**Router Guards**

Guards are used to block some routes from loading based on some permissions or blocking a route based if not authenticated.

Guards help us to secure the route or to perform some actions before navigating into a route or leaving the route.

**Use:** We use angular guards to control, whether the user can navigate to or away from the current route.

**Why Guards:**

Allowing the user to navigate all parts of the application is not a good idea. We need to restrict the user until the user performs specific actions like login. Angular provides Route Guards for this purpose.

**Scenario:**

One of the common scenario, where we use Route guards is authentication. We want our App to stop the unauthorized user from accessing the protected route.

We achieve this by using the CanActivate guard, which angular invokes when the user tries to navigate into the protected route.

**Uses of Guards:**

* To Confirm the navigational operation
* Asking whether to save before moving away from a view
* Allow access to certain parts of the application to specific users
* Validating the route parameters before navigating to the route
* Fetching some data before you display the component.

**Types of Guards:**

1.CanActivate guard

2.CanActivate

3.Resolve

4.CanLoad

5.CanActivateChild

**CanActivate**:

This guard decides if a route can be activated (or component gets used). This guard is useful in the circumstance where the user is not authorized to navigate to the target component.

Or the user might not be logged into the system.

**CanDeactivate**:

This Guard decides if the user can leave the component (navigate away from the current route). This route is useful in where the user might have some pending changes, which was not saved.

The CanDeactivate route allows us to ask user confirmation before leaving the component. You might ask the user if it’s OK to discard pending changes rather than save them.

**Resolve**:

This guard delays the activation of the route until some tasks are complete. You can use the guard to pre-fetch the data from the backend API, before activating the route.

**CanLoad**:

The CanLoad Guard prevents the loading of the Lazy Loaded Module. We generally use this guard when we do not want to unauthorized user to be able to even see the source code of the module.

This guard works similar to CanActivate guard with one difference. The CanActivate guard prevents a particular route being accessed. The CanLoad prevents entire lazy loaded module from being downloaded,

Hence protecting all the routes within that module.

**CanActivateChild**:

This guard determines whether a child route can be activated. This guard is very similar to CanActivateGuard. We apply this guard to the parent route.

The Angular invokes this guard whenever the user tries to navigate to any of its child route. This allows us to check some condition and decide whether to proceed with the navigation or cancel it.

**How to Build Angular Route Guards**

Building the Guards are very easy.

1. Build the Guard as Service.

ex:

(import { CanActivate } from '@angular/router';)

(@Injectable()

export class ProductGuardService implements CanActivate {})

2. Implement the Guard Method in the Service.

ex:

canActivate(): boolean {

// Check weather the route can be activated;

return true;

// or false if you want to cancel the navigation;

}

3. Register the Guard Service in the Root Module.

ex:

providers: [ProductService,ProductGuardService]

4. Update the Routes to use the guards.

ex:

{ path: 'product', component: ProductComponent, canActivate : [ProductGuardService]

**Order of execution of route guards**

A route can have multiple guards and you can have guards at every level of a routing hierarchy.

1.CanDeactivate() and CanActivateChild() guards are always checked first. The checking starts from the deepest child route to the top.

2.CanActivate() guard is checked next and checking starts from the top to the deepest child route.

3.CanLoad() is invoked next, If the feature module is to be loaded asynchronously.

4.Resolve() Guard is invoked last.

**summary**:

The angular Guards are a great tool, which helps us to protect the route. They also help us to run some logic, get data from the back end server, etc. You can also create multiple guards against a single route or use the same guard against multiple routes.

**MATERIAL**

UI component library

provides us with components to build awesome user interfaces in quick time.

It is also the implementation of Google's material design specification.

Add angular material to an application.

ng add @angular/material

**Typography:**

Deals with the style and appearance of text on the website.

**Components:**

* Buttons and Indicators

ex: (import {MatButtonModule} from '@angular/material';)

.html

(<button mat-button>Hello</button>)

* Navigation
* Layout
* Form Controls
* Popups and Modals
* Data Tables

We will now import the modules in the parent module - app.module.ts as shown below.

import { MatButtonModule, MatMenuModule, MatSidenavModule } from '@angular/material';

Let us now add the material-css support in styles.css.

@import "~@angular/material/prebuilt-themes/indigo-pink.css";

**Menu**

To add menu, <mat-menu></mat-menu> is used. The file and Save As items are added to the button under mat-menu. There is a main button added Menu. The reference of the same is given the <mat-menu> by using [matMenuTriggerFor]="menu" and using the menu with # in<mat-menu>.

In app.component.html

<button mat-button [matMenuTriggerFor] = "menu">Menu</button>

<mat-menu #menu = "matMenu">

**SideNav**

To add sidenav, we need <mat-sidenav-container></mat-sidenav-container>. <mat-sidenav></mat-sidenav> is added as a child to the container. There is another div added, which triggers the sidenav by using (click)="sidenav.open()".

In app.component.html

<mat-sidenav-container class="example-container" fullscreen>

<mat-sidenav #sidenav class = "example-sidenav">

Angular 7

</mat-sidenav>

**Datepicker**

Let us now add a datepicker using materials. To add a datepicker, we need to import the modules required to show the datepicker.

In app.module.ts, we have imported the following module as shown below for datepicker

import { MatDatepickerModule, MatInputModule, MatNativeDateModule } from '@angular/material';

The app.component.html is as shown below −

<mat-form-field>

<input matInput [matDatepicker] = "picker" placeholder = "Choose a date">

<mat-datepicker-toggle matSuffix [for] = "picker"></mat-datepicker-toggle>

<mat-datepicker #picker></mat-datepicker>

</mat-form-field>

**ANIMATIONS**

Animation was available with Angular 2, from Angular 4 onwards animation is no more a part of the @angular/core library, but is a separate package that needs to be imported in app.module.ts.

To start with, we need to import the library with the below line of code

import { BrowserAnimationsModule } from '@angular/platform-browser/animations';

The BrowserAnimationsModule needs to be added to the import array in app.module.ts as follows

import { BrowserAnimationsModule } from '@angular/platform-browser/animations';

In app.component.html, add the html elements, which are to be animated.

We have to import the animation function that is to be used in the .ts file as shown

import { trigger, state, style, transition, animate } from '@angular/animations';

**VIRTUAL SCROLLING**

Virtual scrolling is one of the feature added to Angular 7 .It is one of the feature added to CDK(Component Development Kit) to display all the dom elements to the user, as the user scrolls, the next list is displayed. This gives faster experience as the full list is not loaded at one go and only loaded as per the visibility on the screen.

**Why Virtual Scrolling Module ?**

Consider you have a UI which has a big list where loading all the data together can have performance issues

To get this feature we should add dependency as follows

npm install @angular/cdk save

Then add the virtual scrolling Module in app.module.ts as follows

import { ScrollDispatchModule } from '@angular/cdk/scrolling';

In which ever component you are working in that respective component.html file you should add tag as follows

<cdk-virtual-scroll-viewport>

**DRAG AND DROP**

The new Drag and Drop feature added to Angular 7 CDK helps to drag and drop the elements from the list. We will understand the working of Drag and Drop Module with the help of an example. The feature is added to cdk. We need to first download the dependency as shown below

npm install @angular/cdk --save

Then add the Drag and Drop Module in app.module.ts as follows

import { DragDropModule } from '@angular/cdk/drag-drop';

Later we need to add the dragdrop cdk properties in app.component.html as shown below −

<div cdkDropList

#personList = "cdkDropList"

[cdkDropListData] = "personaldetails"

[cdkDropListConnectedTo] = "[userlist]"

class = "example-list"

(cdkDropListDropped) = "onDrop($event)" >

To add the drop feature, we need to add the event onDrop in app.component.ts as shown below

import {CdkDragDrop, moveItemInArray, transferArrayItem}

from '@angular/cdk/drag-drop';