\*\*\*\*\*\*\*Data binding is the most sought feature in [**AngularJS**](https://javabeat.net/angularjs/). There are two directives **ng-model and ng-bind** is frequently used for data binding.

**ng-model** or **data-ng-model** directive in AngularJS is one of the greatest strength to bind the variables used in application with input components. This works as two way data binding.

ng-bind is one way data binding used for displaying the value inside html component as inner HTML. This directive can not be used for binding with the variable but only with the HTML elements content.

The angular $[scope](http://www.code-sample.com/2015/09/scope-vs-rootscope-in-angularjs.html) is a JavaScript object which provide the connectivity between the controller and the view. All the model data bind on view (DOM element) using the $scope for a controller.

[**What is $rootScope?**](http://www.code-sample.com/2015/09/scope-vs-rootscope-in-angularjs.html)

The angularjs $[rootScope](http://www.code-sample.com/2015/09/scope-vs-rootscope-in-angularjs.html" \t "_blank) is the scope and one application can have only one $rootScope and It use like global variable and access in across application. The angular $rootScope is parent scope and $scopes are child scopes.

[**$scope vs. $rootScope**](http://www.code-sample.com/2015/09/scope-vs-rootscope-in-angularjs.html)

The $rootScope objects updates under a specific controller $scope not for all global controllers.

The angular $scope can't be used outside the controller and its only for current controller but the $rootScope can be used anywhere and Its available globally in the application.

All app has one-and-only $rootScope and its lifecycle is the same as the app and its all controllers can have.

At a high level, directives are markers on a DOM element (such as an attribute, element name, comment or CSS class) that tell AngularJS's **HTML compiler** ([$compile](https://docs.angularjs.org/api/ng/service/$compile)) to attach a specified behavior to that DOM element (e.g. via event listeners), or even to transform the DOM element and its children.

AngularJS comes with a set of these directives built-in, like ngBind, ngModel, and ngClass. Much like you create controllers and services, you can create your own directives for AngularJS to use.

**What does it mean to "compile" an HTML template?** For AngularJS, "compilation" means attaching directives to the HTML to make it interactive. The reason we use the term "compile" is that the recursive process of attaching directives mirrors the process of compiling source code in [compiled programming languages](http://en.wikipedia.org/wiki/Compiled_languages).

AngularJS **normalizes** an element's tag and attribute name to determine which elements match which directives. We typically refer to directives by their case-sensitive [camelCase](http://en.wikipedia.org/wiki/CamelCase) **normalized** name (e.g. ngModel). However, since HTML is case-insensitive, we refer to directives in the DOM by lower-case forms, typically using [dash-delimited](https://en.wikipedia.org/wiki/Letter_case#Special_case_styles) attributes on DOM elements (e.g. ng-model).

The **normalization** process is as follows:

1. Strip x- and data- from the front of the element/attributes.
2. Convert the :, -, or \_-delimited name to camelCase.

$ sign is used as prefix for the built-in objects in AngularJS.

The ng-app directive tells AngularJS that this is the root element of the AngularJS application.

All AngularJS applications must have a root element.

You can only have one ng-app directive in your HTML document. If more than one ng-app directive appears, the first appearance will be used.

### [ng-app directive initializes an AngularJS application.](https://www.onlineinterviewquestions.com/angular-js-mcq/" \l "collapseUnfiled4)

[Internationalization is the process of designing and preparing your app to be usable in different languages. Localization is the process of translating your internationalized app into specific languages for particular locales.](https://www.onlineinterviewquestions.com/angular-js-mcq/" \l "collapseUnfiled4)

There are 2 ways to create forms in Angular

1. Template Driven Forms

2. Model Driven Forms (Commonly called Reactive Forms)

Both these approaches have their own pros and cons. For example, Template Driven forms are generally used to create simple forms. On the other hand, Reactive forms are used to create complex forms. For example, if you want to add form controls dynamically or perform cross-field validation we use the Reactive forms approach. There are several other differences, between Template driven and Reactive forms. We will discuss those differences in detail, in a later video.

[#employeeForm](https://www.youtube.com/results?search_query=%23employeeForm) is called the template reference variable. Notice we have assigned "ngForm" as the value for the template reference variable employeeForm. So employeeForm variable holds a reference to the form. When Angular sees a form tag, it automatically attaches the ngForm directive to it. The ngForm directive supplements the form element with additional features. It holds all the form controls that we create with ngModel directive and name attribute, and monitors their properties like value, dirty, touched, valid etc.



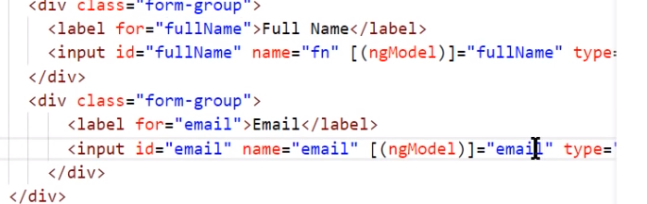
The ngSubmit directive submits the form when we hit the enter key or when we click the Submit button. When the form is submitted, saveEmployee() method is called and we are passing it the employeeForm. The ngForm directive is provided by Angular FormsModule. So for us to be able to use it, we will have to import the FormsModule in our AppModule file (app.module.ts). So please make sure to include the following import statement. Also include "FormsModule" in the imports array of @NgModule decorator.

import { FormsModule } from '@angular/forms'

If "FormsModule" is not imported you will see the following error in the browser developer tools there is no directive with exportas set to ngform.

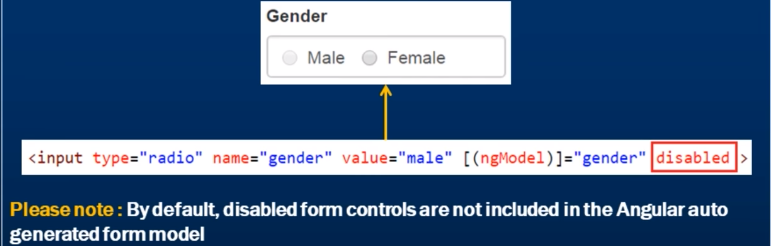


We can send data to component using either ngSubmit or using @ViewChild .

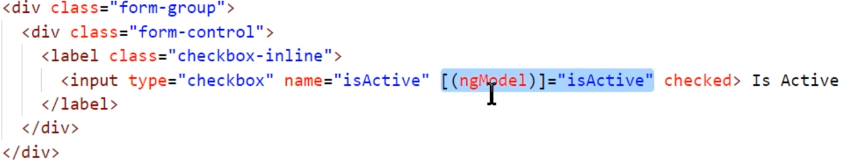


the component class. Angular automatically creates "fullName" property using the value of the "name" attribute of the HTML input element. This is why "name" attribute is also required when we use **ngModel**directive. If we remove the "name" attribute, we get the following error.  
If ngModel is used within a form tag, either the name attribute must be set or the form control must be defined as 'standalone' in ngModelOptions

So the bottom line is, if you want an input element to be tracked by the form make sure to include both the **name attribute**and **ngModel**directive. Otherwise that input element will not be part of the Form model created by Angular.



Bootstrap Checkbox in Angular :-



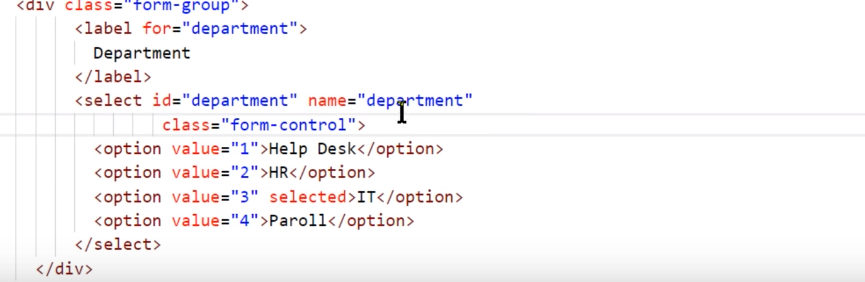
With Angular Template Driven forms, we use "ngModel" directive for two-way data binding. So the moment we put it, the "checked" attribute does not work. To make it work include "isActive" property in the component class and initialise it to true.

At this point you will have "IsActive" checkbox checked by default when the form loads. Now, even if we remove the "checked" attribute from the checkbox it is still checked by default when the form loads. This is because of the two-way data binding that we get with "ngModel" directive.

\*\*\* To disable a checkbox use the disable attribute (instead of checked) …

So we see that ngModel and checked to be used together, we have to define the property in the component class. If any one of them is used (either ngModel or checkedattribute), the value is checked/unchecked in the page.

Dropdown in Angular:

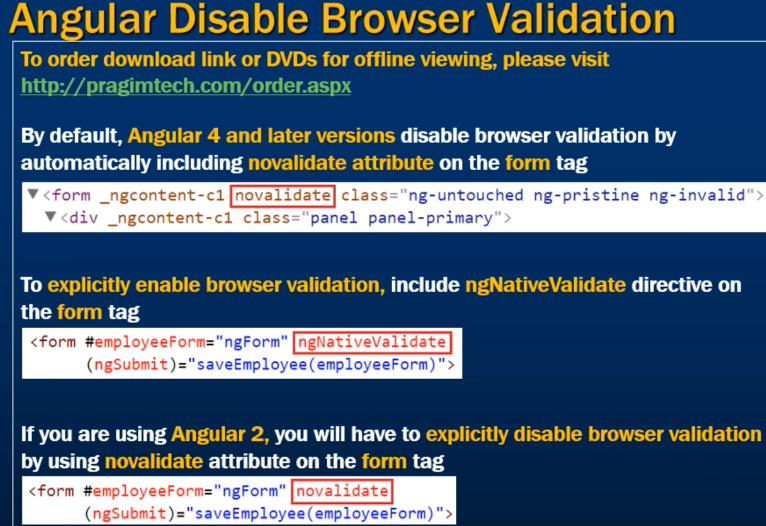


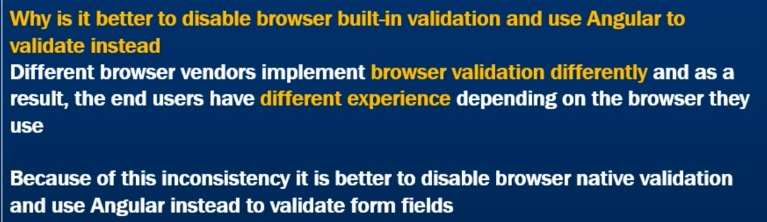
Selected is similar to checked attribute used above. So for selected attribute to be used, we need to place a dept property in component class.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*The code to log the employee form values is in the **saveEmployee**() method and this method should only be called when we click the **"Save"** button.So the question that comes to our mind is, why is the form being submitted when we click "Show Preview" or "Hide Preview" button.This is because of the way we have created the button. If we do not explicitly specify the button **type**attribute, the button behaves like the **"Submit"** button and hence the code in the "saveEmployee()" method is also executed.To prevent this, explicitly set the **type**attribute of the button to "button". This prevents the button from behaving as a Submit button.



* Since ngFor is a structural directive there is an asterisk before it.
* Structural directives modify the DOM, i.e they add or remove the elements from DOM. Adding and removing elements from DOM is different from showing and hiding. We will discuss all these in detail in our upcoming videos.



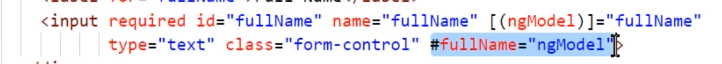


**Angular Form Validation:**

To understand validation in Angular, we need to understand the following 6 Angular validation properties.

1. touched 2. untouched 3. pristine 4. dirty 5. valid 6. invalid

These 6 properties are available at each individual form control level and also at the form level.

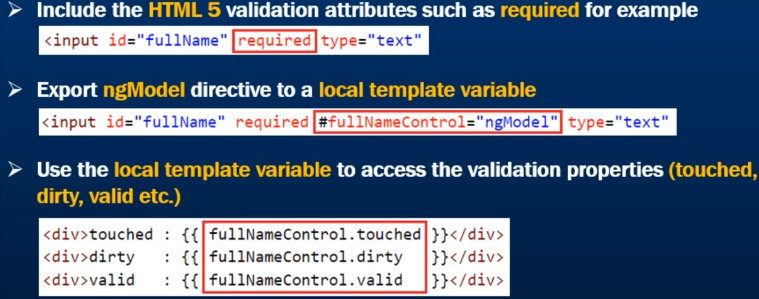


We are exporting NgModel into a local variable called fullName. To do this we are using #fullName="ngModel". This variable fullName is called with different names - local variable, template variable and template reference variable.

At this point if you view the page in the browse you will see the following errorCannot assign to a reference or variable!

We get this error because, Angular generated form model creates a property with name "fullName" and we are also creating a local reference variable with the same name by exporting ngModel to #fullName. Hence we get the error - Cannot assign to a reference or variable.

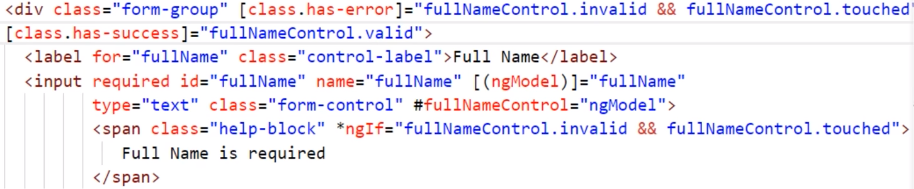
One way to fix this error is, by giving our local template reference variable a different name other than fullName. So if we change #fullName="ngModel" to #fullNameControl="noModel" the error goes away.



Displaying Error Messages:

We will use the following Bootstrap classes for styling validation error messages.

has-error control-label help-block



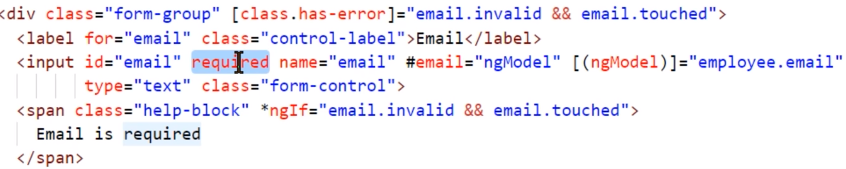
* [class.has-error]="fullNameControl.invalid. This is class binding in angular. If the invalid property returns true, then the Bootstrap class **has-error** is added to the div element, if it is false then the class is removed.
* On the "Full Name" label element we applied **control-label** Bootstrap class. This class turns the label text to red if there is a validation error. Bootstrap **help-block**class on the span element for styling.
* If the user types something, the field is valid, so we want a green border and the label text should also turn green. To achieve this we can use the Bootstrap **has-success**class. The **has-success**class is added when valid property is true and it is removed when it is false

To disable the "Save" button when the form is not valid, bind the invalid property of the **employeeForm**template variable to the disabled property of the button.

<button class="btn btn-primary" type="submit" [disabled]="employeeForm.invalid">

Save</button>.

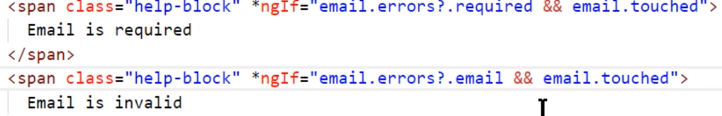
Email Validation:

There are 2 different ways to validate email form field in Angular. We can either use **pattern validator**or **email validator**.

On the email input field, place the **email**



If the email is invalid, angular attaches email key to the errors collection. On the other hand, if the email field is valid, the key email will not be in the errors collection. The question mark here is called the **safe navigation operator**. It protects against null and undefined values in property paths.It checks if error property is not null. It is generally used when we are not sure if a property exists or not. It safely handles null and undefined values, and very useful to prevent null-reference exceptions.



….to be continued



This boolean expression dynamically adds or removes required attribute to the email field depending on whether email contact preference radio button is selected or not.

To create an input property decorate the property with @Input decorator .

Input properties are typically used to pass data from the parent to child component.

To detect and react when an input property value changes, we have 2 options. We can either use

1. Property Setter

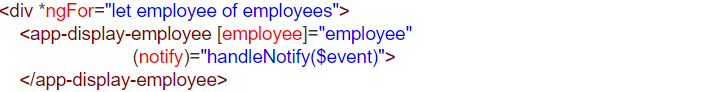
2. ngOnChanges Life Cycle Hook

So the output property is an event defined using EventEmitter object and decorated with @Output decorator.To specify the type of data that we want to pass from the child component to parent component we use the EventEmitter generic argument.

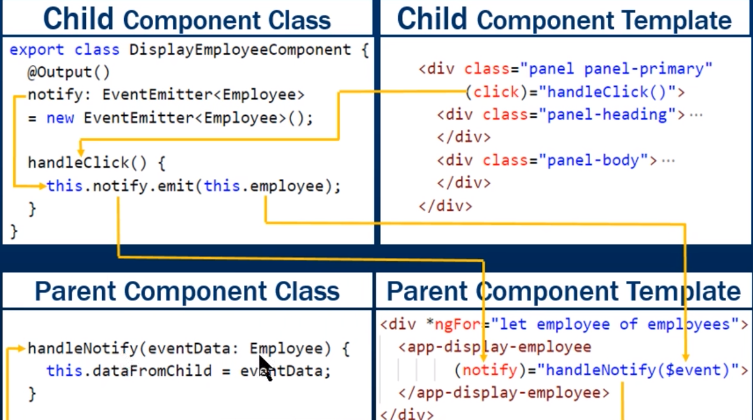
The event data is commonly called event payload.

You can only pass one value using EventEmitter. If you want to pass more than one value using EventEmitter use a custom type like Employee as the generic argument.

To raise the custom event, use the emit() method of the EventEmitter object. The event payload is passed as the argument to the emit() method.



 Notice we are using **event binding** to bind to the notify event. We are handling the notify event in the parent component using handleNotify($event) method. $event is the data from the child component.



We can call child component methods and properties from parent component. One way to do this is by using output properties. Another way, to pass data from the child component to the parent component is by using a **template reference variable**.

Here is the child component class. Notice the class has **employee**property and getNameAndGender() method. We will be calling both of these from the parent component (ListEmployeesComponent) using a template reference variable.

export class DisplayEmployeeComponent {

  @Input() employee: Employee;

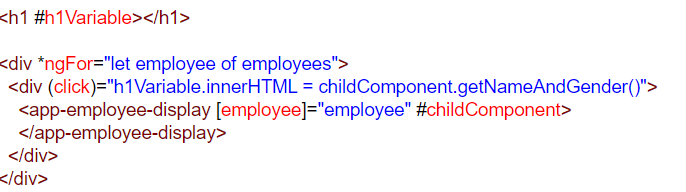
  getNameAndGender(): string {

    return this.employee.name + ' ' + this.employee.gender;

  }

}

Code in Parent Component View Template



* #childComponent is the template reference variable to the child component.
* <div (click)="handleClick(childComponent.getNameAndGender())">. Using the template reference variable we are calling the child component method **getNameAndGender()**. The value this method returns is assigned to the innerHTML property of the <h1> element. #h1Variable is the template reference variable for <h1> element.

**Calling the child component property using template reference variable**

<h1 #h1Variable></h1>

<div \*ngFor="let employee of employees">

  <div (click)="h1Variable.innerHTML = childComponent.employee.name + ' ' + childComponent.employee.gender">

    <app-employee-display [employee]="employee" #childComponent>

    </app-employee-display>

  </div>

</div>

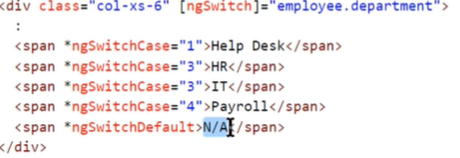
**Important points to remember about Angular service**

* 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.

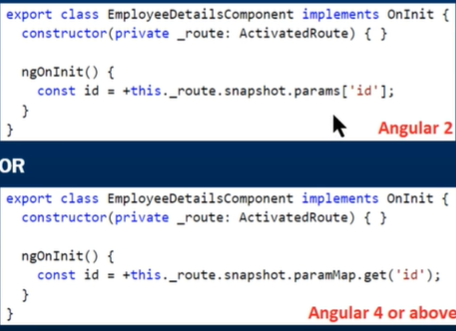
Activating Route parameters declaratively:



Reading the route parameters:

To read the route parameter value use Angular ActivatedRoute service.

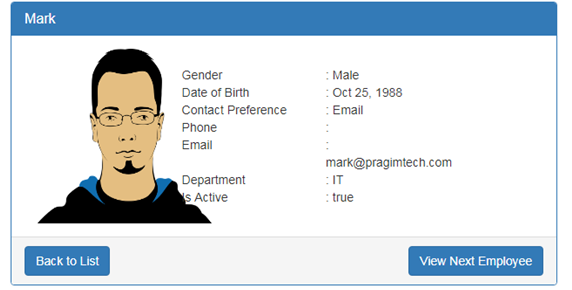
There are 2 ways to read the route parameter value. We can either use the snapshot approach or observable approach.

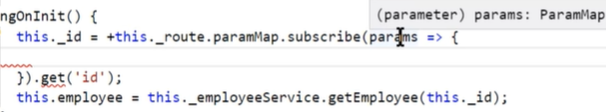


+ symbol is used to convert string to number.

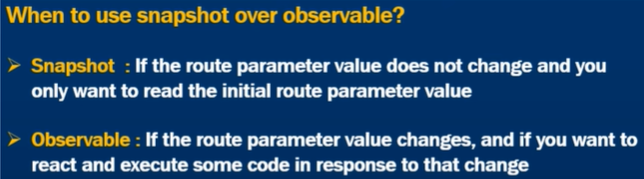
USING OBSERVABLE APPROACH:



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.

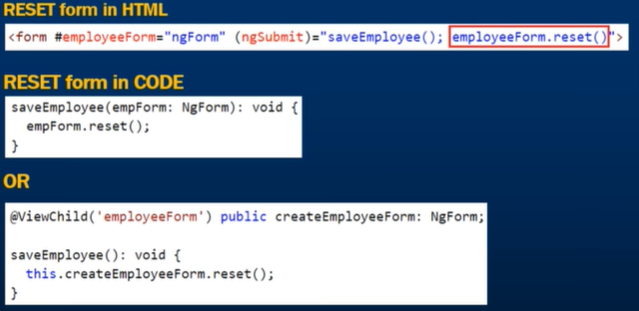


Params is of type paramMap. We can use the get method of paramMap.



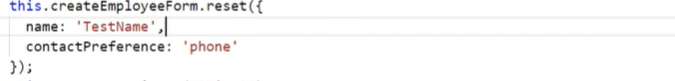
When we subscribe to an observable, we should unsubscribe to it, But Activated Route observable is a type of exception. We do not need to unsubscribe to it explicitly. Angular does it for you.

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.

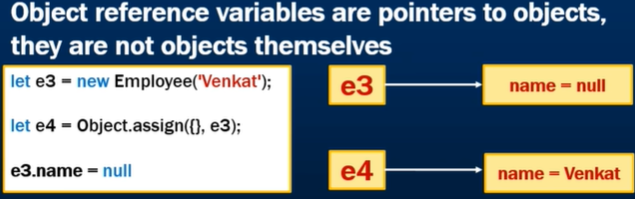
There are wo ways to reset a form:

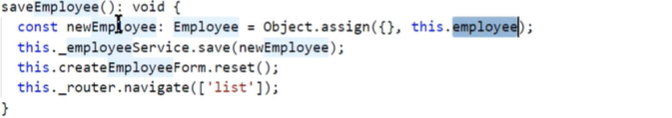
In class ngForm has ’n’ in Capital.

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:



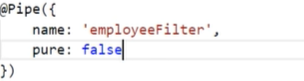


Filter Pipes:

The class implements PipeTransform.

In Angular, there are two categories of pipes

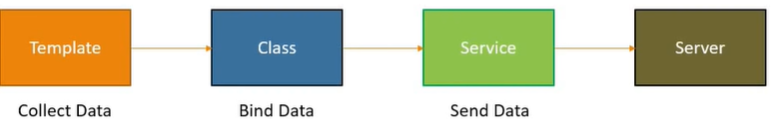
* Pure Pipes
* Impure Pipes  
  When you create a new pipe, it is **pure by default**. To make a pipe impure, set it's **pure**flag to false. Impure pipes can significantly affect the performance of the application. So you have to think very carefully, before you use an impure pipe in your angular application.



Default value of pure is true.

**QUERY PARAMS:**

**Angular Forms:**

****

Much of the code and logic resides in html template.

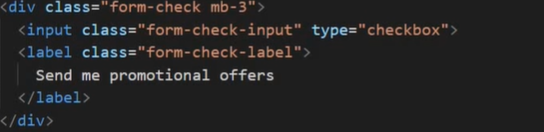
Easy to use and similar to angularJS forms.

Two way data binding with ngModel.

We don’t have to keep track of input field values and react to change the input field values. Angular takes care of that using ngModel directive. As a result we have bulky html code and component component code.

Drawbacks: Unit testing is a challenge.

Readability decreases with complex forms and validations.

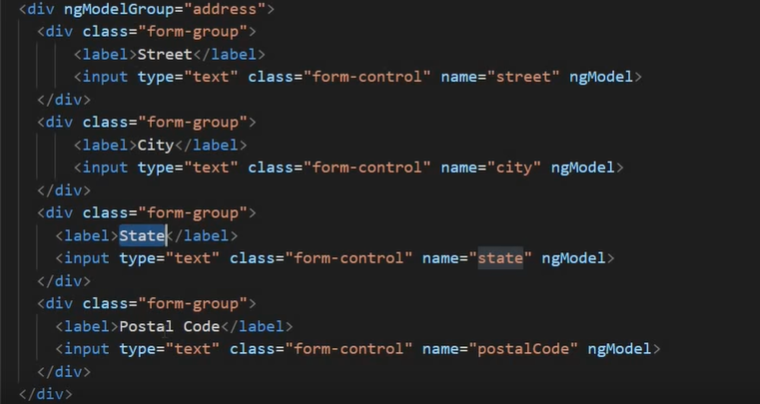


Binding DATA with ngForm:First step with angular forms Is to import FormsModule.

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.

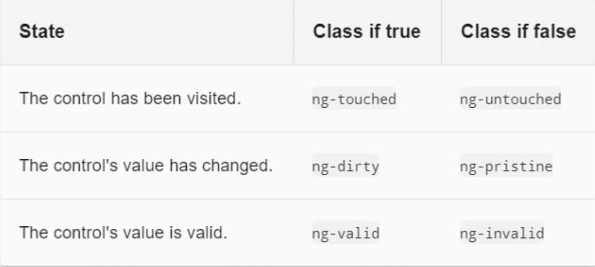
Using the value property, only those form-controls will be tracked having the ngModel Directive.

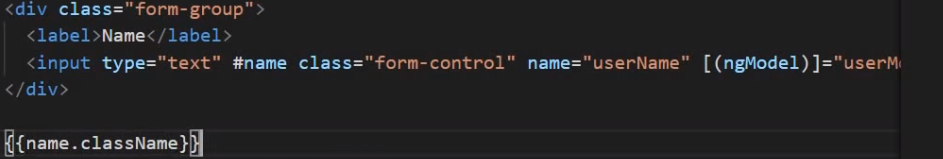
In addition to ngModel, we also have ngModelGroup directive, used to create a subgroup within a form.





Binding Data to model:

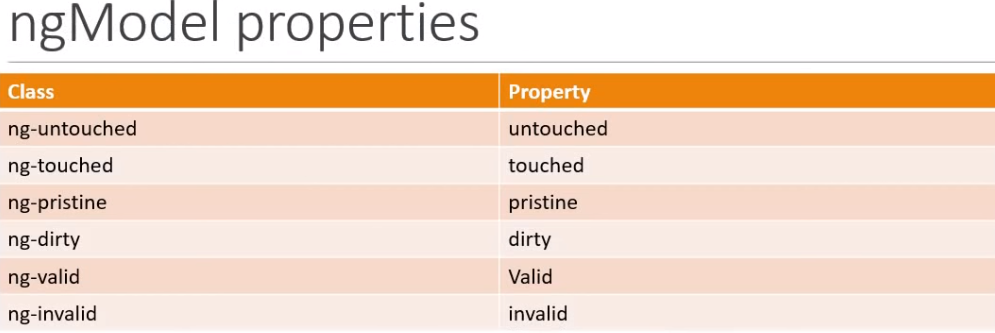




Output



For each of the above classes, angular provides an associated property on ngModel directive.



To get access to these ngModel properties, we should create a reference to ngModel directive.



The Temp ref variable in above code points to input element in DOM.



Now it points to ngModel of form-control and bind to all its properties

So we can use name.property (ex: name.touched, name.invalid)..

Bootstrap 4 provides **is-invalid class**.

Displaying error messages:



To display diff messages for each error , use the errors property on ngModel. Or

For multiple Validation, create a block that get rendered only if the errors object is defined and field is invalid or ….



Way we can decide what has to be shown using the errors property on ngModel.

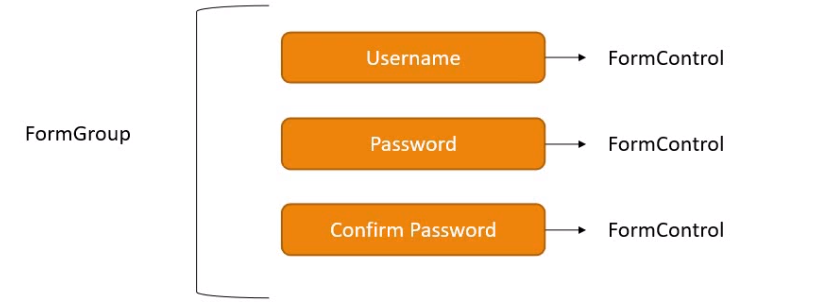
VALIDATION A SELECT CONTROL:

Disabling submit button using validations on an entire form:

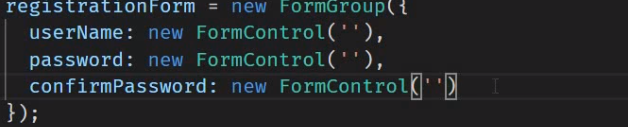


REACTIVE FORMS:

In angular to work with reactive forms , we need reactive forms module.



Each form field is an instance of form-control class and the entire form is an instance of the form-group class.



FormGroup instance with 3 instances of formControl . If we want a default value, pass it in the constructor of formControl.

Now to associate this model with the view in html using formGroup and formControlName.





No need of ngModel and name in Reactive forms.

For nested FormGroup, create a formgroup (address say) inside the formGroup(registration form) of the component.

And in the html, inside a div tag, use FormGroupName to bind it.



Now we have setValue method to set the values programmatically. This method can be called on control class or group class .this method accepts an object that matches the structure of the form group.

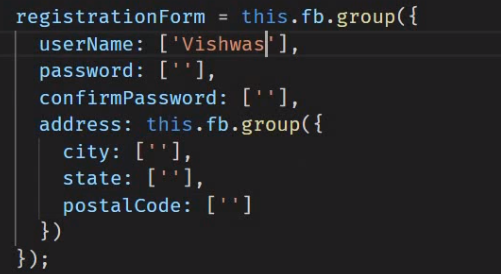
setValue() needs all the values to be passed. But if you want to pass only few values, use patchValue().

Creating from-Control instances manually becomes very repetitive. To avoid this we have form-Builder. Using this , we will be creating instances of form-control but with very lesser code.

Form-Builder is a service, so first import it instead of formGroup and Control.

To create formGroup, we call group method.

It takes in an object with different formControl as keys with value as an array.



In reactive forms, validation rules are applied in the component class.

VALIDATIONS in REACTIVE FORMS:

For validations, Make use of validation class.



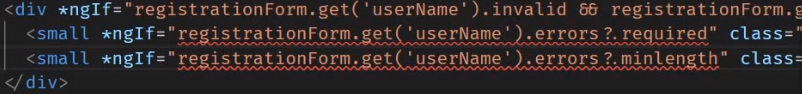




To apply more than one validation, we convert the second element into an array of elements.



If different error messages to be printed, use ngIf:



We can also create a getter for the form control and replace the registrationForm.get(‘userName’) by just userName.

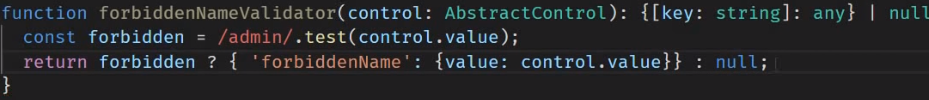


CUSTOM VALIDATIORS:

Custom validator is nothing but a function. It can be written inside a component class. But this validator can be used across many places, so we make a separate file for the validator and export them.

The function accepts one parameter which is the formControl being validated. Its type is abstract control. Import it from angular/forms.

The function returns either of two values. If the validation fails, it returns an object where the key is of type string and value of type any else returns null.



test is the operator to check if input is admin or not. Export the function to use it.



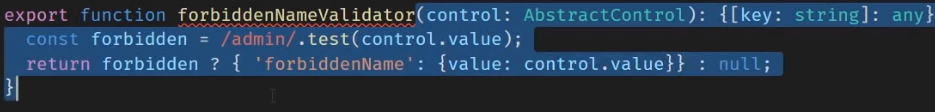
Make sure the validator is in the validators array.

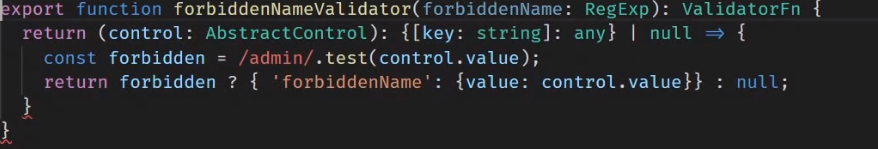


It is possible to pass parameters to custom validators. We can pass a string which can be forbidden to our custom validator. The drawback of validator function is that it can accept only one parameter ie:- the form control.

So we can do is create a factory function that accepts a string as a parameter and returns the validator function.

Copy paste the validator function and add fat arrows after null to make it a proper function.finally return the validator function and export the factory function.



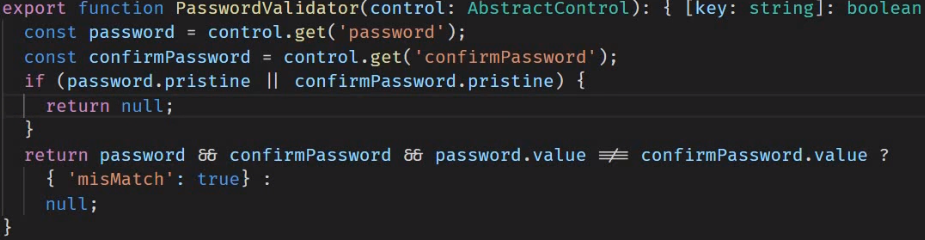


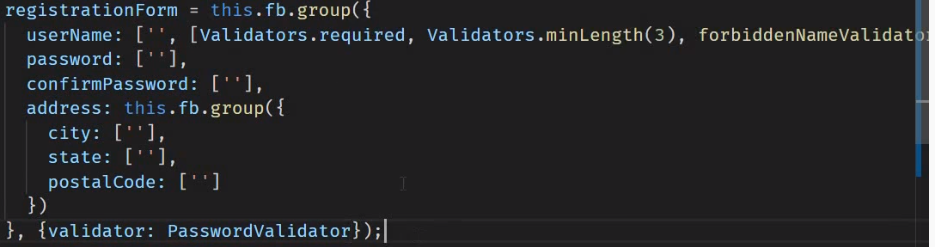
Replace admin with the forbiddenName parameter.

Now in form,

For cross field validation, we have to create a custom validator.

Now in cross field validation, we have to make sure that the control parameter does not refer to an individual formControl but to a formGroup encompassing the various fields being validated.





ANGULAR CRUD TUTORIALS:2,9,10,11,12,24,25-30,36,43,44,46,47….