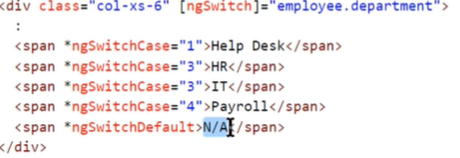
**Important points to remember about Angular service**

* A service in angular is a class
* Irrespective of whether a service has an injected dependency or not, always decorate the angular service class with @Injectable() decorator for consistency and future proof
* If a service is registered at a component level, then that service is available only to that component and to it's children
* If a service is registered at a module level, then that service is available to all the components in the application
* To use a service in a component inject it into the component class constructor

redirecting the user to the "list" route using the **navigate()** method of the angular **Router**service.  
  this.\_router.navigate(['list']);

Switch case in angular is a combination of 3 directives

1. ngSwitch directive 2. ngSwitchCase directive 3. ngSwitchDefault directive



Route Guard Use ------------------ --------------------------------------------

CanDeactivate Guard navigation away from the current route

CanActivate Guard navigation to a route

CanActivateChild Guard navigation to a child route

CanLoad Guard navigation to a feature module loaded asynchronously

Resolve Perform route data retrieval before route activation

There are 3 steps to use a routing guard in Angular.

1. Build the route guard

2. Register the guard with angular dependency injection system

3. Tie the guard to a route Building the route guard.

A route guard can be implemented as a function or a service

Building a route guard:

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

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

import { CreateEmployeeComponent } from './create-employee.component';

@Injectable()

export class CreateEmployeeCanDeactivateGuardService

    implements CanDeactivate<CreateEmployeeComponent> {

    constructor() { }

    canDeactivate(component: CreateEmployeeComponent): boolean {

        if (component.createEmployeeForm.dirty) {

            return confirm('Are you sure you want to discard your changes?');

        }

        return true;

    }

}

**Code Explanation :**

* Since we are implementing the routing guard as a service, decorate the guard class with @Injectable() decorator.
* Since we want to implement CanDeactivate routing guard, make the guard class implement CanDeactivate interface.
* CanDeactivate interface supports generics. In our case, since we are creating a guard for CreateEmployeeComponent, we have specified CreateEmployeeComponent as the argument for the generic type of CanDeactivate interface.
* CanDeactivate interface has one method called canDeactivate(). Our routing guard class needs to provide implementation for this interface method.
* canDeactivate() method returns true or false. True to allow navigation away from the route. False to prevent navigation.
* The first parameter that is passed to the canDeactivate() method is the CreateEmployeeComponent. We are using this parameter to check if the component is dirty. If it is dirty, we are triggering JavaScript confirm dialog to the user.
* If the component is not dirty we simply return true, to allow navigation away from the "create" route.

**How to check if the form is dirty :** Include the following line of code in CreateEmployeeComponent class

@ViewChild('employeeForm') public createEmployeeForm: NgForm;  
  
@ViewChild() decorator provides access to the NgForm directive in the component class. employeeForm which is passed as the selector to the @ViewChild() decorator is the form template reference variable.

**Register the guard with angular dependency injection system :** Since the routing guard is implemented as a service, we need to register it in a module. At the moment we have only one module in our application and that is the root module AppModule. Import and register CreateEmployeeCanDeactivateGuardService in app.module.ts file using the **providers**property.

@NgModule({  
  declarations: […  
  ],  
  imports: […  
  ],  
  providers: [CreateEmployeeCanDeactivateGuardService],  
  bootstrap: [AppComponent]  
})  
export class AppModule { }

**Tie the guard to a route :**Using the route guard, we want to prevent navigating away from the "create" route, so tie the route guard with the "create" route in app.module.ts file as shown below.

const appRoutes: Routes = [

  {

    path: 'list', component: ListEmployeesComponent

  },

  {

    path: 'create',

    component: CreateEmployeeComponent,

    canDeactivate: [CreateEmployeeCanDeactivateGuardService]

  },

  {

    path: '', redirectTo: '/list', pathMatch: 'full'

  }

];

At this point, if you try to navigate away from the "create" route when the form is dirty you get the alert as expected. The browser back and forward buttons also prevent the navigation away from the "create" route.  
  
**CanDeactivate limitations :** CanDeactivate guard does not prevent route deactivation

* If you type a different url in the address bar directly OR
* If you close the tab or the browser window OR
* If you navigate to an external URL

Angular Route Params:

Create a route with parameters : To create a route with parameter include a FORWARD SLASH, a COLON and a place holder for the parameter. The example below, creates a route with parameter id.

{ path: 'employees/:id', component: EmployeeDetailsComponent }

And in the component class,

]

The array in navigate method is called link parameters array.

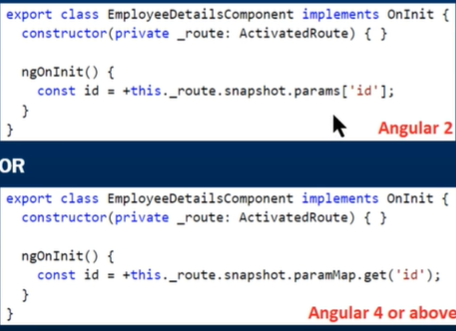
Activating Route parameters declaratively:



Reading the route parameters:

To read the route parameter value use Angular ActivatedRoute service.

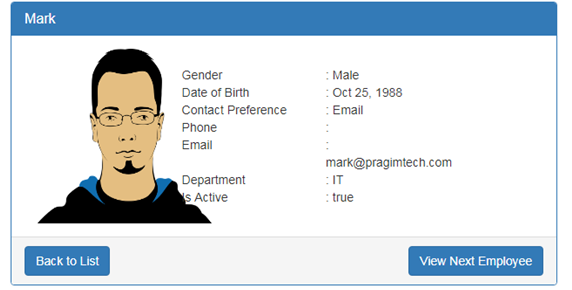
There are 2 ways to read the route parameter value. We can either use the snapshot approach or observable approach.

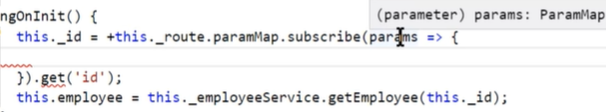


+ symbol is used to convert string to number.

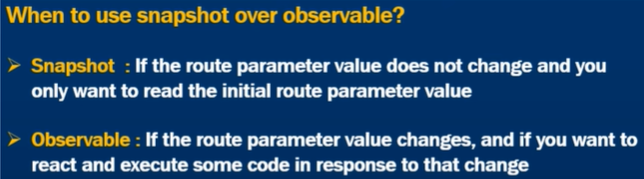
USING OBSERVABLE APPROACH:



Snapshot approach works fine, if you navigate to another component before navigating from the current employee to the next employee. In our case we are always navigating back to the ListEmployeesComponent before navigating to view another employee details.  
  
  
If you expect users to navigate from employee to employee directly, without navigating to another component first, then you need to use the observable approach. Notice, on the EmployeeDetailsComponent we now have **"View Next Employee"** button.

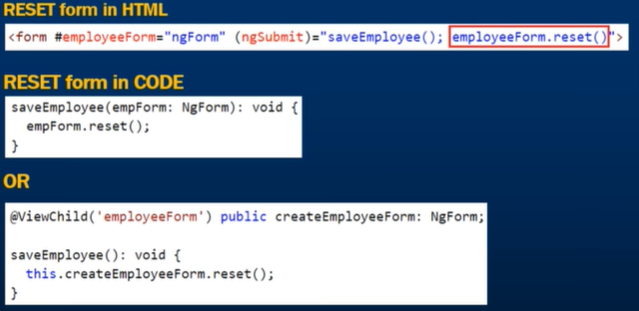


Params is of type paramMap. We can use the get method of paramMap.



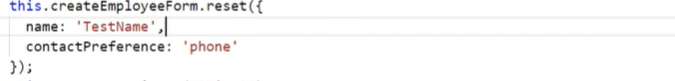
When we subscribe to an observable, we should unsubscribe to it, But Activated Route observable is a type of exception. We do not need to unsubscribe to it explicitly. Angular does it for you.

RESET form:- which will reset all the form control values to their initial state (ie:-it also clears out the form fields) and also resets the form flags like dirty, pristine, valid, touched etc.

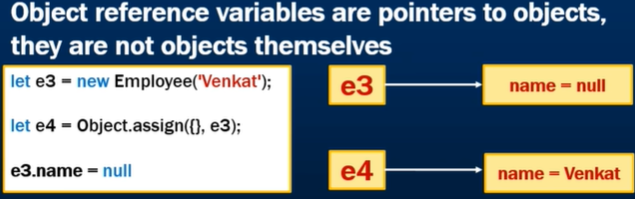
There are wo ways to reset a form:

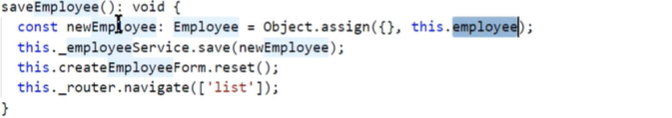
In class ngForm has ’n’ in Capital.

When we reset the form we want to set some default values. This is done by passing an object to the reset method.



Object and reference variables:



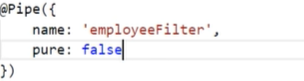


Filter Pipes:

The class implements PipeTransform.

In Angular, there are two categories of pipes

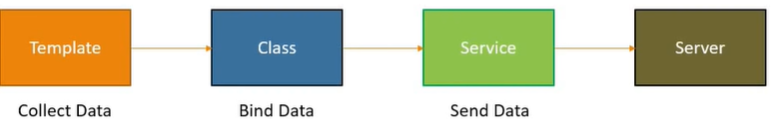
* Pure Pipes
* Impure Pipes  
  When you create a new pipe, it is **pure by default**. To make a pipe impure, set it's **pure**flag to false. Impure pipes can significantly affect the performance of the application. So you have to think very carefully, before you use an impure pipe in your angular application.



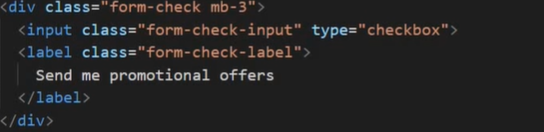
Default value of pure is true.

**QUERY PARAMS:**

**Angular Forms:**

****

Much of the code and logic resides in html template.

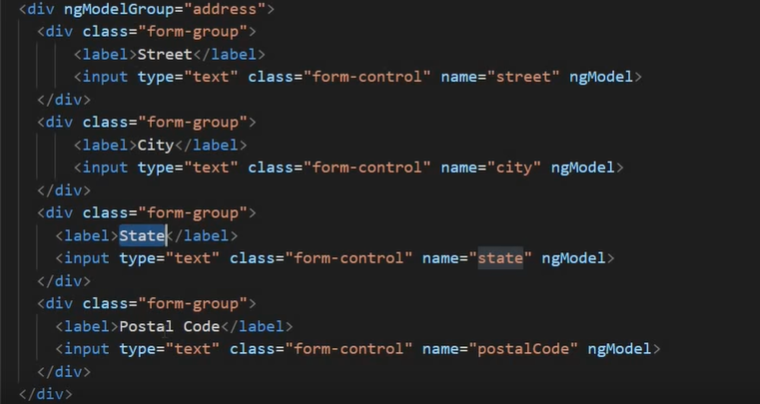


Binding DATA with ngForm:First step with angular forms Is to import FormsModule.

Anytime we use a form tag, angular attaches a ngForm directive to form which gives us info of the form. We get a hold of ngForm directive using template reference variable.

Using the value property, only those form-controls will be tracked having the ngModel Directive.

In addition to ngModel, we also have ngModelGroup directive, used to create a subgroup within a form.





Binding Data to model: