|  |
| --- |
|  |
| Angular 8 |
|  |
| Angular – Services And AuthGuard |

**TechBrain Express**

December 17, 2019

Authored by: Ashu

Angular – Service and AuthGuard

# Demonstrate implementation of service

# Demonstrate protecting routes using auth guard

Follow: F:\AngularOct19\ revatureBookStoreAddBookLogin-CTSOBL

<https://howtodoinjava.com/angular/angular2-service/>

# Why Service?

Components shouldn't fetch or save data directly and they certainly shouldn't knowingly present fake data. They should focus on presenting data and delegate data access to a service.

Services are a great way to share information among classes that *don't know each other*.

 An Angular service is a stateless object and provides some very useful functions. These functions can be invoked from any component of Angular, like Controllers, Directives, etc. This helps in dividing the web application into small, different logical units which can be reused.

For example, your controller is responsible for the flow of data and binding the model to view. An Angular application can have multiple controllers, to fetch data which is required by the entire application. Making an AJAX call to the server from the controller is redundant, as each controller will use similar code to make a call for the same data. In such cases, it's extremely useful to use a service, as we can write a service which contains the code to fetch data from the server and inject the service into the controller. Services will have functions to make a call. We can use these functions of services in the controller and make calls to the server, that way we need not write the same code again and it can be used in components other than controllers as well. Also, controllers no longer have to perform the task of fetching the data, as services take care of this, thus achieving the objective of Separation of Concerns.

## Service related Decorators

1. @Injectable - Registering the provider in the @[Injectable](https://angular.io/api/core/Injectable) metadata also allows Angular to optimize an app by removing the service if it turns out not to be used after all.

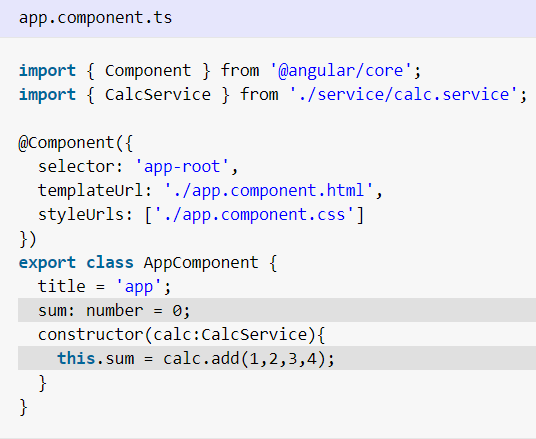
Two ways to apply Service

|  |  |
| --- | --- |
| Global Level | Local Level |
| o inject as global service, inject the **service into root module**. | To inject as local service, inject the **service into component** directly. |
|  |  |

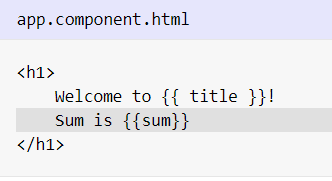
#### Add a method to service



#### 2) Import service to compoment and use the method to update model



#### 3) Update view HTML



#### 4) Run the app

Now run the app with command 'ng serve' and check the output at 'http://localhost:4200/'.

Note: Provide method is not defined because of Providin =’root’ clause in app.service.ts

## Lab: Upgrade above code to show current date and time on service initialization

1. Update calcservice.ts

import { Injectable } from '@angular/core';

@Injectable({

providedIn: 'root'

})

export class CalcServiceService {

constructor() { }

public add(...params : number[]) : number{

let result=0;

for( let val of params )

{

result+=val;

}

return result;

}

showTodayDate() {

let ndate = new Date();

return ndate;

}

}

1. Update app.component.ts .

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

import { CalcServiceService } from './calc-service.service';

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class AppComponent {

title = 'ServiceDemo';

sum:number=0;

constructor(private calc:CalcServiceService)

{console.log("hi");

this.sum=calc.add(1,2,3,4);

console.log(this.sum);

}

todaydate: Date;

ngOnInit() {

this.todaydate = this.calc.showTodayDate();

}

}

1. Update app.component.html

<h1>

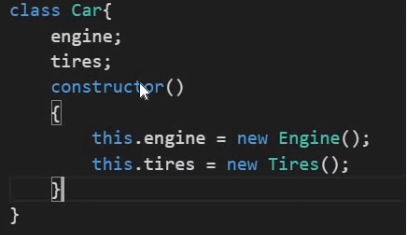
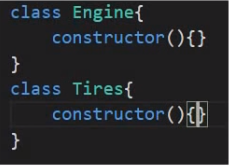
Welcome to Calculate Service {{title}}

Sum is {{sum}} Todayz Date: {{todaydate}}

</h1>

# Dependency Injection

## What is drawback without DI?

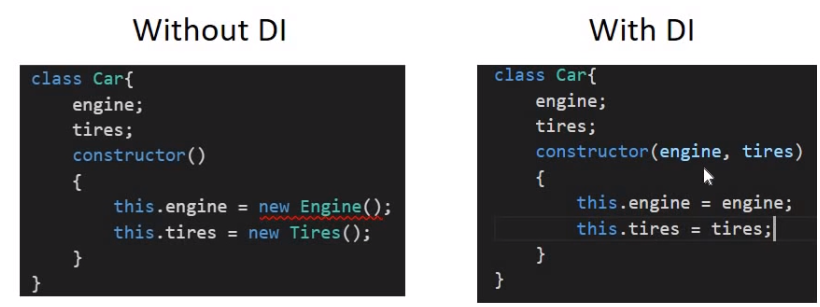


Issue – 1. In case there is a change engine class , there would be change in car Car constructor.so there is a FLEXIBILITY issue. As dependency change car needs to be changed.

2.Testing not flexible.

## DI as Design Pattern

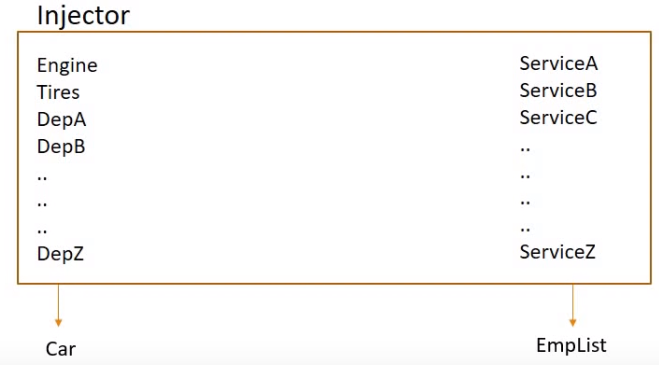
DI is design pattern where class receives the dependencies from external sources rather than creating them itself.



Here there is no creation of dependency in Car.

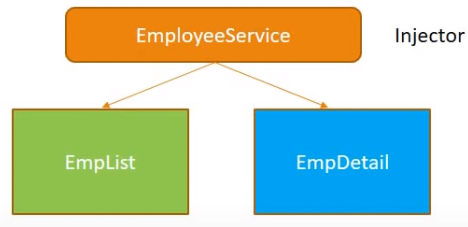
## DI as Angular Provide

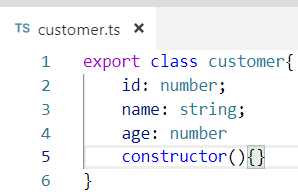
DI in angular Provides injector using that you inject all your dependencies. Injectors are container of all dependencies such as engine/Tires /….. so if Car ask for dependency it will go to Injetcor



**DI as Framework**

1. Define a CustomerService Class
2. Register with Injector
3. Declare as dependency in CustList and CustDetail component



1. 
2. CustomerService.ts

import { Injectable } from '@angular/core';

@Injectable({

providedIn: 'root'

})

export class CustomerServiceService {

constructor() { }

getCustomers()

{

return [

{"id":100, "name": "Ashu", "Age":20},

{"id":200, "name": "Ashish", "Age":30},

{"id":300, "name": "Ani", "Age":10}

];

}

}

1. Customer.service.ts

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

import { CustomerServiceService } from '../customer-service.service';

@Component({

selector: 'app-customer-list',

//templateUrl: './customer-list.component.html',

template:`<h2> Customer List</h2>

<ul \*ngFor="let customer of customers">

<li>{{customer.name}}</li>

</ul>

`,

styleUrls: ['./customer-list.component.css']

})

export class CustomerListComponent implements OnInit {

public customers = [] ;

constructor(private custservice:CustomerServiceService ) {}

ngOnInit() {

this.customers=this.custservice.getCustomers();

console.log(this.customers[0].name)

}

}

# Demonstrate protecting routes using AUTHGuard

## Why AuthGuard?

At the moment, ***any* user can navigate *anywhere*** in the application *anytime*. That's not always the right thing to do.

* Perhaps the **user is not authorized to navigate** to the target component.
* Maybe **the user must login (*authenticate***) first.
* Maybe **you should fetch some data before you display the target** component.
* You might want **to save pending changes before leaving a component**.
* You might ask the **user if it's OK to discard pending changes** rather than save them.

## How Guard Works?

A guard's return value controls the router's behavior:

* If it returns true, the navigation process continues.
* If it returns false, the navigation process stops and the user stays put.
* If it returns a [UrlTree](https://angular.io/api/router/UrlTree), the current navigation cancels and a new navigation is initiated to the [UrlTree](https://angular.io/api/router/UrlTree) returned.
* The guard *might* return its boolean answer synchronously.

## What is scenario where gurad can behave asynchronously?

the guard can't produce an answer synchronously. The guard could ask the user a question, save changes to the server, or fetch fresh data. These are all asynchronous operations.

* Accordingly, a routing guard can return an Observable<boolean> or a Promise<boolean> and the router will wait for the observable to resolve to true or false.

## Guard Interfaces?

* [**CanActivate**](https://angular.io/api/router/CanActivate) to mediate navigation *to* a route. The [CanActivate](https://angular.io/api/router/CanActivate) guard is the tool to manage these navigation business rules.
* [**CanActivateChild**](https://angular.io/api/router/CanActivateChild) **to** mediate navigation *to* a child route.
* [**CanDeactivate**](https://angular.io/api/router/CanDeactivate)to mediate navigation *away* from the current route.
* [**Resolve**](https://angular.io/api/router/Resolve)to perform route data retrieval *before* route activation.
* [**CanLoad**](https://angular.io/api/router/CanLoad)to mediate navigation *to* a feature module loaded *asynchronously*.

## Steps :

1. Build Route guard
2. Register guard with angular dependency injection system
3. Tie guard to a route

# CanDeactivate

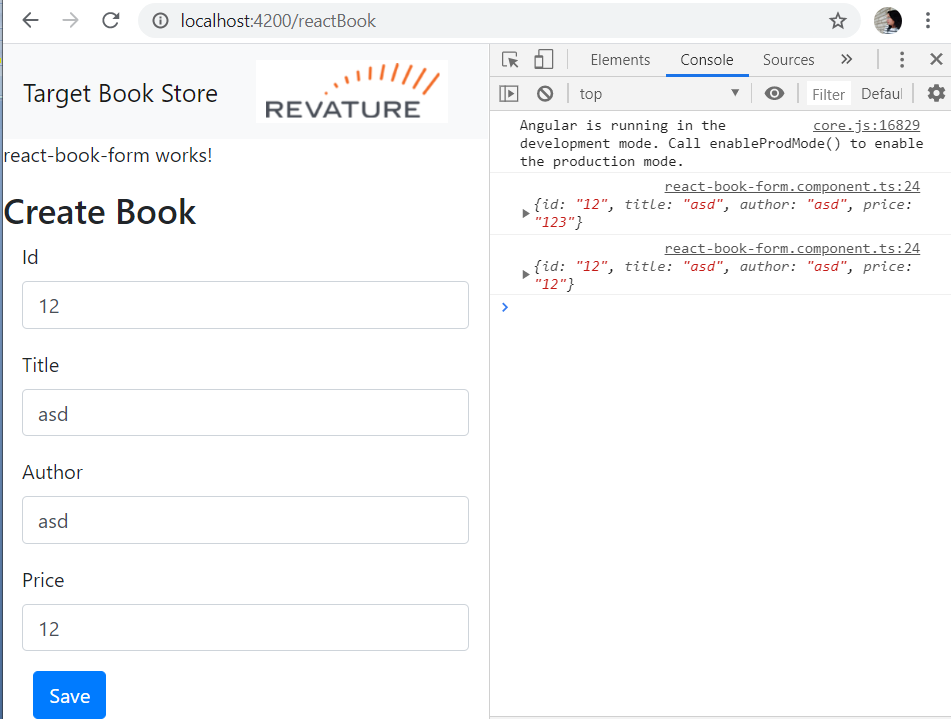
## Why deActivate AuthGurad?

When you enter an dprior to saving switch another link, data is lost, ideally it should ask Do you want to discard data?

Based on answer it follow further.

CanDeactivate – Guard navigation away from current navigation.

Step 1:



## Step1 : Build Route guard

**F:\AngularOct19\revatureBookStoreAddBookLogin-CTSOBL\can-deactivate-guard.service**

import{Injectable} from '@angular/core';

import {CanDeactivate} from '@angular/router';

import { Observable } from 'rxjs';

export interface CanComponentDeactivate{

canDeactivate:() => Observable <boolean> | Promise<boolean> | boolean;

}

@Injectable()

export class CanDeactivateGuard implements CanDeactivate<CanComponentDeactivate>{

canDeactivate(component: CanComponentDeactivate)

{

//if passed componenet has canDeactive method

return component.canDeactivate ? component.canDeactivate():true;

}

}

**F:\AngularOct19\revatureBookStoreAddBookLogin-CTSOBL\ReactBookFormComponent.ts**

export class ReactBookFormComponent implements OnInit {

bookForm: FormGroup;

//We have bind this bookFrom in html and same with belw controls

constructor() { }

ngOnInit() {

this.bookForm=new FormGroup({

id: new FormControl(),

title: new FormControl(),

author: new FormControl(),

price: new FormControl(),

});

}

onSubmit(): void{

console.log(this.bookForm.value);

}

canDeactivate() {

console.log('i am navigating away');

// if the editName !== this.user.name

if (this.bookForm.dirty)

return window.confirm('Discard changes?');

return true;

}

}

## Step 2: Register guard with angular dependency injection system

**F:\AngularOct19\revatureBookStoreAddBookLogin-CTSOBL\angular.modular.ts**

providers: [BookServiceService,AuthService, CanDeactivateGuard],

1. Tie guard to a route

## Step 3: Tie guard to a route

**F:\AngularOct19\revatureBookStoreAddBookLogin-CTSOBL\app-routing.modular.ts**

const routes: Routes = [

{ path: 'Login', component: LoginComponent },

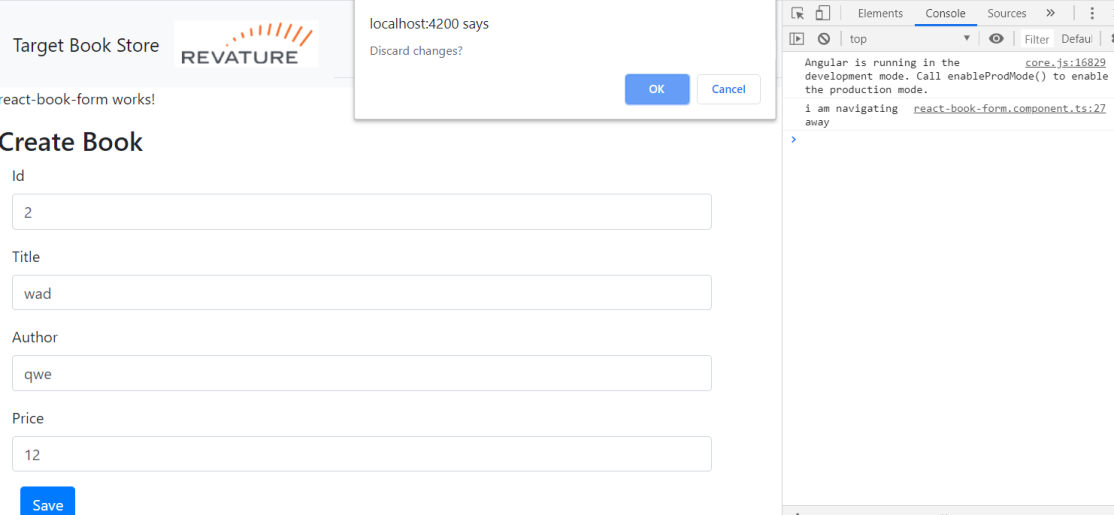
{ path: 'Home', component: HomeComponent },// if getservletpath()=Home, forward to HomeComponent.ts

{ path: 'addBook', component: BookComponent },

{ path: 'reactBook', component: ReactBookFormComponent, canDeactivate: [ CanDeactivateGuard ] },

];

## Verify :



# CanActivate

## Why CanActivate()?

A CanActivate guard is useful when we want to check on something before a component gets used.

This is extremely useful for scenarios like:

* checking if a user is authenticated
* checking if a user has permission

<https://scotch.io/courses/routing-angular-applications/canactivate-and-canactivatechild>

F:\AngularOct19\revatureBookStoreAddBookLogin-CTSOBL\src\app\shared\guards\auth-guard.service.ts

## Step 1: Build Guard - AuthGuard

import { Injectable } from '@angular/core';

import { CanActivate, CanActivateChild } from '@angular/router';

@Injectable()

export class AuthGuard implements CanActivate, CanActivateChild {

canActivate() {

console.log('i am checking to see if you are logged in');

return true;

}

canActivateChild() {

console.log('checking child route access');

return true;

}

}

## Verify – Click on Add button to check

## 

## Step 2: Register Guard

F:\AngularOct19\revatureBookStoreAddBookLogin-CTSOBL\src\app\app.module.ts

providers: [BookServiceService,AuthService,AuthGuard, CanDeactivateGuard],

## Step 3: Set Map Route with Path

const routes: Routes = [

{ path: 'Login', component: LoginComponent },

{ path: 'Home', component: HomeComponent },// if getservletpath()=Home, forward to HomeComponent.ts

{ path: 'addBook', component: BookComponent , canActivate: [AuthGuard],},

{ path: 'reactBook', component: ReactBookFormComponent, canDeactivate: [ CanDeactivateGuard ] },

{ path: 'Author', component: ReactValidateAuthorComponent },