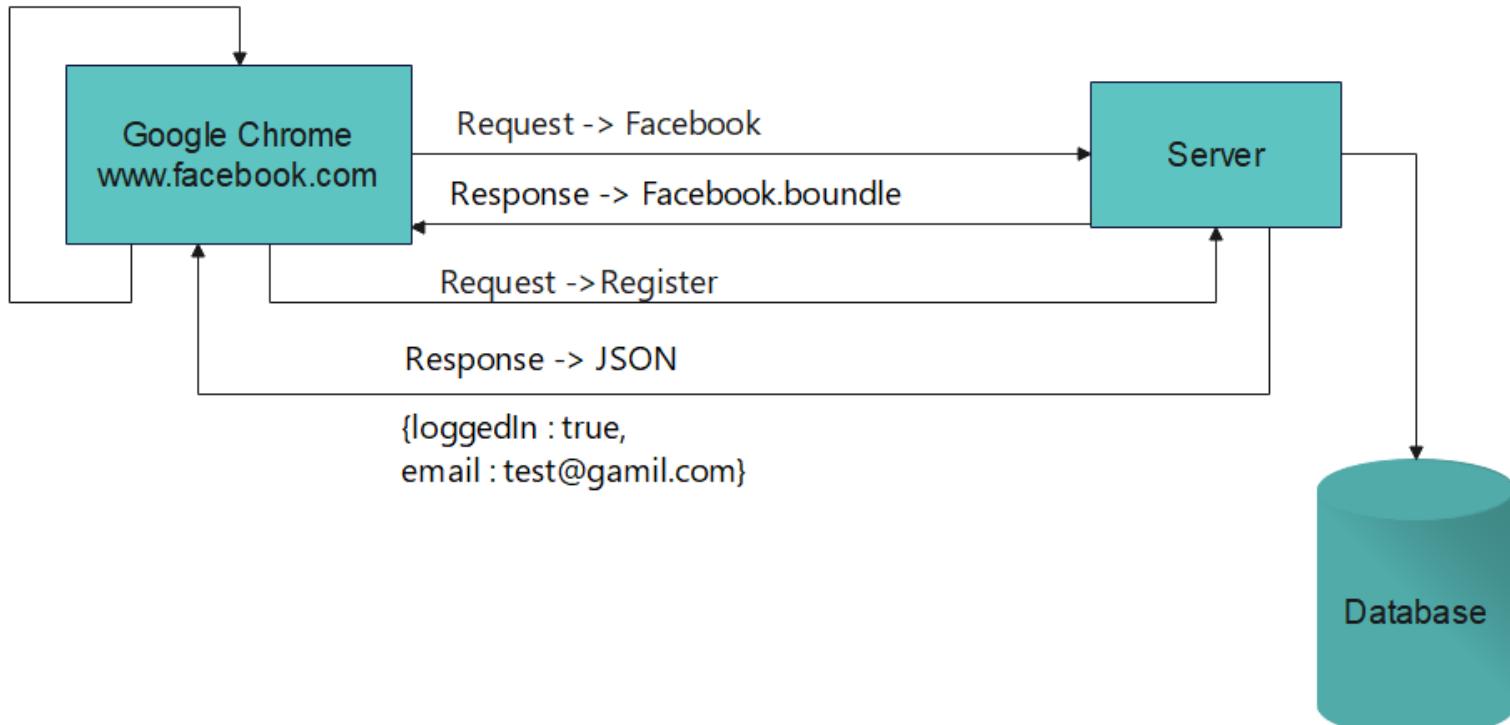


Single Page Applications

The concept of a single page application refers to a web application that loads a single HTML page and only a part of the page gets updated on each mouse click rather than the entire page. There is no reloading of the page or transfer of control to another page during the process. The result is high performance and faster loading pages.



Single Page Application





Facebook



Gmail



Google Maps



Twitter



Google Drive



Git Hub

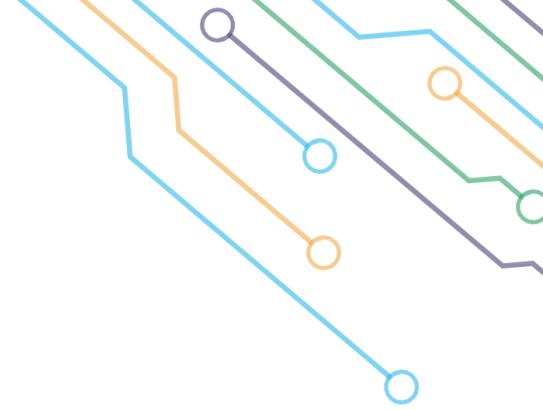


PayPal



Angular CLI

Angular CLI is a command-line tool for developing, scaffolding and maintaining Angular applications.



Angular CLI

To install the Angular CLI - The CLI tool for Angular:
→ `npm i -g @angular/cli`

Read more about the angular package:
<https://www.npmjs.com/package/@angular/cli>



Create Angular Project



Create Angular Project

After installing the angular package (@angular/cli), use this command to create the Angular project:

→ `ng new project_name`

- PS C:\Users\d.kanaan.ext\Desktop> `ng new TheLearningHub`
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? CSS





Create Angular Project

To run the project, use this command:

→ ng serve -o

```
PS C:\Users\d.kanaan.ext\Desktop\TheLearningHub> ng serve -o
```

```
? Would you like to share pseudonymous usage data about this project with the Angular Team  
at Google under Google's Privacy Policy at https://policies.google.com/privacy. For more  
details and how to change this setting, see https://angular.io/analytics. Yes
```





Create Angular Project

To know what the Angular version , use this command :

→ ng version

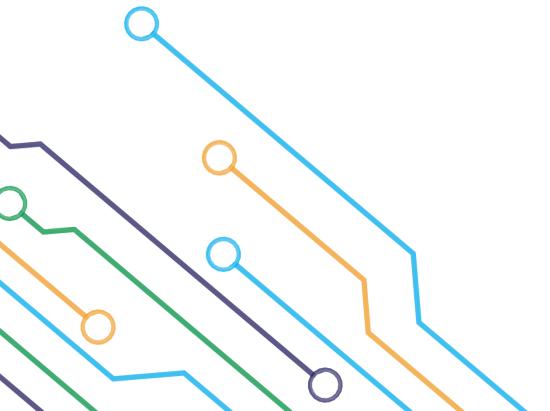
```
PS C:\Users\d.kanaan.ext\Desktop\TheLearningHub> ng version
```



```
Angular CLI: 16.0.4
Node: 18.16.0
Package Manager: npm 9.5.1
OS: win32 x64
```

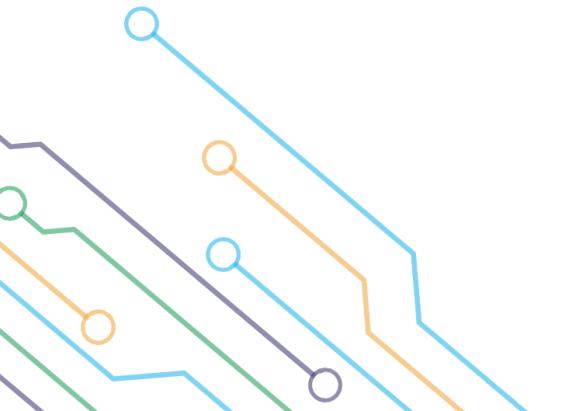
```
Angular: 16.0.4
... animations, cli, common, compiler, compiler-cli, core, forms
... platform-browser, platform-browser-dynamic, router
```

Package	Version
<hr/>	
@angular-devkit/architect	0.1600.4
@angular-devkit/build-angular	16.0.4



Exercise

Search about What happens when we do NG serve?



Note: By default, angular project run on port **4200**.

- PS C:\Users\d.kanaan.ext\Desktop\TheLearningHub> `ng serve -o`
✓ Browser application bundle generation complete.

Initial Chunk Files	Names	Raw Size
vendor.js	vendor	2.26 MB
polyfills.js	polyfills	328.94 kB
styles.css, styles.js	styles	226.38 kB
main.js	main	48.12 kB
runtime.js	runtime	6.53 kB

Initial Total | 2.86 MB

Build at: 2023-06-05T13:05:58.878Z - Hash: 5339c3b817c18923 - Time: 7629ms

** Angular Live Development Server is listening on localhost:4200, open your browser on <http://localhost:4200/> **

✓ Compiled successfully.



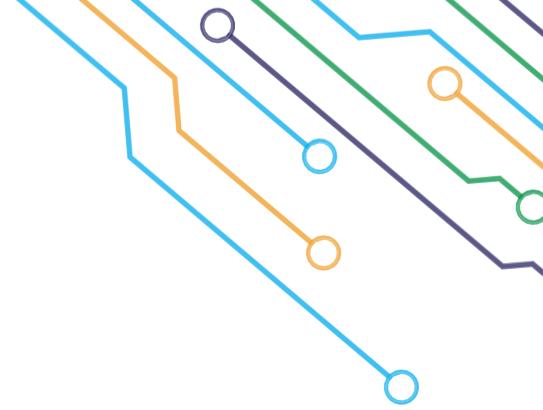
Create Angular Project

We will create an LMS website (TheLearningHub) during this course, to create the angular project for our demo, use this command:

→ ng new TheLearningHub

- PS C:\Users\d.kanaan.ext\Desktop> cd .\TheLearningHub\
PS C:\Users\d.kanaan.ext\Desktop\TheLearningHub> █





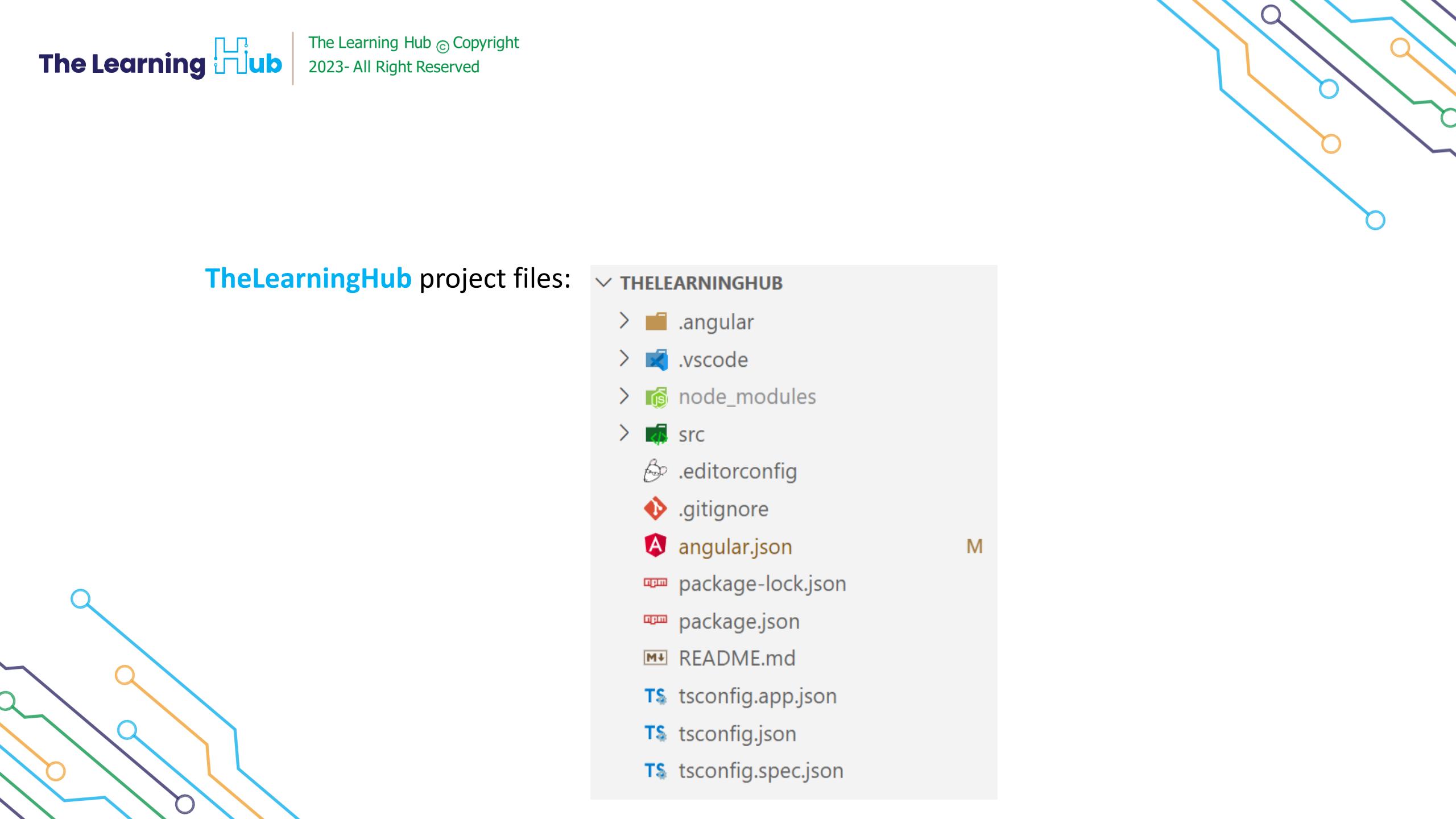
Create Angular Project

Go to The Learning Hub project using the cd command.

- PS C:\Users\d.kanaan.ext\Desktop> `cd .\TheLearningHub`
- PS C:\Users\d.kanaan.ext\Desktop\TheLearningHub> `code .`
- PS C:\Users\d.kanaan.ext\Desktop\TheLearningHub> `code .`

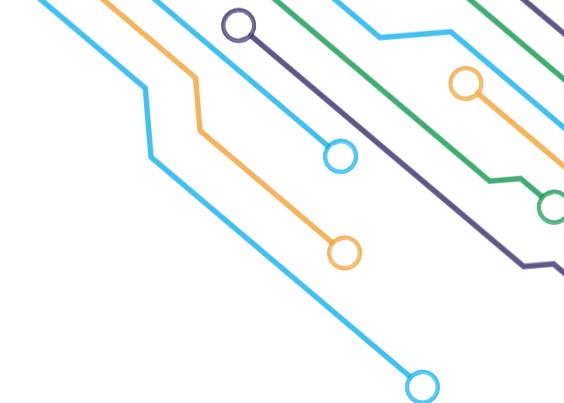


TheLearningHub project files:



```
THELEARNINGHUB
> .angular
> .vscode
> node_modules
> src
  .editorconfig
  .gitignore
  angular.json
  package-lock.json
  package.json
  README.md
  tsconfig.app.json
  tsconfig.json
  tsconfig.spec.json
```

M

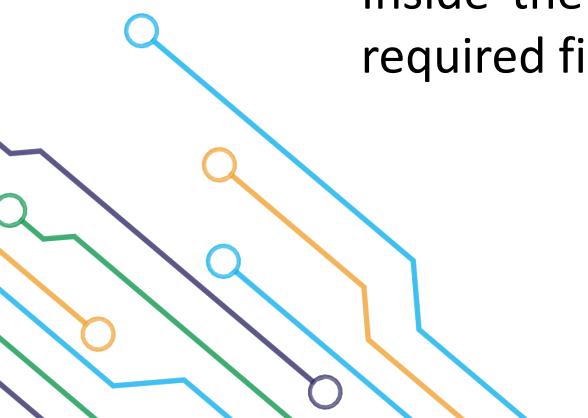


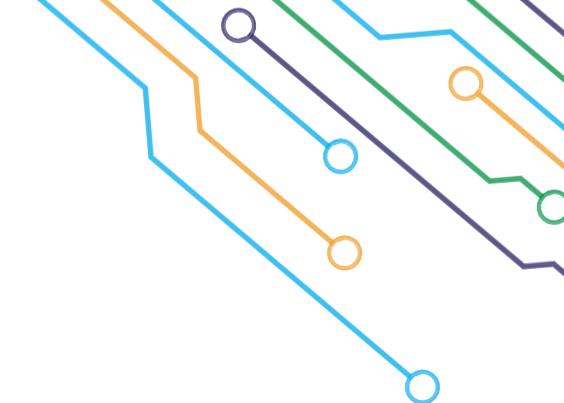
Angular project files

node_modules: You can think of the node_modules folder as a cache for the external modules that your project depends upon. NPM installs these modules with the NPM service, which downloads them from the web and copies them into the node_modules folder.

src: The project will be worked on in this folder.

Inside the src, the app folder was created during the project setup and holds all the required files for the project.





Angular project files

assets: It contains the resources such as images, videos, audio, and bootstrap files.

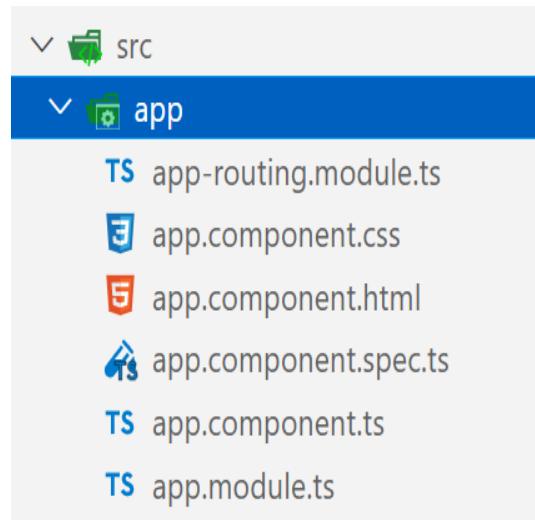
index.html: It is the first file that will be loaded in the angular project.

styles.css: It contains the CSS style that you would apply to the whole project.

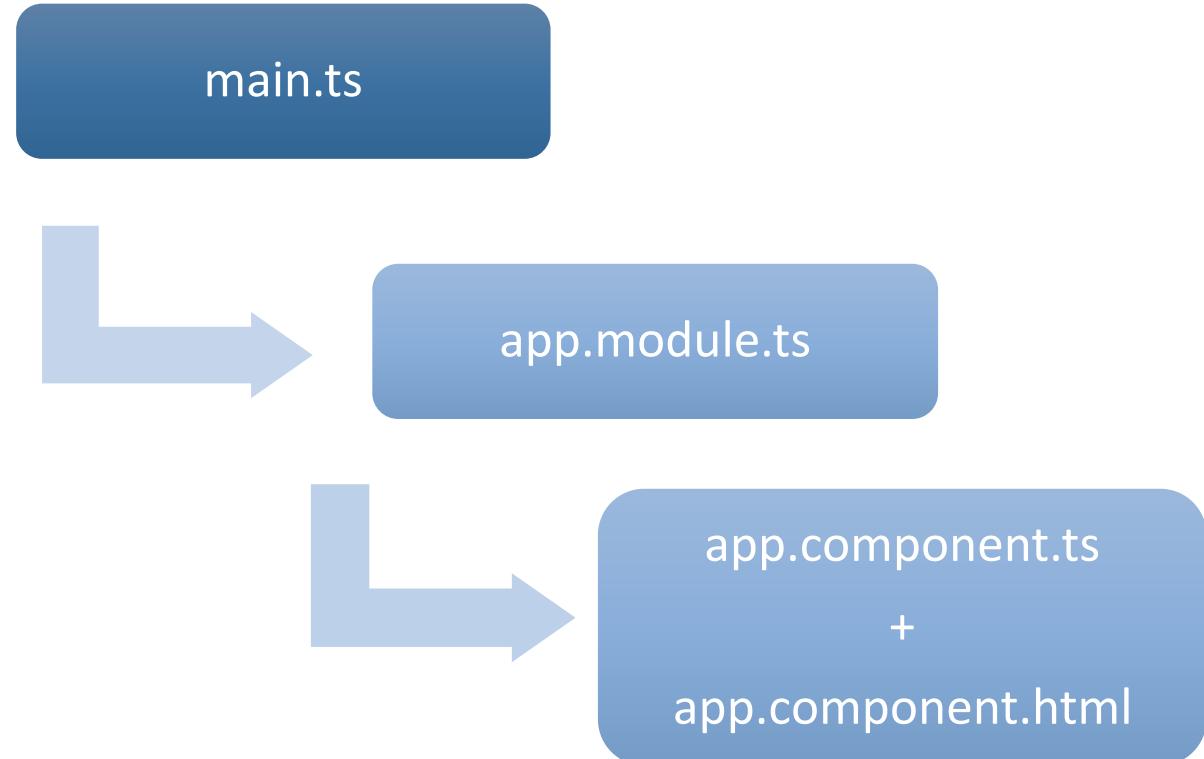


Angular project files

By default, the angular project contains one component called the **app component**.



The Flow of Execution of Angular App.



The Flow of Execution of Angular App.

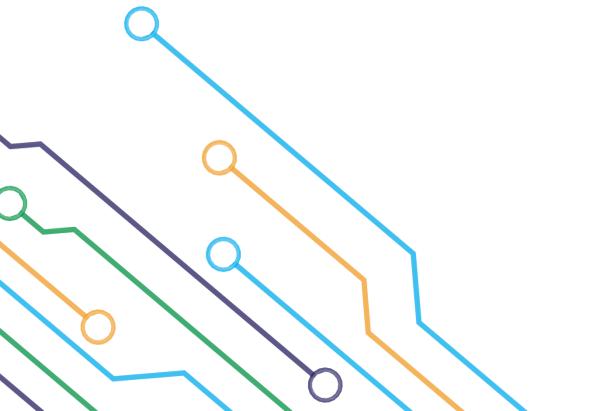
Component in Angular

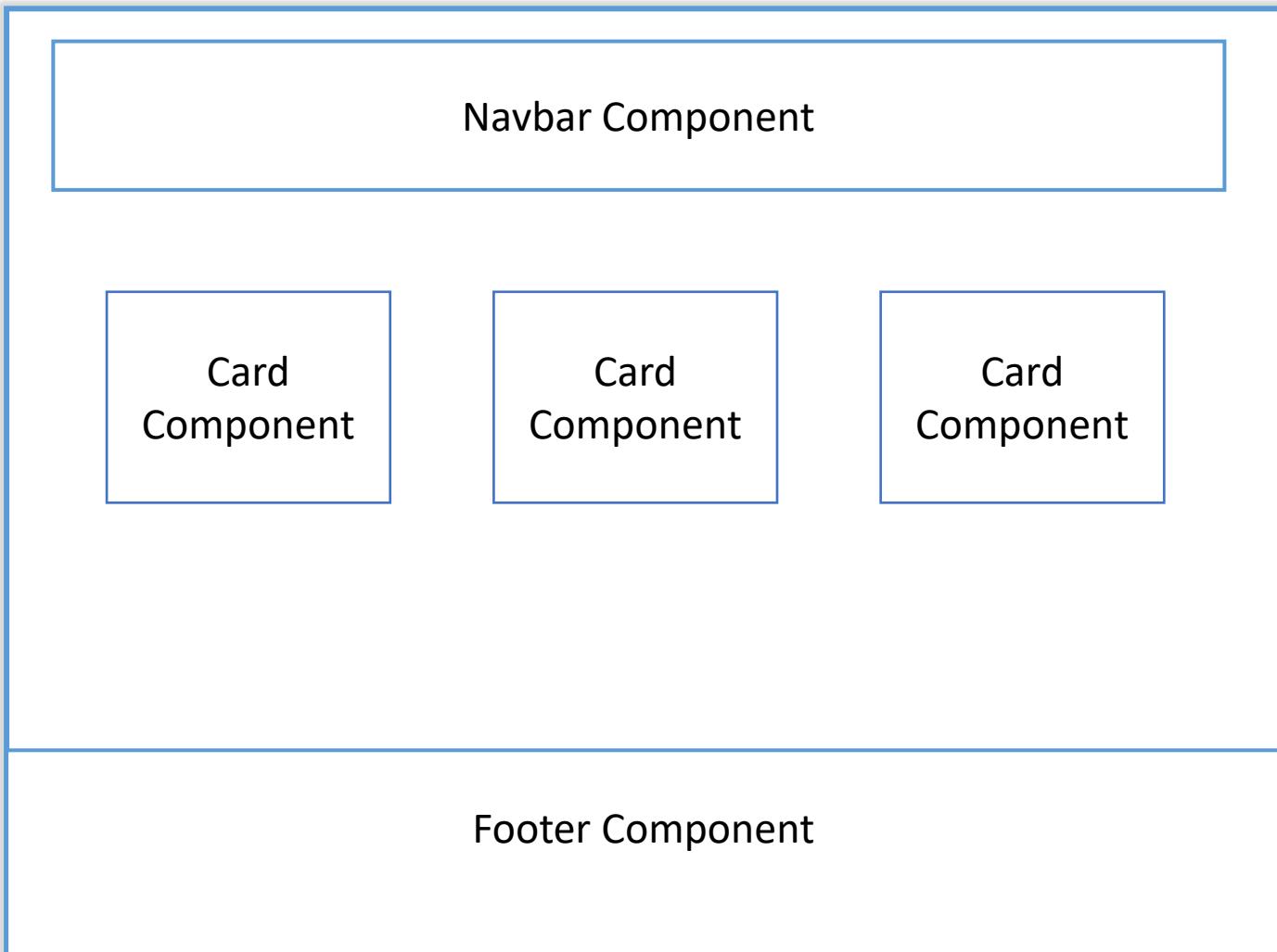


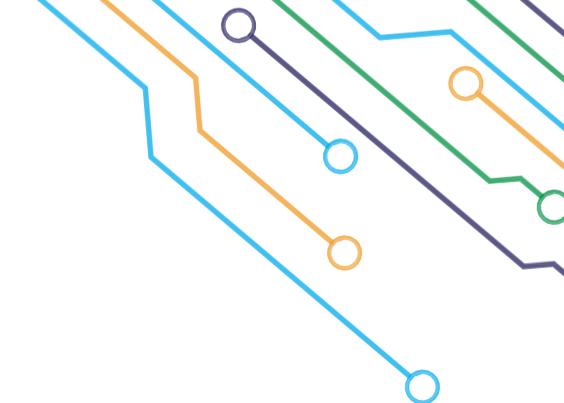
Overview of Angular Component

Components are the basic building blocks of an Angular application.

The Component defines the view and its data, which determine how the view appears and behaves.





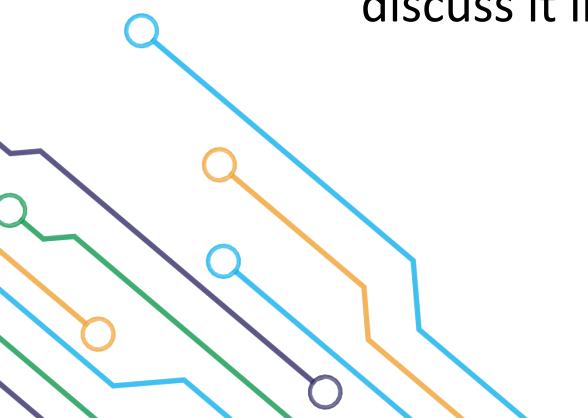


Overview of Angular Component

Components in Angular are JavaScript/TypeScript classes that are defined using @component Decorators.

Using the Decorator, the component can display a view & get metadata about the class.

Data Binding is the process used by the component to pass data to the view. (We will discuss it in Chapter 02).



Angular TS

Education and Training Solutions 2023



1 Data Binding

2 Directives

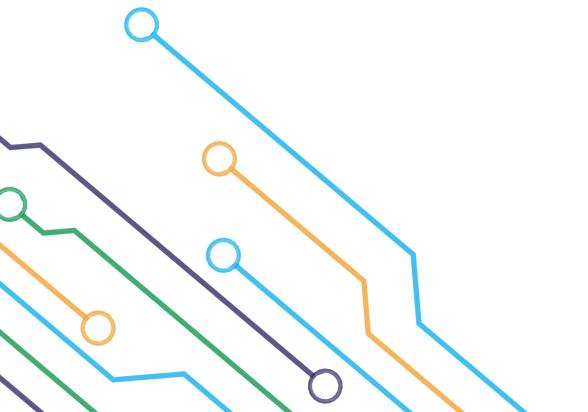


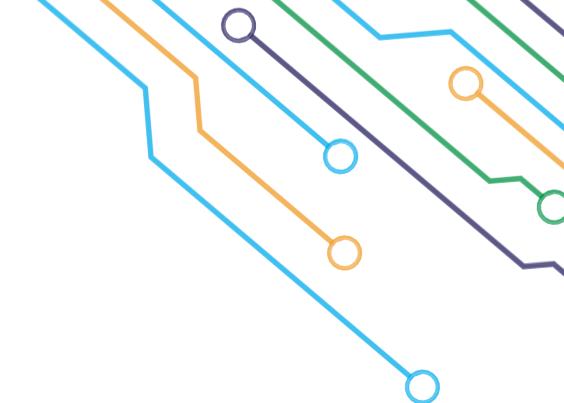
Data Binding



Overview of Data Binding

A data binding technique keeps data synchronized between components and views. Each time the user updates the data in the view, Angular updates the component. Once Angular gets new data for the component, it updates the view.



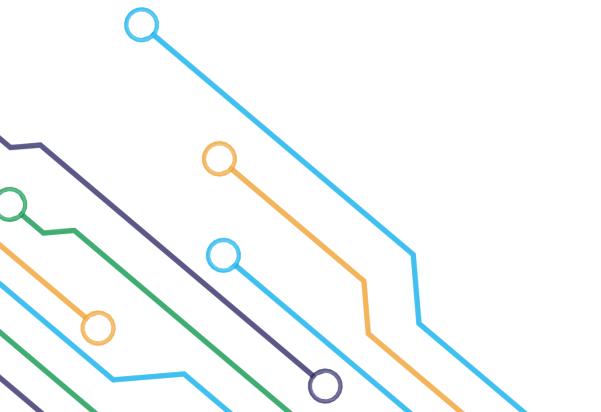


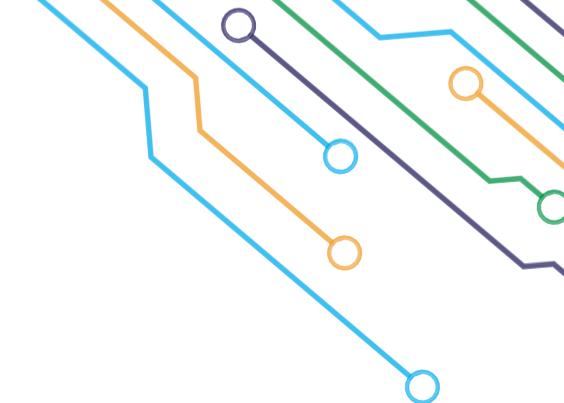
Overview of Data Binding

There are many uses of data binding. You can show models to the user and change element styles dynamically as well as respond to user events.

There are two basic types of data binding in Angular:

1. One-way binding.
2. Two-way binding.



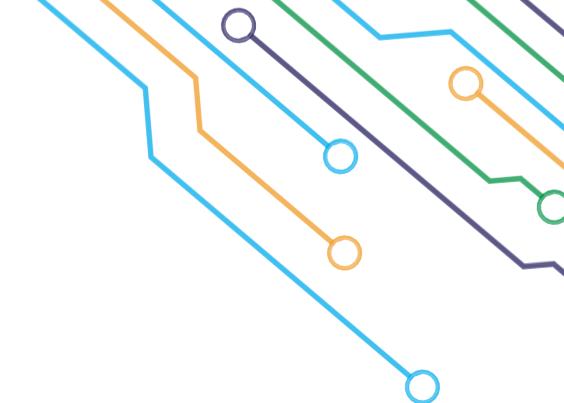


One-way data-binding

One-way binding allows for data to flow in one direction only. Either from the view to the component or from the component to the view.

Using interpolation and property binding, you can bind data from components to views.

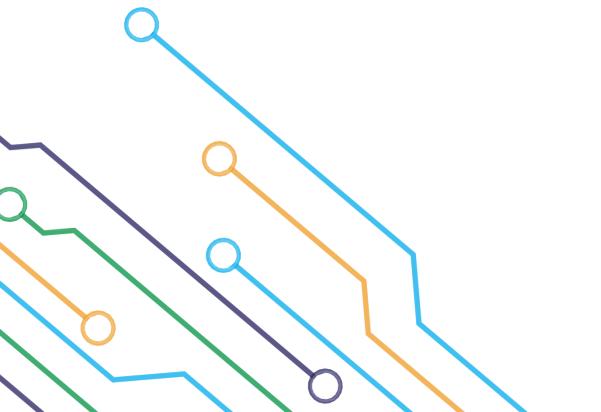




Interpolation Binding

Using interpolation, we can include expressions as part of any string literal in our HTML. The angular evaluates the expression into a string and replaces it with the original string and then updates the view.

Anywhere in the view that uses a string literal, interpolation can be used.
Angular uses double curly braces ({{ }}) in the template to indicate interpolation.



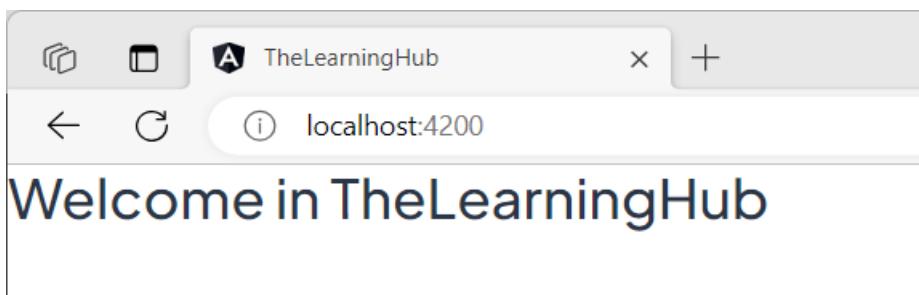
Interpolation Binding Example

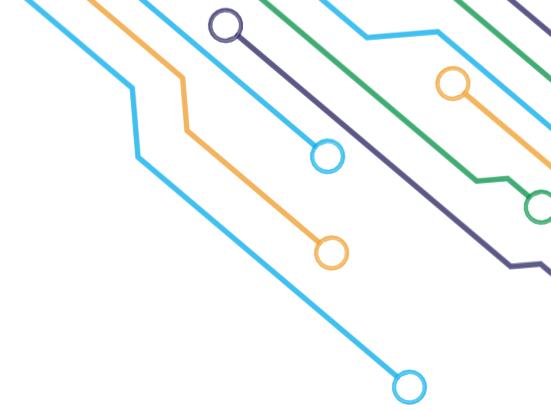
In app.component.html

```
<h2>Welcome in {{title}}</h2>
```

Interpolation Binding Example

The Result:



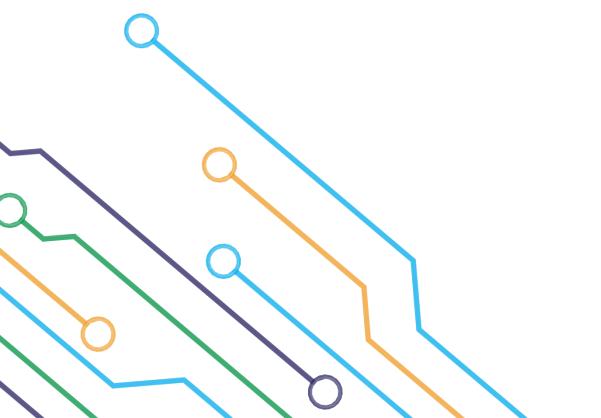


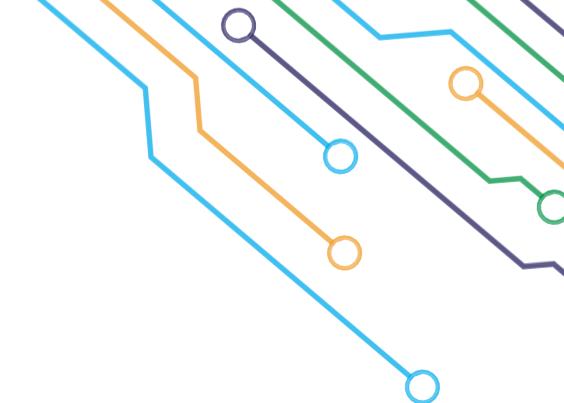
Interpolation Binding

When you launch the app, you will see TheLearningHub in the output.

Angular replaces {{title}} with the title values of the component.

Also, whenever the values of the title change, Angular updates the view. However, not the other way around.



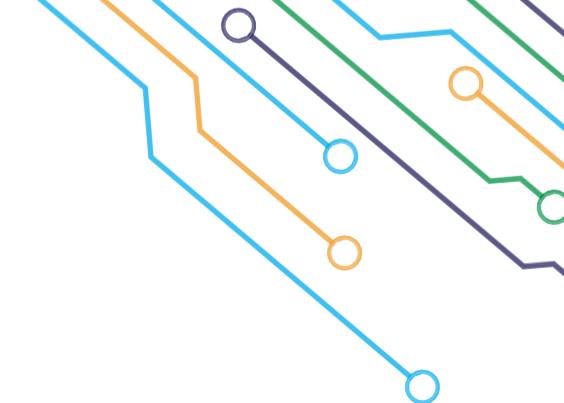


Property Binding

The binding property of an HTML element's property to a property in the component is called property binding.

Angular updates the element's property when the component's value changes.



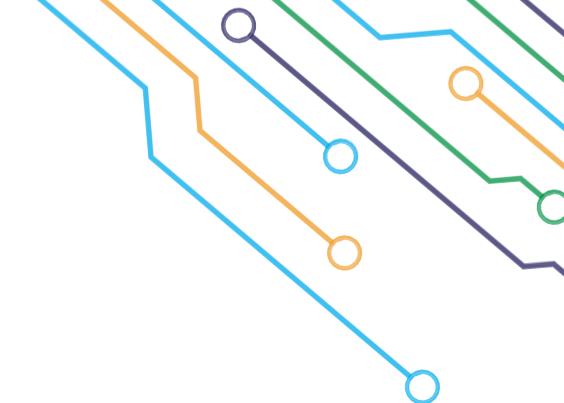


Property Binding

property binding allows you to set properties such as class, href, src, textContent, etc.

It can also be used to set properties of components or custom directives (properties decorated with Input).



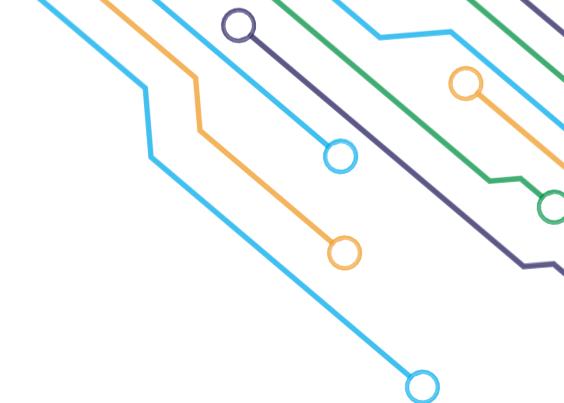


Property Binding Syntax

[binding-target] = "binding-source"

Within the square brackets [] there is the property binding target (or property target).
The property name should match the property name of the enclosing element.



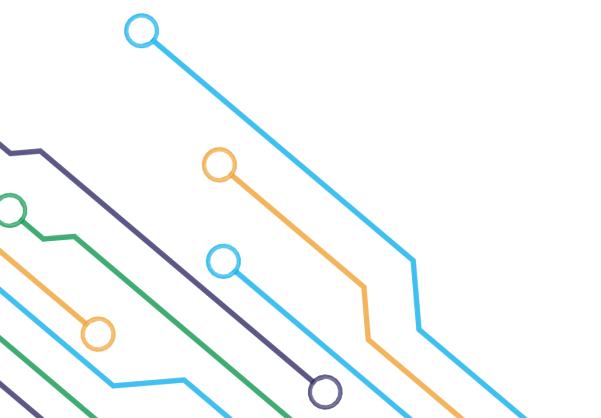


Property Binding Syntax

The binding source is enclosed in quotation marks and assigned to the binding target.

The Binding source must be a template expression.

The expression can be a property in the component, a method in the component, a template reference variable, or a combination of all three.



Property Binding Example

In app.component.html.

```
<input type ="text" placeholder="your name" [value]="name" />  
<img [src]="imagepath" alt="imagehere">
```

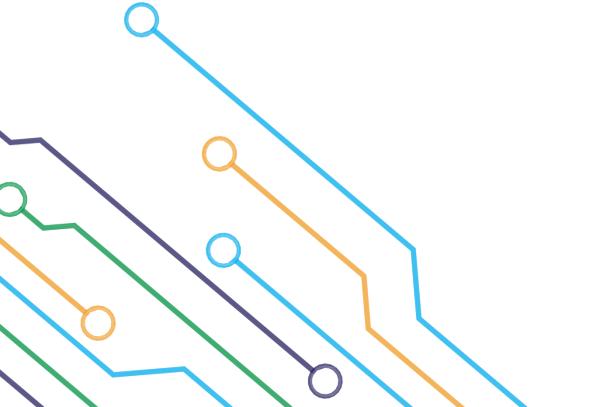


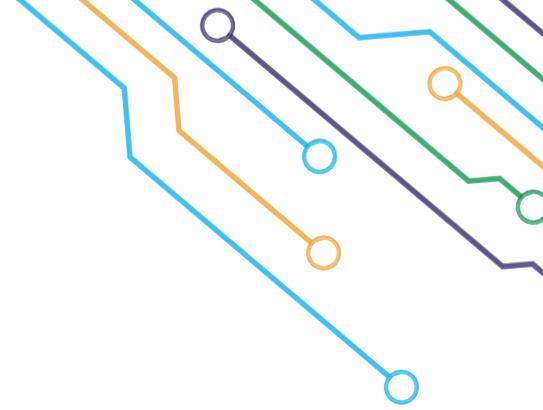
Property Binding Example

The name and image path must be defined in the typescript file.

→ In app.component.ts:

```
name:string = 'Dana Kanaan';
imagepath:string='https://miro.medium.com/max/
512/1*FKD2Uy_Q6r6AviZA2VD4RQ.png';
```





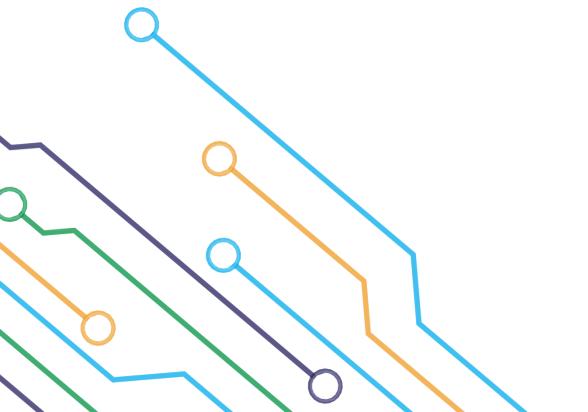
Property Binding Example

→ In app.component.html

```
<input type="text" placeholder="your name" [value]="name"  
       (change)="handleNameInputChange()" />
```

→ In app.component.ts

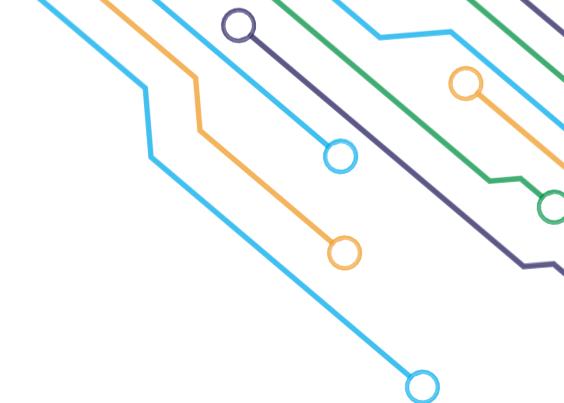
```
handleNameInputChange() {  
    alert('The value is changed!'); }
```



Handling Events

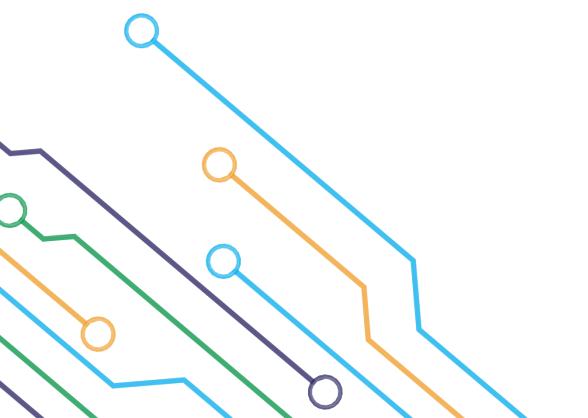
The most common way to handle events is to pass the event object, \$event, to the method that handles it.

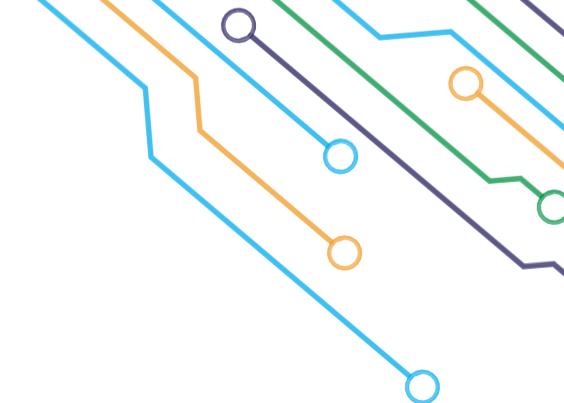
A \$event object contains information needed by the method, such as the name of the user, or the URL of an image.



Handling Events

Target events determine the shape of \$event objects. When the target event is a native DOM element event, \$event is a DOM event object that has properties such as target and target.value.





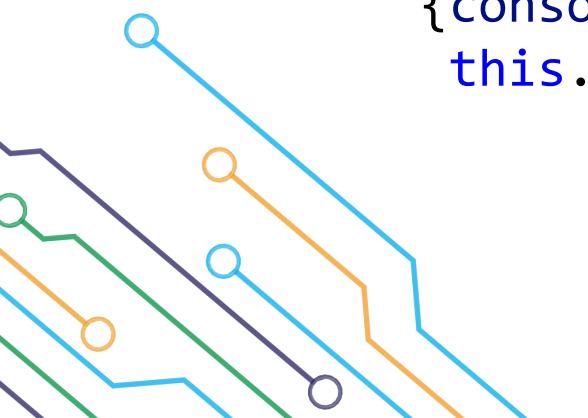
Handling Events Example

→ In app.components.html

```
<input type="text" placeholder="your name"  
[value]="name"(change)="handleNameInputChange($event)" />
```

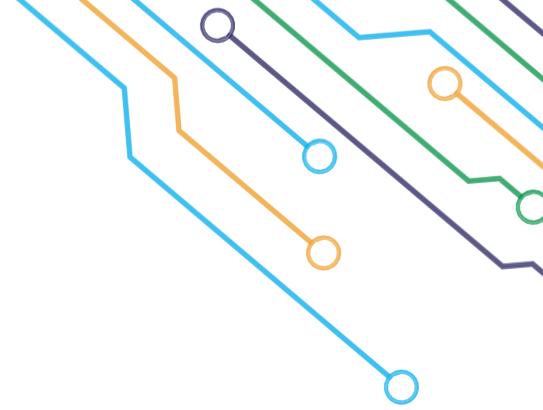
→ In app.components.ts

```
handleNameInputChange = (inputhead: any) =>  
{console.log(inputhead.target.value);  
this.name = inputhead.target.value; }
```



Two-way data-binding

By using two-way binding, any changes we make to our model in the component are propagated to the view, and updates to the view are immediately propagated to the underlying component.

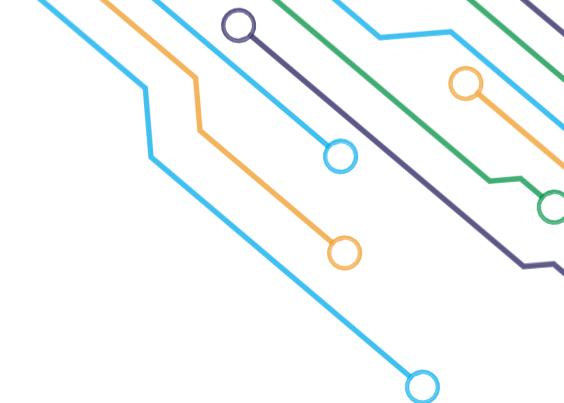


Two-way data-binding Syntax

The syntax for tow-way data-binding :

```
<Element [(ngModel)]="property"></Element>.
```



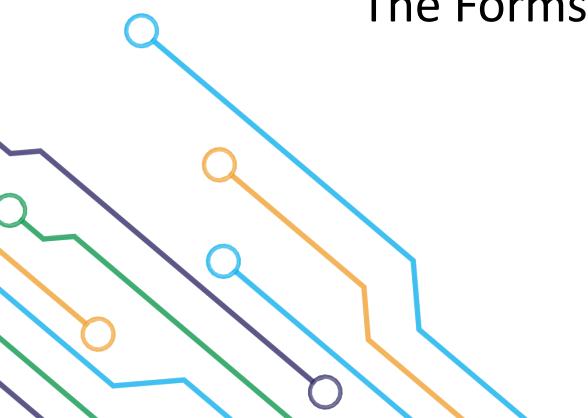


ngModel

For Angular, the ngModel directive is used to achieve the two-way binding on HTML Form elements. ngModel binds to form elements like input, select, and select an area.

The ngModel directive is not part of the Angular Core library; instead, it is part of @angular/forms.

The FormsModule package should be imported into your Angular module.





How to use ngModel

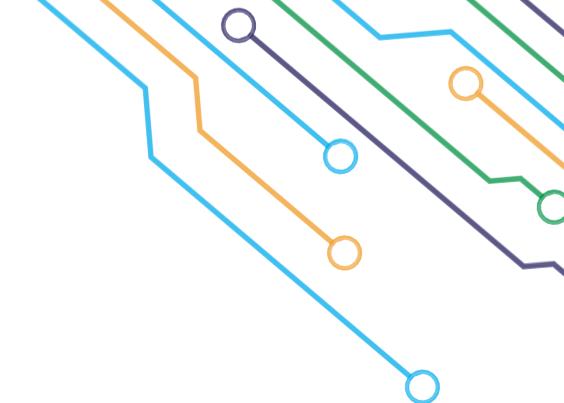
→ In app.module.ts

```
import { FormsModule } from '@angular/forms';
```

→ In the import section.

```
imports: [  
  BrowserModule,  
  AppRoutingModule,  
  FormsModule],
```





Two-way data-binding Example

Creates a simple form using two-way data binding which contains:

- Name
- Email
- Salary
- And then Calculate the annual salary.



Example:

→ In app.component.ts

```
export class AppComponent {  
  title = 'TheLearningHub';  
  name: string = '';  
  email: string = '';  
  salary: number = 0;  
}
```

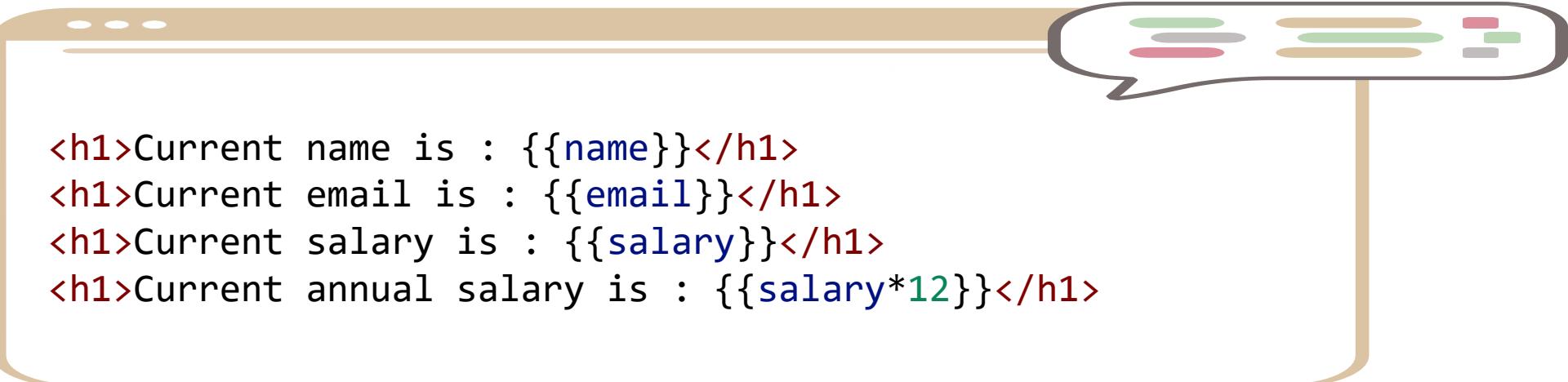
Example:

→ In app.component.html

```
<input type="text" placeholder="your name"  
[(ngModel)]="name" />  
  
<input type="text" placeholder="your email"  
[(ngModel)]="email" />  
  
<input type="number" placeholder="your Salary"  
[(ngModel)]="salary" />
```

Example:

→ And this code reads the value from the typescript file.



```
<h1>Current name is : {{name}}</h1>
<h1>Current email is : {{email}}</h1>
<h1>Current salary is : {{salary}}</h1>
<h1>Current annual salary is : {{salary*12}}</h1>
```

Example:

→ In app.component.css

```
input {  
    display: block;  
    width: 300px;  
    padding: 10px;  
    font-size: 1em;  
    margin-top: 10px;  
}
```

Example:

→ In app.component.html

```
<input type="text" placeholder="your name" [(ngModel)]="name"  
(ngModelChange)="handlechange($event)"/>
```

Exercise

Add to the form a button called clear to clear all data
on the HTML page.



Directives



Overview of Directives

Angular directives allow us to manipulate the DOM.

Directives can be used to change the appearance, behaviour, or layout of DOM elements.

The directives in Angular are divided into three types :

1. Component Directive.
2. Structural directives.
3. Attribute directives.

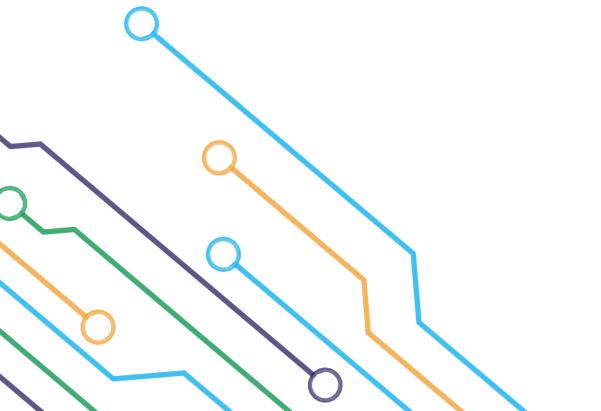




Component Directives

Component directives are used for specifying the template/HTML for the Dom Layout.

Component directives are simple classes decorated with the @component decorator.



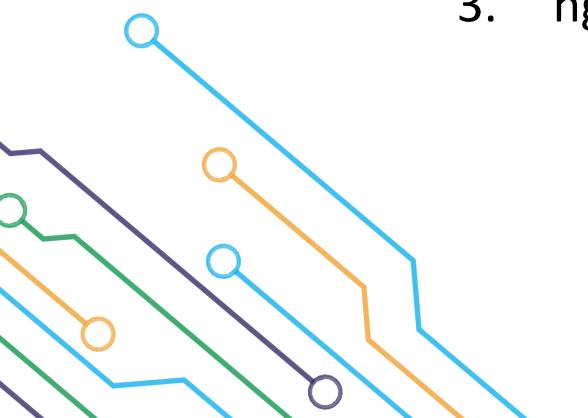


Structural Directives

A structural directive can change the layout of the DOM by adding or removing elements.

There are three common structural directives:

1. ngFor
2. ngIf
3. ngSwitch



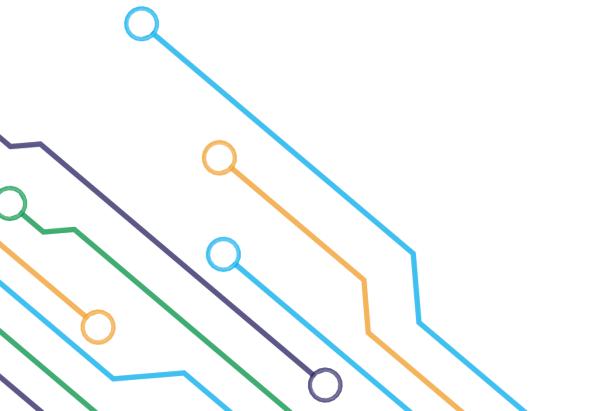


ngFor Structural Directives

ngFor is an Angular directive that repeats a portion of the HTML template once per iteration of an I EnumerableList (collection).

Syntax:

ngFor="let obj of collection"



ngFor Example:

→ In app.component.ts

```
student = [
  {
    firstName: "Mones",
    lastName: "Ahmad",
    course:["API", "C#", "angular"],
    gender: "Male",
    email: "mones989@hotmail.com",
    issPassed: true
  },
  {
    firstName: "Mohammad",
    lastName: "Shourman",
    course:["Web Design" , "Oracle"],
    gender: "Male",
  }
]
```

ngFor Example:

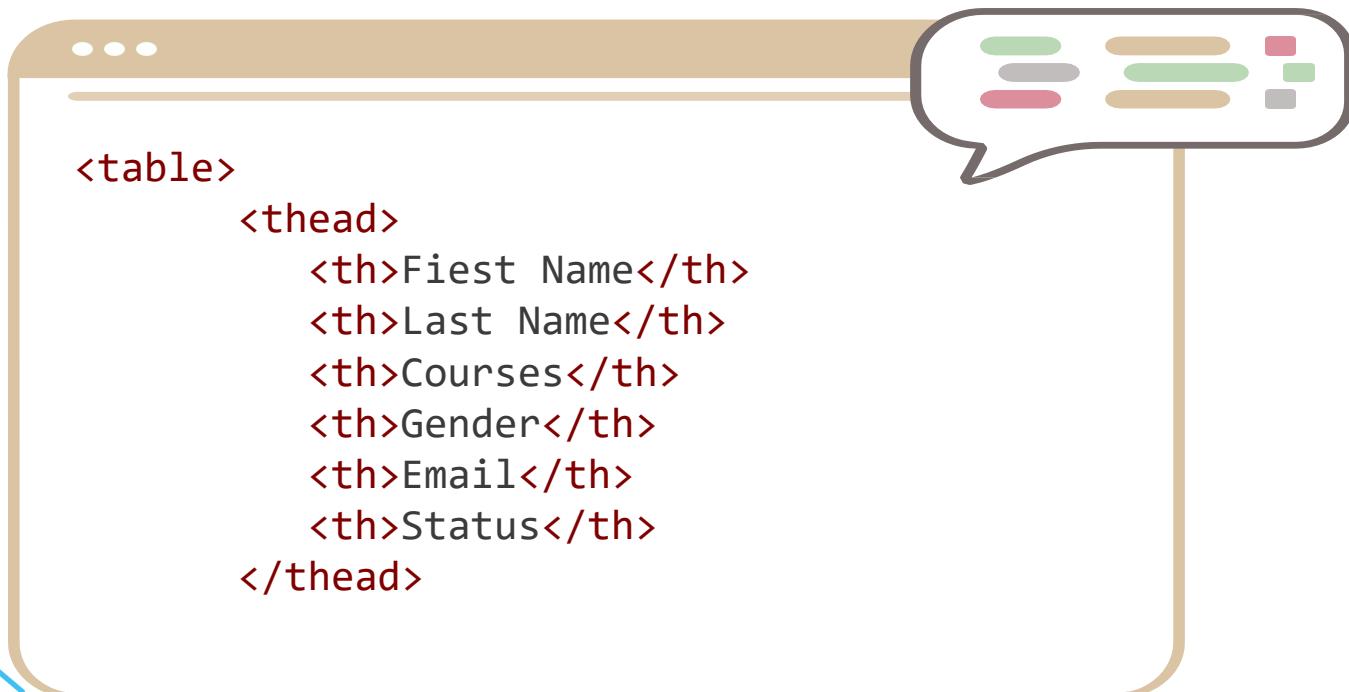
→ In app.component.ts



```
email: "shurman@meo.com",
isPassed: false
},
{
  firstName: "Mohammad",
  lastName: "Fodeh",
  course: ["MVC"],
  gender: "Male",
  email: "mohammad@meo.com",
  isPassed: true
}]
```

ngFor Example:

→ In app.component.html



ngFor Example:

→ In app.component.html

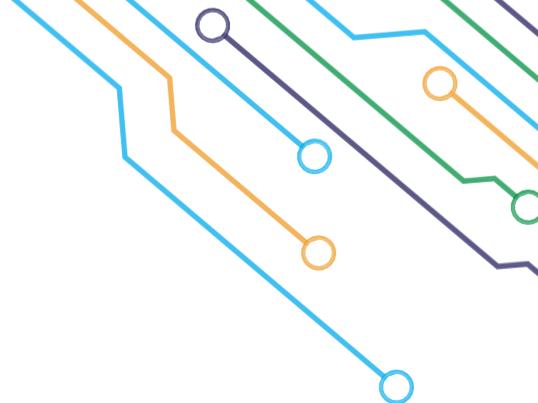
```
<tbody>
  <tr *ngFor="let c of student">
    <td>{{c.firstName}}</td>
    <td>{{c.lastName}}</td>
    <td>{{c.course}}</td>
    <td>{{c.gender}}</td>
    <td>{{c.email}}</td>
    <td>{{c.isPassed}}</td>
  </tr>
</tbody>
</table>
```

ngIf

By using the ngIf Directive, HTML elements can be added or removed based on an expression.

Boolean values must be returned from the expression.

The element is removed if the expression is false, otherwise, the element is inserted.



ngIf Example

In app.component.html

```
<div *ngIf="toggle">  
    This is shown if condition is true  
</div>
```

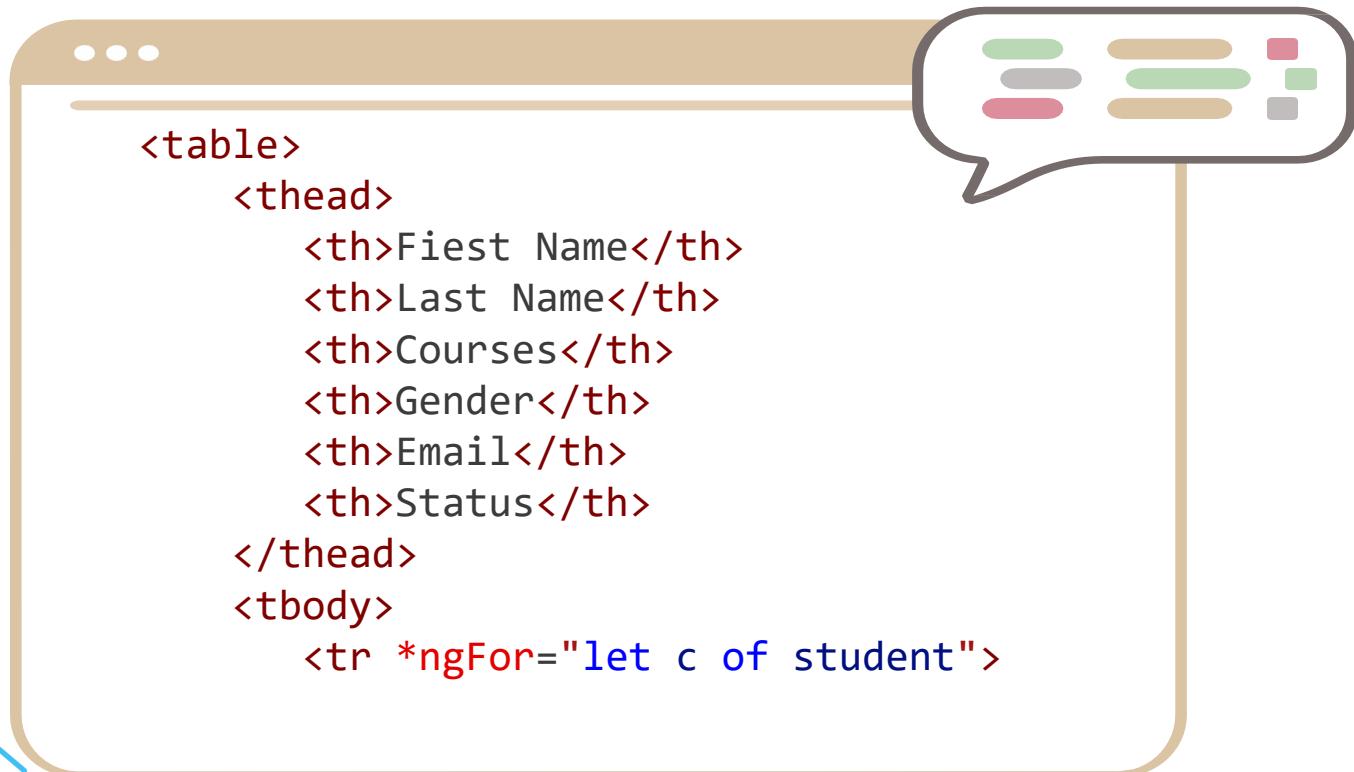
In app.component.ts

```
toggle:boolean=true;
```



ngIf Example:

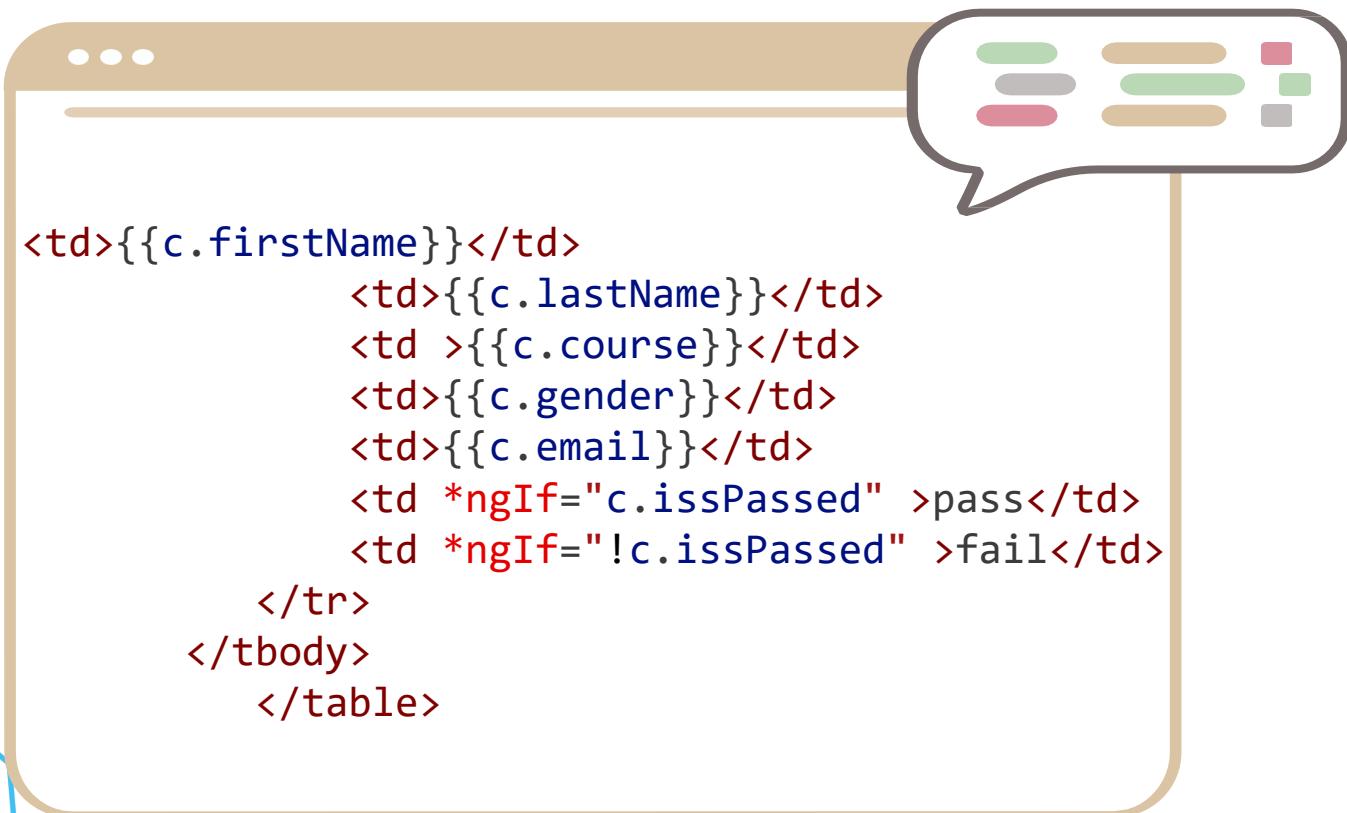
→ In app.component.html



```
<table>
  <thead>
    <th>First Name</th>
    <th>Last Name</th>
    <th>Courses</th>
    <th>Gender</th>
    <th>Email</th>
    <th>Status</th>
  </thead>
  <tbody>
    <tr *ngFor="let c of student">
```

ngIf Example:

→ In app.component.html



```
<td>{{c.firstName}}</td>
    <td>{{c.lastName}}</td>
    <td>{{c.course}}</td>
    <td>{{c.gender}}</td>
    <td>{{c.email}}</td>
    <td *ngIf="c.issPassed" >pass</td>
    <td *ngIf="!c.issPassed" >fail</td>
</tr>
</tbody>
</table>
```

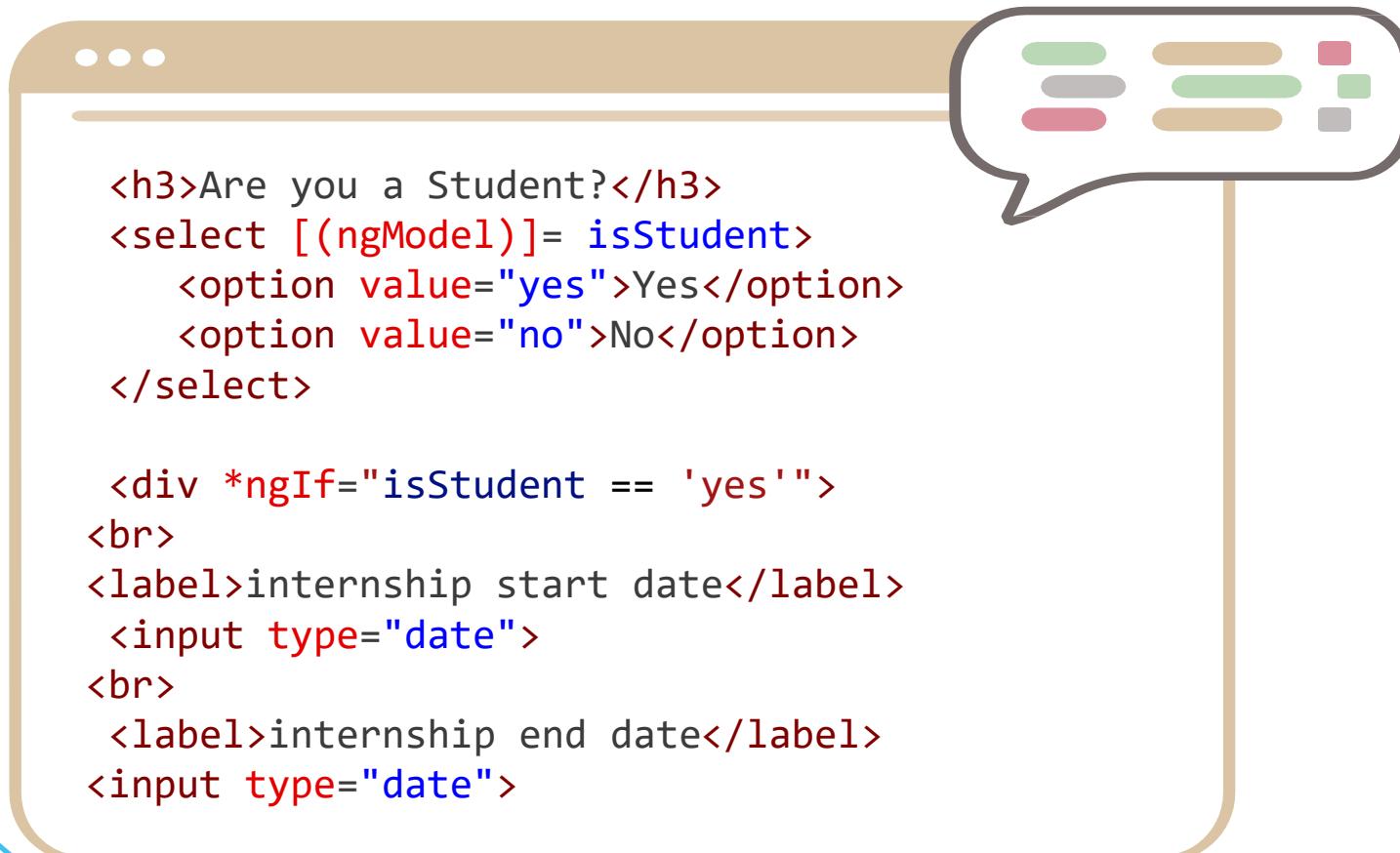
ngIf Example

In app.component.ts

```
isStudent = "no"
```

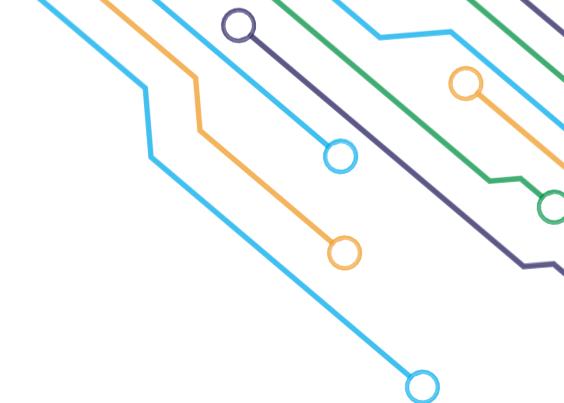
ngIf Example:

→ In app.component.html



```
<h3>Are you a Student?</h3>
<select [(ngModel)]="isStudent">
  <option value="yes">Yes</option>
  <option value="no">No</option>
</select>

<div *ngIf="isStudent == 'yes'">
<br>
<label>internship start date</label>
<input type="date">
<br>
<label>internship end date</label>
<input type="date">
```



ngSwitch

By using the ngSwitch directive, you can add or remove HTML elements based on match expressions.

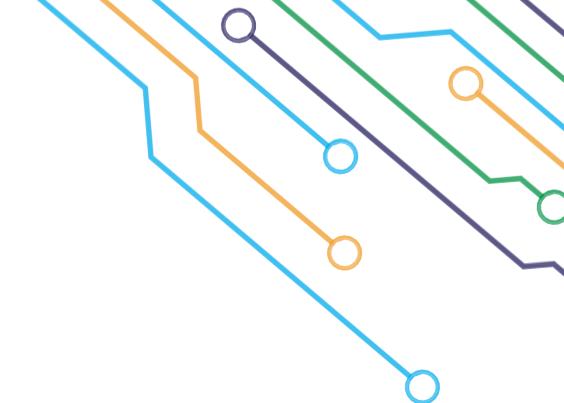
ngSwitch directive used with ngSwitchCase and ngSwitchDefault.



ngSwitch Example:

→ In app.component.html

```
<div [ngSwitch]="isStudent">
  <p *ngSwitchCase="'yes'">you are a student</p>
  <p *ngSwitchCase="'no'">you are a
graduated</p>
  <p *ngSwitchDefault>Please select from the
above list </p>
</div>
```

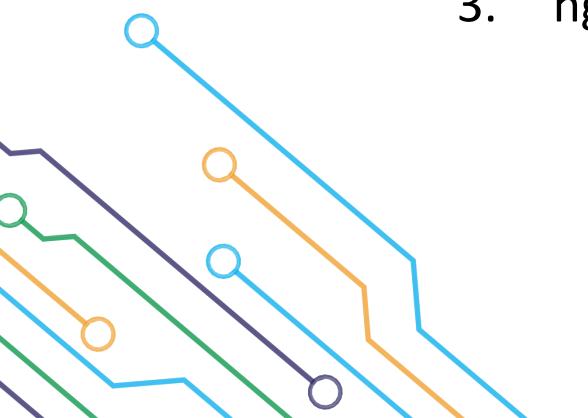


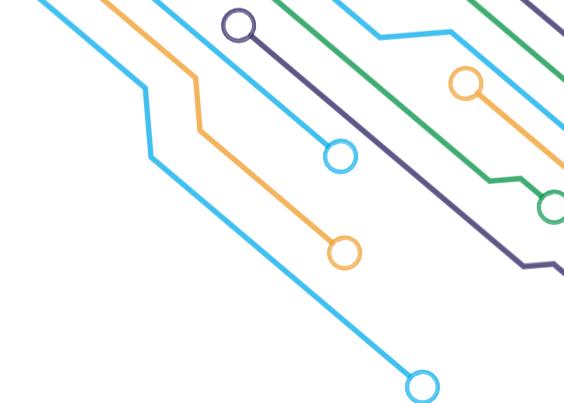
Attribute Directives

By using a style directive or attribute, we can change how an element appears or behaves.

There are three common Attribute directives

1. ngModel
2. ngClass
3. ngStyle





Attribute Directives

1. ngModel

In order to achieve the two-way data binding, the ngModel directive is used.

2. ngClass

CSS classes are added or removed from HTML elements using the ngClass property.



ngClass Example

In app.component.css

```
.red { color: red;  
background-color: gray; }  
.size20 { font-size: 20px; }
```



ngClass Example

In app.component.html

```
<div [ngClass]="'red size20'"> Red Text with Size 20px </div>
```

The Result



Red Text with Size 20px

ngClass Example

→ In app.component.html

```
<table>
<thead [ngClass]="'tableHead'>
  <th>First Name</th>
  <th>Last Name</th>
  <th>Courses</th>
  <th>Gender</th>
  <th>Email</th>
  <th>Status</th>
</thead>
<tbody>
```

ngClass Example

→ In app.component.html

```
<tr *ngFor="let c of student"
[ngClass]="'failStudent' : !c.issPassed">
  <td>{{c.firstName}}</td>
  <td>{{c.lastName}}</td>
  <td [ngClass]="'courses' :
c.course.length > 1">{{c.course}}</td>
    <td [ngClass]="'male' : c.gender == 'Male'
, 'female' : c.gender == 'Female'
">{{c.gender}}</td>
    <td>{{c.email}}</td>
    <td>{{c.issPassed}}</td>
</tr>
</tbody>
</table>
```

ngClass Example

→ In app.component.css

```
.failStudent {  
background-color: red; }  
.courses {  
color: blue; }  
.tableHead{  
background-color: darkblue;  
color: white; }  
.male {  
background-color: lightblue; }  
.female {  
background-color: lightpink; }
```

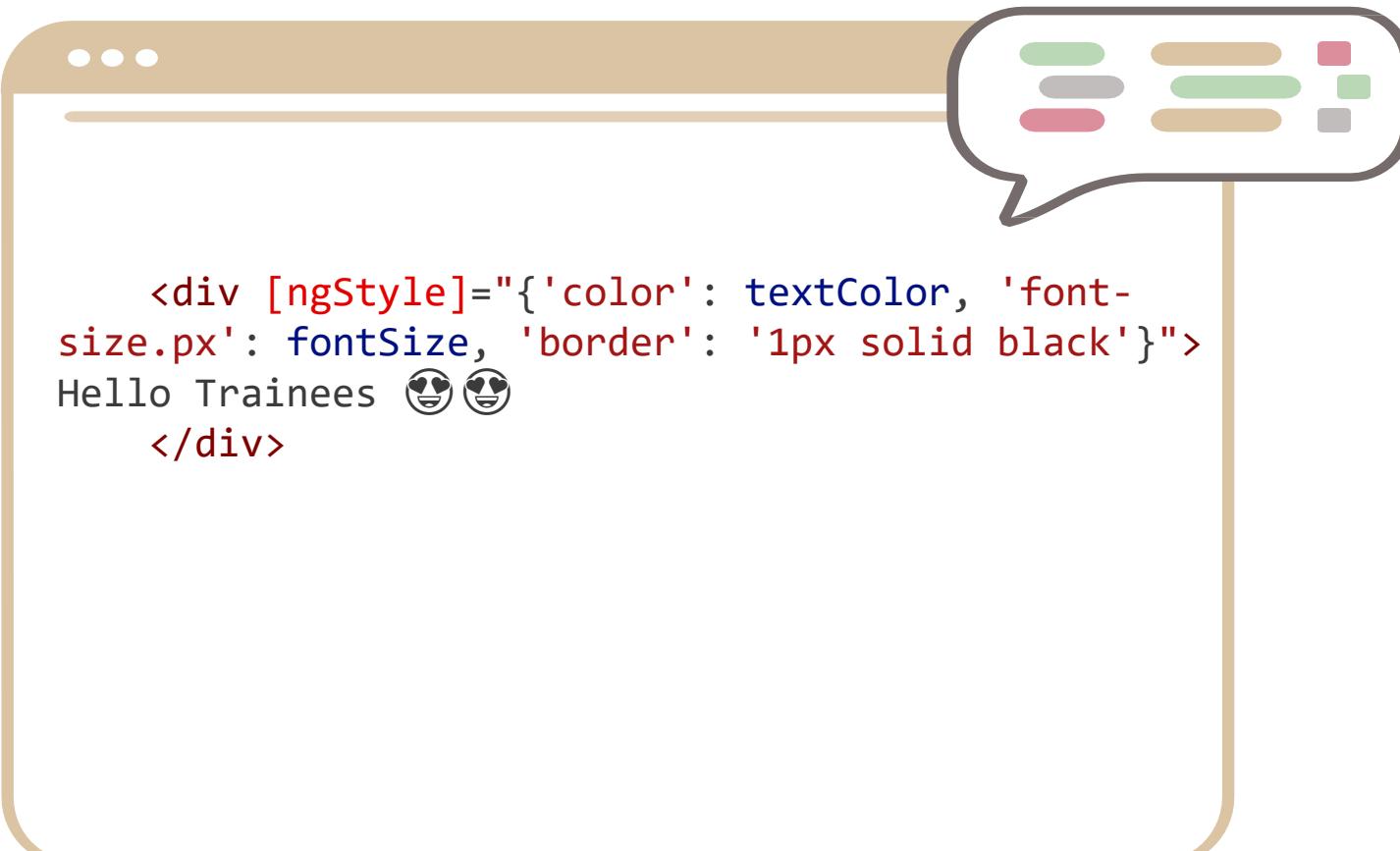
ngStyle Example

→ In app.component.ts

```
textColor = 'red'  
fontSize = '32'
```

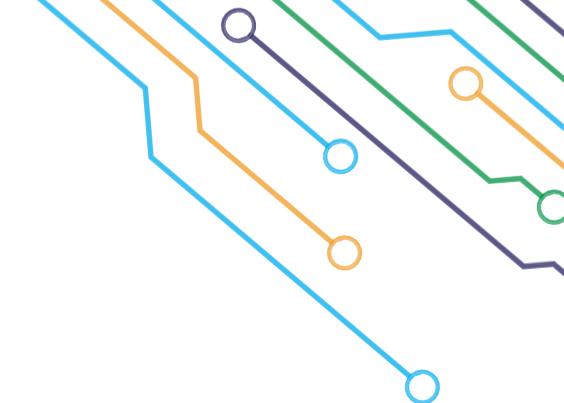
ngStyle Example

→ In app.component.html



```
<div [ngStyle]="{'color': textColor, 'font-size.px': fontSize, 'border': '1px solid black'}">  
Hello Trainees 😍😍  
</div>
```

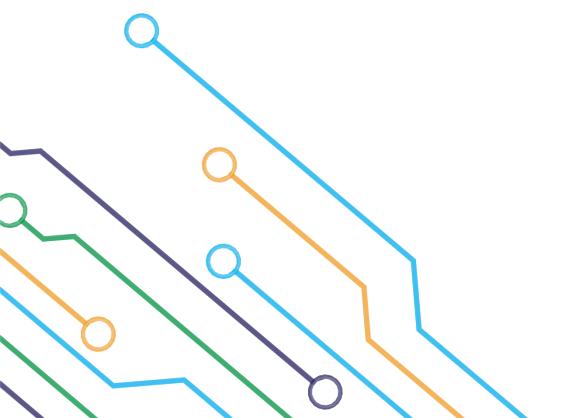
Module in Angular

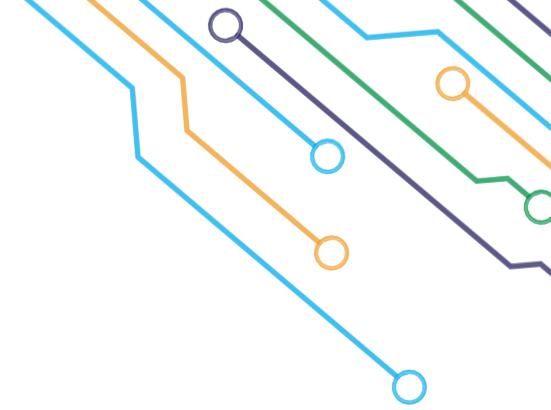


Overview of Module

It is a deployment subset within your full Angular application.

Using it, you can divide an application into smaller parts and load each one separately, as well as create libraries of components that can be imported directly into other applications.



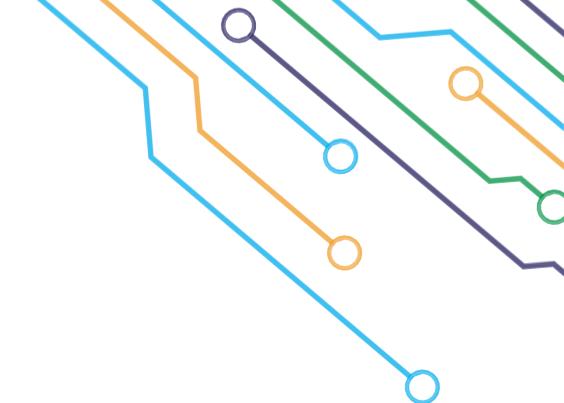


Overview of Module

An NgModule is defined by a class decorated with `@NgModule()`.

This decorator is a function that takes an object that describes a module as its single metadata property.



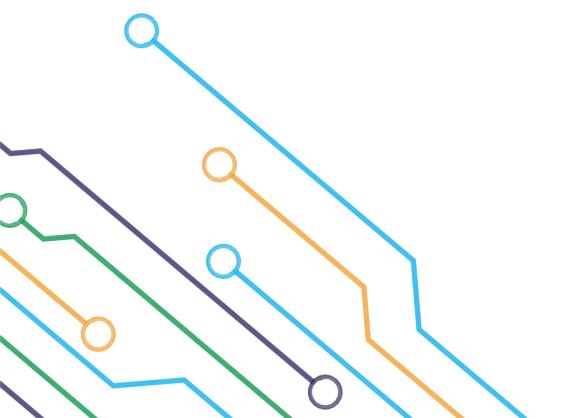


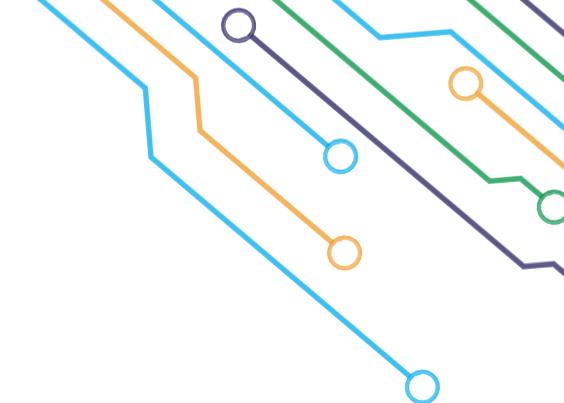
Properties for NgModule

The most important properties of NgModule:

Declarations: The components, directives, and pipes of this module.

Imports: The subset of NgModule declarations that must be visible and useable in other NgModule templates.

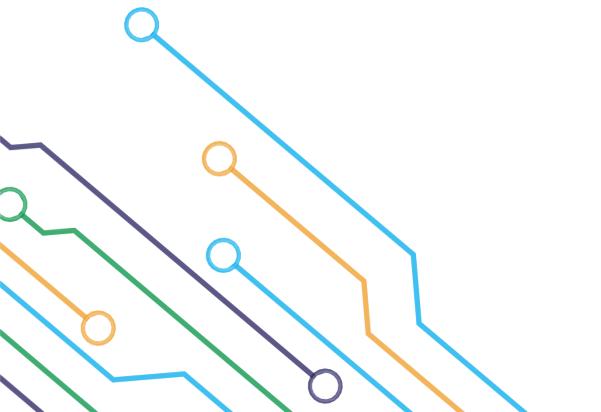


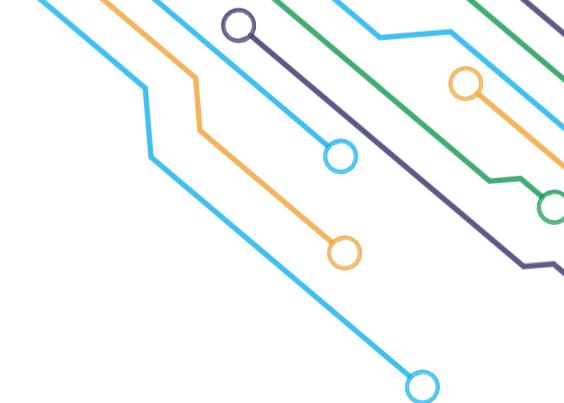


Properties for NgModule

Exports: This NgModule depends on other modules whose exported classes are needed by this component template.

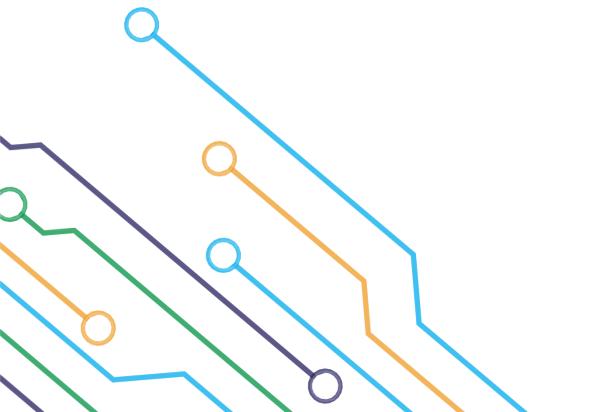
Providers: Services created by this NgModule become accessible in all parts of the application since this NgModule contributes to the global collection of services. (It is also possible to specify providers at the component level.)





Properties for NgModule

bootstrap: The main view of the application, called the root component, is where all other views reside. It's only the root NgModule that needs to set the bootstrap property.





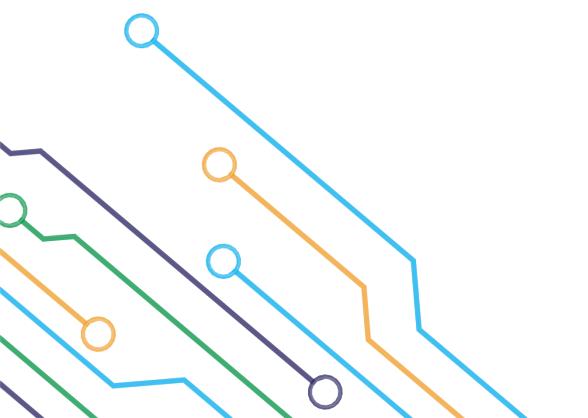
Generate a New Module

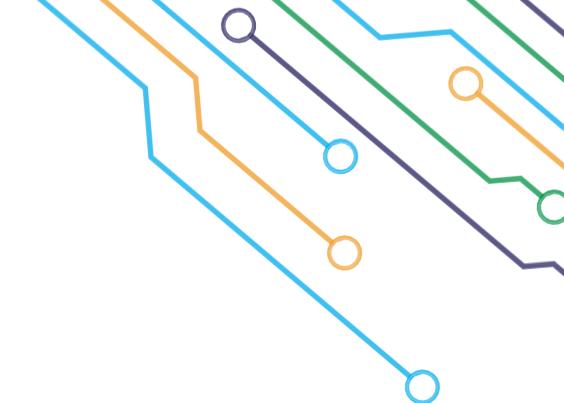
Use this command in the terminal to generate a new module.

→ `ng generate module module_name --routing`

Or

→ `ng g m module_name --routing`





Generate a New Module Example

In our project (**The Learning Hub**), create a new module called **auth**, and for this module generate two components inside it:

login and **register**.



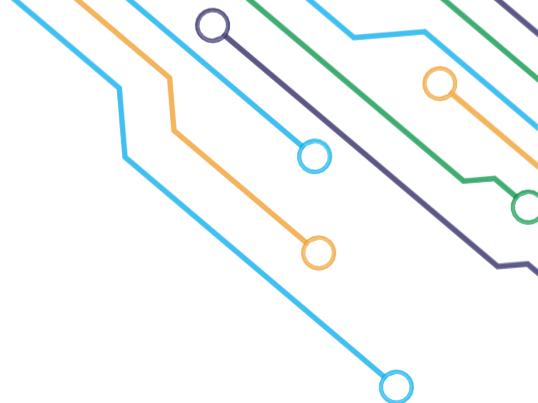


Generate a New Module Example

Generate auth module:

- PS C:\Users\d.kanaan.ext\Desktop\TheLearningHub> `ng g m auth --routing`
`CREATE src/app/auth/auth-routing.module.ts (247 bytes)`
`CREATE src/app/auth/auth.module.ts (272 bytes)`
- PS C:\Users\d.kanaan.ext\Desktop\TheLearningHub>





Generate a New Module Example

Generate a login component in auth module.

To determine, these components related to this module, write
moduleName/componentName.

- PS C:\Users\d.kanaan.ext\Desktop\TheLearningHub> `ng g c auth/login`
CREATE src/app/auth/login/login.component.html (20 bytes)
CREATE src/app/auth/login/login.component.spec.ts (552 bytes)
CREATE src/app/auth/login/login.component.ts (198 bytes)
CREATE src/app/auth/login/login.component.css (0 bytes)
UPDATE src/app/auth/auth.module.ts (352 bytes)





Generate a New Module Example

Generate a Register component in auth module.

- PS C:\Users\d.kanaan.ext\Desktop\TheLearningHub> `ng g c auth/register`
`CREATE src/app/auth/register/register.component.html (23 bytes)`
`CREATE src/app/auth/register/register.component.spec.ts (573 bytes)`
`CREATE src/app/auth/register/register.component.ts (210 bytes)`
`CREATE src/app/auth/register/register.component.css (0 bytes)`
`UPDATE src/app/auth/auth.module.ts (442 bytes)`



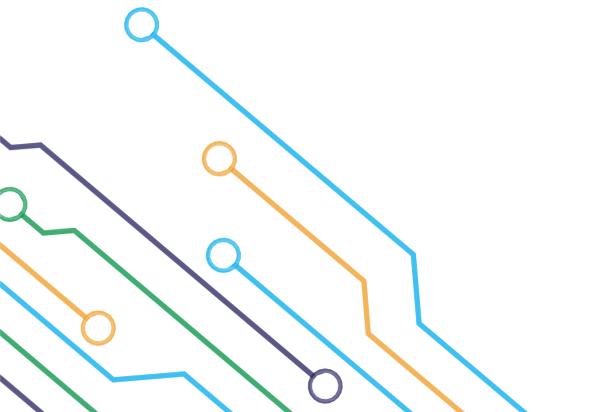
Shared Module

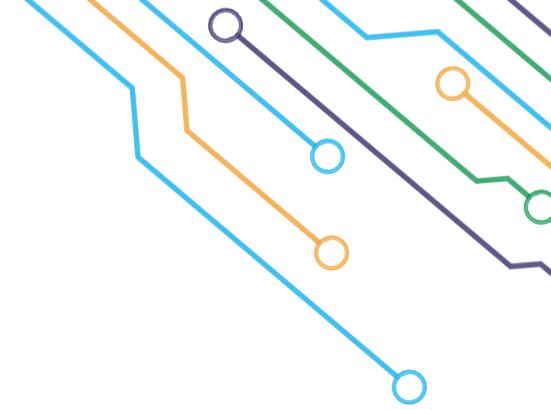


Overview of Shared Module

Your code can be organized and streamlined by creating shared modules.

Directives, pipes, and components that are commonly used can be put into this module, which you can then import wherever you need them in your application.





Overview of Shared Module

In order to generate a shared module, the same command to generate any other module will be used.

- PS C:\Users\d.kanaan.ext\Desktop\TheLearningHub> `ng g m shared`
`CREATE src/app/shared/shared.module.ts (192 bytes)`



Consider the following

The CommonModule is imported because the module's component needs common directives.

The module declares and exports utility pipes, directives, and components.

It re-exports all modules, components, pipes, and directives that are declared and imported.

```
@NgModule({
  declarations: [],
  imports: [
    CommonModule,
    FormsModule,
    ReactiveFormsModule,
    MatFormFieldModule,
    MatInputModule
  ],
  exports: [
    FormsModule,
    ReactiveFormsModule,
    MatFormFieldModule,
    MatInputModule
])
})
```

Note

You can create a template that you will use multiple times, such as navbars, footers, or sidebars, in the shared module.

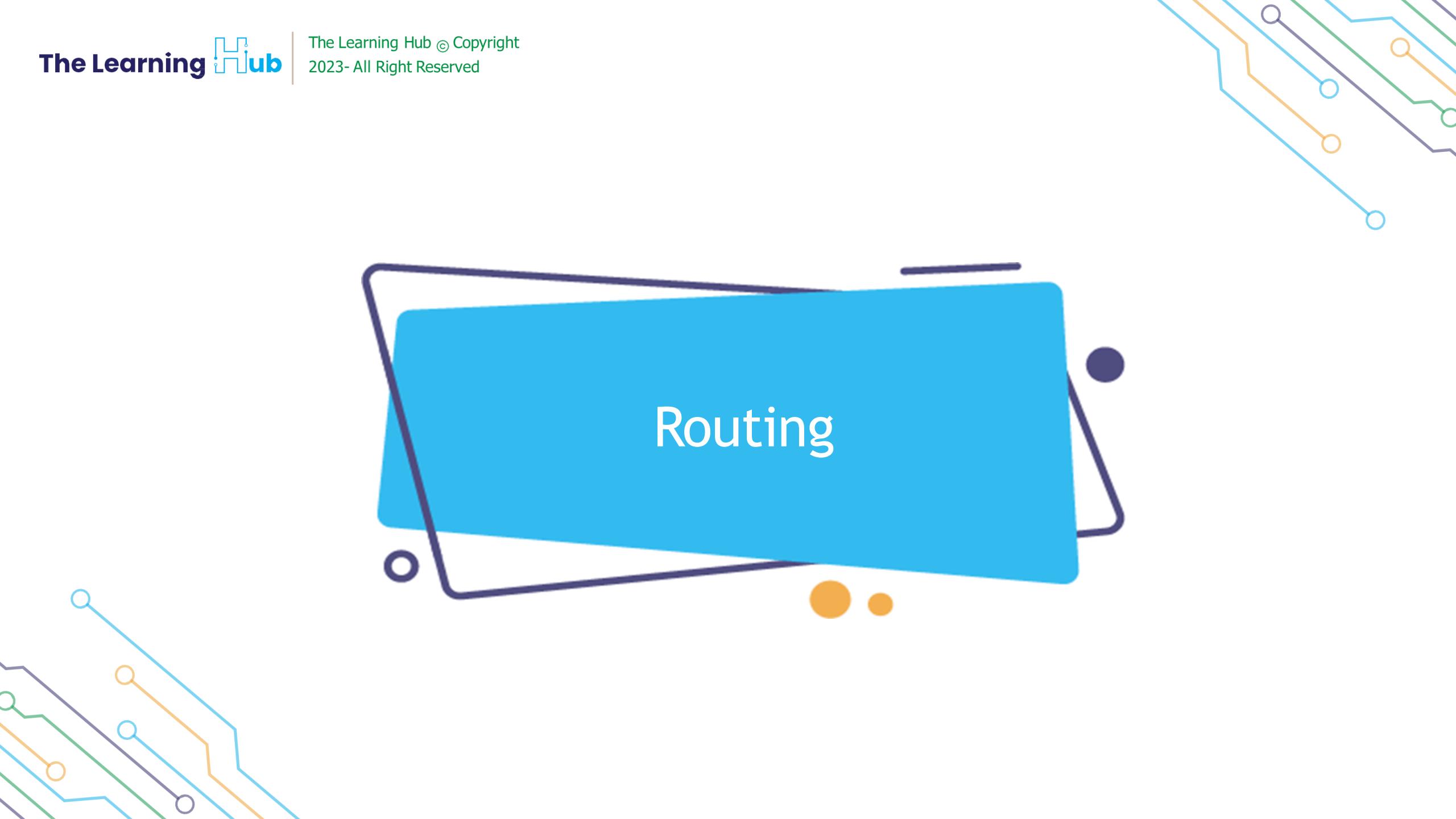
Exercise

Create a shared module that's include a navbar and footer components and transfer the template from the app module to the shared module.

Note: Don't forget to include the shared module in which you want to

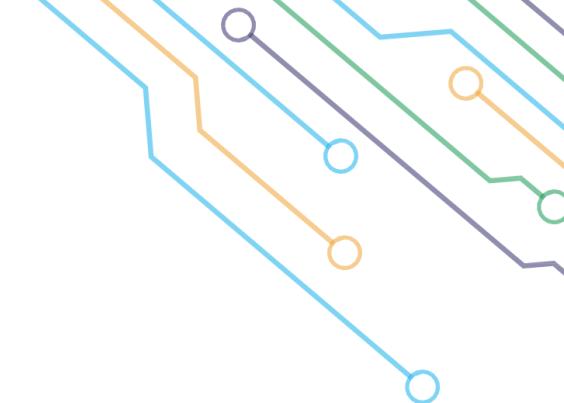
Use the navbar and footer components.





Overview of Routing

Routing allows you to move from one part of the application to another part or one View to another View.



Overview of Routing

Since the app component is the first component that is loaded when the project has run and if you want to go to another page, we will use routing in the app component.

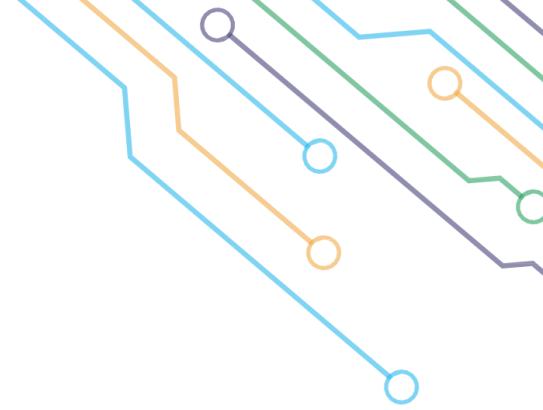
Add this tag in the app.component.html:

→ <router-outlet></router-outlet>



Example of Routing

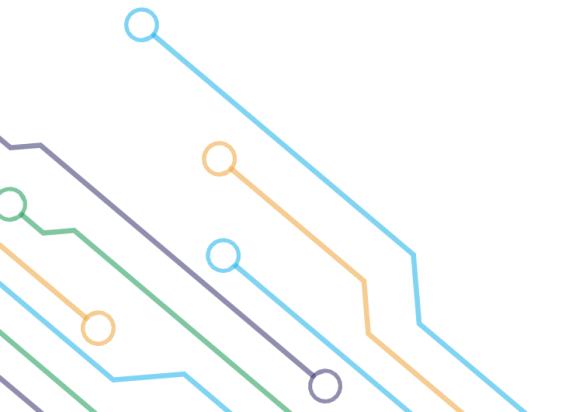
Generate a new component called about us and another component called contact us in the app module.



Example of Routing

Generate the about us component in the app module:

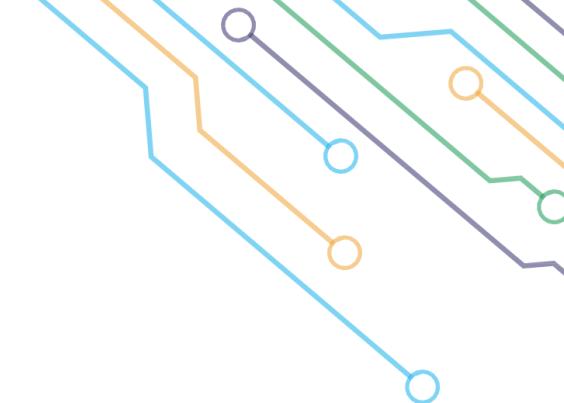
```
PS C:\Users\d.kanaan.ext\Desktop\TheLearningHub> ng g c aboutUs
CREATE src/app/about-us/about-us.component.html (23 bytes)
CREATE src/app/about-us/about-us.component.spec.ts (567 bytes)
CREATE src/app/about-us/about-us.component.ts (209 bytes)
CREATE src/app/about-us/about-us.component.css (0 bytes)
UPDATE src/app/app.module.ts (819 bytes)
```



Example of Routing

Generate the contact us component in the app module:

- PS C:\Users\d.kanaan.ext\Desktop\TheLearningHub> `ng g c contactUs`
CREATE src/app/contact-us/contact-us.component.html (25 bytes)
CREATE src/app/contact-us/contact-us.component.spec.ts (581 bytes)
CREATE src/app/contact-us/contact-us.component.ts (217 bytes)
CREATE src/app/contact-us/contact-us.component.css (0 bytes)
UPDATE src/app/app.module.ts (727 bytes)

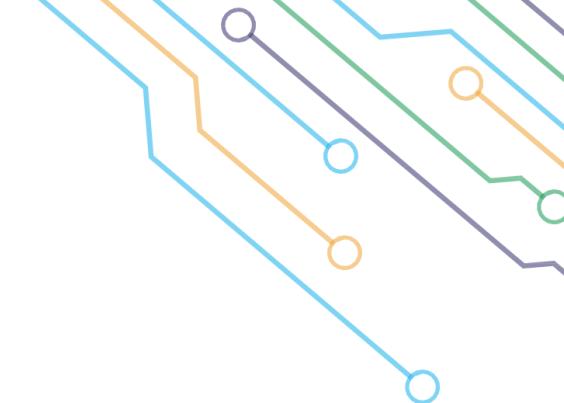


Overview of Routing

The first page will load **app components** and if you want to go to the about us component in the same module (**App Module**) you should use routing.

In-app-routing.module.ts. There is an array called **routes**
this array is used to add the route for all components and other modules.





Overview of Routing

Each route is an object, and each route consists of a path and the component name.

path: where you want to go for example /about us, but you must write the page name without using /.

component: component name which you want to display.



Example of Routing

```
const routes: Routes = [  
  {  
    path: 'about',  
    component: AboutusComponent  
  }  
];
```

NOTE: Once you write the component name it will import this component like this:

```
import { AboutusComponent } from './aboutus/aboutus.component';
```

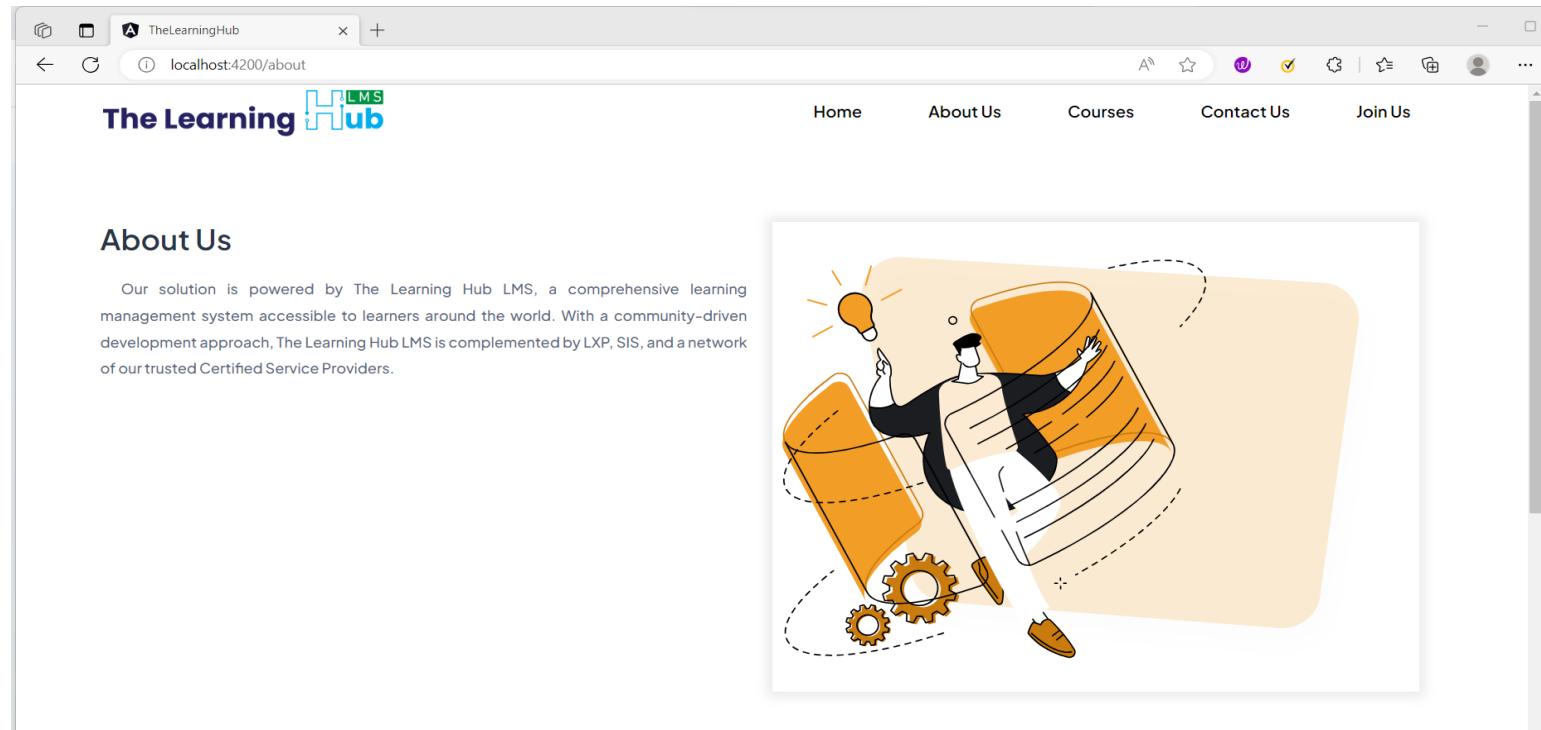
Example of Routing

To add more than one route separate them with a comma.

```
const routes: Routes = [
  {
    path: 'about',
    component: AboutusComponent
  },
  {
    path: 'contact',
    component: ContactusComponent
  }
];
```

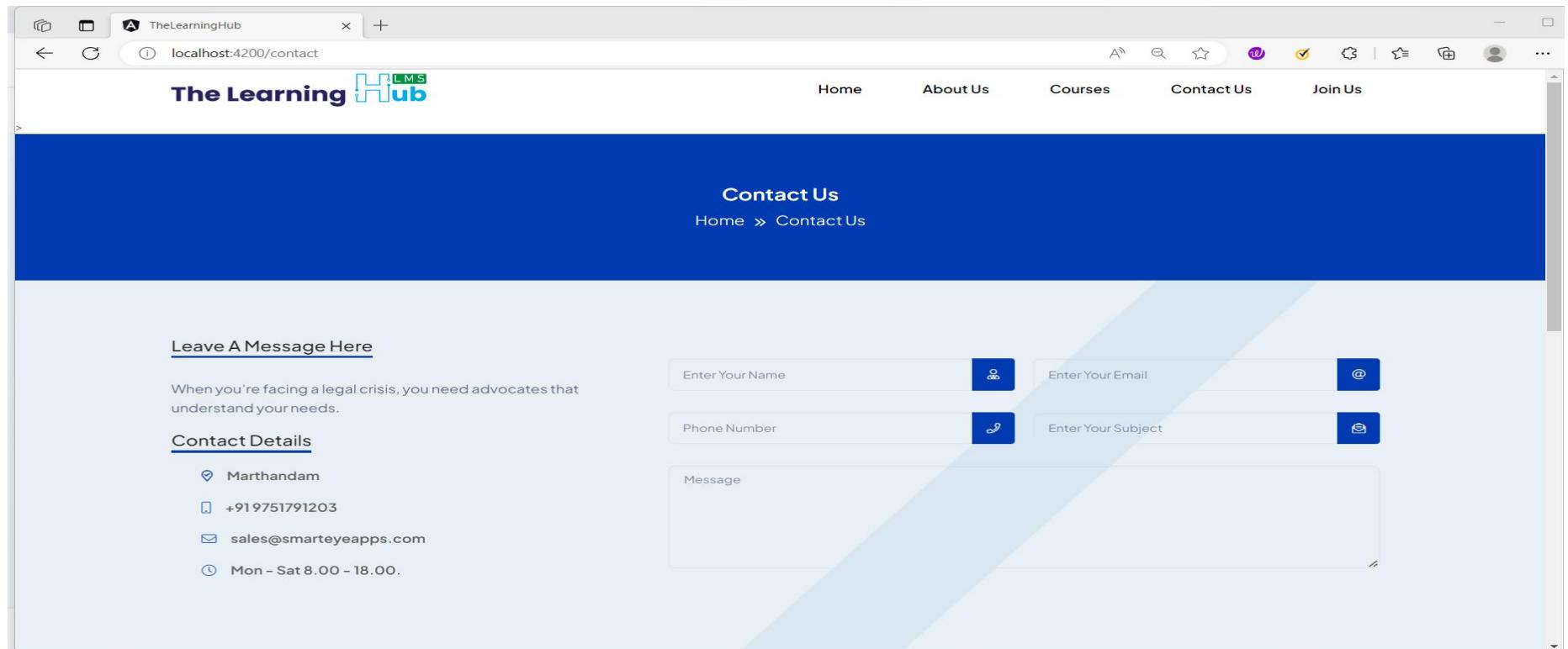
Example of Routing

Because the about us component in the app module use /about in the URL.

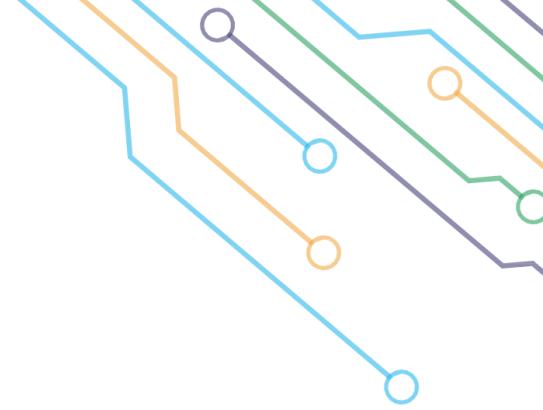


Example of Routing

Contact us component.



Default Routing

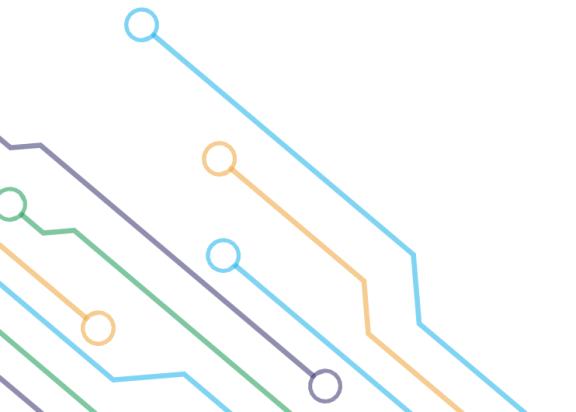


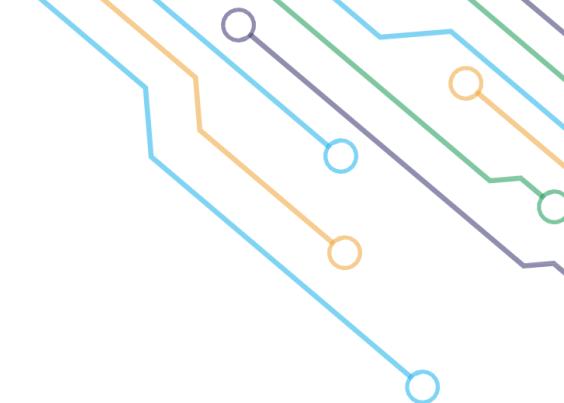
Overview of Default Routing

The default route is redirected -for example- to the home path.

This means that, when you navigate to the root of your application /, you are redirected to the home path (/home).

The path is empty, indicating the default route.



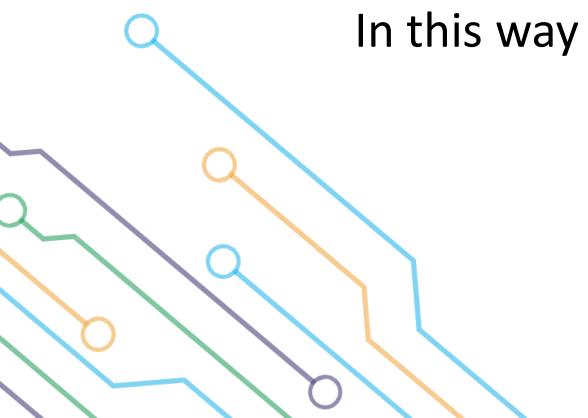


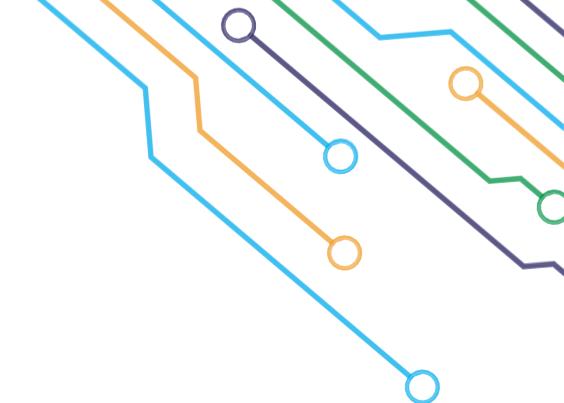
Overview of Default Routing

The default route of the project is the About Us component.

```
const routes: Routes = [  
  {  
    path: '',  
    component: AboutComponent  
  },  
]
```

In this way when the project was run the about us component will load.



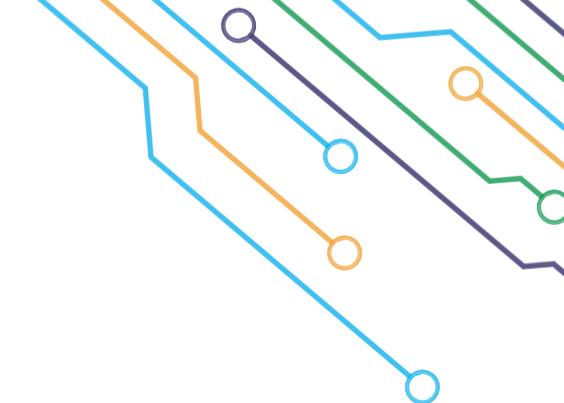


Eager loading

The eager loading of a resource occurs the moment the code is executed. The eager loading process also involves preloading related entities linked to a resource.

In other words, eager loading refers to loading modules before the application starts.





Lazy Loading

NgModules are eagerly loaded by default, so as soon as an application loads, all its NgModules load too, regardless of whether they are required or not.

It's a design pattern that loads NgModules as needed.

Lazy loading helps keep initial bundle sizes smaller, which in turn helps decrease load times.



Lazy Loading

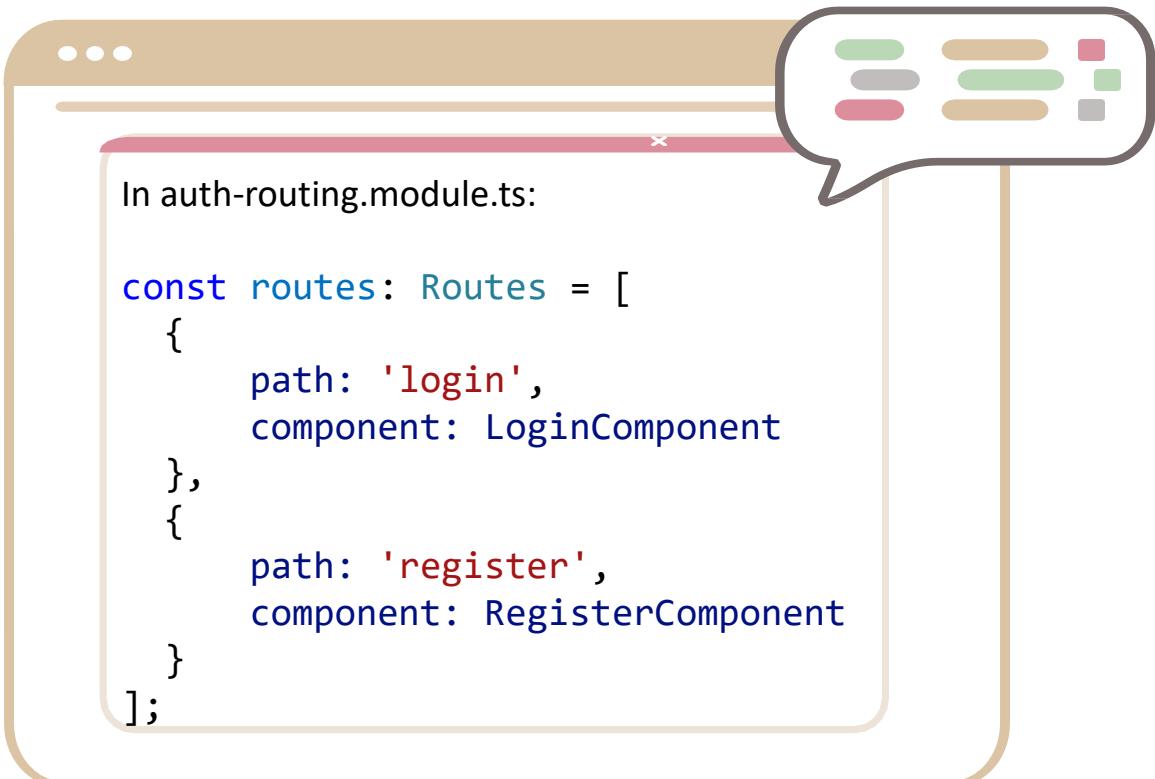
It is necessary to load a feature module lazily using the **loadChildren** property in route configuration and that feature module must not be imported into the application module.

Exercise

Create a route for auth module to login and register components.



Exercise Solution :

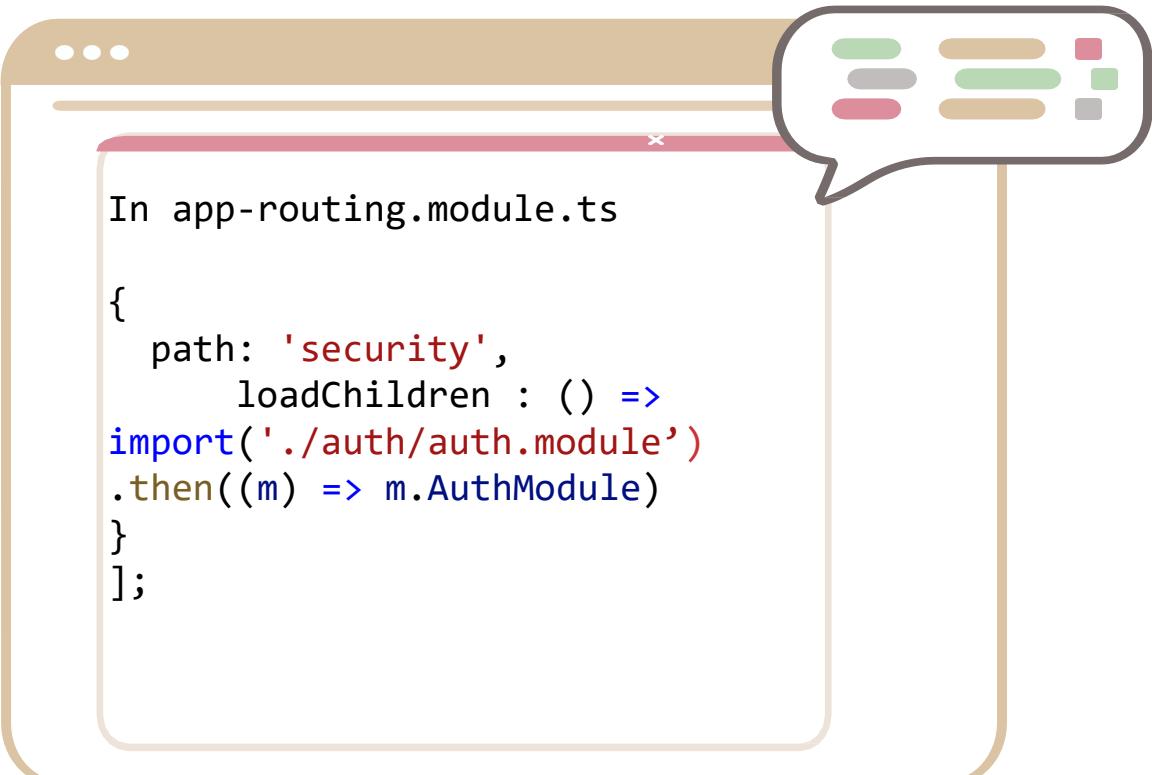


In auth-routing.module.ts:

```
const routes: Routes = [
  {
    path: 'login',
    component: LoginComponent
  },
  {
    path: 'register',
    component: RegisterComponent
  }
];
```



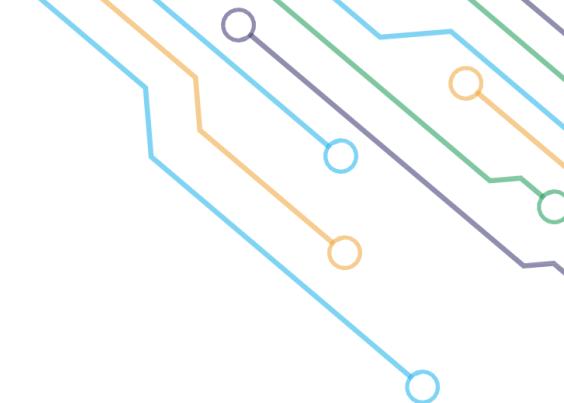
Exercise Solution :



```
In app-routing.module.ts

{
  path: 'security',
  loadChildren : () =>
import('./auth/auth.module')
.then((m) => m.AuthModule)
}
];
```



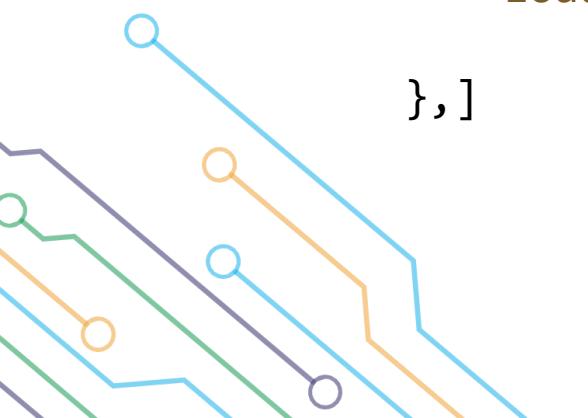


Example of Default Routing

If you want to make the login page the first page to be loaded.

In-app-routing.module.ts

```
const routes: Routes = [
  {
    path: '',
    loadChildren: () => import('./auth/auth.module').
      then((m) => m.AuthModule)
  },
]
```



Example of Default Routing

And In auth-routing.module.ts :

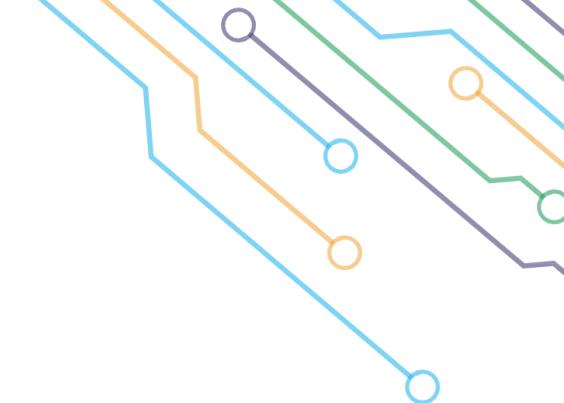
```
const routes: Routes = [
  {
    path: '',
    component: LoginComponent
  },
  {
    path: 'register',
    component: RegisterComponent
  }
];
```

Angular

Tahaluf Training Center 2023



Services In Angular

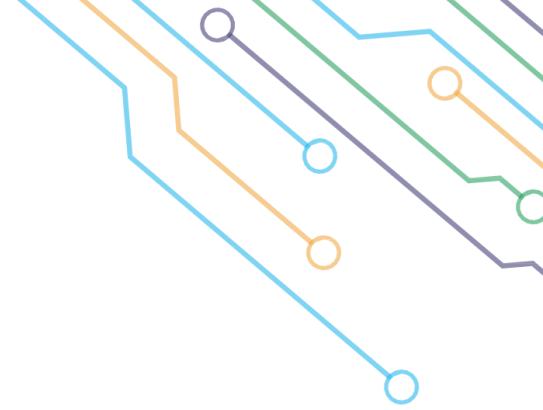


Overview of Services

Service is reusable pieces of code with a defined purpose. Your application will use this code across many components.

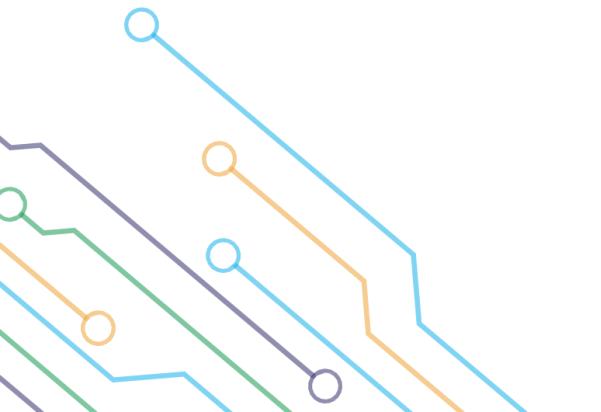
Is basically just another piece in your Angular app, another class you can add that acts as a central repository.





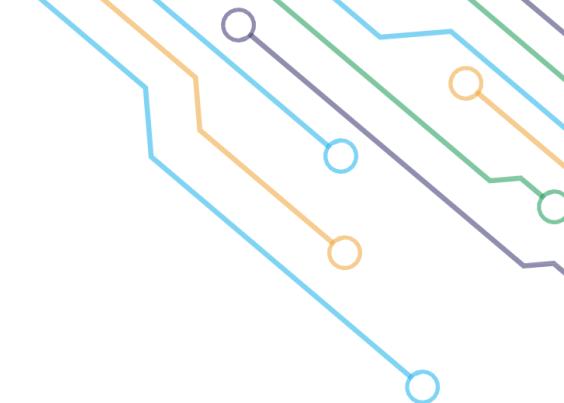
The purpose of Services

- Is to provide features independent of components.
- Sharing logic or data among components.
- Encapsulate interactions with external parties, such as data access.



Advantages of Services

- Testing and debugging are easier on services.
- The service can be reused in many places.



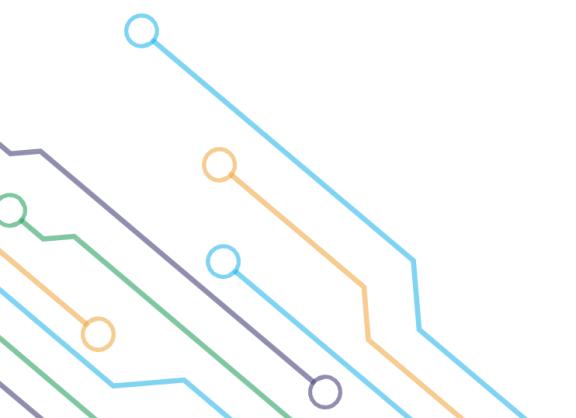
How to generate a new services

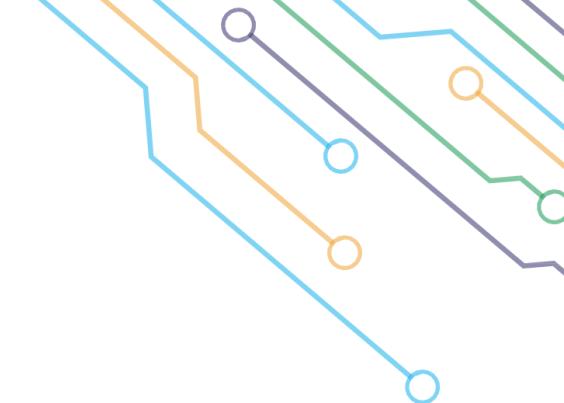
To create a service, use this command on the terminal:

→ `ng generate service service_name`

or you can use this shortcut.

→ `ng g s service_name`



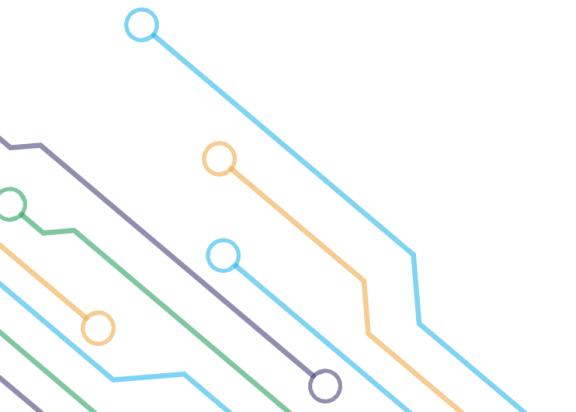


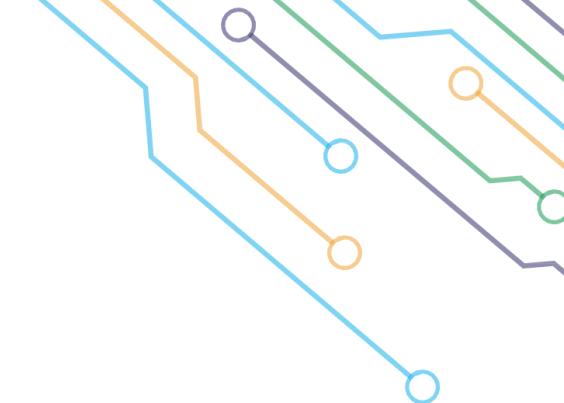
Example

Generate a new service called home.

Here, the Services is a folder that will contain a home service.

```
PS C:\Users\d.kanaan.ext\Desktop\TheLearningHub> ng g s Services/home
CREATE src/app/Services/home.service.spec.ts (347 bytes)
CREATE src/app/Services/home.service.ts (133 bytes)
```





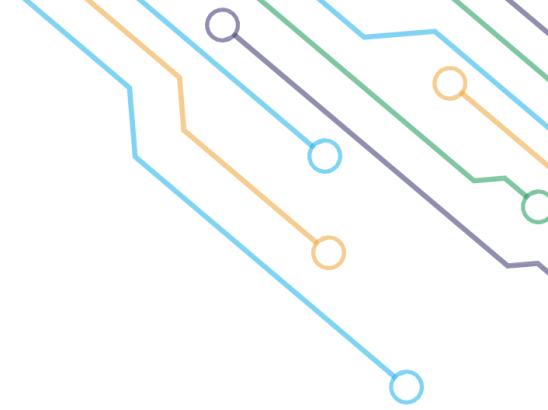
Example

In the home service, define a new property and read the value of it in the home component.

In home.service.ts:

```
export class HomeService {  
  constructor() { }  
  message:string= "Welcome, this message from Home Service :)"  
}
```



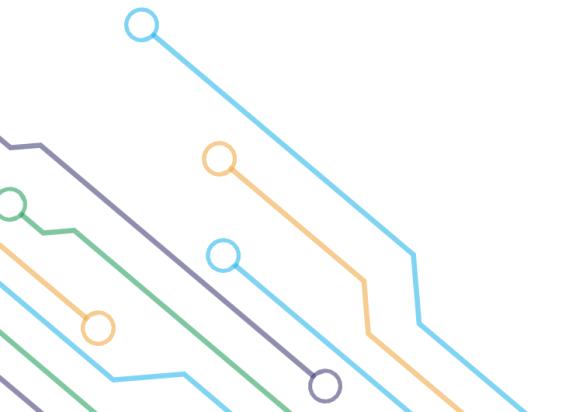


Angular Dependency Injection

You can access the service from any component using the [Angular Dependency Injection](#).

[Angular Dependency Injection](#) injects an instance of this class into our component automatically.

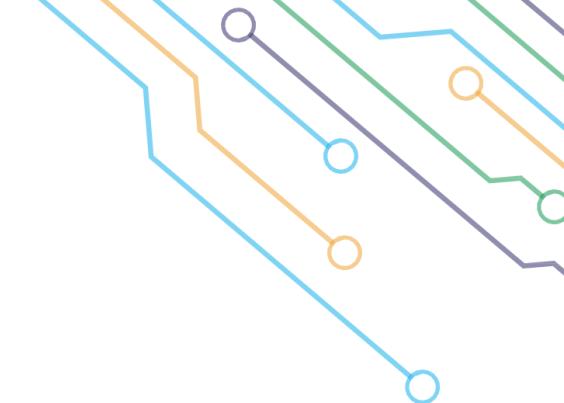
To inform Angular that we require such an instance of the services, we add a constructor to the class and inject the service inside it.



Example

```
export class HomeComponent {  
constructor(public home:HomeService){}  
}
```





Example

```
<div class="container custom-container">
  <div class="slider-content">
    <div class="slider-head">
      <div class="main-content">
        <h2> {{home.message}} </h2>
        <h1> broaden your <span>knowledge</span> with tutors </h1>
```



Exercise

Read the value of the message property from the login component and if the user logged in successfully, update the message to “You are logged In ”.



Exercise

Generate a new Service called Auth service, the purpose of this service is to add the functionality of the login and register components.

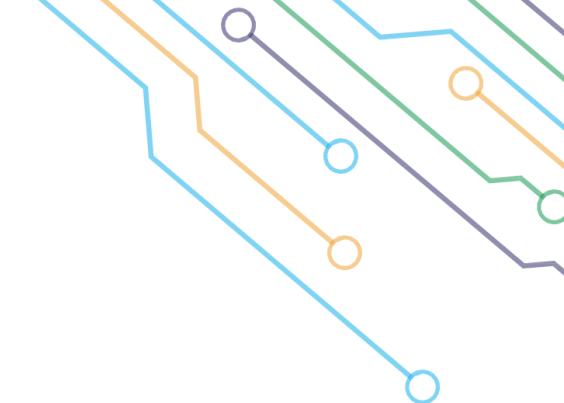


Communicating with backend services using HTTP

Overview of Communicating with backend services using HTTP

To access various back-end services and download data, most front-end applications communicate with a server via the HTTP protocol.

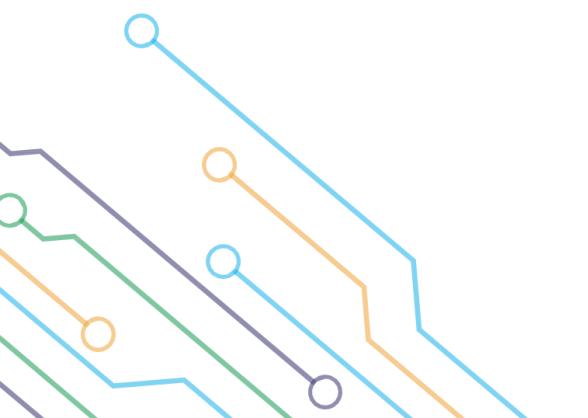
There is an HTTP API provided by Angular, the HttpClient class in `@angular/common/http`.

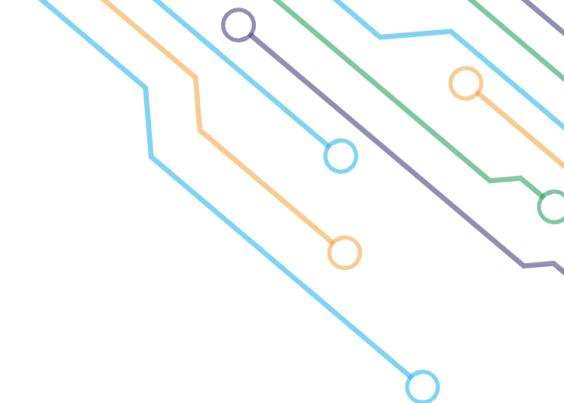


Overview of HttpClient

The HttpClient service is an injectable service that provides powerful methods for communicating with the remote server.

HttpClient API makes it easy to perform an HTTP POST, GET, PUT, and DELETE requests.





Overview of HttpClient

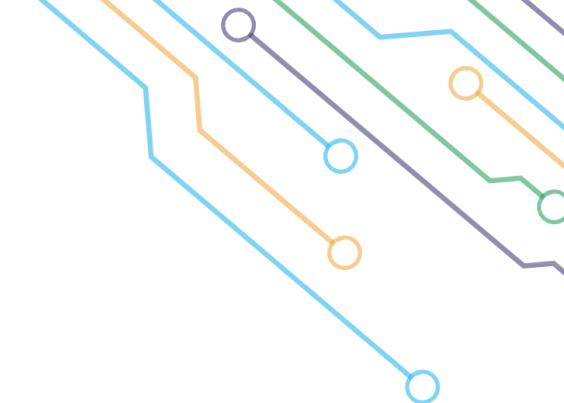
Angular provides a built-in package called HTTP.

To use this package, first, you must add the module for this package to the shared module for our project.

```
import{HttpClientModule}from '@angular/common/http'
```

And add `HttpClientModule` in the import and export array.



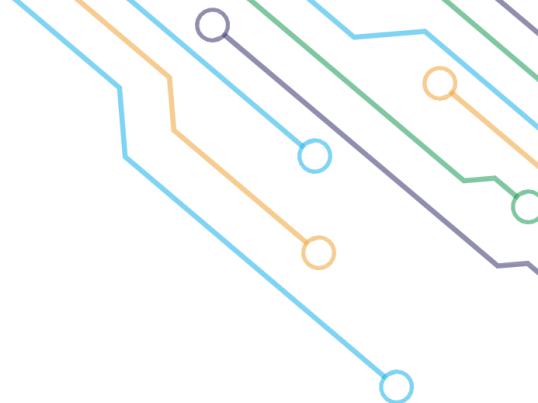


Overview of Cross-Origin Resource Sharing (CORS)

Is a protocol that manages communication between two or more domains.

Is a method based on HTTP headers that allow browsers to identify requests coming from an allowed domain list while also filtering out those requests coming from unknown sources.





Overview of Cross-Origin Resource Sharing (CORS)

This error will be in the console of the browser.

Angular is running in development mode. Call enableProdMode() to enable production mode.

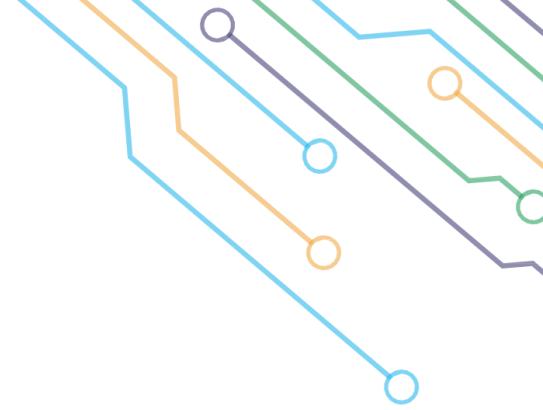
[core.js:28047](#)

[WDS] Live Reloading enabled.

[index.js:52](#)

✖ Access to XMLHttpRequest at '<https://localhost:44379/api/Course>' [home:1](#) ⓘ
from origin '<http://localhost:4200>' has been blocked by CORS
policy: No 'Access-Control-Allow-Origin' header is present on the requested
resource.





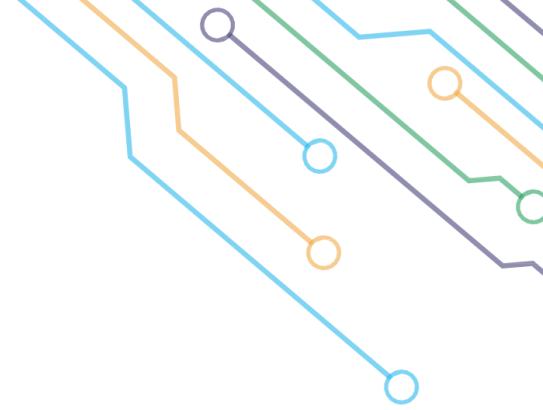
Overview of Cross-Origin Resource Sharing (CORS)

To fix this error add the following lines of code to your API project.

In the **Startup.cs** file → at the beginning of **ConfigureServices** method.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddCors(corsOptions =>
    {
        corsOptions.AddPolicy("policy",
builder =>
{
    builder.AllowAnyOrigin().AllowAnyMethod().AllowAnyHeader();
});
    });
}
```





Overview of Cross-Origin Resource Sharing (CORS)

Then in the **Startup.cs** file → **Configure** method add this line of code:

```
app.UseCors("policy");
```

```
    app.UseRouting();  
  
    app.UseCors("policy");  
  
    app.UseAuthorization();
```

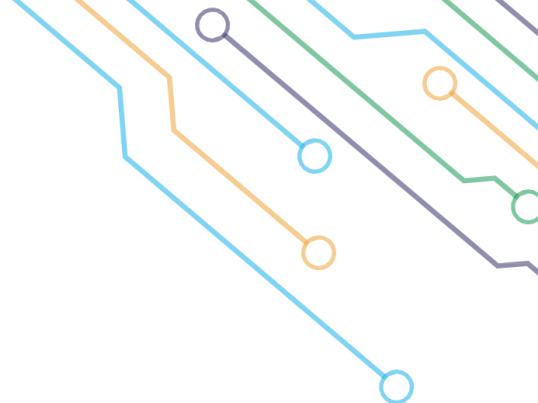


Create Admin Dashboard

Overview of Admin Dashboard

The Dashboard provides a visual representation of your company's reports in a very easy way.

Charts display real-time information (e.g., technician productivity by work type, resource usage by a several number of work orders, etc.)



Create a dashboard for our project

Step one: Create a new module called Admin .

- PS C:\Users\d.kanaan.ext\Desktop\TheLearningHub> `ng g m Admin --routing`
`CREATE src/app/admin/admin-routing.module.ts (248 bytes)`
`CREATE src/app/admin/admin.module.ts (276 bytes)`
- PS C:\Users\d.kanaan.ext\Desktop\TheLearningHub>

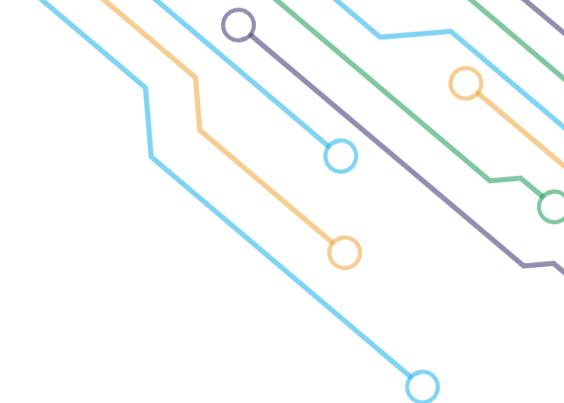


Create a dashboard for our project

Step two: add the route of the admin module in the root module.

In-app-routing.module.ts

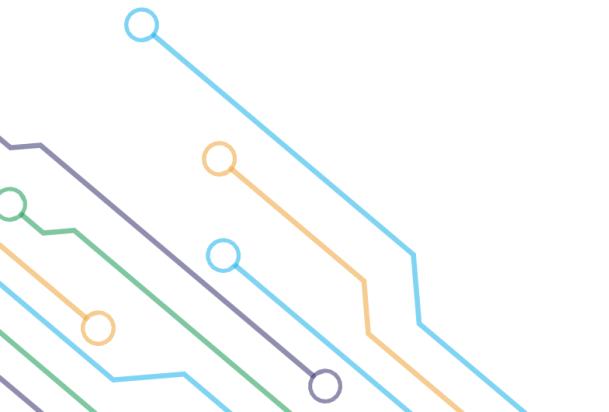
```
{  
  path: 'admin',  
  loadChildren: () => AdminModule  
}
```



Create a dashboard for our project

Step three: Create a new component called a sidebar inside the Admin module and another one called a dashboard.

Note: The purpose of this component is to add a sidebar template for the admin responsibilities.



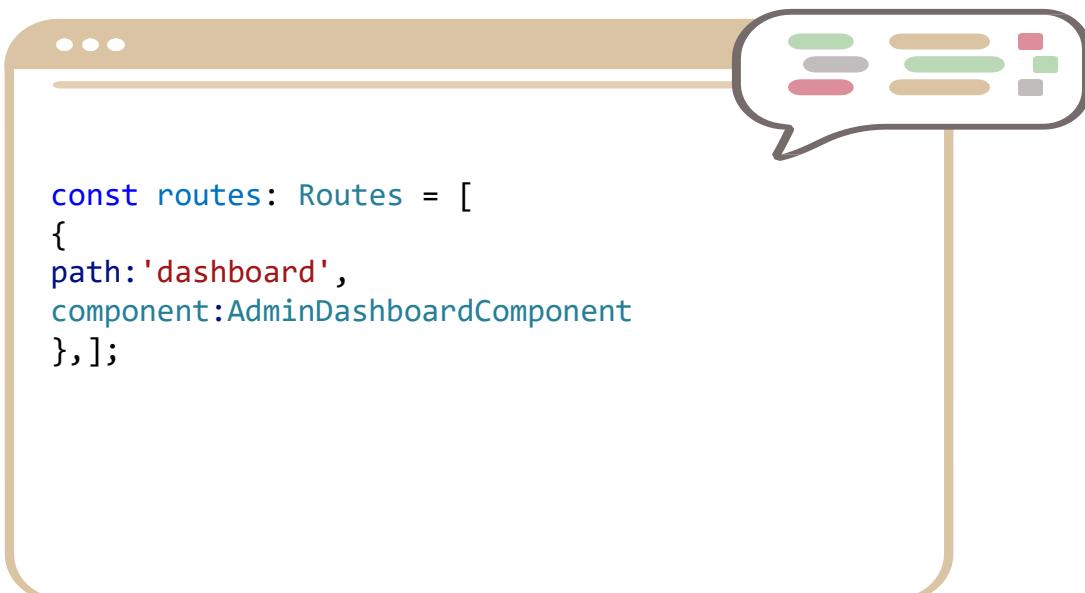
Create a dashboard for our project

- PS C:\Users\d.kanaan.ext\Desktop\TheLearningHub> `ng g c admin/sidebar`
`CREATE src/app/admin/sidebar/sidebar.component.html (22 bytes)`
`CREATE src/app/admin/sidebar/sidebar.component.spec.ts (566 bytes)`
`CREATE src/app/admin/sidebar/sidebar.component.ts (206 bytes)`
`CREATE src/app/admin/sidebar/sidebar.component.css (0 bytes)`
`UPDATE src/app/admin/admin.module.ts (588 bytes)`

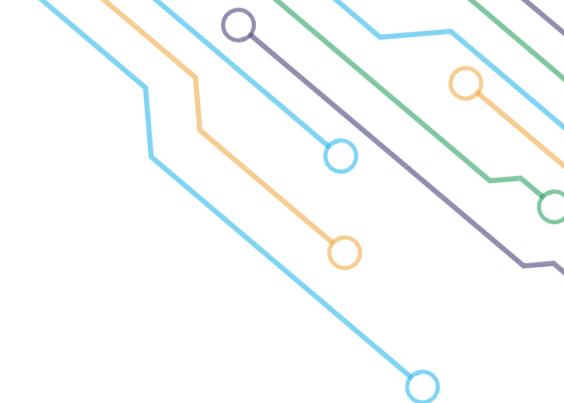
- PS C:\Users\d.kanaan.ext\Desktop\TheLearningHub> `ng g c admin/AdminDashboard`
`CREATE src/app/admin/admin-dashboard/admin-dashboard.component.html (30 bytes)`
`CREATE src/app/admin/admin-dashboard/admin-dashboard.component.spec.ts (616 bytes)`
`CREATE src/app/admin/admin-dashboard/admin-dashboard.component.ts (237 bytes)`
`CREATE src/app/admin/admin-dashboard/admin-dashboard.component.css (0 bytes)`
`UPDATE src/app/admin/admin.module.ts (394 bytes)`

Create a dashboard for our project

Step four: Add the routing for the dashboard inside the admin module.



```
const routes: Routes = [
  {
    path: 'dashboard',
    component: AdminDashboardComponent
  },
]
```



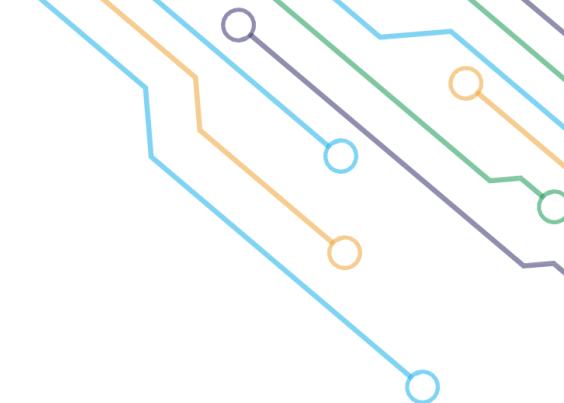
Create a dashboard for our project

Step five: Use this link to add the sidebar style to the sidebar component.

<https://themewagon.com/themes/modernize/>

Note: Modify the sidebar to fit the admin's responsibilities according to the project.



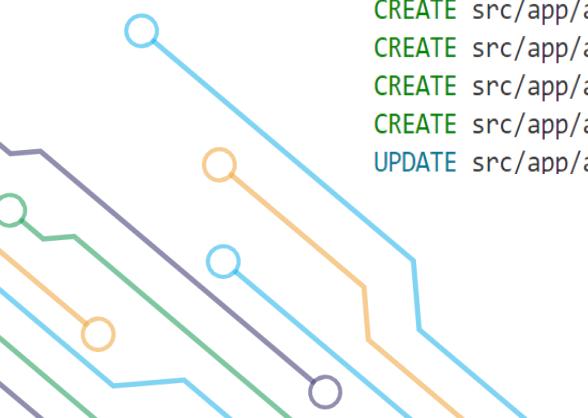


Create a dashboard for our project

Create a new component called manage-course inside the Admin module.

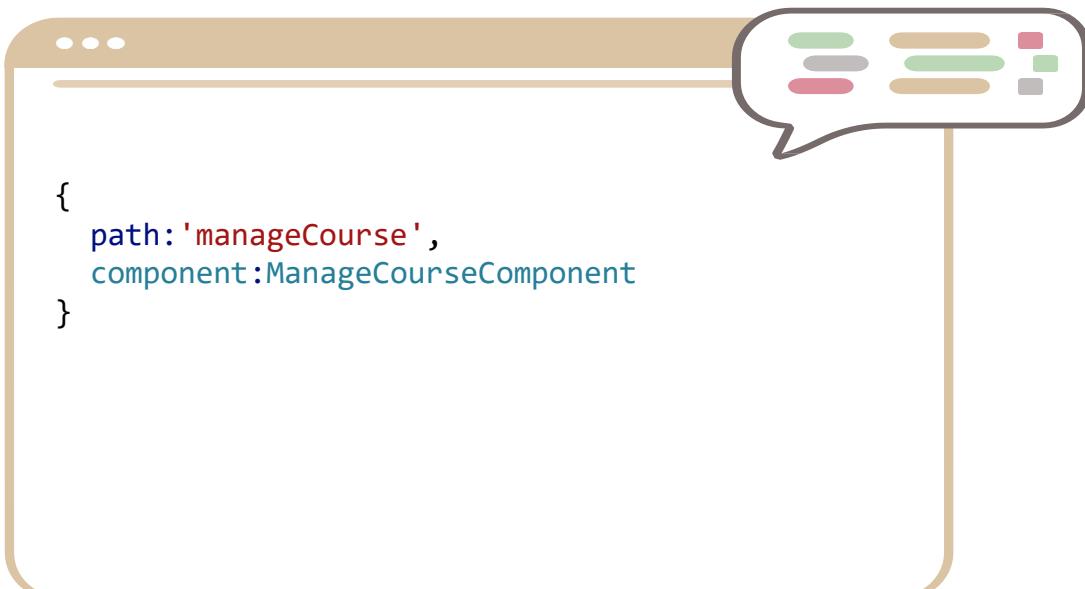
Note: The purpose of this component is to add all functionality for the course table such as : get all courses, create a new course, update specific courses, and delete a specific one.

- PS C:\Users\d.kanaan.ext\Desktop\TheLearningHub> `ng g c admin/ManageCourse`
CREATE src/app/admin/manage-course/manage-course.component.html (28 bytes)
CREATE src/app/admin/manage-course/manage-course.component.spec.ts (602 bytes)
CREATE src/app/admin/manage-course/manage-course.component.ts (229 bytes)
CREATE src/app/admin/manage-course/manage-course.component.css (0 bytes)
UPDATE src/app/admin/admin.module.ts (502 bytes)

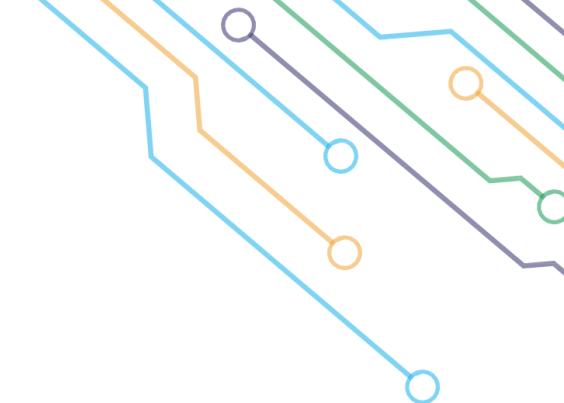


Create a dashboard for our project

Step Six: Add the routing for the manage course component inside the admin module.



API Calls (Get All)



How to retrieve data from a table using HttpClient Service

Step one: Create an object of httpClient service in the constructor.

Step two: Use this object to access the HTTP methods like (get, post, put, delete).

Step three: The data will be retrieved as a JSON object, so we need to declare an array to store this data.

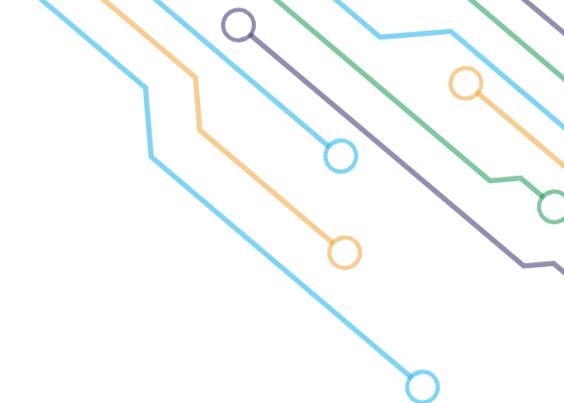


Example using HttpClient methods

Retrieve the data from the course table and display this data in the manage course component.

Step one: In the home service creates an instance of httpClient service.

```
export class HomeService {  
  constructor(private http:HttpClient) { }  
}
```



Example using HttpClient methods

Step two: Declare an array to store the data that will be retrieved.

In-home service.ts

```
courses: any=[{}];
```

Step three: use the HTTP method (get) to receive the data from the course table and save it in the courses array that is declared in the previous step.



Example using HttpClient methods

→ In home.service.ts

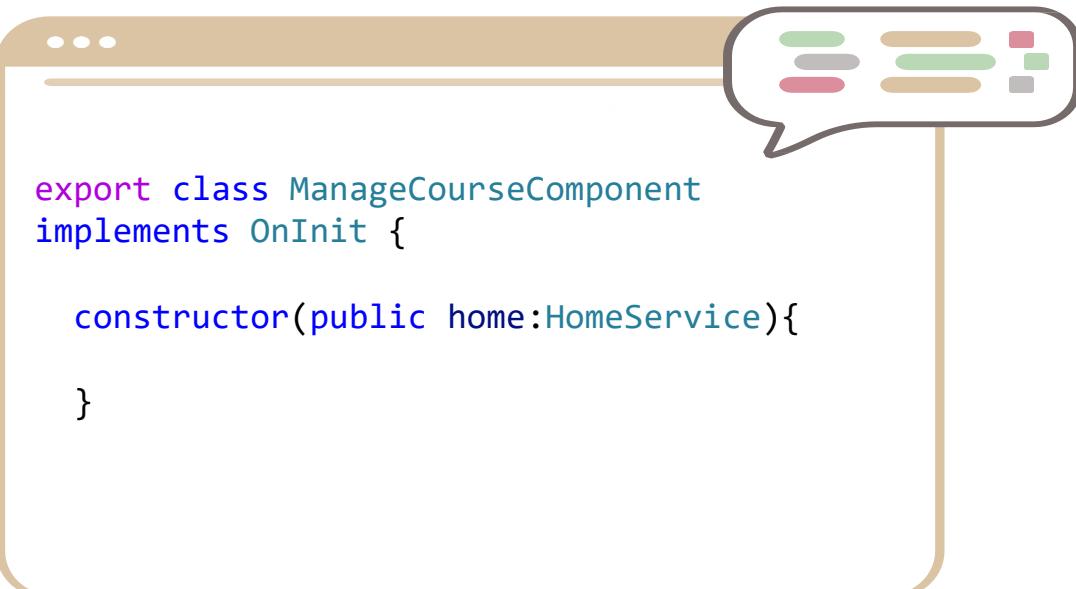


```
export class HomeService {
  constructor(private http:HttpClient) { }
  courses:any=[{}];
  getAllCourses(){
    debugger;
    this.http.get('https://localhost:7074/api/course')
      .subscribe({
        next: (x)=> {this.courses = x},
        error: (x)=> {console.log(x)};
      })
  }
}
```

Example using HttpClient methods

Step four: Create an object of home Service in the Manage Course component.

In manageCourse.component.ts



```
export class ManageCourseComponent
  implements OnInit {
  constructor(public home:HomeService){}
```

Example using HttpClient methods

Step five: Call the get-all method from the home service in the ngOnInit method.

In manageCourse.component.ts



```
ngOnInit(): void {
  this.home.getAllCourses();
}
```

Note: Make sure the component implements the **OnInit** interface

Example using HttpClient methods

Step Six: Bind the data from the course array.

In manageCourse.component.html

```
<app-sidebar></app-sidebar>
<div class="col-lg-8 d-flex align-items-stretch" style="width: 100%; font-size: 15px; margin-left: 285px; overflow-x: hidden;">
  <div class="card w-100">
    <div class="card-body p-4">
      <h3 class="card-title fw-semibold mb-4">All Courses</h3>
      <button>Create One</button>
      <div class="table-responsive">
        <table class="table text-nowrap mb-0 align-middle">
          <thead class="text-dark fs-4">
            <tr>
              <th class="border-bottom-0">
                <h4 class="fw-semibold mb-0">Id</h4>
              </th>
              <th class="border-bottom-0">
                <h4 class="fw-semibold mb-0">Course Name</h4>
              </th>
              <th class="border-bottom-0">
                <h4 class="fw-semibold mb-0">Price</h4>
              </th>
              <th class="border-bottom-0">
                <h4 class="fw-semibold mb-0">Image</h4>
              </th>
              <th class="border-bottom-0">
                <h4 class="fw-semibold mb-0">Operation</h4>
              </th>
            </tr>
          </thead>
```

Example using HttpClient methods

Step Six: Bind the data from the course array.

In manageCourse.component.html

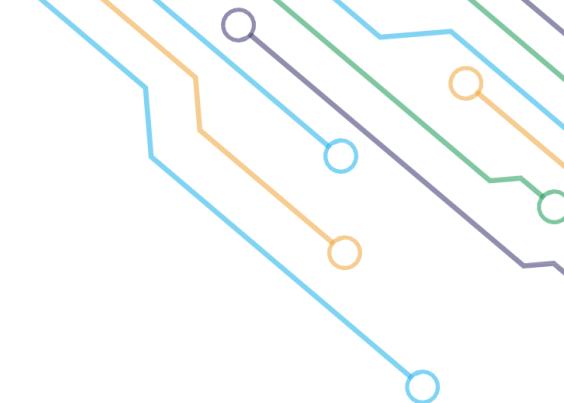
```
<tbody>
  <tr *ngFor="let course of home.courses">
    <td>{{course.course_Id}}</td>
    <td>{{course.course_Name}}</td>
    <td>{{course.price}}</td>
    <td></td>
    <td>
      <button type="button" class="btn btn-primary m-1">Update</button>
      <button type="button" class="btn btn-danger m-1">Delete</button>
    </td>
  </tr>
</tbody>
</table>
</div>
</div>
```

Example using HttpClient methods

The Result

All Courses				
		Create Course		
ID	Course Name	Price	Image	Operation
106	Agile	12		<button>Update</button> <button>Delete</button>
22	Angular	500		<button>Update</button> <button>Delete</button>
46	API	450		<button>Update</button> <button>Delete</button>

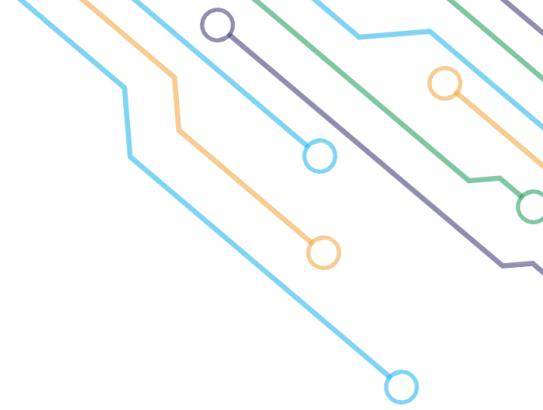
Angular Material



Overview of Angular Material

Angular Material is a User Interface (UI) component library that developers can use in their Angular projects to speed up the development of elegant and consistent user interfaces. Angular Material offers you reusable and beautiful UI components like Cards, Inputs, Data Tables, Datepickers, and much more.





Install Angular Material

Use this command to set up the angular material for your project.

→ ng add @angular/material

It will ask some questions:

1. Choose the name of the theme.

✓ Package information loaded.

The package `@angular/material@16.0.3` will be installed and executed.

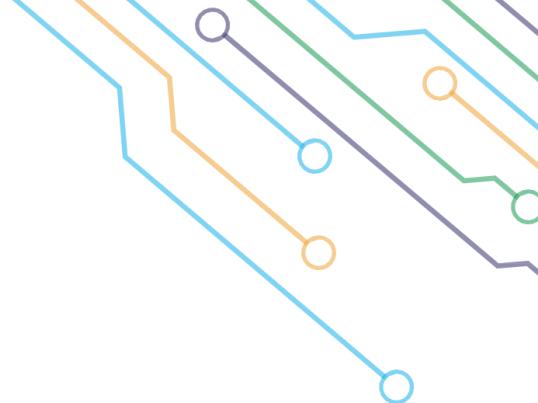
Would you like to proceed? Yes

✓ Packages successfully installed.

? Choose a prebuilt theme name, or "custom" for a custom theme: (Use arrow keys)

- > Indigo/Pink [Preview: <https://material.angular.io?theme=indigo-pink>]
- Deep Purple/Amber [Preview: <https://material.angular.io?theme=deeppurple-amber>]
- Pink/Blue Grey [Preview: <https://material.angular.io?theme=pink-bluegrey>]
- Purple/Green [Preview: <https://material.angular.io?theme=purple-green>]
- Custom





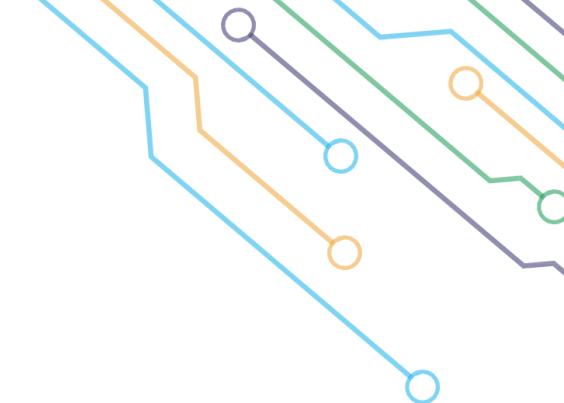
Install Angular Material

2. Set up global Angular Material typography styles.
3. Set up browser animations for Angular Material.

? Set up global Angular Material typography styles? Yes

? Include the Angular animations module? Include and enable animations





Use a component from Angular Material

In order to include any component from angular material in your angular project :

1. From the toolbar select the components.
2. Then select the name of the component you want to add to your project.
3. Go to API and copy the import statement and add it to the shared module.
4. From the Example choose the template.



API Calls (Delete)

How to deal with the Delete function

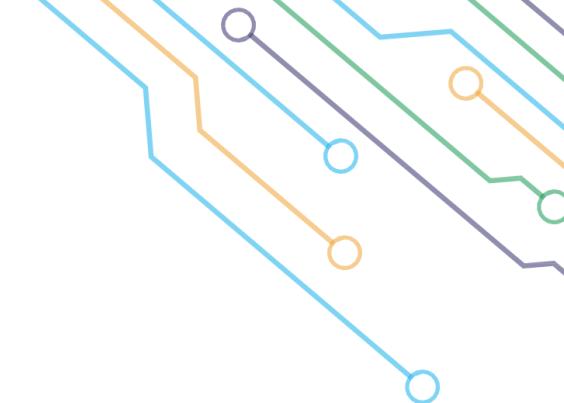
Step one: Add a new function in the home service that hits API to the Delete function

```
deleteCourse(id:number)
{
  this.http.delete('https://localhost:44382/api/course/
DeleteCourse/'+id).subscribe((resp:any)=>{
  console.log('Deleted');
  alert('The Course Deleted Successfully')
},err=>{
  console.log('Something went wrong');
})
}
```

How to deal with the Delete function

Step two: Add a new button called Delete in the HTML file inside the table in the manage course component and send the id as a parameter.

```
<button type="button"  
(click)="openDeleteDailog(course.course_Id)"  
class="btn btn-danger m-1">Delete</button>
```



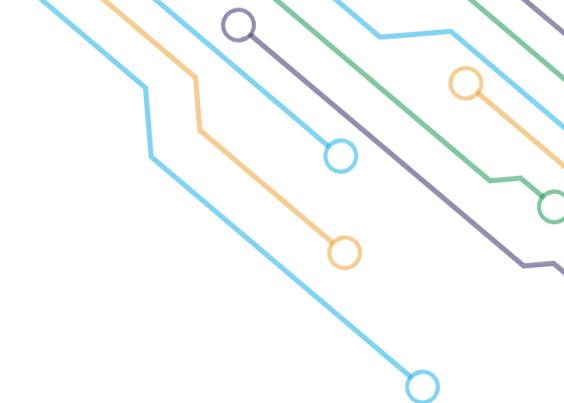
How to deal with the Delete function

Step three: Add the template of the delete form in the HTML file of the manage course component using the dialog from Angular material.

- 1) From the Component go to the dialog.
- 2) Add the API and the name of the module in the shared module in array.

```
imports: [
  CommonModule,
  HttpClientModule,
  MatDialogModule
],
exports: [
  NavbarComponent,
  FooterComponent,
  HttpClientModule,
  MatDialogModule
]
```





How to deal with the Delete function

- 3) Import the shared module in the admin module.

```
imports: [  
    CommonModule,  
    AdminRoutingModule,  
    SharedModule  
]
```



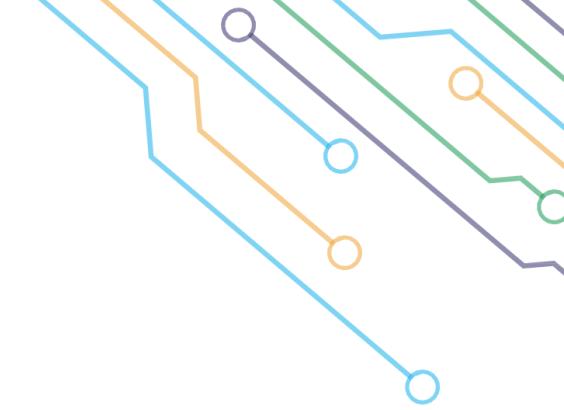
How to deal with the Delete function

Step three: Add the template of the delete form in the HTML file of the manage course component.

Note: Use the `ng-template` which is used to render HTML within Angular templates. Although, it is not rendered directly to the DOM.

The `<ng-template>` is an Angular element for rendering HTML. It is never displayed directly.

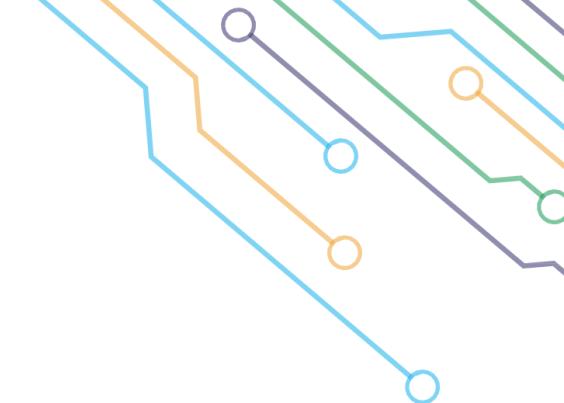
```
<ng-template #callDeleteDialog>
  <div style="padding:15px">
    <h2>Are you sure you want to delete this item?</h2>
    <button class="btn btn-primary m-1" mat-dialog-close="yes">Yes</button>
    <button class="btn btn-danger m-1" mat-dialog-close="no">No</button>
  </div>
</ng-template>
```



How to deal with the Delete function

@ViewChild: Property decorator for configuring the view query. Whenever a change is detected, the change detector looks for the first element or directive matching the selector in the view DOM. As soon as a new child matches the selector in the view DOM, the property is updated.

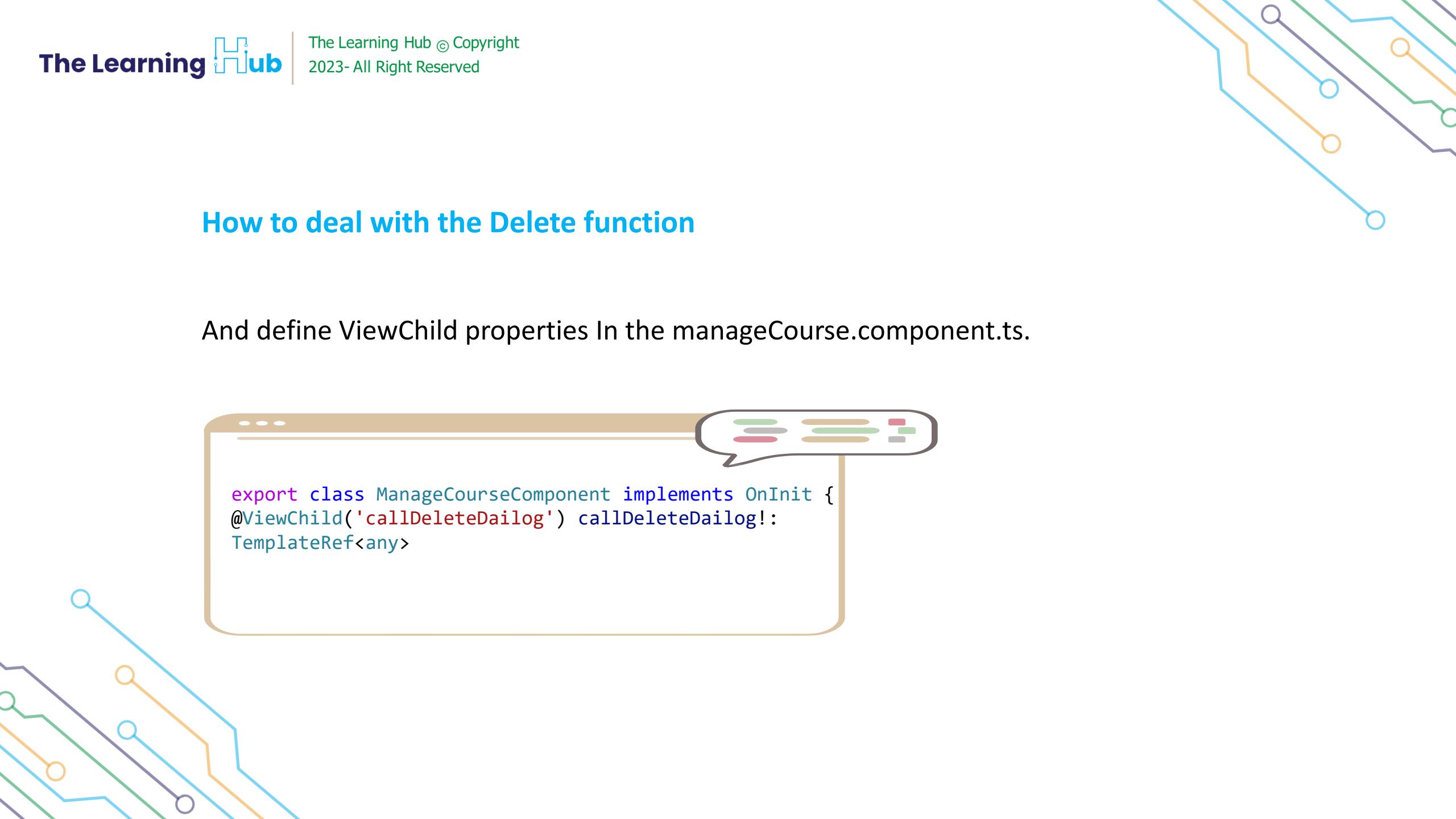




How to deal with the Delete function

And define ViewChild properties In the manageCourse.component.ts.

```
export class ManageCourseComponent implements OnInit {  
  @ViewChild('callDeleteDailog') callDeleteDailog!:  
  TemplateRef<any>
```



How to deal with the Delete function

Step four: Implement the openDeleteDailog function In the managecourse.component.ts



```
constructor(public home: HomeService, public dialog: MatDialog) { }
openDeleteDailog(id: number) {
  const dialogRef = this.dialog.open(this.callDeleteDailog);
  dialogRef.afterClosed().subscribe(result => {
    if (result != undefined) {
      if (result == 'yes')
        this.home.deleteCourse(id);
      else if (result == 'no')
        console.log('Thank you');
    } }) }
```

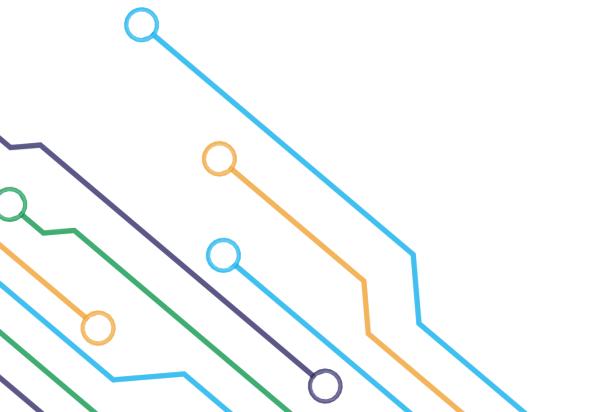
Reactive Forms

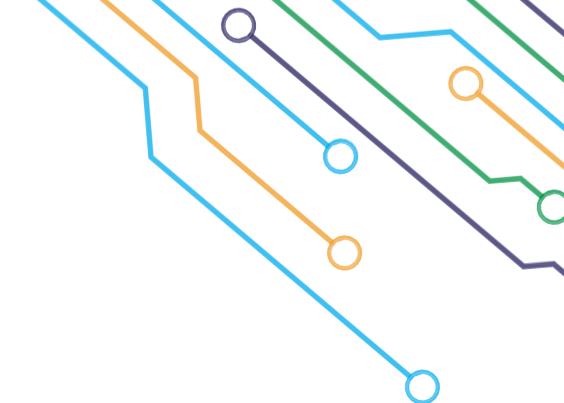


Overview of Reactive Form

Many common applications handle input from users with forms.

Applications use forms to enable users to log in, update a profile, enter sensitive information, and perform many other tasks.



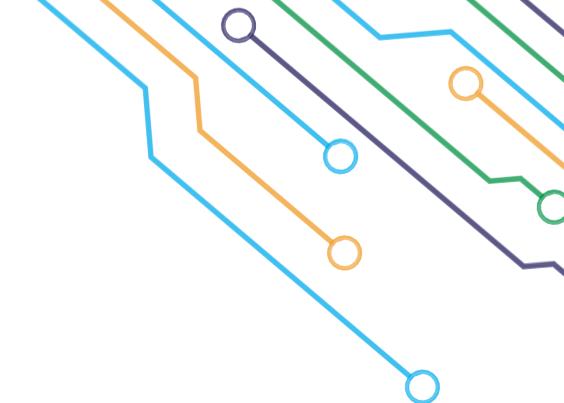


Overview of Reactive Form

Angular supports two ways of handling user input through forms: reactive and template-driven.

It captures user input events from the view, validates the input, creates a form model and data model, and tracks changes.





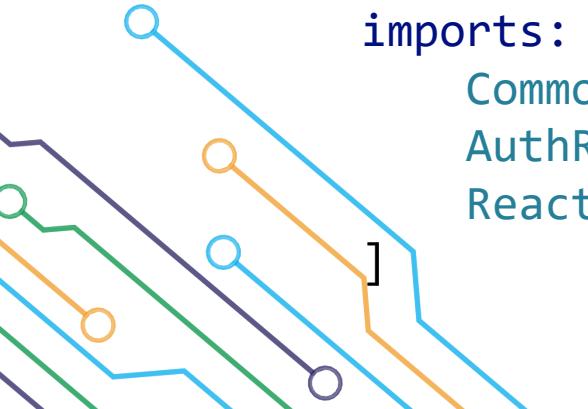
Overview of Reactive Form

To generate an internal representation of a template-driven form, template directives are used.

In reactive forms, you create your own form representation in the component class.

```
import { ReactiveFormsModule } from '@angular/forms';
```

```
imports: [  
  CommonModule,  
  AuthRoutingModule,  
  ReactiveFormsModule]
```



Advantages of Reactive Form

1. Using custom validators.
2. Changing validation dynamically.
3. Dynamically adding form fields

Form Control

Form Group

Form Control



Overview of Form Control

Form Control: Form input instance with multiple functionalities, ex: validation, flag, and data binding.

An Angular form control encapsulates both the value of data and the validation information for each form element.

In a reactive form, every input should be bound to a form control.

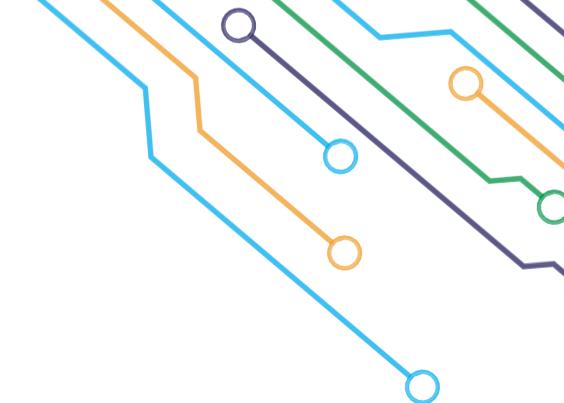


Syntax of Form Control

Use this syntax to create an instance of the form control class:

```
Instance_name = new FormControl(form state, Validation Option)
```

Form Group

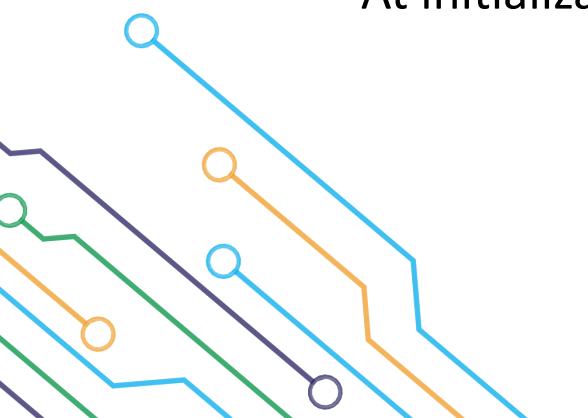


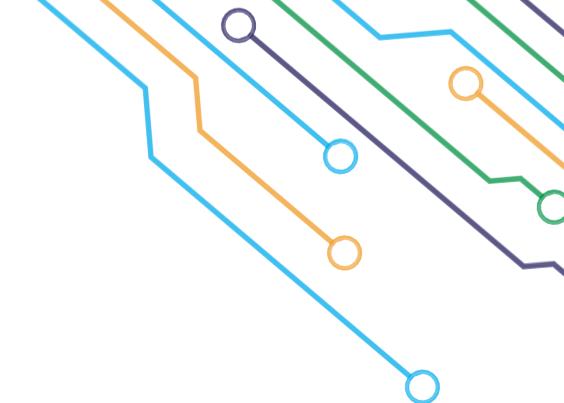
Overview of Form Group

A form group encapsulates a collection of form controls.

Controls give you access to the state of elements, while groups give you access to the state of wrapped controls.

At initialization, every form control in a form group is identified by its name.





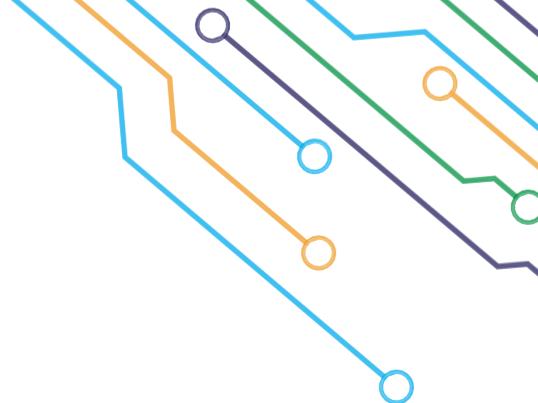
How to deal with a reactive form

To work with reactive forms, you will be using the `ReactiveFormsModule` instead of the `FormsModule`.

Step 1:

Open the `Shared.module.ts` and add `ReactiveFormsModule` in the import and export array.





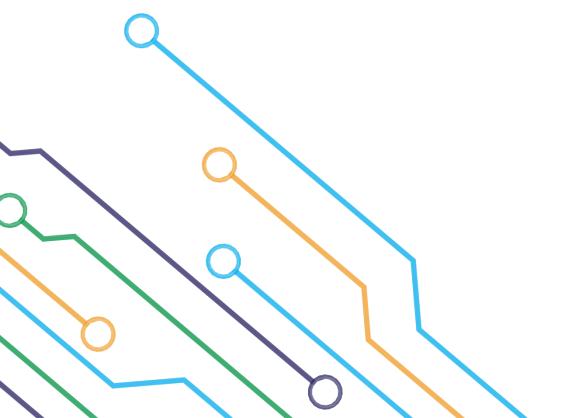
How to deal with a reactive form

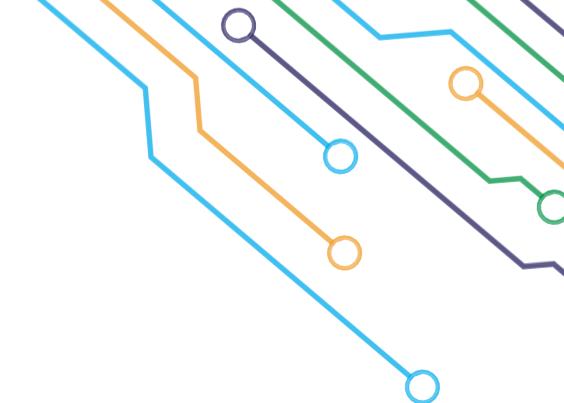
Step 2:

Adding a Form to the Component Template.

Step 3:

Building the Component Class.



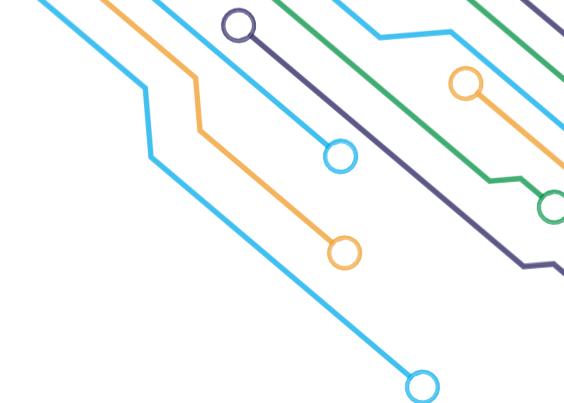


Overview of Form Group

FormGroups are collections of Form controls that track the values and validity of control instances in the group.

One of the building blocks of angular forms is the FormGroup.



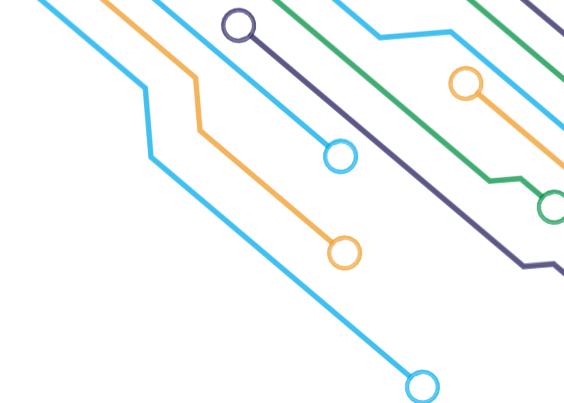


Overview of Form Group

FormGroups encapsulate all information related to a collection of Form Controls that addresses this problem.

Each of these controls has a value and a validation status.



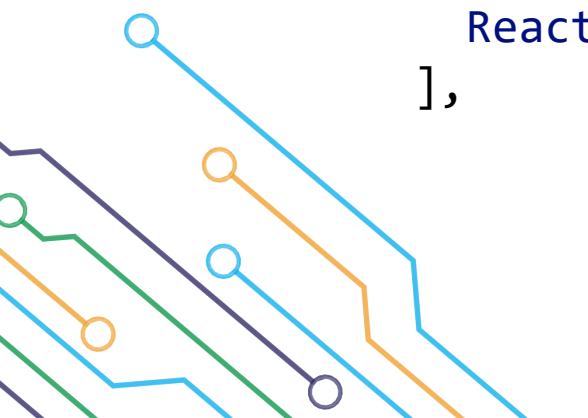


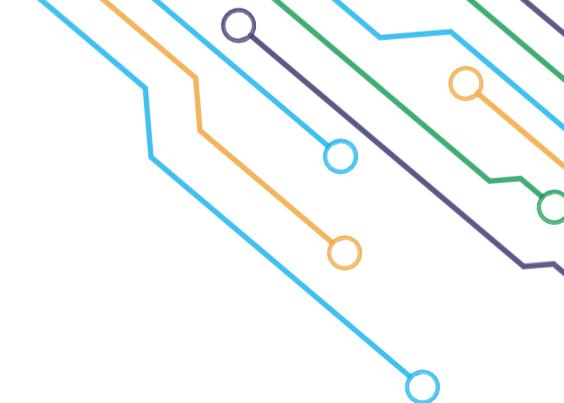
Overview of Form Group

Angular forms can be used by importing the `FormsModule` (for template-driven forms) and `ReactiveFormsModule` (for reactive forms) from the `@angular/forms` package.

```
import { FormsModule, ReactiveFormsModule } from '@angular/forms';

imports: [
  FormsModule,
  ReactiveFormsModule
],
```





Overview of Form Group

FormGroup Demo will be in the register component.

Register form template link:

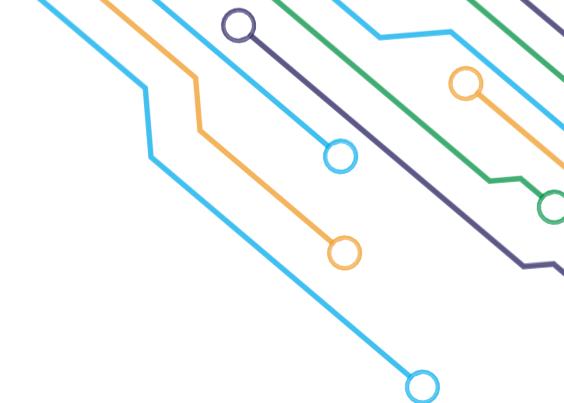
<https://mdbbootstrap.com/docs/standard/extended/registration/>

Angular material link:

<https://material.angular.io/components/input/overview>



API Calls (Create)



Overview Of Http Post

This service is provided as an injectable class that includes methods for making HTTP requests.

During form submission, apps often use POST requests to transmit data to the server.



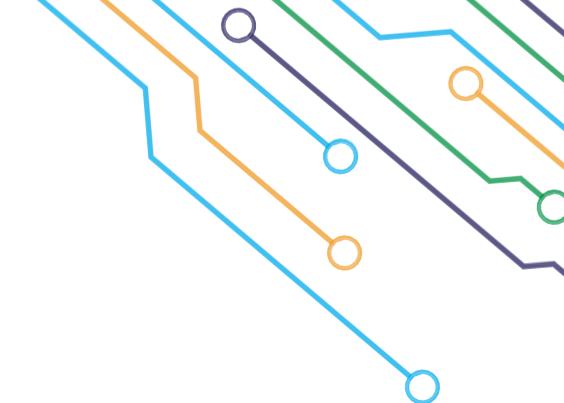


Example HTTP Post

Inside the admin module, create a new component to contain the template of the create new course form.

- PS C:\Users\d.kanaan.ext\Desktop\TheLearningHub> `ng g c admin/createCourse`
`CREATE src/app/admin/create-course/create-course.component.html (28 bytes)`
`CREATE src/app/admin/create-course/create-course.component.spec.ts (602 bytes)`
`CREATE src/app/admin/create-course/create-course.component.ts (229 bytes)`
`CREATE src/app/admin/create-course/create-course.component.css (0 bytes)`
`UPDATE src/app/admin/admin.module.ts (770 bytes)`



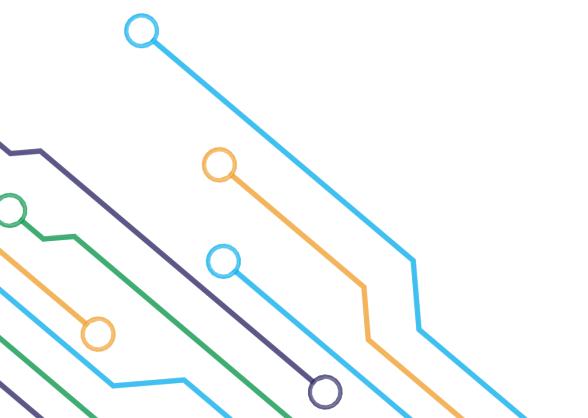


Example HTTP Post

In our project, we'll use an angular material dialog box in the create component.

So, add the API for the dialog in the Shared module.

```
import {MatDialogModule} from '@angular/material/dialog';
```



Example HTTP Post

Add the name of the dialog module in the import and export array.

```
imports: [CommonModule,  
RouterModule,  
FormsModule,  
ReactiveFormsModule,  
MatFormFieldModule,  
MatInputModule,  
MatDialogModule]
```

```
exports:[FormsModule,  
ReactiveFormsModule,  
MatFormFieldModule,  
MatInputModule,  
NavbarComponent,  
FooterComponent,  
MatDialogModule]
```

Example HTTP Post

Define a form group in the create component and use it to send data to the DB.

```
export class CreateCourseComponent {  
 createForm : FormGroup= new FormGroup({  
    course_Name :new FormControl('',[Validators.required]),  
    price :new FormControl('',[Validators.required]),  
    startdate :new FormControl('',[Validators.required]),  
    enddate :new FormControl('',[Validators.required]),  
    imagename :new FormControl('',[Validators.required])  
})
```

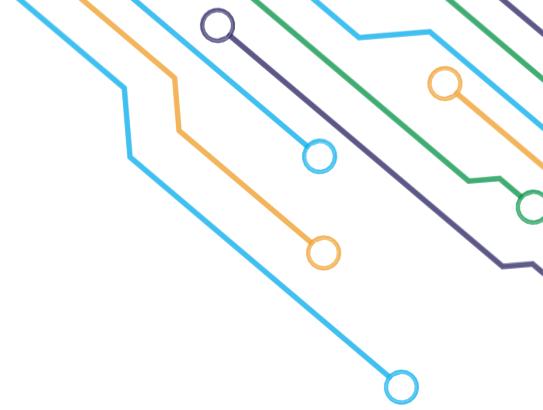
Example HTTP Post

Add the template form in the HTML file for Create component.

```
<h2 mat-dialog-title>Create New Course </h2>
<mat-dialog-content class="mat-typography">
  <form class="example-form" [formGroup]="createForm">
    <mat-form-field class="example-full-width" appearance="fill">
      <mat-label>Course Name </mat-label>
      <input type="text" matInput formControlName="course_Name">
      <mat-error *ngIf="createForm.controls['course_Name'].hasError('required')">
        Course Name is <strong>required</strong>
      </mat-error>
    </mat-form-field>
    <br>
    <mat-form-field class="example-full-width" appearance="fill">
      <mat-label>Price</mat-label>
      <input type="number" matInput formControlName="price">
      <mat-error *ngIf="createForm.controls['price'].hasError('required')">
        price is <strong>required</strong>
      </mat-error>
    </mat-form-field>
    <br>
```

Example HTTP Post

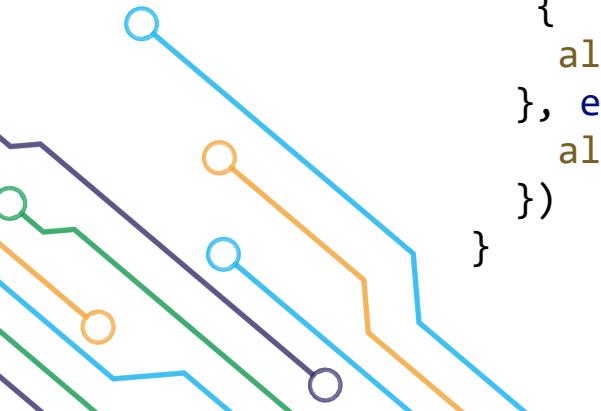
```
<mat-form-field class="example-full-width" appearance="fill">
  <mat-label>Start Date </mat-label>
  <input type="text" matInput formControlName="startdate">
  <mat-error *ngIf="createForm.controls['startdate'].hasError('required')">
    Start Date is <strong>required</strong>
  </mat-error>
</mat-form-field>
<br>
<mat-form-field class="example-full-width" appearance="fill">
  <mat-label>End Date </mat-label>
  <input type="text" matInput formControlName="enddate">
  <mat-error *ngIf="createForm.controls['enddate'].hasError('required')">
    End Date is <strong>required</strong>
  </mat-error>
</mat-form-field>
<br>
```



Example HTTP Post

Create a createCourse function in the home service which calls the API function using the post method.

```
createCourse(body: any) {
  //hits Api (create function)
  debugger
  this.http.post('https://localhost:44382/api/course/'
, body).subscribe((resp: any) =>
  {
    alert('Created Sucessfully');
  }, err => {
    alert('Something went wrong');
  })
}
```



Example HTTP Post

Create a saveCourse function in the create component to call createCourse function from home services.

```
saveCourse()
{
    this.home.createCourse(this.createForm.value)
}
```

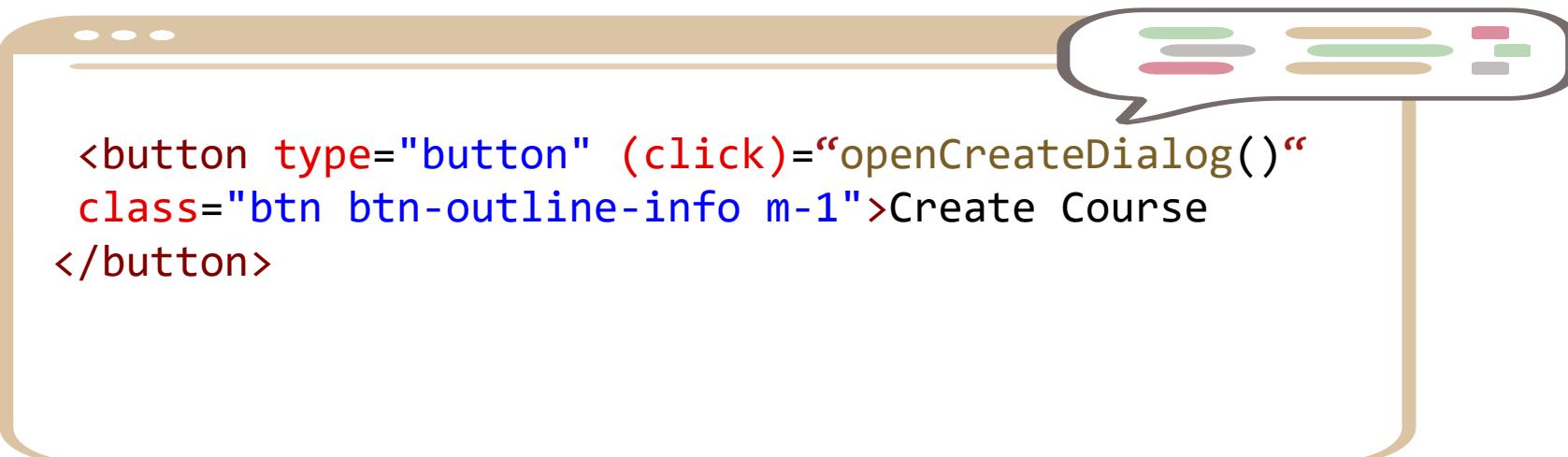
Example HTTP Post

Add a button in the HTML file of the create component to call the SeveCourse function.

```
<mat-dialog-actions align="end">
  <button mat-button mat-dialog-close>Cancel</button>
  <button mat-button (click)="saveCourse()" [mat-dialog-close]="true"
    cdkFocusInitial>Save</button>
</mat-dialog-actions>
```

Example HTTP Post

Add a button in the HTML file of the Manage course component to open the create dialog .



```
<button type="button" (click)="openCreateDialog()"  
class="btn btn-outline-info m-1">Create Course  
</button>
```

Example HTTP Post

In the typescript file for managecourse component create an object of MatDialog Service to use an open method from this service.

```
constructor(private dialog :MatDialog) { }

openCreateDialog() {
  const dialogRef = this.dialog.open(CreateCourseComponent);
}
```

Note: The open method takes the name of the component as a parameter.

Example HTTP Post



Create New Course

Course Name *

Price *

Start Date *
dd/mm/yyyy 

End Date *
dd/mm/yyyy 

[Cancel](#) [Save](#)

Angular

Tahaluf Training Center 2023



- 1 API Calls (Update)
- 2 API Calls (Upload & Retrieve Image).
- 3 Spinner.
- 4 Toastr.





API Calls (Update)

API Calls (Update)

Step one: Add a new function in the home service that calls API.

```
updateCourse(body: any) {
    //hits Api (create function)
    debugger
    this.http.put('https://localhost:44382/api/course/',
body).subscribe((resp: any) =>
{
    alert('Updated Sucessfully');
}, err => {
    alert('Something went wrong');
    console.log(err);
})
```

API Calls (Update)

Step two: Add a form group in an HTML file in the Manage course component to receive the new data for a specific row.

```
updateForm : FormGroup= new FormGroup({  
    course_Id:new FormControl(),  
    course_Name :new FormControl(''),  
    price :new FormControl(),  
    startdate :new FormControl(''),  
    enddate :new FormControl(''),  
    imagename :new FormControl('')  
})
```

API Calls (Update)

Step three: Add a new button in the HTML file inside the table in the Mange course component and send the previous data as a parameter.

```
<button type="button" (click)="openUpdateDailog(course)"  
class="btn btn-primary m-1">Update</button>
```

API Calls (Update)

Step four: define an object to receive previous data.

```
previousData:any={} ;
openUpdateDialog(courseObj:any){
  this.previousData={
    course_Id:courseObj.course_Id,
    course_Name:courseObj.course_Name,
    price:courseObj.price,
    startdate:courseObj.startdate,
    enddate:courseObj.enddate }
  console.log(this.previousData);
  this.updateForm.controls['course_Id'].setValue
  (this.previousData.course_Id); }
```

API Calls (Update)

Step five: Add the template of the update form in the HTML file of the Manage course component.

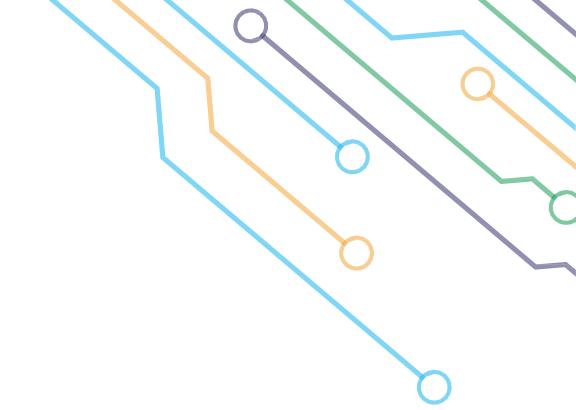
API Calls (Update)

```
<ng-template #callUpdateDialog>
  <h2 mat-dialog-title>Create New Course </h2>
  <mat-dialog-content class="mat-typography">
    <form class="example-form" [formGroup]="updateForm">
      <mat-form-field class="example-full-width" appearance="fill">
        <mat-label>Course Name </mat-label>
        <input type="text" matInput formControlName="course_Name"
               [(ngModel)]="previousData.course_Name" >
      </mat-form-field>
      <br>
      <mat-form-field class="example-full-width" appearance="fill">
        <mat-label>Price</mat-label>
        <input type="number" matInput formControlName="price"
               [(ngModel)]="previousData.price" >
      </mat-form-field>
      <br>
      <mat-form-field class="example-full-width" appearance="fill">
        <mat-label>Start Date </mat-label>
        <input type="date" matInput formControlName="startdate"
               [(ngModel)]="previousData.startdate" >
      </mat-form-field>
      <br>
```

API Calls (Update)

```
<mat-form-field class="example-full-width" appearance="fill">
  <mat-label>End Date </mat-label>
  <input type="date" matInput formControlName="enddate"
    [(ngModel)]="previousData.enddate">
</mat-form-field>

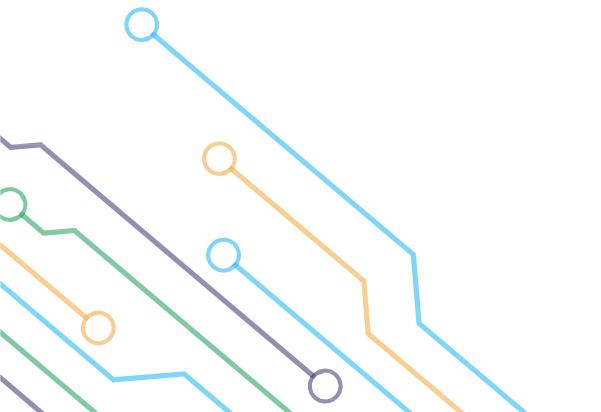
<div class="example-container">
  <input type="file" #file formControlName="imagename"
    (change)="uploadImage(file.files)" >
</div>
<br>
</form>
```



API Calls (Update)

In the managecourse.component.ts:

```
@ViewChild('callUpdateDialog') callUpdateDialog!: TemplateRef<any>
```



API Calls (Update)

Then complete the OpenUpdateDailog.

```
openUpdateDailog(courseObj: any) {
  this.previousData = {
    course_Id: courseObj.course_Id,
    course_Name: courseObj.course_Name,
    price: courseObj.price,
    startdate: courseObj.startdate,
    enddate: courseObj.enddate
  }
  console.log(this.previousData);
  this.updateForm.controls['course_Id'].setValue(this.previousData.course_Id);
  this.dialog.open(this.callUpdateDailog);
```

API Calls (Update)

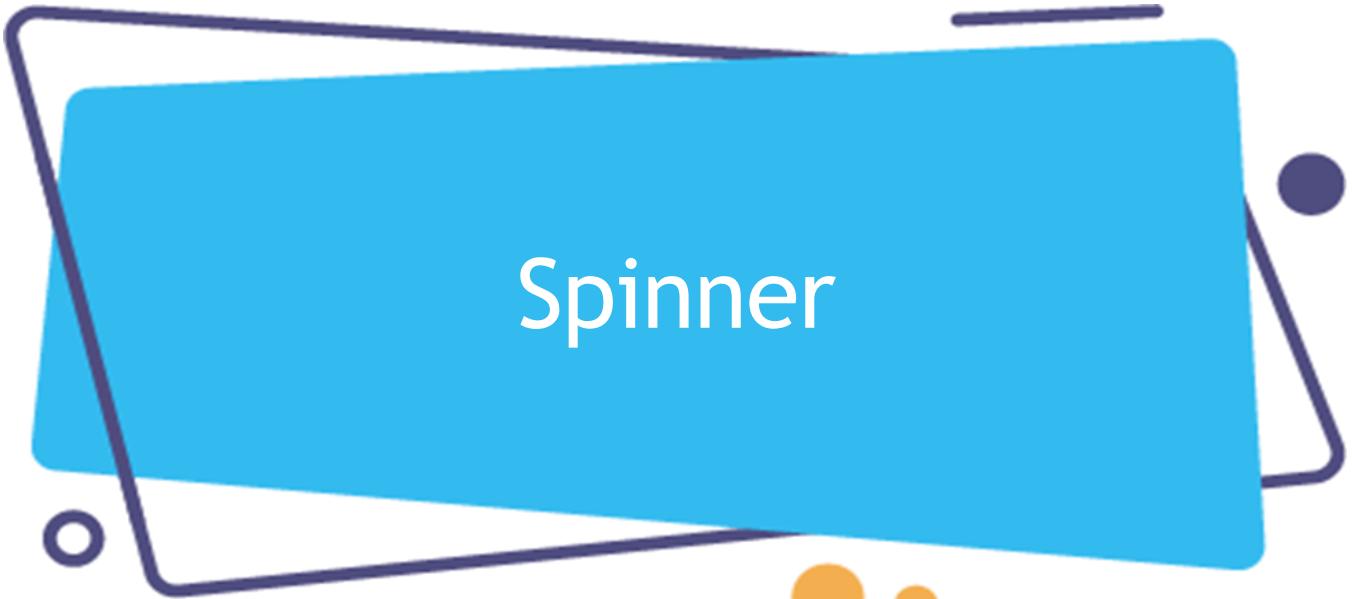
Step six: Add a new button in the update form.

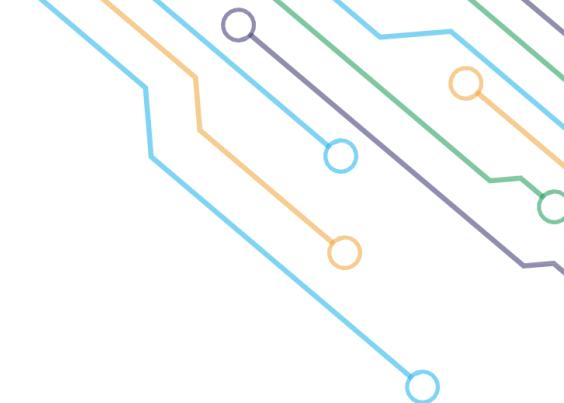
```
<button mat-button (click)="updateCourse()"  
[mat-dialog-close]="true" cdkFocusInitial>Update</button>
```

API Calls (Update)

Step seven: Implement the update course method in the typescript file.

```
updateCourse() {  
    this.home.updateCourse(this.updateForm.value);  
}
```



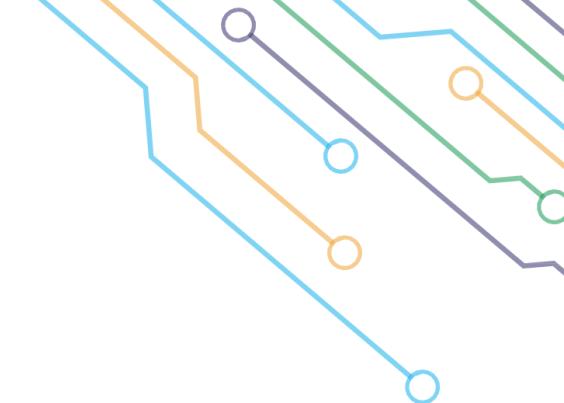


Overview of Spinner

An Angular library for loading spinners, ngx spinners shows that data is loading in real-time to the user and can be used to give the user a notification.

A loader is used to display the loading state of data in an application.





How to add spinner

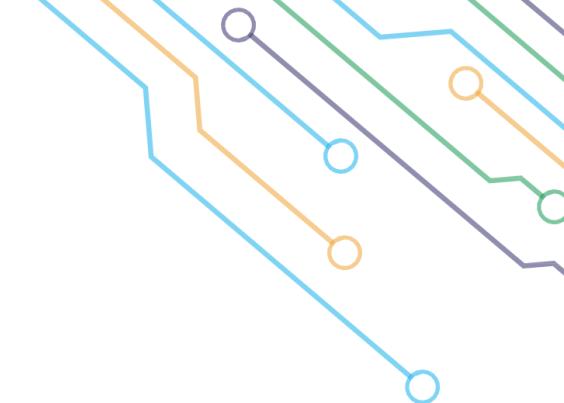
Step one: from the npm website (node package manager) search for the ngx-spinner library.

Step two: Install the library using the terminal.

This is the installation command.

→ `npm i ngx-spinner`





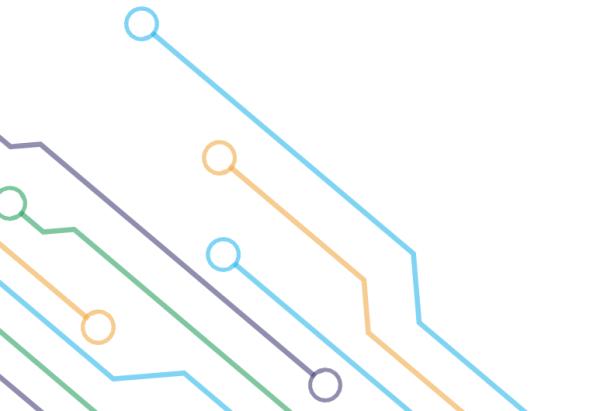
How to add spinner

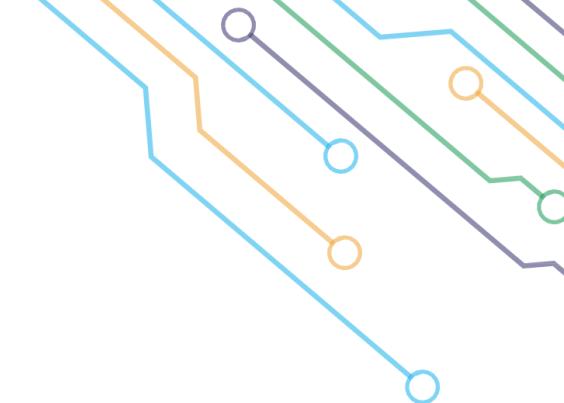
Step three: import the NgxSpinnerModule in the root module (App Module).

```
import { NgxSpinnerModule } from "ngx-spinner";
```

Step four: Adding the template you want in the App Component.

Step five: Add CSS animation files to angular.json config.



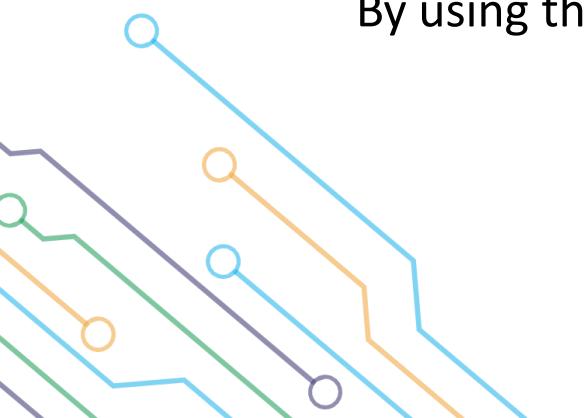


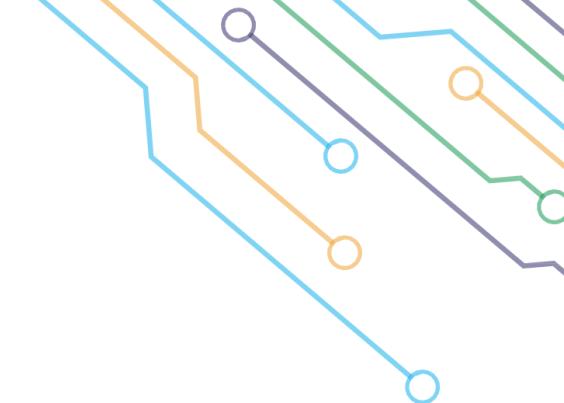
How to add spinner

Step six: Add NgxSpinnerService service wherever you want to use the ngx-spinner and create an object of this service in the constructor.

```
import { NgxSpinnerService } from "ngx-spinner";  
  
constructor(private spinner: NgxSpinnerService){}
```

By using this object, you can show and hide the spinner using the show and hide.





Example

Add the spinner in the register component when the user clicks on submit button.

In the app.module.ts:

```
import { NgxSpinnerModule } from "ngx-spinner";
```

And add the NgxSpinnerModule in the import array.



Example

2. Add the template in the app component:

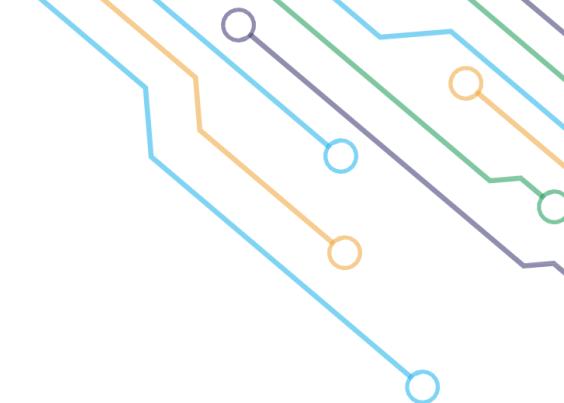
```
<ngx-spinner bdColor = "rgba(0, 0, 0, 0.8)" size = "medium"  
color = "#fff"  
type = "ball-fussion"  
[fullScreen] = "true">  
<p style="color: white" > Loading... </p>  
</ngx-spinner>
```

Example

3. Add CSS animation files to angular.json config.

```
"node_modules/ngx-spinner/animations/ball-scale-multiple.css"

"styles": [
  ./node_modules/@angular/material/prebuilt-themes/
  indigo-pink.css,
  "src/styles.css",
  "node_modules/ngx-spinner/animations/
  ball-scale-multiple.css"
],
```



Example

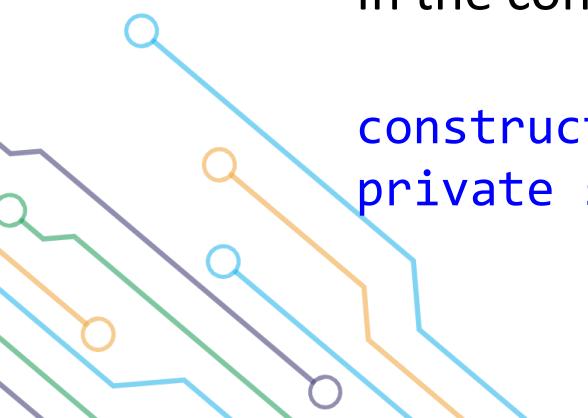
4. Add NgxSpinnerService service in the register component

In register.component.ts

```
import { NgxSpinnerService } from "ngx-spinner";
```

In the constructor :

```
constructor(private router:Router,  
private spinner: NgxSpinnerService)
```



Example

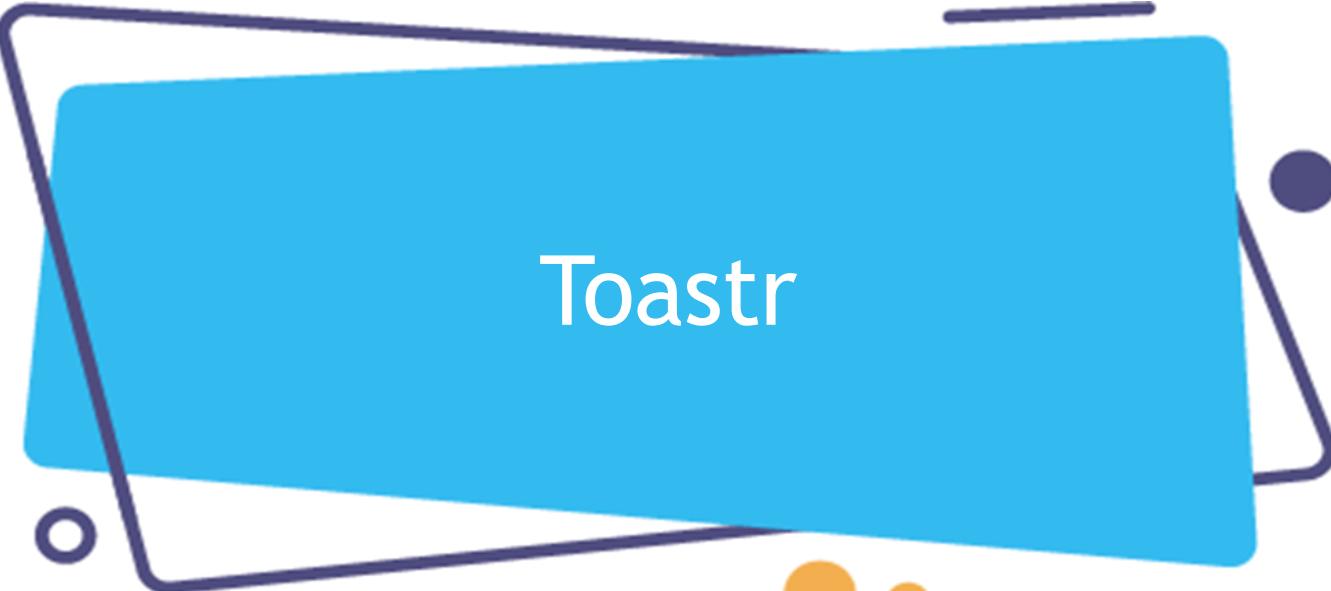
5. Use the show and hide functions in the submit method.

```
submit(){
    this.spinner.show();
    setTimeout(() => {
        /** spinner ends after 5 seconds */
        this.spinner.hide();
    }, 5000);
}
```

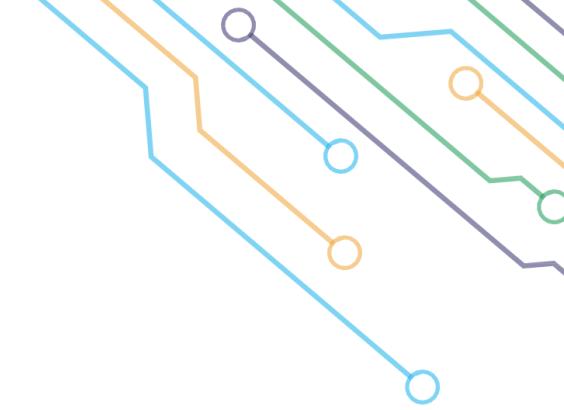
Example

5. Use the show and hide functions in the submit method.

```
submit(){
    this.spinner.show();
    setTimeout(() => {
        /** spinner ends after 5 seconds */
        this.spinner.hide();
    }, 5000);
}
```



Toastr



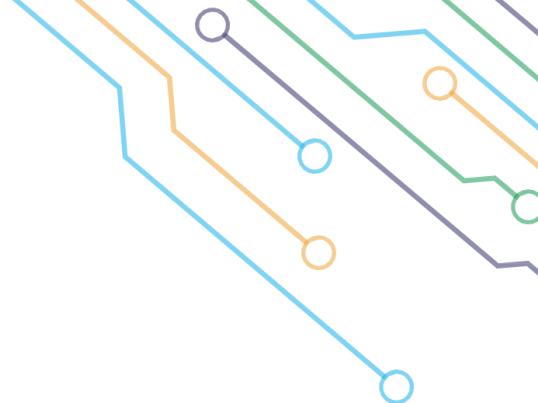
Overview of Toastr

Toastr is a JavaScript library that creates pop-up notifications.

It is simple, easy to use, and can be extended.

It allows you to create simple toasts using HTML5 and JavaScript.





Step to download the toaster package

Step One: Install the toaster package using this command:

```
npm install ngx-toastr --save
```

```
PS C:\Users\d.kanaan.ext\Desktop\TheLearningHub> npm i ngx-spinner
```

```
added 1 package, and audited 1015 packages in 11s
```

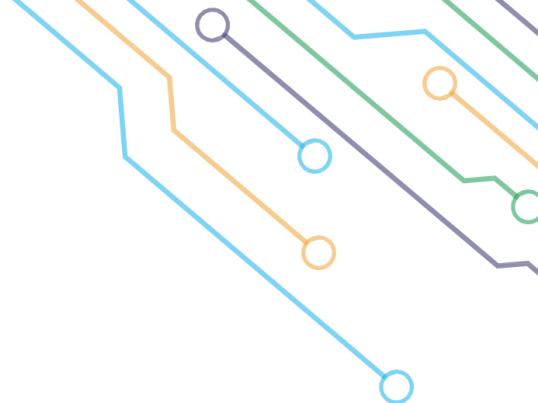
```
107 packages are looking for funding
  run `npm fund` for details
```

```
2 high severity vulnerabilities
```

```
To address all issues, run:
  npm audit fix
```

```
Run `npm audit` for details.
```





Step to download the toastr package

Step Two: Add the style of the toastr in the angular.json file.

```
"styles": [  
    "./node_modules/@angular/material/prebuilt-themes/indigo-pink.css",  
    "src/styles.css",  
    "node_modules/ngx-spinner/animations/ball-fussion.css",  
    "node_modules/ngx-toastr/toastr.css"  
,
```



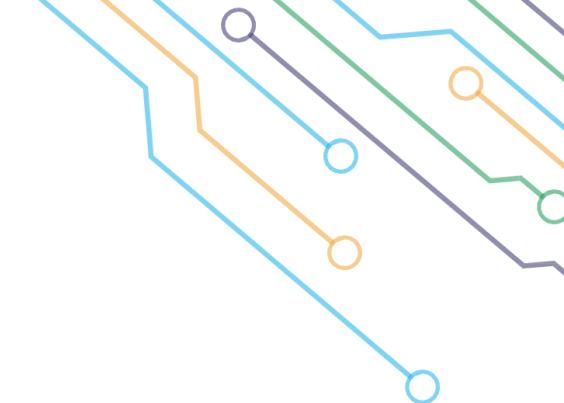
Step to download the toastr package

Step Three: Import the ToasterModule and ToastrNoAnimationModule in the root module(app.module.ts).

```
import {ToastrModule, ToastNoAnimation,  
ToastNoAnimationModule}  
from 'ngx-toastr'
```

Note: Don't forget to add the module's name in the import array.

```
imports: [  
  BrowserModule,  
  AppRoutingModule,  
  BrowserAnimationsModule,  
  NgxSpinnerModule,  
  SharedModule,  
  ToastNoAnimationModule.forRoot(),  
  ToastrModule.forRoot()  
],
```



Example

To add the toastr in the home component

1. Import the ToastrService in the typescript of the component.

So, In home.component.ts.

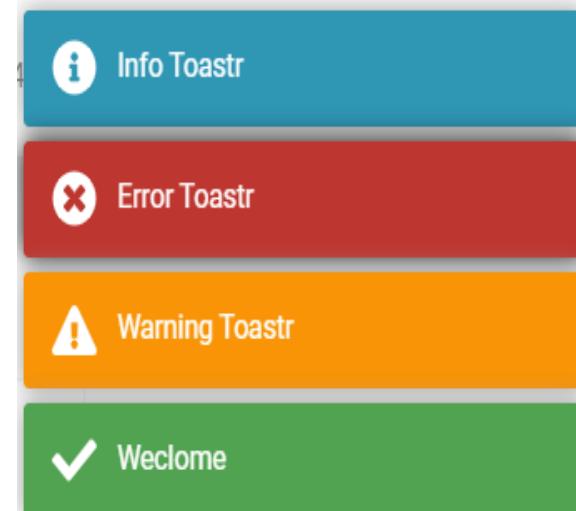
```
import {ToastrService} from 'ngx-toastr';
```



Example

2. Create an instance of toastr service in the constructor of the component and then use this object to access the toastr method like success, warning, error, and info.

```
export class HomeComponent implements OnInit {  
  
  constructor(private toastr: ToastrService) {}  
  
  ngOnInit(): void {  
    this.toastr.success('Weclome');  
    this.toastr.warning('Warning Toastr');  
    this.toastr.error('Error Toastr');  
    this.toastr.info('Info Toastr');  
  }  
}
```



Exercise

Update the API functionality for whole home services as follows:

1. Show the spinner.
2. Connect to the API.
3. Display a toastr as per the response statutes.
4. Hide the spinner.



Break

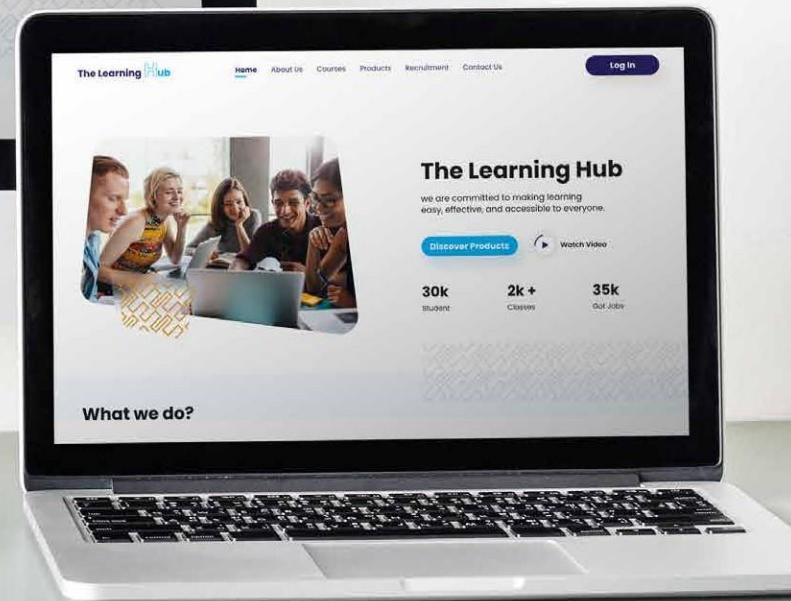
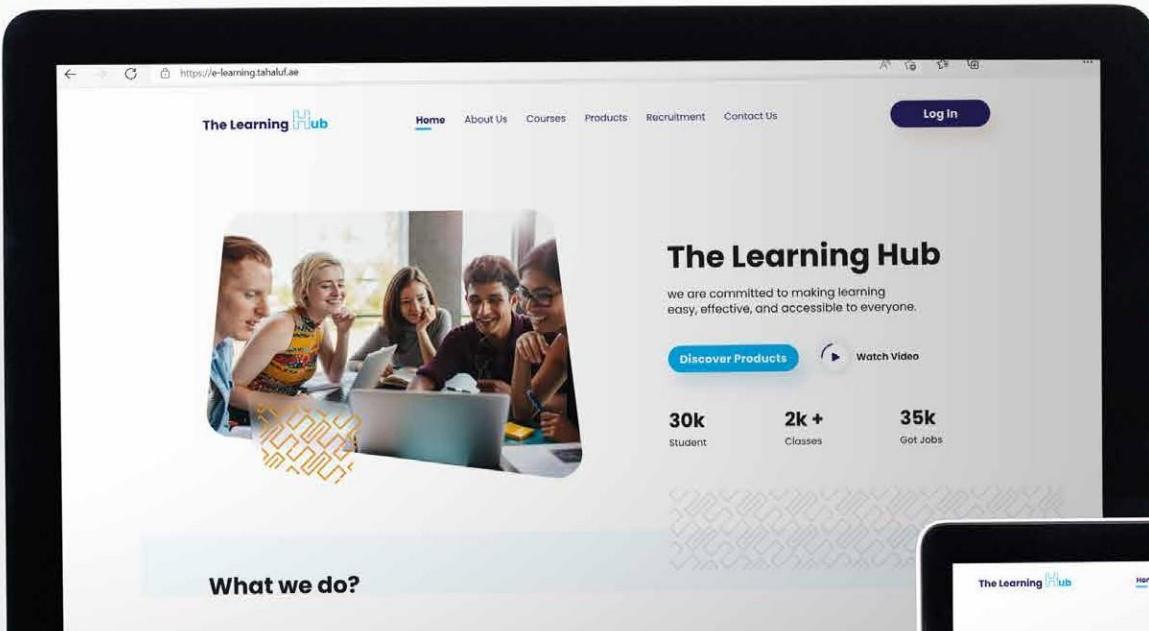
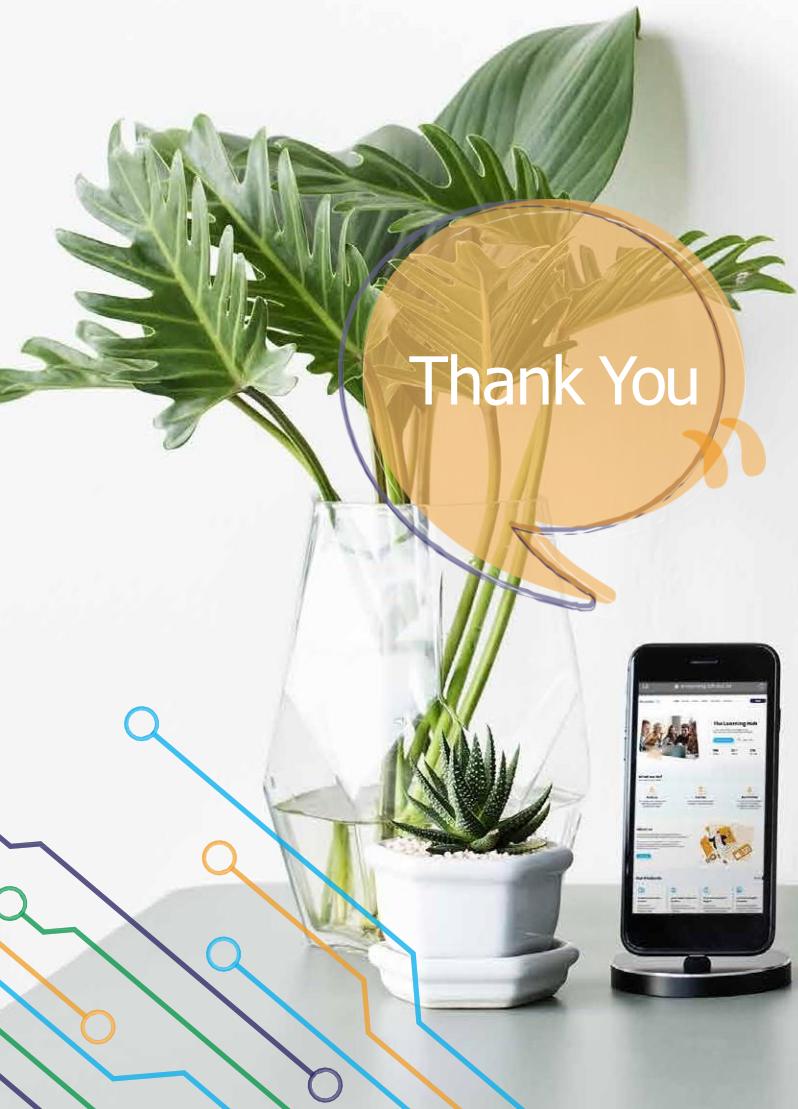
9:00 - 10:00



References :

1. Angular, “Angular,” *Angular.io*, 2019. <https://angular.io/>
2. “Complete Angular Tutorial For Beginners,” *TekTutorialsHub*.
<https://www.tektutorialshub.com/angular-tutorial/>
3. “npm | build amazing things,” *Npmjs.com*, 2019. <https://www.npmjs.com/>
4. “Angular Tutorial for Beginners | Simplilearn,” *Simplilearn.com*.
<https://www.simplilearn.com/tutorials/angular-tutorial> (accessed Aug. 19, 2022).





Angular TS

Education and Training Solutions 2023



1 Reusable Component.

2 Input Decorator.

3 Output Decorator.





Reusable Component.



Reusable Component

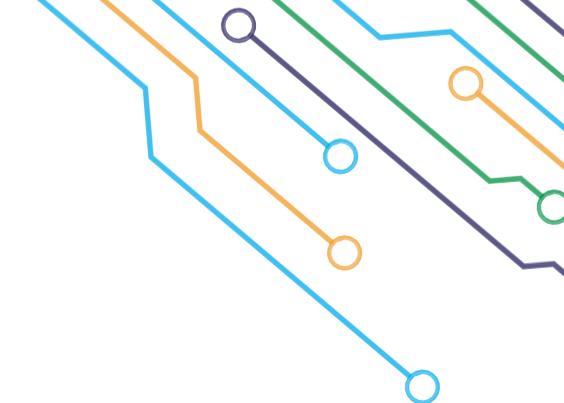
It defines a base structure and behaviour that can be used in different contexts/dynamic content.

With every reusable component, we have a parent component.

In the reusable component, this flexible content comes from parent content and ends up in a dedicated slot.

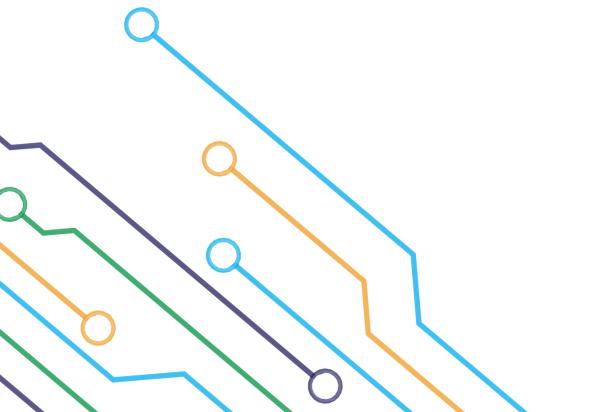
In other words, it is projected down to the parent component.





The Purpose of Reusable Component

- Efficiency: Using the same markup across components, making future changes easy.
- Consistency: Whenever reusable components are updated; they get affected across all their uses.
- Easy to Test: Testing becomes easier when SRPs are followed.





Reusable Component Demo

Create a new component called CourseCard.

Note: This component will contain reusable code.

- PS C:\Users\d.kanaan.ext\Desktop\TheLearningHub> `ng g c courseCard`
`CREATE src/app/course-card/course-card.component.html (26 bytes)`
`CREATE src/app/course-card/course-card.component.spec.ts (588 bytes)`
`CREATE src/app/course-card/course-card.component.ts (221 bytes)`
`CREATE src/app/course-card/course-card.component.css (0 bytes)`
`UPDATE src/app/app.module.ts (1465 bytes)`



Reusable Component Demo

Create another component called courses.

- PS C:\Users\d.kanaan.ext\Desktop\TheLearningHub> `ng g c courses`
`CREATE src/app/courses/courses.component.html (22 bytes)`
`CREATE src/app/courses/courses.component.spec.ts (566 bytes)`
`CREATE src/app/courses/courses.component.ts (206 bytes)`
`CREATE src/app/courses/courses.component.css (0 bytes)`
`UPDATE src/app/app.module.ts (1555 bytes)`

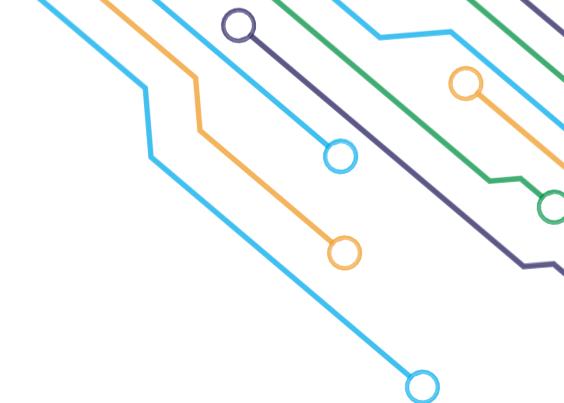


Reusable Component Demo

Add the routing for the course component.

In-app-routing.module.ts

```
{  
  path: 'course',  
  component:CoursesComponent  
}
```



Reusable Component Demo

The course card component template will be the cards for each course in the system.

So instead of repeating the card more than once according to the number of courses in the system, we resort to using the principle of reusable components.



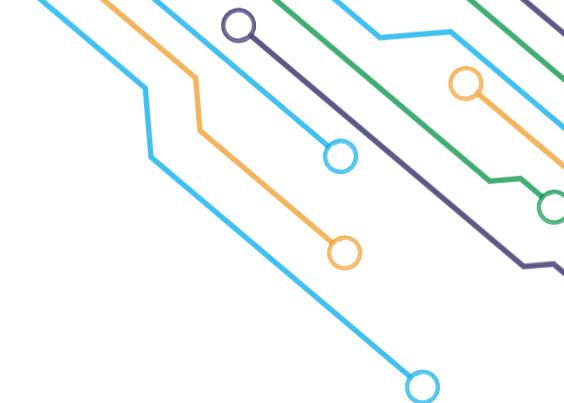
Reusable Component Demo

Add the template of the cards in the courseCard.compoment.html:

Reusable Component Demo

```
<div class="blog-main-card d-flex">
  <article class="blog-sub">
    <div class="blog-content">
      
    </div>
    <div class="blog-content-section">
      <div class="blo-content-title">
        <h4>Course Name</h4>
        <span>The Course Number is</span>
        <p>Lorem, ipsum dolor sit amet consectetur adipisicing elit. Neque at numquam, asperiores aut
           facilis ratione! Voluptatibus neque dignissimos ipsa atque veniam sint omnis in blanditiis
      </div>
      <div class="blog-admin">
        <ol>
          <li><i class="fal fa-user-tie"></i> By Admin</li>
          <li><i class="fal fa-calendar-alt"></i> july 28, 2020</li>
        </ol>
      </div>
    </div>
  </article>
```

Input Decorator.



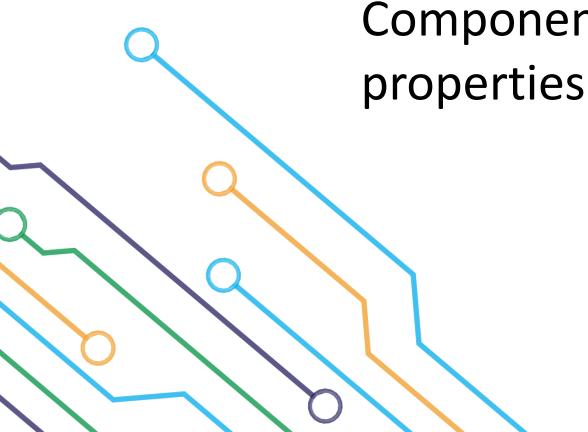
Overview of Input Decorator

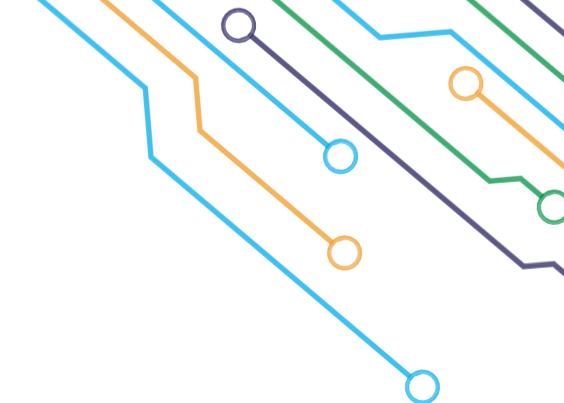
@Input is a decorator for marking a property as input.

It can be used to specify a component property.

Allows data to be passed between parent and child components (property binding).

Component properties should be annotated with the @Input decorator to act as input properties.



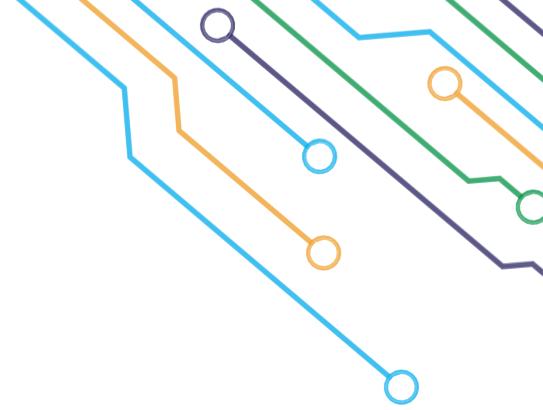


Input Decorator Example

Step One:

Start with a call the get all courses from home service inside the course component.

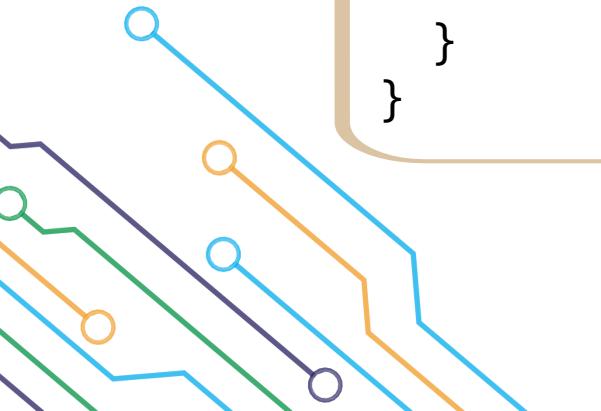


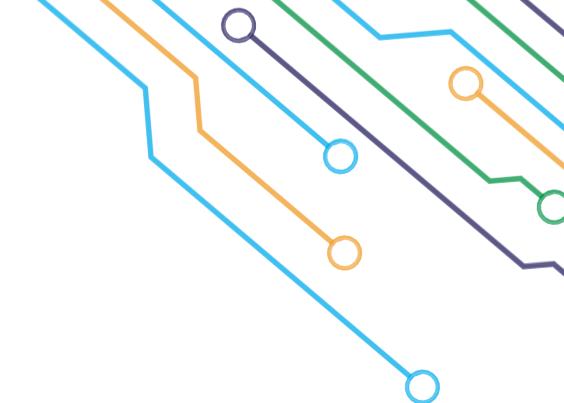


Input Decorator Example

In courses component.ts :

```
export class CoursesComponent implements OnInit
{
  constructor(public home:HomeService){}
  ngOnInit(): void {
    this.home.getAllCourses();
  }
}
```

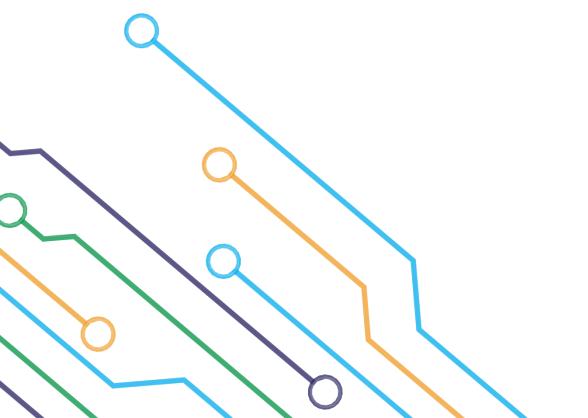




Input Decorator Example

Step Two:

You will need to open the parent component views (courses.component.html), then pass this variable to the child component.



Input Decorator Example

```
<div class="row">
<app-course-card class="col-4" *ngFor="let course of
home.courses"
[courseName] = "course.course_Name"
[courseId] = "course.course_Id"
[price] = "course.price"
[startdate] = "course.startdate" [courseimage] = "course.imagename"
[enddate] = "course.enddate"></app-course-card>
</div>
```



Input Decorator Example

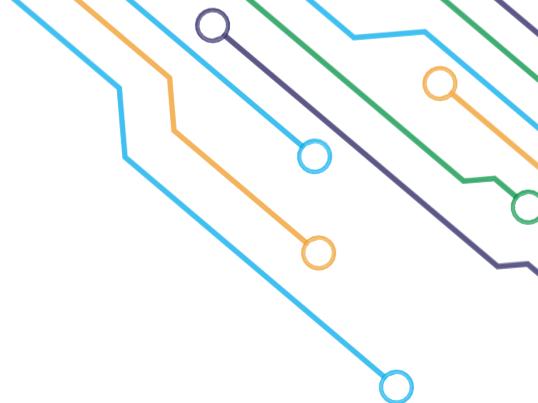
Step Three:

Import the input on the child component (course card component).

So, in the courseCard.component.ts:

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





Input Decorator Example

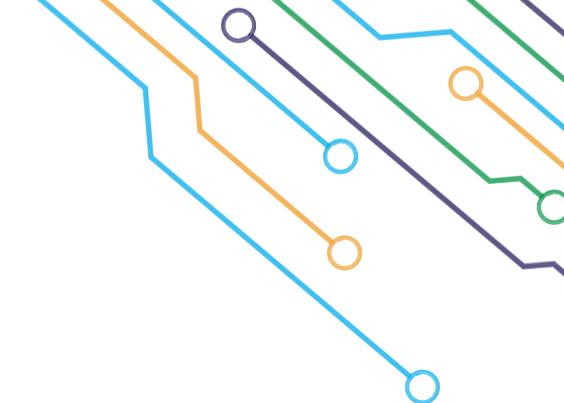
Step Four:

Create variables that are passed from the parent component to the child component.

In the coursCard.component.ts:

```
export class CourseCardComponent {  
  @Input() courseName :string|undefined;  
  @Input() courseId :number|undefined;  
  @Input() price :number|undefined;  
  @Input() startdate :string|undefined;  
  @Input() enddate :string|undefined;  
  @Input() courseimage:string|undefined;  
}
```





Input Decorator Example Result

The Learning Hub LMS

OUR COURSES —

The Agile - Scrum Framework

The Course Number is 106
The Course cost is 12

By Admin
Dec 12, 1111

ORACLE CLOUD

Oracle

The Course Number is 121
The Course cost is 90

By Admin
Dec 12, 1998

ANGULAR

Angular

The Course Number is 22
The Course cost is 500

By Admin
Dec 12, 1111

C#

C#

The Course Number is 107
The Course cost is 200

By Admin
Dec 12, 1998

ORACLE CLOUD

Oracle

The Course Number is 108
The Course cost is 200

By Admin
Dec 12, 1998

API

Application Programming Interface

The Course Number is 46
The Course cost is 450

By Admin
Dec 12, 1998

Decorative graphic element consisting of several colored lines (blue, orange, green) with circular endpoints, forming a stylized path or flow.

Output Decorator.

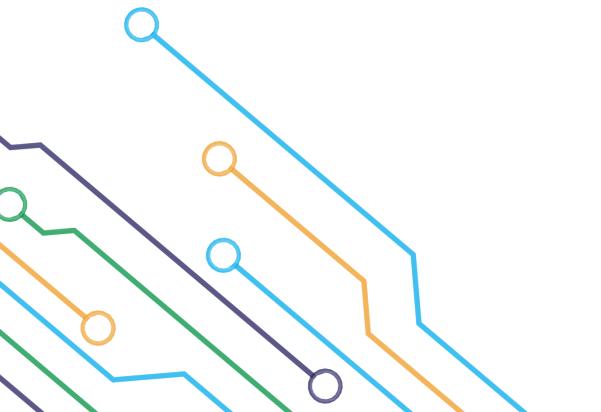


Output Decorator

In the `@Output` decorator, data is passed from the child to the parent component.

This annotation binds a property of type Angular Event Emitter.

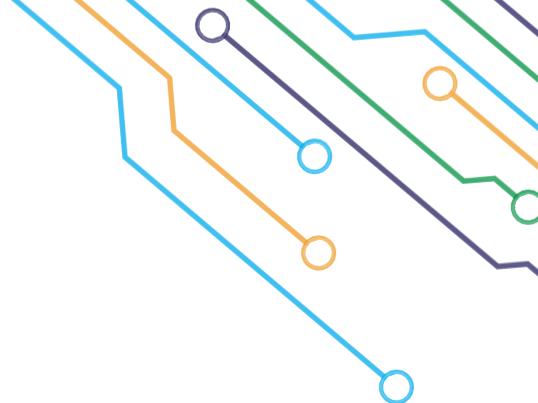
Use it in components with the `@Output` directive to emit custom events synchronously or asynchronously.



Exercise

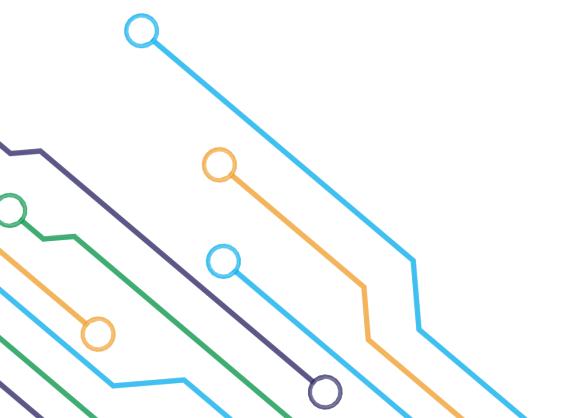
Generate a new component called profile and give it a route.





Exercise Solution

- PS C:\Users\d.kanaan.ext\Desktop\TheLearningHub> `ng g c profile`
`CREATE src/app/profile/profile.component.html (22 bytes)`
`CREATE src/app/profile/profile.component.spec.ts (566 bytes)`
`CREATE src/app/profile/profile.component.ts (206 bytes)`
`CREATE src/app/profile/profile.component.css (0 bytes)`
`UPDATE src/app/app.module.ts (1645 bytes)`



Exercise Solution

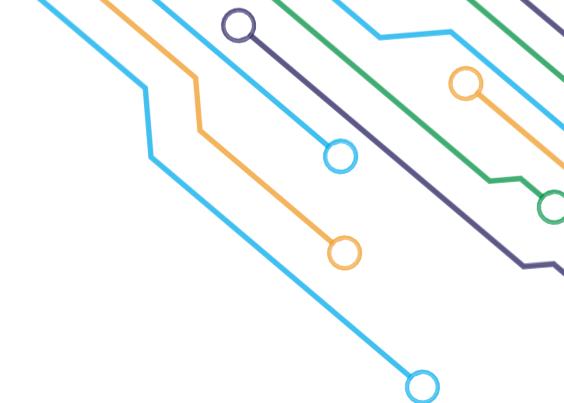
In the app.module.routing.ts

```
{  
  path: 'profile',  
  component: ProfileComponent  
}
```

Exercise

Create an output event that redirects to the profile component when the user clicks on the course image.





Exercise Solution 2

Using the output decorator to pass the data from the child to the parent component.

The show profile function will emit the open profile that is passed to the parent component (course component).



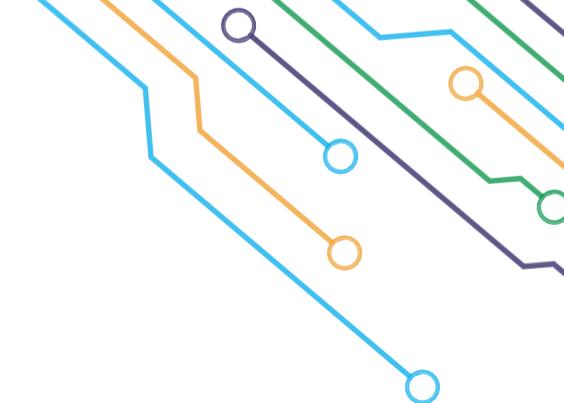
Exercise Solution 2

Step One:

Import the output decorator and the Event Emitter in the child component (course card component).

In courseCard.component.ts:

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



Exercise Solution 2

Step Two:

Create an instance of the eventEmmitter class In the child component to emit the output event in the parent component.

```
@Output() opneProfile=new EventEmitter();
```



Exercise Solution 2

Step Three:

In the child component views (courseCard.component.html)

```
<div class="blog-content">

</div>
```

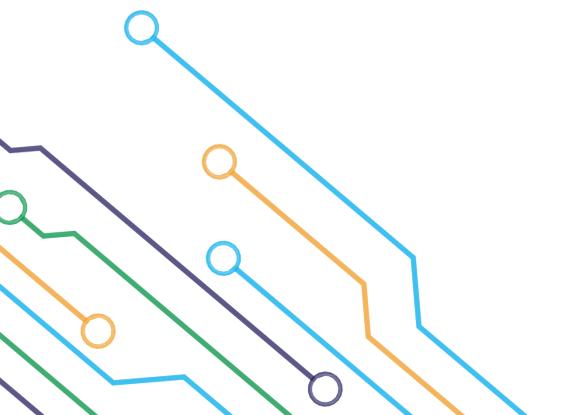


Exercise Solution 2

Step Four:

In the courseCard.component.ts

```
showProfile(){
    this.openProfile.emit();
}
```

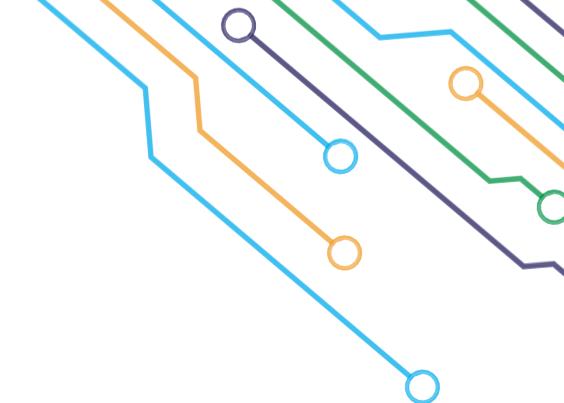


Exercise Solution 2

Step Five:

In the courses.component.html add an output event called show profile.

```
| <div class="row">  
|   <app-course-card class="col-4" *ngFor="let course of home.courses"  
|     [courseName] = "course.course_Name"  
|     [courseId] = "course.course_Id"  
|     [price] = "course.price"  
|     [startdate] = "course.startdate"  
|     [courseimage] = "course.imagename"  
|     [enddate] = "course.enddate"  
|     (openProfile) = "goToProfile()"></app-course-card>  
</div>
```



Exercise Solution 2

Step Six:

In the courses.component.ts add the implementation of the go-to profile function which navigates to the profile component.

```
constructor(public home:HomeService,private
router:Router){}
goToProfile(){
  this.router.navigate(['profile'])
}
```



Break

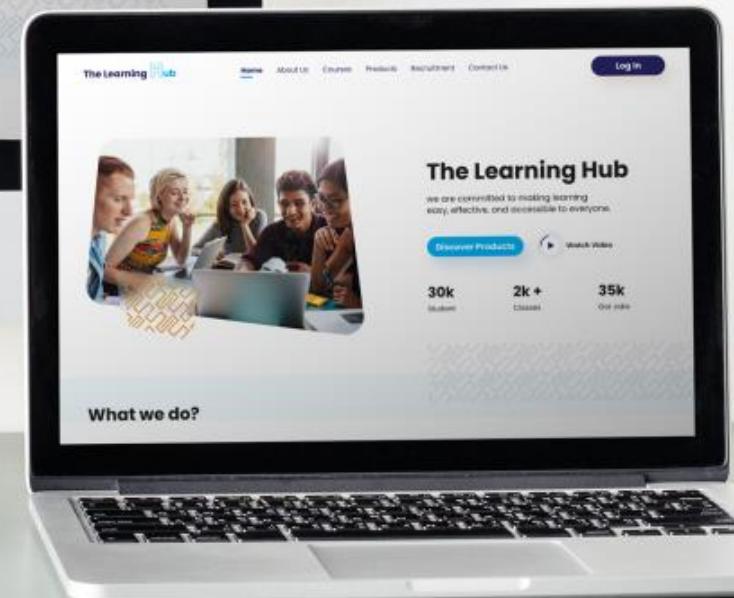
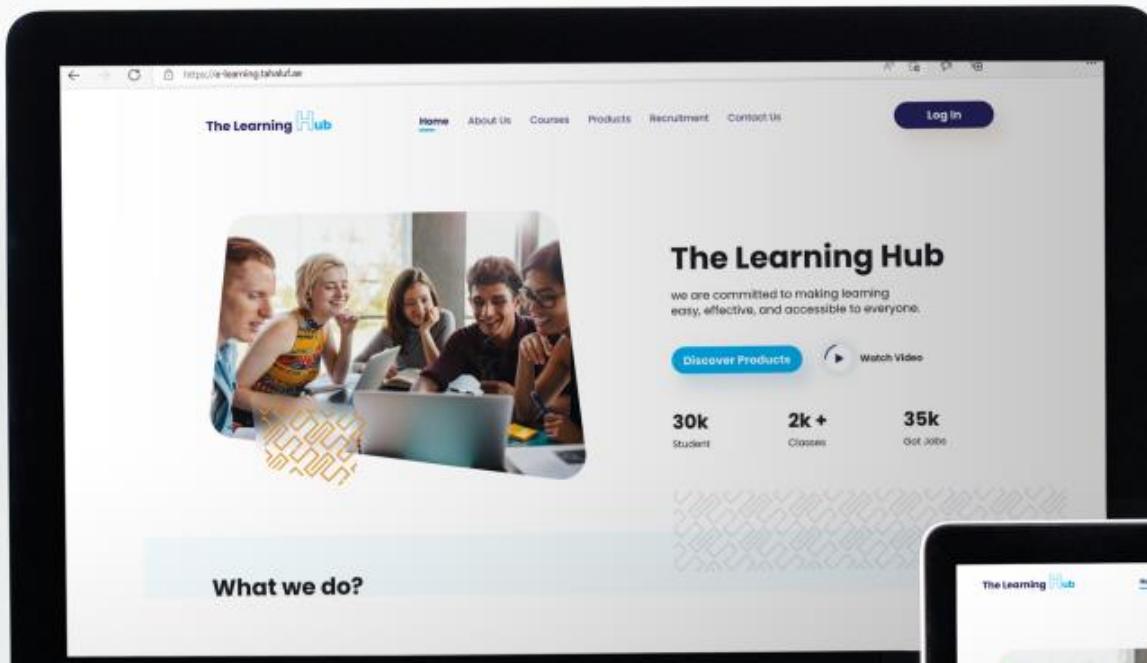
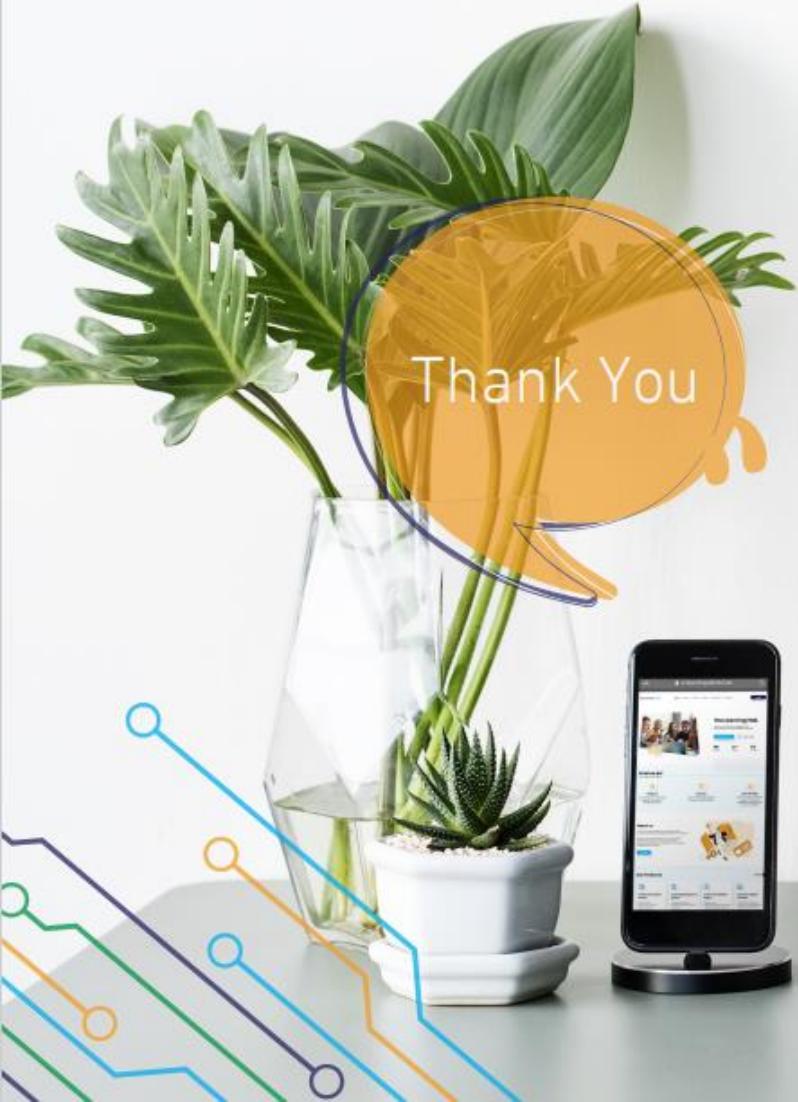
9:00 - 10:00



References :

1. Angular, “Angular,” *Angular.io*, 2019. <https://angular.io/>
2. “Complete Angular Tutorial For Beginners,” *TekTutorialshub*.
<https://www.tektutorialshub.com/angular-tutorial/>
3. “npm | build amazing things,” *Npmjs.com*, 2019. <https://www.npmjs.com/>
4. “Angular Tutorial for Beginners | Simplilearn,” *Simplilearn.com*.
<https://www.simplilearn.com/tutorials/angular-tutorial> (accessed Aug. 19, 2022).





Angular

Tahaluf Training Center 2023



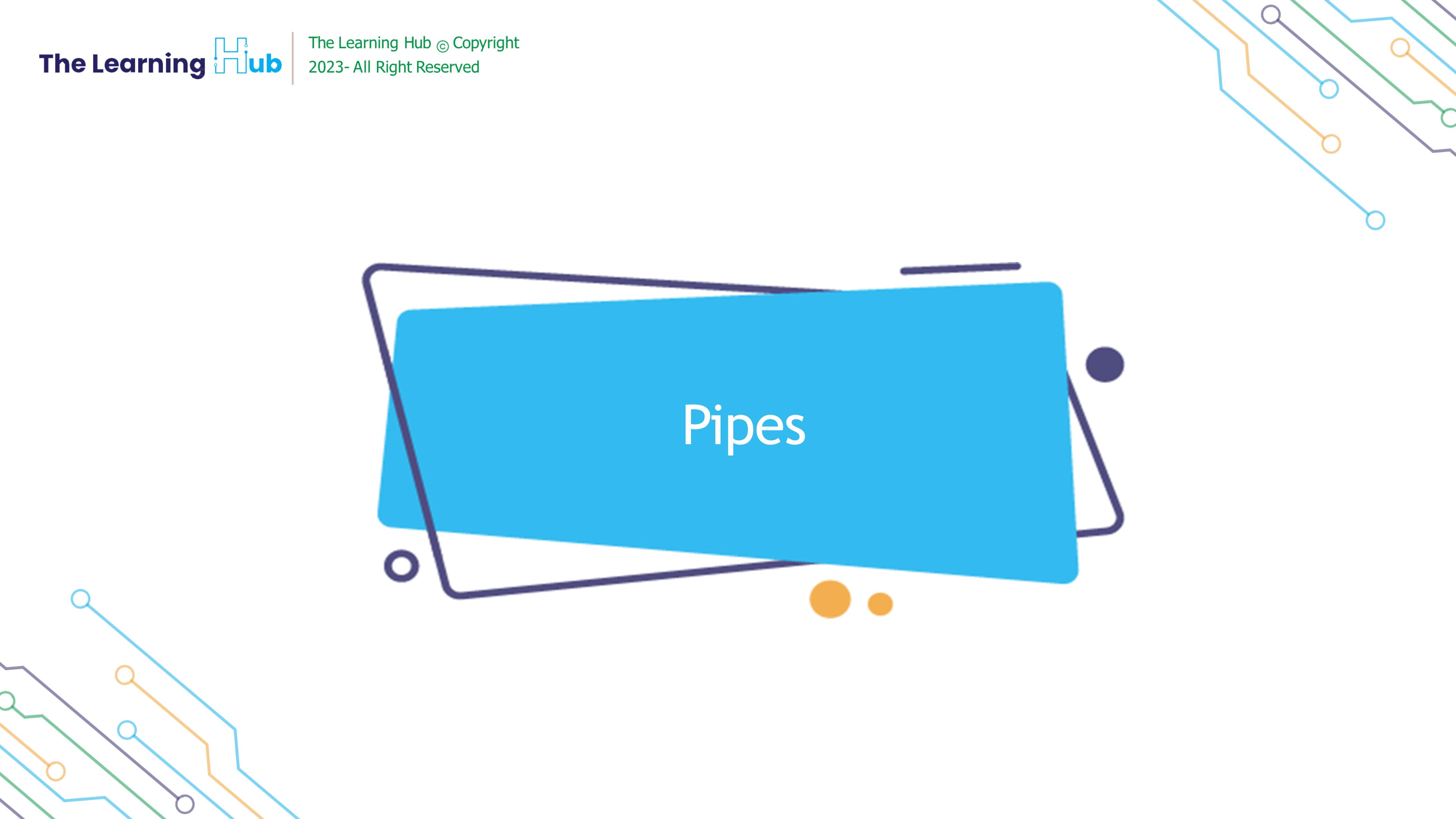
1 Pipes.

2 Custom Pipe (Filter)

3 Hits API (Log in).



Pipes



Overview of Pipe

Use pipes to transform strings, currency amounts, dates, and other data for display. Pipes are simple functions to use in template expressions to accept an input value and return a transformed value. Pipes are useful because you can use them throughout your application, while only declaring each pipe once.

Overview of Pipe

Angular has built-in pipes:

- Date: returns a formatted date.
- Uppercase: return upper case formatted.
- Lowercase: returns lowercase formatted.
- Percent: converts a value to a percentage.

Pipe Example

In course-card.component.html:

```
<h4>{{courseName | uppercase}}</h4>
<h4>{{courseName | lowercase}}</h4>
<li><i class="fal fa-calendar-alt"></i> {{startdate|date}}</li>
<span>The Course cost is{{price | percent}}</span>
```

Custom Pipe (Filter)

How to generate a custom pipe

Using this command, you can generate a custom pipe.

→ `ng generate pipe Pipe_Name`

Example to generate a custom pipe

Generate a new pipe called filter, the purpose of this pipe is to filter the courses based on the course name.

Example to generate a custom pipe

In the terminal:

- PS C:\Users\d.kanaan.ext\Desktop\TheLearningHub> ng g p Pipes/filter
CREATE src/app/Pipes/filter.pipe.spec.ts (187 bytes)
CREATE src/app/Pipes/filter.pipe.ts (217 bytes)
UPDATE src/app/app.module.ts (1715 bytes)

Example to generate a custom pipe

Add the name of the pipe in the shared module in the declaration and export array.

```
declarations: [  
  NavbarComponent,  
  FooterComponent,  
  FilterPipe  
,
```

Example to generate a custom pipe

Add the input field in the manage course component to enter the name of the course.

```
<!-- In the manageCourse.html -->
<input type="text"
[(ngModel)]="_filterText" class="form-control"
id="inlineFormInput" placeholder="Course Name">

<!-- In the manageCourse.ts -->
_filterText:string='';
```

Example to generate a custom pipe

Edit the transform function in the FilterPipe class.
And apply the FilterPipe in the manageCourse component .

```
import { Pipe, PipeTransform } from '@angular/core';
@Pipe({
  name: 'filter'
})
export class FilterPipe implements PipeTransform {
  constructor(){}
  transform(value: any[],filterText:string) {
    if(filterText=='')
    {
      return value;
    }
    else return value.filter(course:any)=>{
      return course.coursename.toLowerCase() ==filterText.toLowerCase();
    }
  }
}
```

Example to generate a custom pipe

```
import { Pipe, PipeTransform } from '@angular/core';
@Pipe({
  name: 'filter'
})
export class FilterPipe implements PipeTransform {
  constructor(){}
  transform(value: any[],filterText:string) {
    if(filterText=='')
    {
      return value;
    }
    else return value.filter(course:any)=>{
      return course.coursename.toLowerCase() ==filterText.toLowerCase();
    }
  }
}
```

API Calls(Login)

API Calls (Log in)

Step One:

Create an instance of the FormControl class in the login.component.ts :

```
email: FormControl = new FormControl('Ex@example.com',  
[Validators.required, Validators.email]);  
password: FormControl = new FormControl('*****',  
[Validators.required, Validators.minLength(8)]);
```

API Calls (Log in)

Step One:

Don't forget the import statement.

```
import { FormControl, Validators } from  
  '@angular/forms';
```

API Calls (Log in)

Step Two:

Add the template for the login page in the login. component.html

<https://statics.teams.cdn.office.net/evergreen-assets/safelinks/1/atp-safelinks.html>

API Calls (Log in)

Step Three:

Make the email and the password read from the form control and add the error message of each validation.

API Calls (Log in)

```
<div class="row px-3" >
<label class="mb-1"><h6 class="mb-0 text-sm">Email
Address</h6></label>
<input class="mb-4" type="text" [formControl]="email"
placeholder="Enter a valid email address">
    <span class="text-danger"
*ngIf="email.hasError('email') &&
!email.hasError('required')>Please enter a valid email
address</span>
    <span class="text-
danger"*ngIf="email.hasError('required')>Email is
required</span>
</div>
```

API Calls (Log in)

```
<div class="row px-3" >
<label class="mb-1"><h6 class="mb-0 text-sm">Email
Address</h6></label>
<input class="mb-4" type="text" [formControl]="email"
placeholder="Enter a valid email address">
    <span class="text-danger"
*ngIf="email.hasError('email') &&
!email.hasError('required')>Please enter a valid email
address</span>
    <span class="text-
danger"*ngIf="email.hasError('required')>Email is
required</span>
</div>
```

API Calls (Log in)

Step Four:

- Add the function in the auth service that calls the login function from the API.
- Save the token in the local storage.
- Install the Jwt-decode package from npm to decode the token.
<https://www.npmjs.com/package/jwt-decode>
- Using the result from decoded token to check the role of the entered user.

API Calls (Log in)

```
export class AuthService {
  constructor(private http:HttpClient,private router:Router,
private toastr:ToastrService) { }
  submit(email:any, password:any)
{
  var body ={
    username : email.value.toString(),
    password: password.value.toString()
}
  const headerDirc= {
    'Content-Type':'application/json',
    'Accept':'application/json'
}
  const requestOptions={
    headers:new HttpHeaders(headerDirc) }
```

API Calls (Log in)

```
debugger;
this.http.post('https://localhost:44382/api/jwt/Login',
body,requestOptions).subscribe((resp:any)=>{
  console.log(resp); //token
  const responce = {
    token : resp.toString() //save on localstorage
  }
  localStorage.setItem('token',responce.token);
```

API Calls (Log in)

```
let data:any = jwt_decode(responce.token);
localStorage.setItem('user',JSON.stringify(data))
if(data.role=="1")
{
    this.toastr.success('Welcome On Admin Dashbaord');
    this.router.navigate(['admin/dashboard']);
}
else if(data.role=="2")
{
    this.toastr.success('Welcome On Courses Page');
    this.router.navigate(['course']);
}
```

API Calls (Log in)

```
},err=>{
    console.log(err);

    this.toastr.error('Something wrong!!');
    this.toastr.error(err.message);

})
```

API Calls (Log in)

Add the button in the login component.

```
<div class="row mb-3 px-3">
<button class="btn btn-blue text-center"
(click)="Login()">Login</button>
</div>
```

API Calls (Log in)

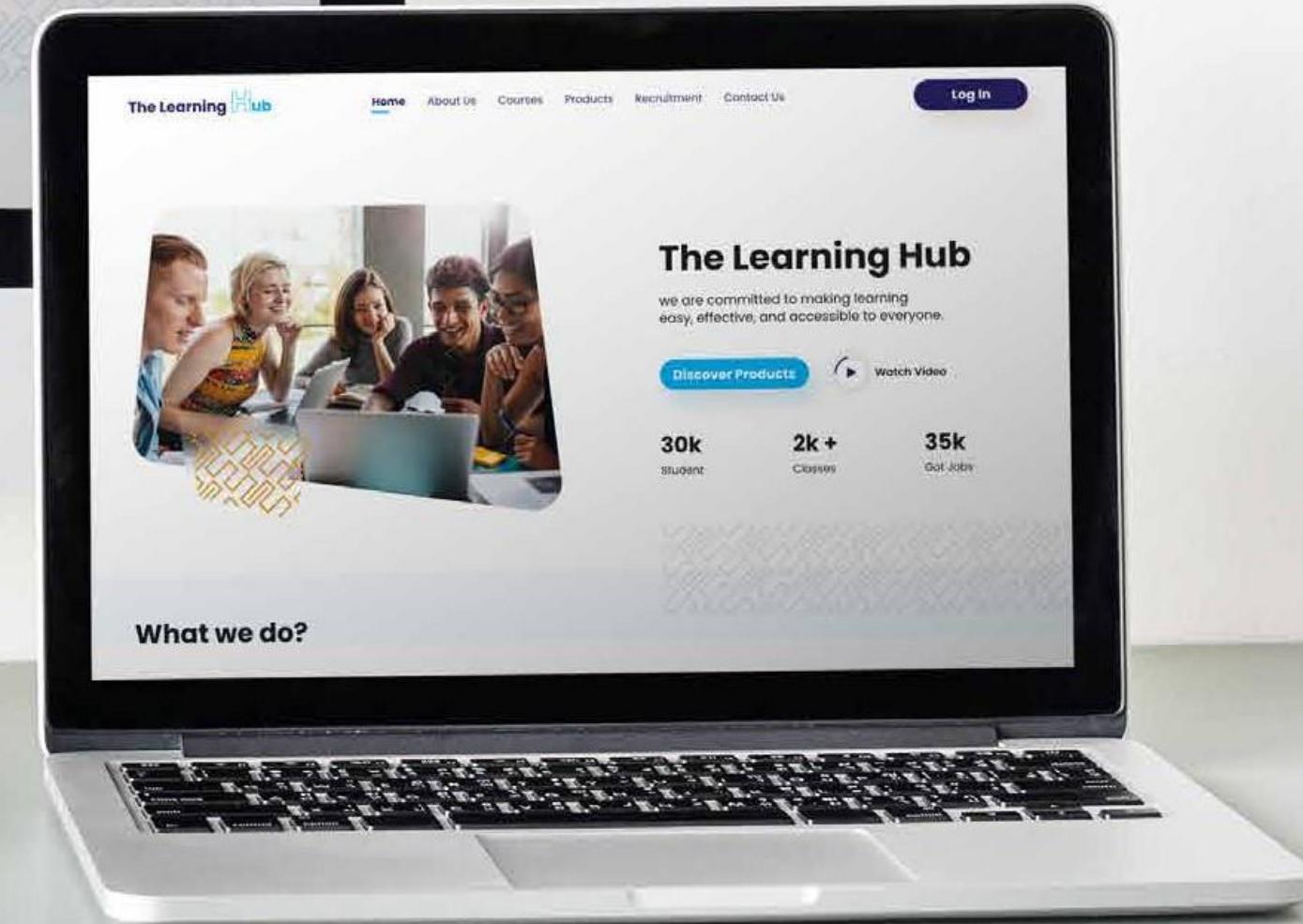
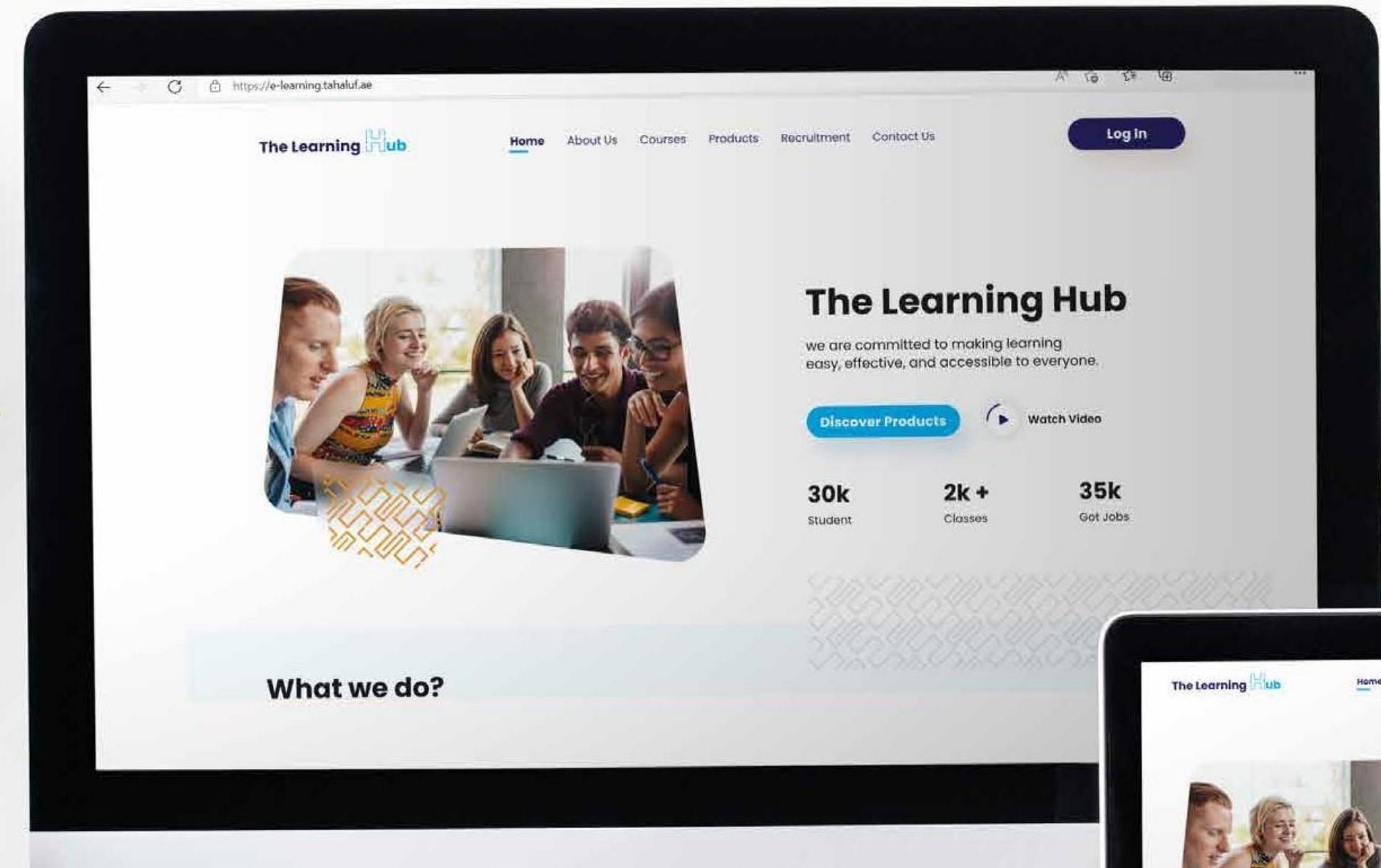
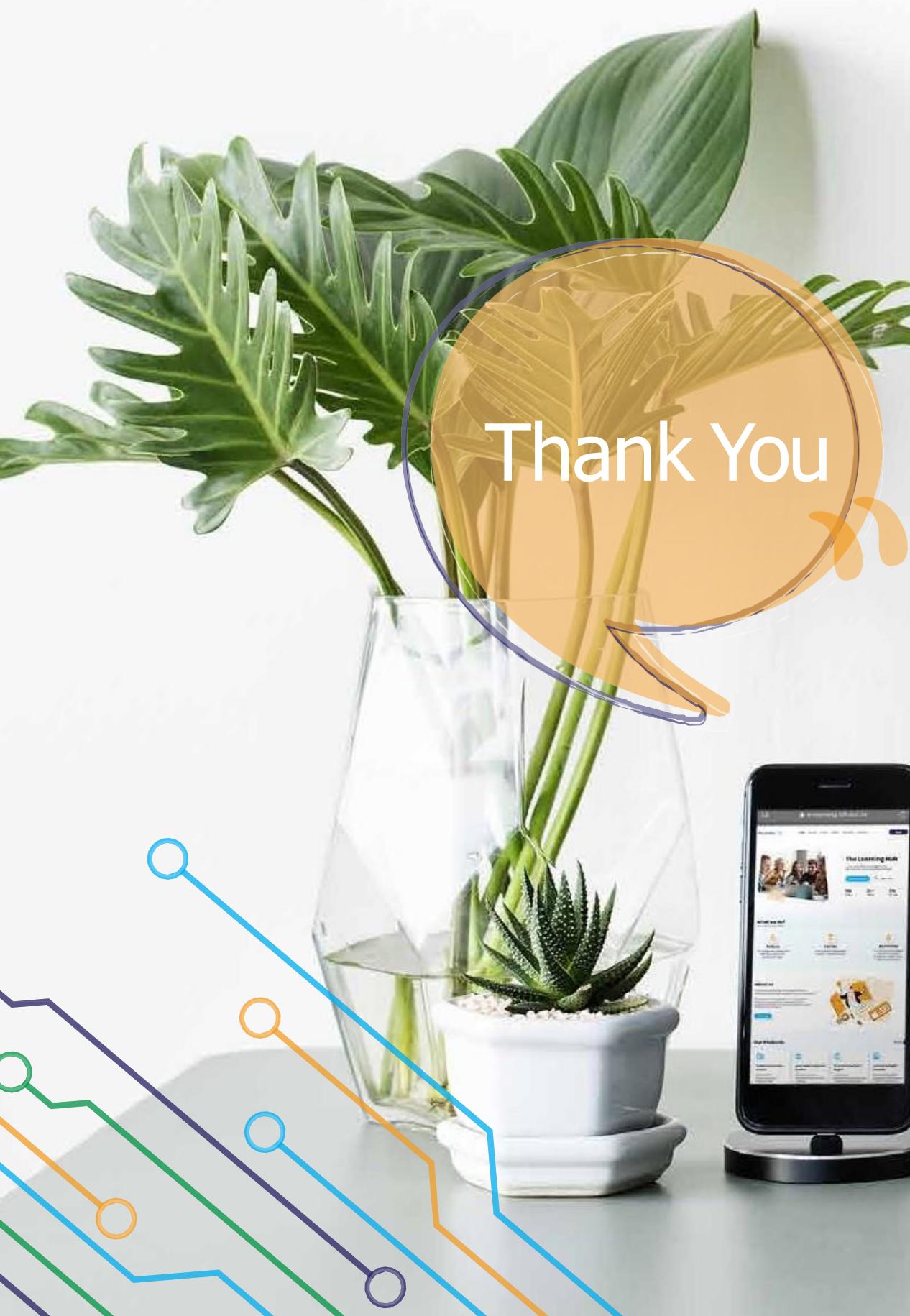
Implement the login function to send the email and password to the submit function in the auth services

```
export class LoginComponent {
  constructor(public router: Router, public auth: AuthService, private spinner: NgxSpinnerService) { }
  login() {
    debugger
    this.spinner.show();
    setTimeout(() => {
      this.spinner.hide();
    }, 3000);
    this.auth.submit(this.email, this.password); }
```

Break

9:00 - 10:00

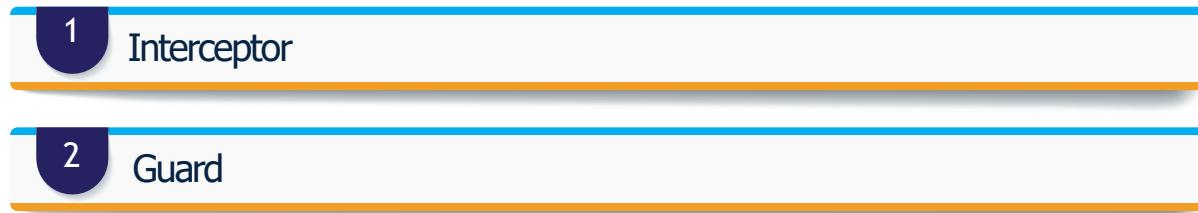




Angular TS

Education and Training Solutions 2023







Overview of Interceptor

HttpInterceptors are used to handle and intercept the HttpRequest and HttpResponseMessage.

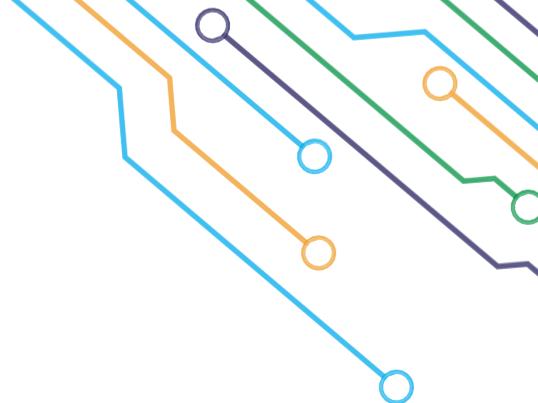


Overview of Interceptor

Interceptors transform outgoing requests before passing them to the next interceptor, with the call `next.handle(transformedReq)`.

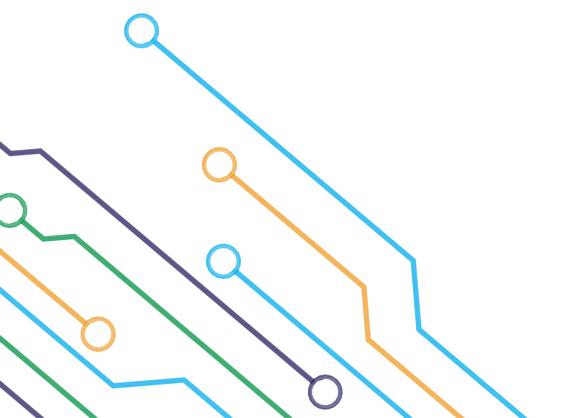
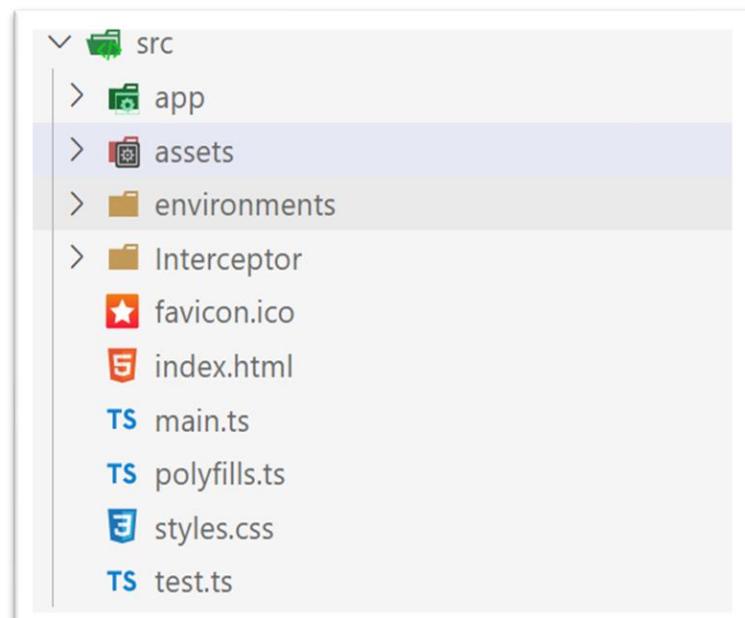
Interceptors can also transform response event streams through the use of RxJS operators on the stream returned by `next.handle ()`.





How to deal with the interceptor

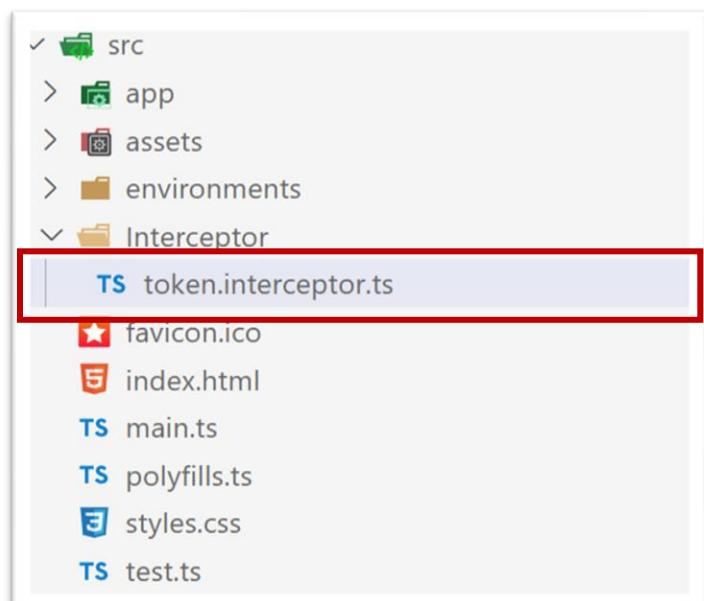
1. Create a new folder called **Interceptor** in the src folder.





How to deal with the interceptor

2. In this folder create a new file called **token.interceptor.ts**.



How to deal with the interceptor

In token.interceptor.ts:

```
import { HttpEvent, HttpHandler, HttpInterceptor, HttpRequest } from "@angular/common/http";
import { Observable } from "rxjs";

export class TokenInterceptor implements HttpInterceptor{
    intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
        const token = localStorage.getItem('token');
        console.log(token);
        req= req.clone({
            setHeaders:{
                Authorization:`Bearer ${token}`
            }
        })
        return next.handle(req);
    }
}
```



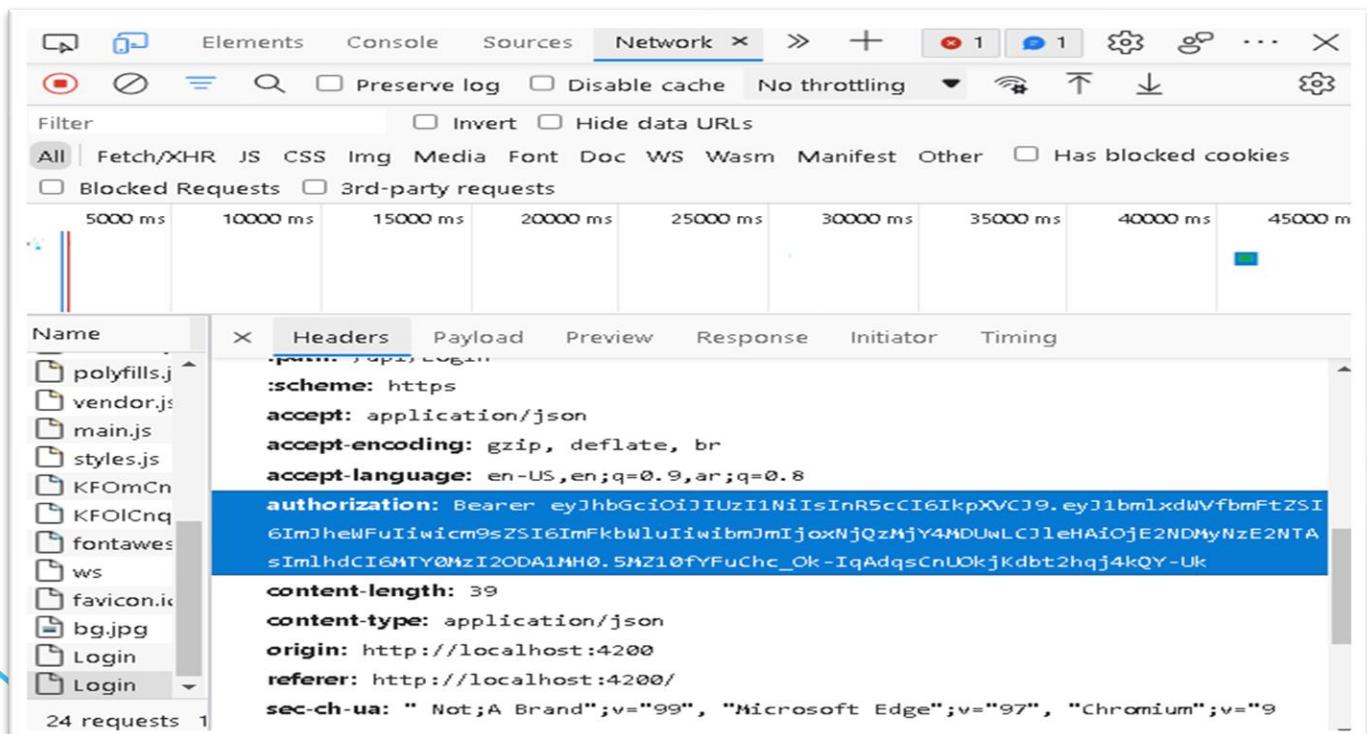
How to deal with the interceptor

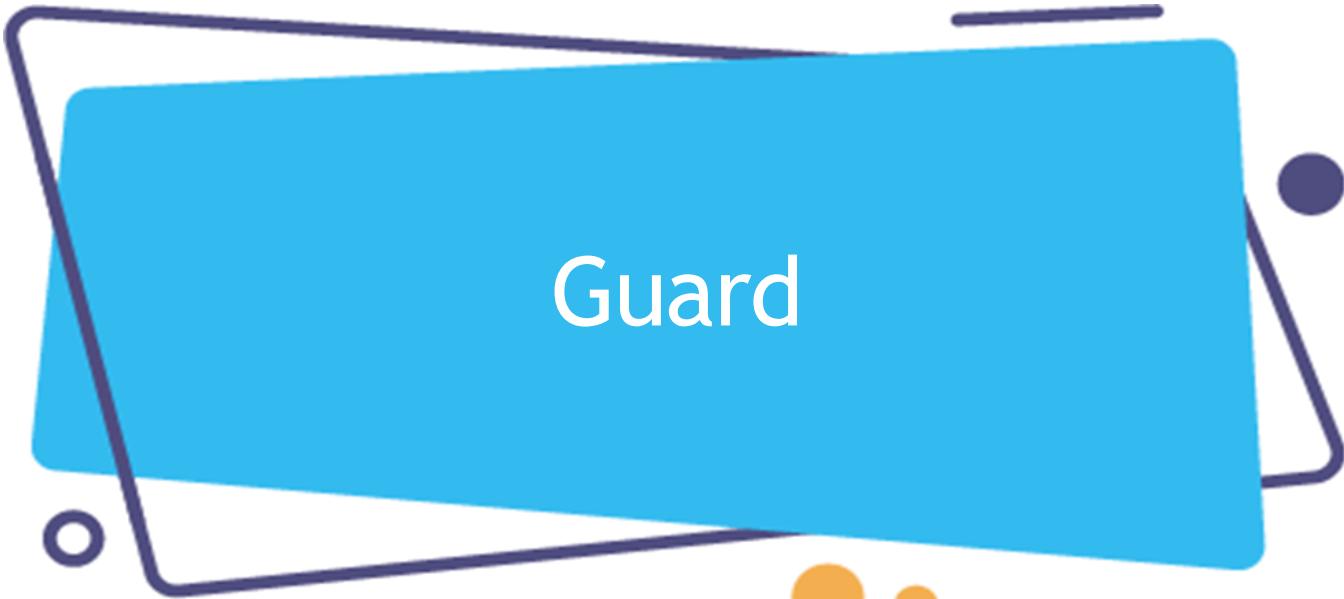
3. Apply the interceptor class In the App Module.
In app.module.ts (In providers array):

```
providers: [ {  
    provide:HTTP_INTERCEPTORS,  
    useClass:TokenInterceptor,  
    multi:true  
}],
```

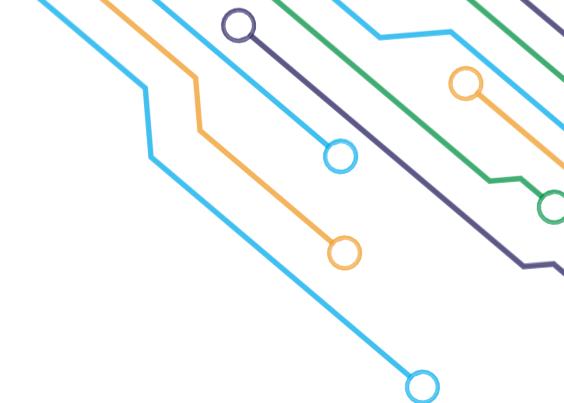


How to deal with the interceptor



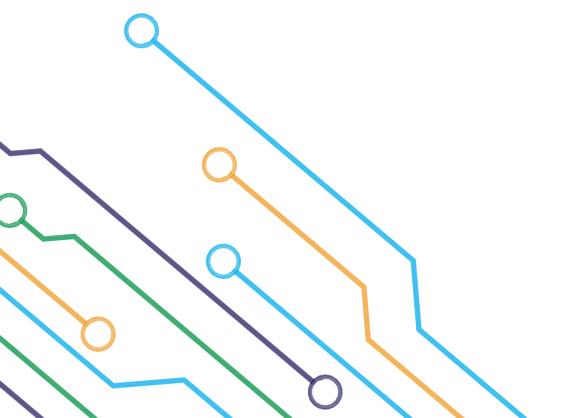


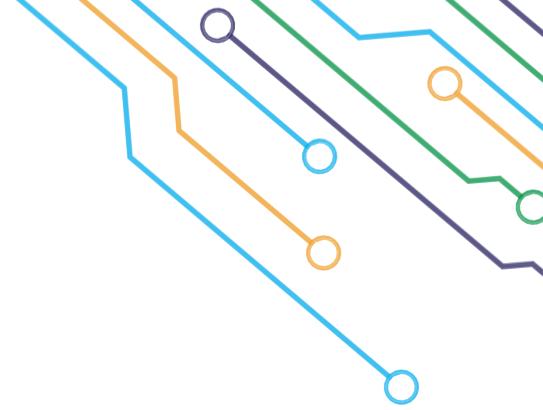
Guard



Overview of Guard

CanActivate: This interface lets classes implement guards that decide whether a route can be activated. Navigation continues if all guards return true. Navigation is canceled if any guard returns false. The current navigation is canceled if any guard returns a UrlTree and new navigation begins based on the UrlTree.



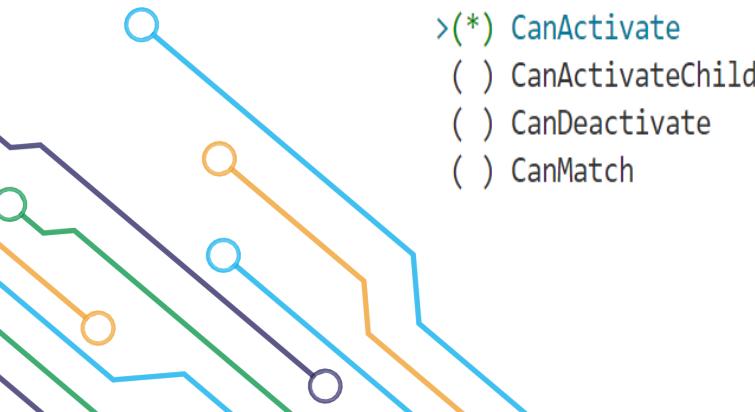


How to include the guard in the project

Generate a new guard via this command :

ng generate guard guardName or ng g g guardName. (Chose canActivate)

```
PS C:\Users\d.kanaan.ext\Desktop\TheLearningHub> ng g g authorization
? Which type of guard would you like to create? (Press <space> to select, <a> to toggle all, <i>
to invert selection, and <enter> to proceed)
>(* ) CanActivate
( ) CanActivateChild
( ) CanDeactivate
( ) CanMatch
```

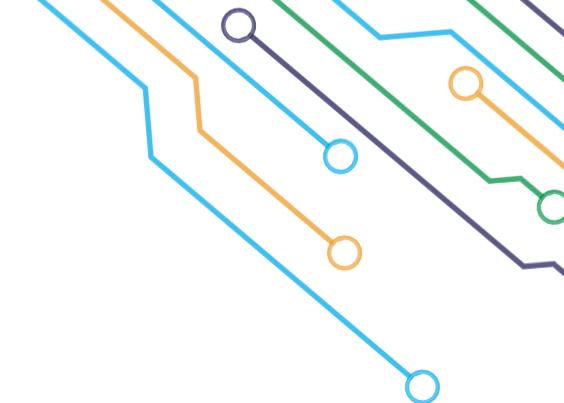


How to include the guard in the project

How to include the guard in the project

How to include the guard in the project

```
//in the App-routing.module.ts
{
  path: 'admin',
  loadChildren: () => AdminModule,
  canActivate:[AutherizationGuard]
},
```



How to include the guard in the project

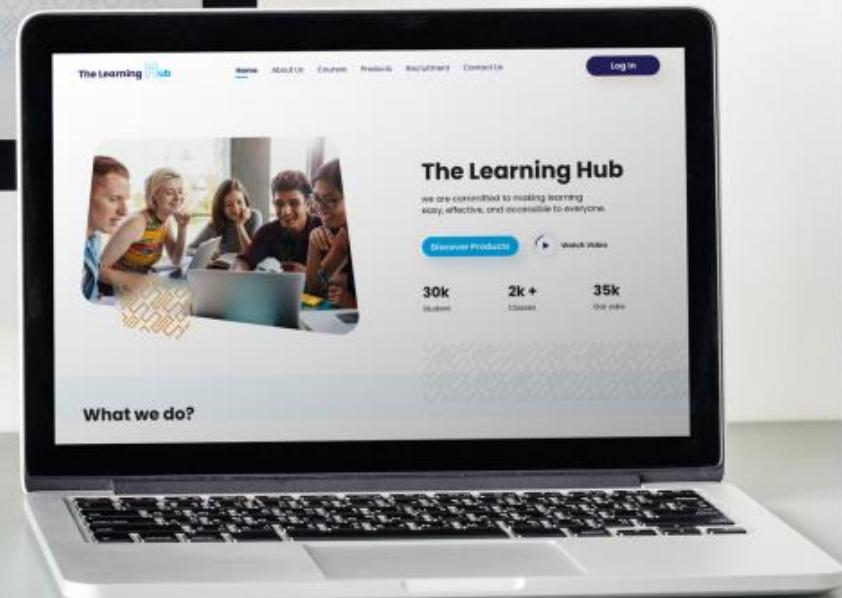
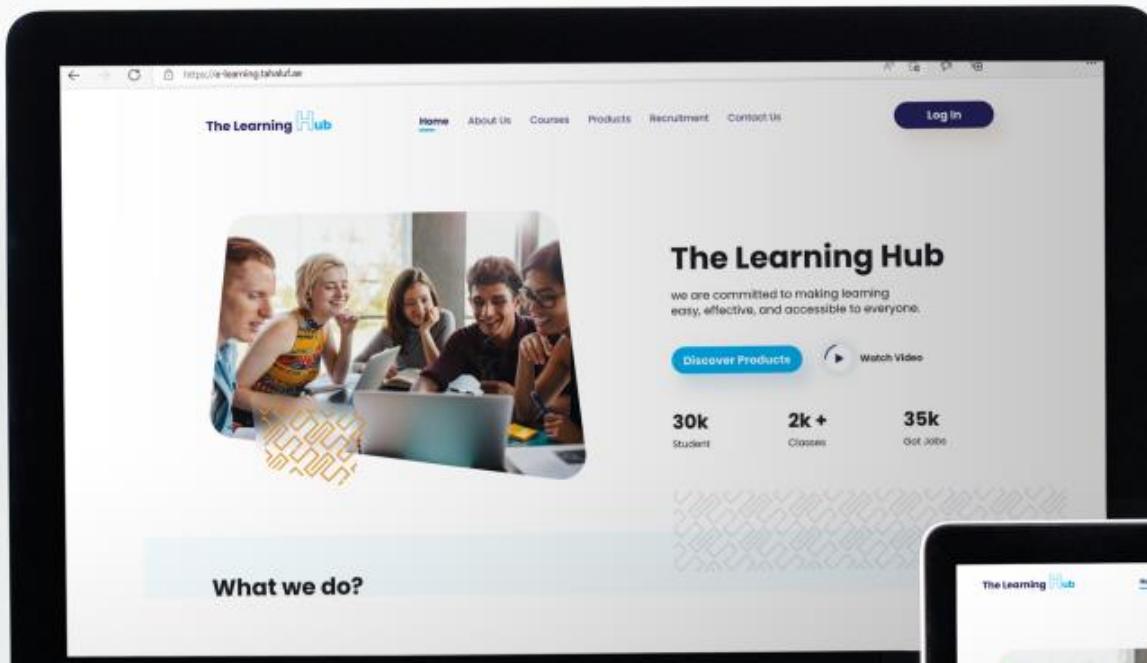
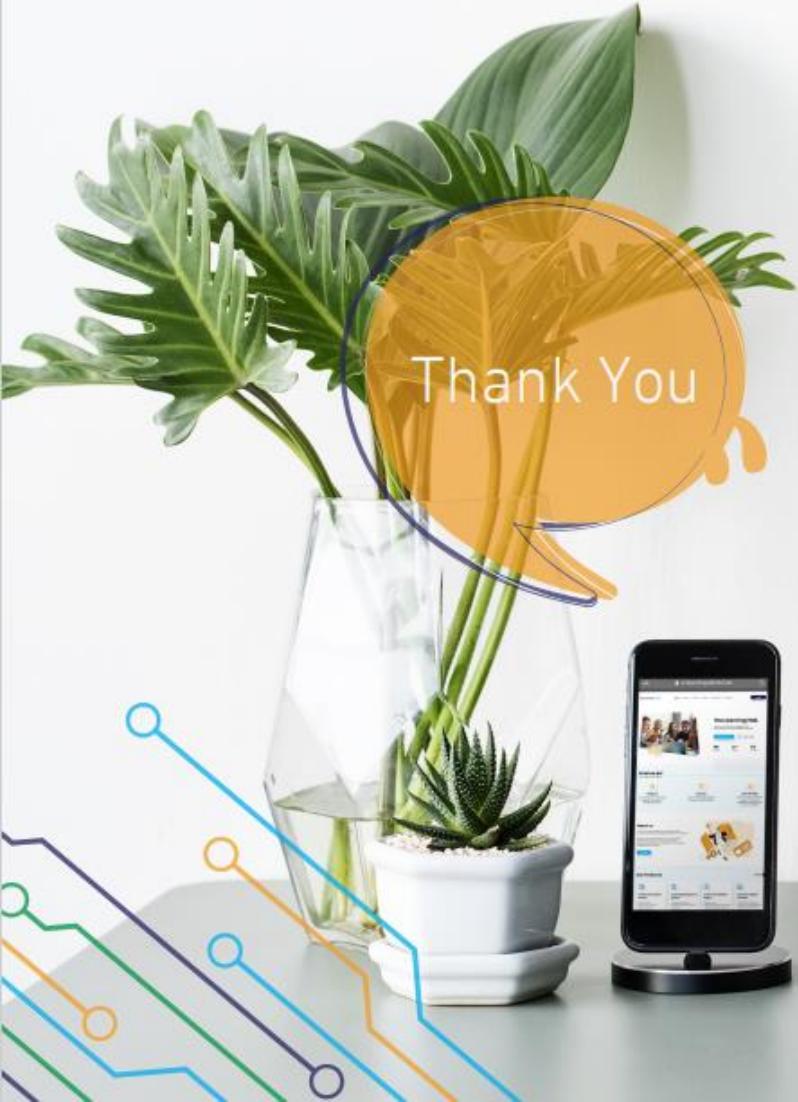
Output :



The screenshot shows a web browser window with the following details:

- Address Bar:** The address bar shows the URL `localhost:4200`.
- Page Content:** The main content area displays the "The Learning Hub LMS" logo and a navigation menu with links to "Home", "About Us", and "Courses".
- Banner:** A large orange horizontal banner is positioned at the bottom right of the page. It contains the text "this page for admin" in white, accompanied by a yellow warning icon.





Break

9:00 - 10:00

