Contents

[1) Introduction 2](#_Toc509317463)

[2) Reading data in angular 3](#_Toc509317464)

[3) Angular routing and navigation 10](#_Toc509317465)

[4) Angular base href 14](#_Toc509317466)

[5) Angular forms tutorial 17](#_Toc509317467)

[6) Bootstrap radio buttons in Angular 20](#_Toc509317468)

[7) Angular radio button checked by default 24](#_Toc509317469)

[8) Bootstrap checkbox in angular 25](#_Toc509317470)

[9) Angular bootstrap select list 27](#_Toc509317471)

[10) Angular select options from array 30](#_Toc509317472)

[11) Angular datepicker tutorial 32](#_Toc509317473)

[12) Customise ngx-bootstrap datepicker 37](#_Toc509317474)

[13) Angular ngif directive 40](#_Toc509317475)

[14) Angular disable browser validation 44](#_Toc509317476)

[15) Angular form validation 45](#_Toc509317477)

[16) Displaying angular form validation error messages 50](#_Toc509317478)

[17) Model binding in angular template driven forms 53](#_Toc509317479)

[18) Angular email validation example 55](#_Toc509317480)

[19) Angular regular expression validation 57](#_Toc509317481)

[20) Angular radio button validation 59](#_Toc509317482)

[21) Add required attribute dynamically in angular 60](#_Toc509317483)

[22) Angular checkbox validation 63](#_Toc509317484)

[23) Angular select list validation 65](#_Toc509317485)

[24) angular value vs ngvalue 67](#_Toc509317486)

[25) Angular custom validator example template driven forms 68](#_Toc509317487)

[26) Angular select list required custom validator 70](#_Toc509317488)

[27) Angular password and confirm password validation 73](#_Toc509317489)

[28) Angular trigger validation manually 78](#_Toc509317490)

[29) Angular form group validation 80](#_Toc509317491)

[30) Angular 5 services tutorial 82](#_Toc509317492)

[31) Create operation in angular 86](#_Toc509317493)

[32) Angular switch case example 88](#_Toc509317494)

# Introduction

This is Part 1 of Angular CRUD tutorial. In this video and in our upcoming videos in this series we will discuss performing CRUD operations in Angular i.e Creating, Reading, Updating and Deleting in Angular with simple examples.

Setting up a new angular project from scratch is a tedious and time consuming process. However, Angular CLI makes it super fast and easy.

With angular development, we write same boiler plate code to create components, pipes, services, directives etc.

Manually creating these consumes lot of time. Angular CLI can generate these with lightning speed while still following Angular's best practices and conventions. So basic knowledge of Angular CLI is very helpful.

The prerequisites for this course are basic knowledge of Bootstrap, Angular CLI and Angular 2. If you are new to these please check out our courses using the links below. Angular CLI : [http://csharp-video-tutorials.blogspo...](https://www.youtube.com/redirect?event=video_description&v=JYPyy-hvjYc&redir_token=oPgaqmmpDt4foOO6AbWYM1yi08x8MTUyMDg5NzE2M0AxNTIwODEwNzYz&q=http%3A%2F%2Fcsharp-video-tutorials.blogspot.com%2F2017%2F10%2Fangular-cli-tutorial-for-beginners.html) Angular 2 : [http://csharp-video-tutorials.blogspo...](https://www.youtube.com/redirect?event=video_description&v=JYPyy-hvjYc&redir_token=oPgaqmmpDt4foOO6AbWYM1yi08x8MTUyMDg5NzE2M0AxNTIwODEwNzYz&q=http%3A%2F%2Fcsharp-video-tutorials.blogspot.com%2F2017%2F06%2Fangular-2-tutorial-for-beginners_12.html) Bootstrap : [http://csharp-video-tutorials.blogspo...](https://www.youtube.com/redirect?event=video_description&v=JYPyy-hvjYc&redir_token=oPgaqmmpDt4foOO6AbWYM1yi08x8MTUyMDg5NzE2M0AxNTIwODEwNzYz&q=http%3A%2F%2Fcsharp-video-tutorials.blogspot.com%2F2016%2F05%2Fbootstrap-tutorial-for-beginners.html)

Installing the tools required Node : Install the latest version of node. Here is the link to download and install the latest version [https://nodejs.org/en/download/](https://www.youtube.com/redirect?event=video_description&v=JYPyy-hvjYc&redir_token=oPgaqmmpDt4foOO6AbWYM1yi08x8MTUyMDg5NzE2M0AxNTIwODEwNzYz&q=https%3A%2F%2Fnodejs.org%2Fen%2Fdownload%2F) To check the version of node

**node -v**

Angular CLI :

Install the latest version of Angular CLI by executing the following command from the windows command prompt.

**npm install -g @angular/cli**

To verify the version of Angular CLI installed on your machine execute the following command. **ng -v**

Another question that you might have is, which version of Angular are we using with this course. As of this recording, the latest version is Angular 5, which is what we will be using.

After you have the latest version of Node and Angular CLI installed, launch windows command prompt as an administrator and execute the following command.

This creates a new AngularProject with name AngularCrud. We do not want test files to be generated for the root component AppComponent, so we have set "skip-tests" option to true.

We will discuss unit testing components in a later video.

**ng new AngularCrud --skip-tests true**

**code .**

to open in visual studio code

Once you have the Angular project opened in Visual Studio Code, open package.json file and notice that we are using Angular 5.

We will be using Bootstrap for styles in our application. So install Bootstrap by executing the following command from the command prompt.

**npm install bootstrap@3 --save**

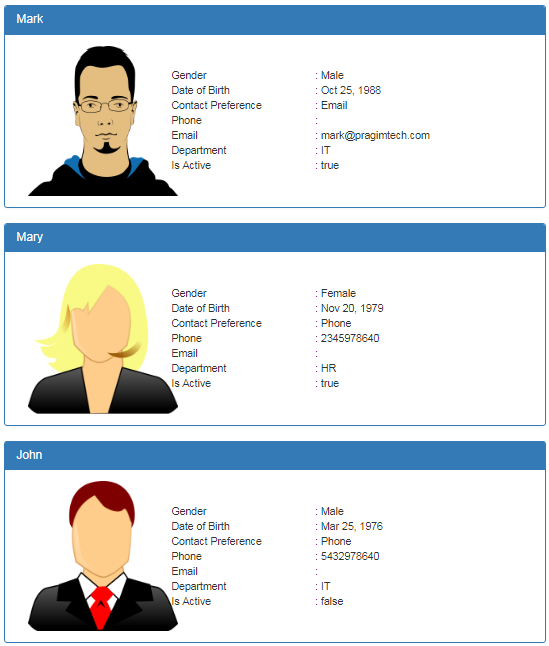
Once Bootstrap is installed, open .angular-cli.json file and specify the path to the Bootstrap stylesheet (bootstrap.min.css) in the styles property as shown below.

"styles": [ "../node\_modules/bootstrap/dist/css/bootstrap.min.css", "styles.css"]

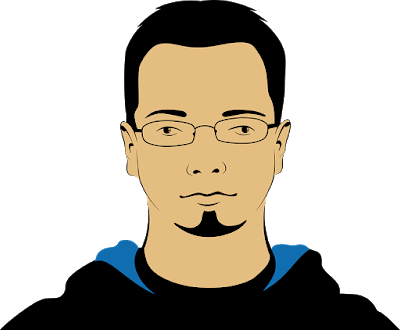
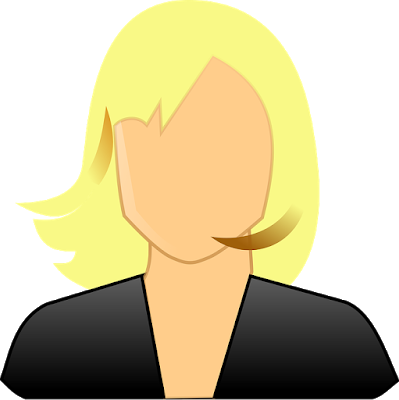
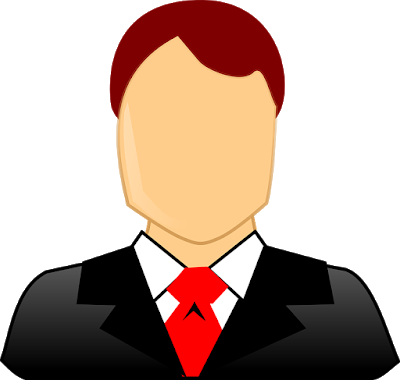
**ng serve -o**

To run and open in new browser

# Reading data in angular

**Suggested Videos**  
[Part 1 - Angular project setup](https://www.youtube.com/watch?v=JYPyy-hvjYc) | [Text](http://csharp-video-tutorials.blogspot.com/2017/12/angular-project-setup.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2017/12/angular-project-setup-slides.html)   
  
In this video we will discuss performing the **READ operation in Angular**. In our upcoming videos, we will discuss the rest of the CRUD operations i.e Creating, Updating and Deleting.   
  
  
Let us understand implementing the READ operation with an example. We want to display list of employees as shown below.   
   
  
  
At the moment, we do not have **Employee**model. First, let's create the Employee model.    
  
**Creating Employee model :**

1. In Visual Studio Code, expand the "src" folder
2. Right click on the "App" folder, and select "New Folder" from the context menu
3. Name the folder "models". We will place all our models in this folder
4. Now add a new file in the "models" folder
5. Name it "employee.model.ts"
6. Copy and paste the following code in it

export class Employee {  
    id: number;  
    name: string;  
    gender: string;  
    email?: string;  
    phoneNumber?: number;  
    contactPreference: string;  
    dateOfBirth: Date;  
    department: string;  
    isActive: boolean;  
    photoPath?: string;  
}  
  
Next, create a component to display the list of employees. Name it **ListEmployeesComponent.**  
  
**Creating ListEmployeesComponent :** Use the following AngularCLI command to create ListEmployeesComponent. We will place all employee CRUD components in "employees" folder. This is the reason we prefixed the "employees" folder name in the command. Also, notice we have set --flat option to true as we do not want to place the ListEmployeesComponent files in it's own dedicated folder.   
  
**ng g c employees/listEmployees --spec false --flat true**  
  
The above command not only creates the **ListEmployeesComponent**, it also updates the **AppModule**. In the app.module.ts file it has imported ListEmployeesComponent and included it in the declarations array. So the Angular CLI has generated lot of boiler plate code, that we would have to write manually otherwise.  
  
**Creating images folder :**We will place all the images that we are going to use in "images" folder. We will have the images folder in the "assets" folder. So add a new folder in the "assets" folder and name it "images" and copy the following 3 images. Name the images mark.png, mary.png and john.png.   
  
[](https://2.bp.blogspot.com/-xG8wtGFhwd4/WjFWoJpjSPI/AAAAAAAAns0/-UCndX6XnlEsTYS-LeZyXa2_DsgQQlkQgCLcBGAs/s1600/mark.png)   
  
[](https://2.bp.blogspot.com/-VWh7J1pgrGo/WjFW-lYK-aI/AAAAAAAAns4/cjh5i2X7vhMhboHwirOlp-QLD3lDOFXXACLcBGAs/s1600/mary.png)   
  
[](https://2.bp.blogspot.com/-L8hccI1RetU/WjFXFp717LI/AAAAAAAAns8/zmoWhYWycR0SNT_u7Fxl5e73hEYsTMVIgCLcBGAs/s1600/john.png)   
  
**Changes in list-employees.component.ts :**The changes are commented and self-explanatory

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

// import Employee Model

import { Employee } from '../models/employee.model';

@Component({

  selector: 'app-list-employees',

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

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

})

export class ListEmployeesComponent implements OnInit {

  // Hard code the employee data. In a later video we will discuss

  // how to retrieve this employees data from a database table

  employees: Employee[] = [

    {

      id: 1,

      name: 'Mark',

      gender: 'Male',

      contactPreference: 'Email',

      email: 'mark@pragimtech.com',

      dateOfBirth: new Date('10/25/1988'),

      department: 'IT',

      isActive: true,

      photoPath: 'assets/images/mark.png'

    },

    {

      id: 2,

      name: 'Mary',

      gender: 'Female',

      contactPreference: 'Phone',

      phoneNumber: 2345978640,

      dateOfBirth: new Date('11/20/1979'),

      department: 'HR',

      isActive: true,

      photoPath: 'assets/images/mary.png'

    },

    {

      id: 3,

      name: 'John',

      gender: 'Male',

      contactPreference: 'Phone',

      phoneNumber: 5432978640,

      dateOfBirth: new Date('3/25/1976'),

      department: 'IT',

      isActive: false,

      photoPath: 'assets/images/john.png'

    },

  ];

  constructor() { }

  ngOnInit() {

  }

}

**Changes in list-employees.component.html :** Replace the existing HTML, with the following HTML. Notice we are using Bootstrap for styling.

<div class="panel panel-primary" \*ngFor="let employee of employees">

  <div class="panel-heading">

    <h3 class="panel-title">{{employee.name}}</h3>

  </div>

  <div class="panel-body">

    <div class="col-xs-10">

      <div class="row vertical-align">

        <div class="col-xs-4">

          <img class="imageClass" [src]="employee.photoPath" />

        </div>

        <div class="col-xs-8">

          <div class="row">

            <div class="col-xs-6">

              Gender

            </div>

            <div class="col-xs-6">

              : {{employee.gender}}

            </div>

          </div>

          <div class="row">

            <div class="col-xs-6">

              Date of Birth

            </div>

            <div class="col-xs-6">

              : {{employee.dateOfBirth | date}}

            </div>

          </div>

          <div class="row">

            <div class="col-xs-6">

              Contact Preference

            </div>

            <div class="col-xs-6">

              : {{employee.contactPreference}}

            </div>

          </div>

          <div class="row">

            <div class="col-xs-6">

              Phone

            </div>

            <div class="col-xs-6">

              : {{employee.phoneNumber}}

            </div>

          </div>

          <div class="row">

            <div class="col-xs-6">

              Email

            </div>

            <div class="col-xs-6">

              : {{employee.email}}

            </div>

          </div>

          <div class="row">

            <div class="col-xs-6">

              Department

            </div>

            <div class="col-xs-6">

              : {{employee.department}}

            </div>

          </div>

          <div class="row">

            <div class="col-xs-6">

              Is Active

            </div>

            <div class="col-xs-6">

              : {{employee.isActive}}

            </div>

          </div>

        </div>

      </div>

    </div>

  </div>

</div>

**Changes in list-employees.component.css :** Include the following CSS classes

.imageClass{

    width:200px;

    height:200px;

}

.vertical-align{

    display: flex;

    align-items: center;

}

**Changes in app.component.html :**Include the ListEmployeesComponent selector (app-list-employees) as a directive in the root component (app.component.html)

<div class="container">

    <app-list-employees></app-list-employees>

</div>

At this point, save all the changes and run the angular project using the following command. This command not only compiles the angular application, it also launches your default browser and displays the list of employees as expected.  
ng serve -o  
  
We have just seen how to perform the READ operation. In our next video, we will set up routing.

# Angular routing and navigation

**Suggested Videos**  
[Part 1 - Angular project setup](https://www.youtube.com/watch?v=JYPyy-hvjYc) | [Text](http://csharp-video-tutorials.blogspot.com/2017/12/angular-project-setup.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2017/12/angular-project-setup-slides.html)   
[Part 2 - Reading data in angular](https://www.youtube.com/watch?v=tPySwBVmGvg) | [Text](http://csharp-video-tutorials.blogspot.com/2017/12/reading-data-in-angular.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2017/12/reading-data-in-angular-slides.html)   
  
In this video we will discuss setting up routing in our sample application. Here are the steps.   
  
  
**Step 1 :** At the moment, in our application we have only one component (ListEmployeesComponent). Let's create another component. In our upcoming videos we will discuss how to create a new employee. So let's add CreateEmployeeComponent. Use the following Angular CLI command to generate the component.   
  
ng g c employees/createEmployee --spec false --flat true 

**Step 2 :**Set <base href="/"> in the application host page which is index.html. The <base href> tells the angular router how to compose navigation URLs. This is already done for us by the Angular CLI, when we created this project.

<base href="/">

We will discuss the significance of this base href element in detail in our next video.  
  
**Step 3 :**Import the RouterModule into the application root module AppModule. The Router Module contains the Router service and Router directives such as (RouterLink, RouterLinkActive, RouterOutlet etc). So for us to be able to implement routing, we first need to import the Router Module in our AppModule. So in app.module.ts make the following changes. Notice the changes are commented and self-explanatory.

// Import RouterModule

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

// Include RouterModule in the "imports" array of the @NgModule() decorator

@NgModule({

  declarations: [...

  ],

  imports: [

    BrowserModule,

    RouterModule

  ],

  providers: [],

  bootstrap: [AppComponent]

})

export class AppModule { }

**Step 4 :**Configure the application routes.   
  
To configure routes, we first need to import Routes type from '@angular/router'. If you look at the definition of Routes type, it is actually an array of Route objects. This Routes type is not required for the application to work. However, using it provides us intellisense and compile time checking. For example, mis-spelled properties of the Route object will be reported as errors.

import { RouterModule, Routes } from '@angular/router';

// Each route maps a URL path to a component

// The 3rd route specifies the route to redirect to if the path

// is empty. In our case we are redirecting to /list

// pathMatch property value can be full or prefix. For now we

// will set it to full as we want to do a full match. In our upcoming videos,

// we will discuss the difference between prefix and full in detail.

const appRoutes: Routes = [

  { path: 'list', component: ListEmployeesComponent },

  { path: 'create', component: CreateEmployeeComponent },

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

];

// To let the router know about the routes configured above,

// pass "appRoutes" constant to forRoot(appRoutes) method

// We also have forChild() method. We will discuss the difference

// and when to use one over the other in our upcoming videos

@NgModule({

declarations: [...

],

imports: [

  BrowserModule,

  RouterModule.forRoot(appRoutes)

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

**Step 5 :**Create the application menu and tie the routes to it. Notice we are using routerLink directive. This directive tells the router where to navigate when the user clicks the link. We are also using router-outlet directive. This directive specifies where you want the routed component view template to be displayed. We want our navigation menu to be always displayed, so the ideal location for it is the root component AppComponent i.e app.component.html file. When the application first loads, it loads the root AppComponent in index.html. So let's place the following HTML in app.component.html file.

<div class="container">

    <nav class="navbar navbar-default">

        <ul class="nav navbar-nav">

            <li>

                <a routerLink="list">List</a>

            </li>

            <li>

                <a routerLink="create">Create</a>

            </li>

        </ul>

    </nav>

    <router-outlet></router-outlet>

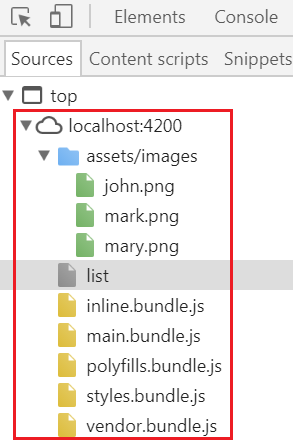
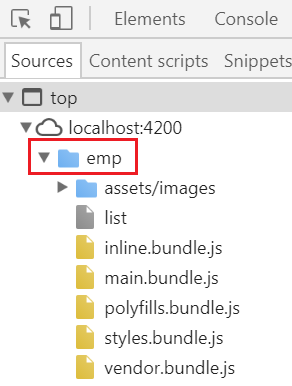
</div>

As you can see, in the HTML we have 2 things - Navigation menu and the <router-outlet> directive. The navigation menu is always displayed. Below the navigation menu, we have the <router-outlet> directive. This is the location where the routed component view template is displayed.   
  
For example, when we click on the "List" link in the navigation menu, the route changes to "/list" and the ListEmployeesComponent view template is displayed at the location where we have the <router-outlet> directive. At this point, if we click on "Create" link, 2 things happen  
  
1. The route changes from "/list" to "/create"   
2. ListEmployeesComponent view template is replaced with CreateEmployeeComponent view template  
  
Also notice, if we navigate to the root of the application i.e if we do not include "/list" or "/create" in the URL, the router automatically redirects us to "/list". This is because of the following empty route we have in our route configuration.

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

We are using [Bootstrap navbar component](https://www.youtube.com/watch?v=KRsSSpMaaKg&list=PL6n9fhu94yhXd4xnk-j5FGhHjUv1LsF0V&index=28)to create the navigation menu. We discussed Bootstrap navbar component in [Part 28](https://www.youtube.com/watch?v=KRsSSpMaaKg&list=PL6n9fhu94yhXd4xnk-j5FGhHjUv1LsF0V&index=28) of [Bootstrap tutorial](https://www.youtube.com/watch?v=314m7YBRFvQ&list=PL6n9fhu94yhXd4xnk-j5FGhHjUv1LsF0V).  
  
Since we are now routing to ListEmployeesComponent we no longer need it's selector. So remove the selector specified in the @Component decorator of ListEmployeesComponent  
  
At the moment both the menu items, i.e the active and the inactive menu items are styled the same way. As a result we don't know which page the user is on. We will discuss how to address this in our upcoming videos.  
  
In our next video, we will discuss the significance of the <base href="/"> element in index.html page.

# Angular base href

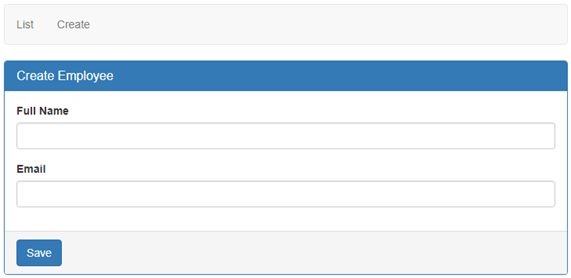
**Suggested Videos**  
[Part 1 - Angular project setup](https://www.youtube.com/watch?v=JYPyy-hvjYc) | [Text](http://csharp-video-tutorials.blogspot.com/2017/12/angular-project-setup.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2017/12/angular-project-setup-slides.html)   
[Part 2 - Reading data in angular](https://www.youtube.com/watch?v=tPySwBVmGvg) | [Text](http://csharp-video-tutorials.blogspot.com/2017/12/reading-data-in-angular.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2017/12/reading-data-in-angular-slides.html)   
[Part 3 - Angular routing and navigation](https://www.youtube.com/watch?v=pcOaAU_iaD4) | [Text](http://csharp-video-tutorials.blogspot.com/2017/12/angular-routing-and-navigation.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2017/12/angular-routing-and-navigation-slides.html)  
  
In this short video we will discuss the **significance of the base href element in Angular**.   
  
  
When setting up routing in an angular application, the first step is to set the base path using the base href element. The base path tells the angular router, how to compose the navigation URLs. The browser uses the <base href> value to prefix relative URLs when referencing CSS files, scripts, and images.   
  
  
During development we usually set this to a single forward slash as shown below.   
<base href="/">  
  
This means all the URLs now will be relative to the root of the application. So when we navaigate to "/list", the path "/list" will be appended to root UR and the complete URL will be as shown below. Notice "/list" is relative to the root URL.  
http://localhost:4200/list  
  
Along the same lines, when we navigate to "/create", the complete URL is http://localhost:4200/create  
  
When we deploy our application to a server, we typically deploy it to a sub folder on the server. For example, if we are deploying our application in a sub-folder called **"emp"**, then we set the base href element to /emp/ as shown below.  
<base href="/emp/">  
  
This means all the URLs now will be relative to the "emp" base path and will be as shown below.  
http://serverName/emp/list  
http://serverName/emp/create  
  
During development we usually set base href element to a single forward slash as shown below.   
<base href="/">  
  
At this point, if we execute the following command, all the URLs will be relative to the root URL "http://localhost:4200"  
ng serve -o   
  
Also, on the **"sources"** tab in the browser developer tools, you will find all the Script,  Images and Template files are relative to the root URL **"http://localhost:4200"** as shown in the image below.   
   
  
During development, if you want a different base path other than "/", simply execute the "ng serve" command with --base-href option set to your desired base path as shown below.  
ng serve -o --base-href /emp/  
  
At this point all the URLs will be relative to "http://localhost:4200/emp" as we have set the --base-href to /emp/. You can confirm this by looking at the URLs in the address bar and the "Sources" tab in the browser developer tools.   
   
  
On your local development machine, if you set the base href element in index.html to "/emp/" instead of a single "/" and if you run ng serve -o command without the "base-href" option  you will not see anything on the browser. When you open the browser developer tools, you will see the JavaScript bundle files failed to load. To fix this execute ng serve command along with the base href option as shown below.  
ng serve -o --base-href /emp/   
  
On your local development machine, if you set the base href element in index.html to a single forward slash and if you want to deploy your application on a server on sub-folder called "emp", then you will have to remember to update the base href element value in index.html to "/emp/". There are 2 ways we can do this.

1. Manually update the index.html file OR
2. Use the --base-href option along with the ng build command as shown below. This will update the "base href" element value index.html.ng build --base-href /emp/

# Angular forms tutorial

**Suggested Videos**  
[Part 2 - Reading data in angular](https://www.youtube.com/watch?v=tPySwBVmGvg) | [Text](http://csharp-video-tutorials.blogspot.com/2017/12/reading-data-in-angular.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2017/12/reading-data-in-angular-slides.html)  
[Part 3 - Angular routing and navigation](https://www.youtube.com/watch?v=pcOaAU_iaD4) | [Text](http://csharp-video-tutorials.blogspot.com/2017/12/angular-routing-and-navigation.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2017/12/angular-routing-and-navigation-slides.html)   
[Part 4 - Angular base href](https://www.youtube.com/watch?v=Std1QJpMEiE) | [Text](http://csharp-video-tutorials.blogspot.com/2017/12/angular-base-href.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2017/12/angular-base-href-slides.html)  
  
In [Part 2](https://www.youtube.com/watch?v=tPySwBVmGvg) of [Angular CRUD tutorial](https://www.youtube.com/watch?v=JYPyy-hvjYc&list=PL6n9fhu94yhXwcl3a6rIfAI7QmGYIkfK5), we discussed performing the READ operation. In this video and in the next few videos we will discuss performing the CREATE operation. To understand the CREATE operation, let us build a form that help us create a new employee. For this we will use the **createEmployee**component that we already created in one of our previous videos in this series. Along the way, we will also discuss performing validation and displaying meaningful error messages to the user.   
  
  
**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.   
  
In this video, we will use the Template driven approach to build the **"Create Employee"**form. As the name implies, template driven forms are heavy on the template. This means we do most of the work in the view template of the component.   
  
We want to design our "Create Employee" form as shown below. To keep this simple, at the moment we only have 2 fields (Full Name & Email). We will add the other fields like Gender, Department, Phone Number etc.. later. Also, at the moment, we only have textboxes on our form. In our upcoming videos we will discuss working with radio buttons, checkbox, dropdownlist etc   
   
  
Replace the HTML in "create-employee.component.html" file with the following HTML

<form #employeeForm="ngForm" (ngSubmit)="saveEmployee(employeeForm)">

  <div class="panel panel-primary">

    <div class="panel-heading">

      <h3 class="panel-title">Create Employee</h3>

    </div>

    <div class="panel-body">

      <div class="form-group">

        <label for="fullName">Full Name</label>

        <input id="fullName" type="text" class="form-control"

               name="fullName" [(ngModel)]="fullName">

      </div>

      <div class="form-group">

        <label for="email">Email</label>

        <input id="email" type="text" class="form-control"

               name="email" [(ngModel)]="email">

      </div>

    </div>

    <div class="panel-footer">

      <button class="btn btn-primary" type="submit">Save</button>

    </div>

  </div>

</form>

Angular Generated Form Model : {{employeeForm.value | json}}   
  
**Code Explanation:**  
We are using Bootstrap CSS classes like panel, panel-primary, panel-heading, panel-title etc to style the form. There is no Angular here. If you are new to bootstrap, [please click here to check out our Bootstrap tutorial](http://csharp-video-tutorials.blogspot.com/2016/05/bootstrap-tutorial-for-beginners.html).  
  
**Consider the following line of code**  
<form #employeeForm="ngForm" (ngSubmit)="saveEmployee(employeeForm)">  
  
**#employeeForm**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 form also has all these properties. We will discuss these properties at the individual control level and at the form level in detail in our upcoming videos.  
  
**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. We do not have this method yet. We will create it in the component class in just a bit.  
  
**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 toolsthere is no directive with exportas set to ngform  
  
**Consider the following block of code**

<div class="form-group">

  <label for="fullName">Full Name</label>

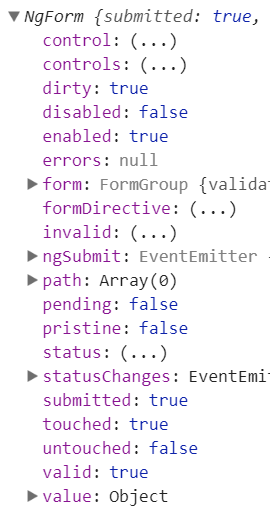
  <input id="fullName" type="text" class="form-control"

          name="fullName" [(ngModel)]="fullName">

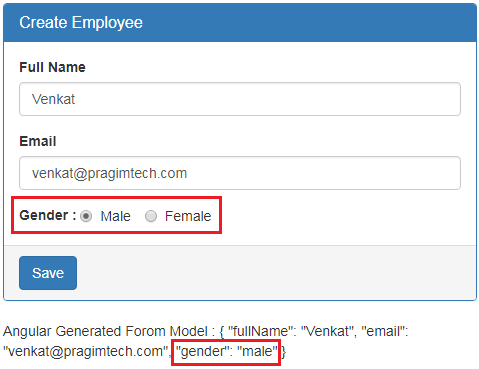
</div>

1. To style the "Full Name" field and it's associated label, we are using Bootstrap. So "form-group" and "form-control" are Bootstrap CSS classes used for styling. There is no Angular here.
2. The "for" attribute on the label, is used to link the label with it's associated "fullName" input control. With the "for" attribute in place, when we click on the label, it's associated input element automatically receives the focus. Again there is no Angular here. It's all standard HTML.
3. The **ngModel**directive is used for creating two-way data binding i.e to keep the HTML element value and it's corresponding component property in sync. We discussed two-way data binding in detail in our [Angular 2 course](https://www.youtube.com/watch?v=WWQZCDegWHg&list=PL6n9fhu94yhWqGD8BuKuX-VTKqlNBj-m6). [Click here to watch two-way data binding video.](https://www.youtube.com/watch?v=aBf1nLGuVz8&index=15&list=PL6n9fhu94yhWqGD8BuKuX-VTKqlNBj-m6)
4. Notice we have set **ngModel** directive to "fullName". We do not have "fullName" property in 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
5. 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.

**Consider the following piece of code :**We are using the value property of the employeeForm to display fullName and email property values of the Form Model that angular automatically generates for us. We are using the Angular "json" pipe to format the JSON data.

**Angular Generated Forom Model : {{employeeForm.value | json}}**  
  
Finally in the CreateEmployeeComponent class include the following saveEmployee() method. At the moment we are simply logging the value of the Angular generated Form model to the console. In our upcoming videos, we will discuss how to save the new employee to a database table.   
  
saveEmployee(employeeForm: NgForm): void {  
  console.log(employeeForm.value);  
}  
  
**Please note :**Make sure to import **NgForm**type from '@angular/forms'  
import { NgForm } from '@angular/forms';  
  
Remember we discussed, The **ngForm**directive supplements the form element with additional features and properties like value, dirty, touched, valid etc. To see all these properties, knock of the value property and log just the employeeForm as shown below.  
  
saveEmployee(employeeForm: NgForm): void {  
  console.log(employeeForm);  
}  
  
At this point, if you fill in the Full Name and Email text boxes and when you submit the form either by click the "Save" button or by pressing the "Enter" key you will see the form logged to the browser console and you can see all these properties.   
   
  
These properties are greatly useful for performing form validation. We will discuss them in detail in our upcoming videos.

# Bootstrap radio buttons in Angular

**Suggested Videos**  
[Part 3 - Angular routing and navigation](https://www.youtube.com/watch?v=pcOaAU_iaD4) | [Text](http://csharp-video-tutorials.blogspot.com/2017/12/angular-routing-and-navigation.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2017/12/angular-routing-and-navigation-slides.html)  
[Part 4 - Angular base href](https://www.youtube.com/watch?v=Std1QJpMEiE) | [Text](http://csharp-video-tutorials.blogspot.com/2017/12/angular-base-href.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2017/12/angular-base-href-slides.html)   
[Part 5 - Angular forms tutorial](https://www.youtube.com/watch?v=pwQ3L3UFEjk) | [Text](http://csharp-video-tutorials.blogspot.com/2017/12/angular-forms-tutorial.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2017/12/angular-forms-tutorial-slides.html)  
  
In this video we will discuss **working with radio buttons in Angular Template Driven forms**.   
  
  
We want to include **"Gender"** radio buttons in the Create Employee form as shown below. When we select employee "Gender" using the radio buttons, the selected gender value should reflect in the Angular generated form model as shown in the image below. Also, we we click the **"Save"** button we want the selected gender value to be logged to the console.   
   
  
  
To achieve this all you have to do is include the following HTML in **create-employee.component.html** file

<div class="form-group">

  <label>Gender</label>

  <div class="form-control">

    <label class="radio-inline">

      <input type="radio" name="gender" value="male" [(ngModel)]="gender">

      Male

    </label>

    <label class="radio-inline">

      <input type="radio" name="gender" value="female" [(ngModel)]="gender">

      Female

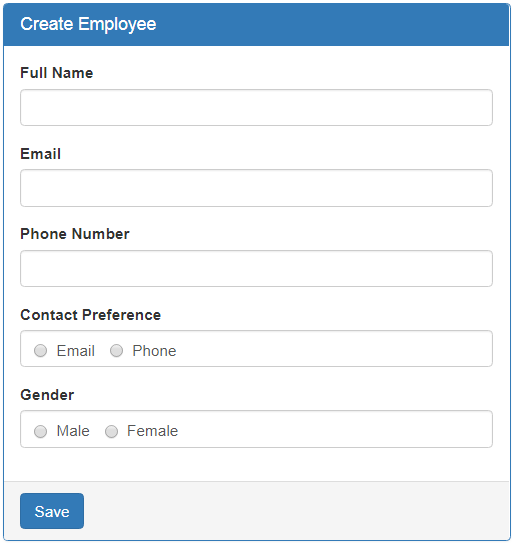
    </label>

  </div>

</div>

**Code Explanation**

* The **name**attribute is required to group the radio buttons as one unit and make the selection mutually exclusive. Make sure both the radio buttons have the same value for the **"name"** attribute. Otherwise the radio button selection won't be mutually exclusive.
* It is also important that you set the **"value"** attribute for each radio button. This value is posted to the server when the form is submitted.

While we are here, let's also include a textbox to capture **"Phone Number"** and **"Contact Preference"** radio button. So the form should now look as as shown below.   
   
  
For your referece, here is the complete HTML in create-employee.component.html

<form #employeeForm="ngForm" (ngSubmit)="saveEmployee(employeeForm)">

  <div class="panel panel-primary">

    <div class="panel-heading">

      <h3 class="panel-title">Create Employee</h3>

    </div>

    <div class="panel-body">

      <div class="form-group">

        <label for="fullName">Full Name</label>

        <input id="fullName" type="text" class="form-control" name="fullName"

        [(ngModel)]="fullName">

      </div>

      <div class="form-group">

        <label for="email">Email</label>

        <input id="email" type="text" class="form-control" name="email"

        [(ngModel)]="email">

      </div>

      <div class="form-group">

        <label for="phoneNumber">Phone Number</label>

        <input id="phoneNumber" type="text" class="form-control" name="phoneNumber"

        [(ngModel)]="phoneNumber">

      </div>

      <div class="form-group">

        <label>Contact Preference</label>

        <div class="form-control">

          <label class="radio-inline">

            <input type="radio" name="contactPreference" value="email"

            [(ngModel)]="contactPreference">

            Email

          </label>

          <label class="radio-inline">

            <input type="radio" name="contactPreference" value="phone"

            [(ngModel)]="contactPreference">

            Phone

          </label>

        </div>

      </div>

      <div class="form-group">

        <label>Gender</label>

        <div class="form-control">

          <label class="radio-inline">

            <input type="radio" name="gender" value="male" [(ngModel)]="gender">

            Male

          </label>

          <label class="radio-inline">

            <input type="radio" name="gender" value="female" [(ngModel)]="gender">

            Female

          </label>

        </div>

      </div>

    </div>

    <div class="panel-footer">

      <button class="btn btn-primary" type="submit">Save</button>

    </div>

  </div>

</form>

Angular Generated Forom Model : {{employeeForm.value | json}}

# Angular radio button checked by default

**Suggested Videos**  
[Part 4 - Angular base href](https://www.youtube.com/watch?v=Std1QJpMEiE) | [Text](http://csharp-video-tutorials.blogspot.com/2017/12/angular-base-href.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2017/12/angular-base-href-slides.html)  
[Part 5 - Angular forms tutorial](https://www.youtube.com/watch?v=pwQ3L3UFEjk) | [Text](http://csharp-video-tutorials.blogspot.com/2017/12/angular-forms-tutorial.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2017/12/angular-forms-tutorial-slides.html)   
[Part 6 - Bootstrap radio buttons in Angular](https://www.youtube.com/watch?v=IjEWmoOHHvM) | [Text](http://csharp-video-tutorials.blogspot.com/2017/12/bootstrap-radio-buttons-in-angular.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2017/12/bootstrap-radio-buttons-in-angular_28.html)  
  
**In this video we will discuss**

* How to have a radio button checked by default
* How to disable a radio button

**How to get a radio button checked by default in Angular :**Consider the following HTML, that displays "Gender" radio buttons

<div class="form-group">

  <label>Gender</label>

  <div class="form-control">

    <label class="radio-inline">

      <input type="radio" name="gender" value="male" [(ngModel)]="gender" >

      Male

    </label>

    <label class="radio-inline">

      <input type="radio" name="gender" value="female" [(ngModel)]="gender">

      Female

    </label>

  </div>

</div>

If we include **checked**attribute on one of the radio buttons, we expect that radio button to be checked by default when the form initially loads. But you will discover that is not the case. In the following example, we have included "checked" attribute on "Male" radio button, but when the form is displayed it is not checked.

<input type="radio" name="gender" value="male" [(ngModel)]="gender" checked>

However, if you remove the "ngModel" directive from the radio button, then it gets checked as expected. Notice the "ngModel" directive is removed from the radio button.   
  
<input type="radio" name="gender" value="male" checked>  
  
With Angular Template Driven forms, we use "ngModel" directive for two-way data binding. So the moment we put it back in place the "checked" attribute does not work. To make it work include **"gender"** property in the component class and initialise to the value of the radio button that you want to have checked by default. In our case, let us say, we want the "Male" radio button to be checked by default. To achieve this include "gender" property initialised to value of "male" in the component class as shown below.  
  
gender = 'male';  
  
At this point you will have "Male" radio button checked by default when the form loads. Now, even if we remove the "**checked**" attribute from the "Male" radio button 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. For our form we do not want any radio button to be checked by default, so remove the "checked" attribute and the "gender" property from the component class.  
  
How to disable a radio button : To disable a radio button, use the disabled attribute on that radio button. "Male" radio button in this case will be disabled when the form initially loads.

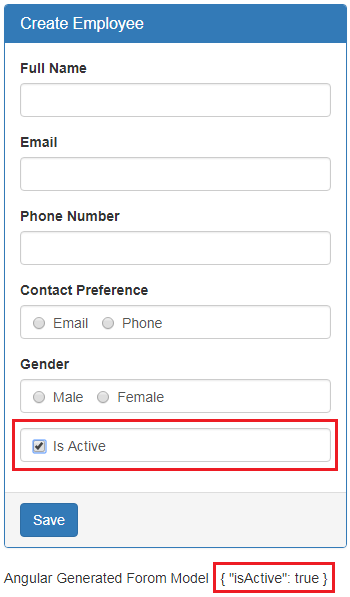
<input type="radio" name="gender" value="male" [(ngModel)]="gender" disabled>

Another important point to keep in mind. By default, disabled form controls are not included in the Angular auto generated form model. Since, the "Male" radio button is disabled, the gender property will not be included in the Angular generated form model.  
  
In our form, we do not want any radio button to be disabled, so please remove the disabled attribute.  
  
In our next video, we will discuss working with **CheckBox control in Angular Template Driven forms**.

# Bootstrap checkbox in angular

**Suggested Videos**  
[Part 5 - Angular forms tutorial](https://www.youtube.com/watch?v=pwQ3L3UFEjk) | [Text](http://csharp-video-tutorials.blogspot.com/2017/12/angular-forms-tutorial.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2017/12/angular-forms-tutorial-slides.html)  
[Part 6 - Bootstrap radio buttons in Angular](https://www.youtube.com/watch?v=IjEWmoOHHvM) | [Text](http://csharp-video-tutorials.blogspot.com/2017/12/bootstrap-radio-buttons-in-angular.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2017/12/bootstrap-radio-buttons-in-angular_28.html)  
[Part 7 - Angular radio button checked by default](https://www.youtube.com/watch?v=er8FD9V3FEU) | [Text](http://csharp-video-tutorials.blogspot.com/2018/01/angular-radio-button-checked-by-default.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/01/angular-radio-button-checked-by-default_2.html)  
  
**In this video we will discuss**

1. Working with a checkbox control in Angular Template Driven forms
2. How to have a checkbox checked by default
3. How to disable a checkbox

**Working with a checkbox in Angular is very similar to working with a radio button**. We want to include "Is Active" checkbox in the **Create Employee**form as shown below. When we check the checkbox, "isActive" property should reflect in the Angular generated for model as shown in the image below. Also, when we click the "Save" button we want the "isActive" property value to be logged to the console.   
   
  
  
To achieve this all you have to do is include the following HTML in **create-employee.component.html**file

<div class="form-group">

  <div class="form-control">

    <label class="checkbox-inline">

      <input type="checkbox" name="isActive" [(ngModel)]="isActive">Is Active

    </label>

  </div>

</div>

If we include **"checked"** attribute on a checkbox, we expect checkbox to be checked by default when the form initially loads. But you will discover that is not the case.

<input type="checkbox" name="isActive" [(ngModel)]="isActive" checked>Is Active

However, if you remove the **"ngModel"** directive from the checbox, then it gets checked as expected. Notice the "ngModel" directive is removed from the checkbox.   
  
<input type="checkbox" name="isActive" checked>Is Active  
  
With Angular Template Driven forms, we use **"ngModel"** directive for two-way data binding. So the moment we put it back in place the **"checked"** attribute does not work. To make it work include **"isActive"** property in the component class and initialise it to true.  
  
isActive = true;   
  
At this point you will have **"Is Active"** 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. For our form we do not want the checkbox to be checked by default, so remove the "checked" attribute and the "isActive" property from the component class.  
  
How to disable a checkbox : To disable a checkbox, use the disabled attribute

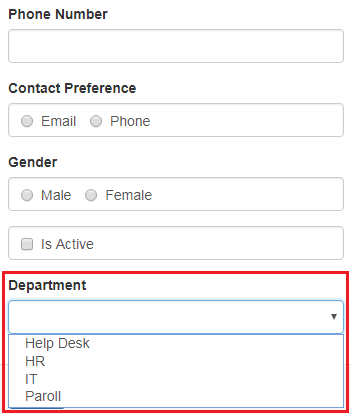
<input type="checkbox" name="isActive" [(ngModel)]="isActive" disabled>Is Active

Another important point to keep in mind. By default, disabled form controls are not included in the Angular auto generated form model. Since, the "Is Active" checkbox is disabled, it will not be included in the Angular generated form model.  
  
In our form, we do not want the checkbox to be disabled, so please remove the **disabled**attribute.

# Angular bootstrap select list

**Suggested Videos**  
[Part 6 - Bootstrap radio buttons in Angular](https://www.youtube.com/watch?v=IjEWmoOHHvM) | [Text](http://csharp-video-tutorials.blogspot.com/2017/12/bootstrap-radio-buttons-in-angular.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2017/12/bootstrap-radio-buttons-in-angular_28.html)  
[Part 7 - Angular radio button checked by default](https://www.youtube.com/watch?v=er8FD9V3FEU) | [Text](http://csharp-video-tutorials.blogspot.com/2018/01/angular-radio-button-checked-by-default.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/01/angular-radio-button-checked-by-default_2.html)   
[Part 8 - Bootstrap checkbox in angular](https://www.youtube.com/watch?v=3qfzuZBoThI) | [Text](http://csharp-video-tutorials.blogspot.com/2018/01/bootstrap-checkbox-in-angular.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/01/bootstrap-checkbox-in-angular-slides.html)  
  
**In this video we will discuss**

* Working with a select list in Angular Template Driven forms
* How to have one of the select list option selected by default
* How to disable select list

Let us **understand working with a select element in Angular with a simple example**. We want to include "Department" select list as shown in the image below.   
   
  
  
**Here is the HTML for the "Department" select list**

<div class="form-group">

  <label for="department">Department</label>

  <select id="department" name="department"

          [(ngModel)]="department" class="form-control">

    <option value="1">Help Desk</option>

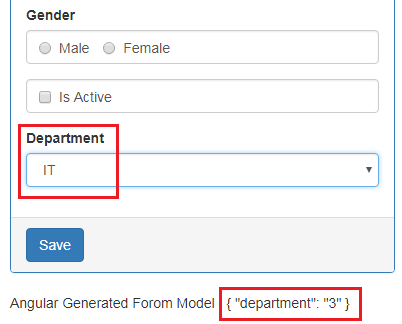
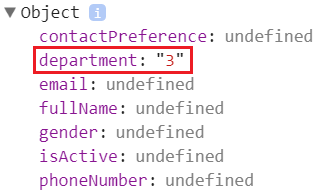
    <option value="2">HR</option>

    <option value="3">IT</option>

    <option value="4">Paroll</option>

  </select>

</div>

At the moment, we have hard coded the select list options in the HTML. In our next video we will discuss, how to get the select list options from the component class. Notice each option also has a corresponding value. The value is the department id which is what we want to save in the database table when the form is submitted. We will discuss, saving the data to a database table in a later video.  
  
At this point, when we select an option, notice the corresponding option value is included against the **"department"**property in the **Angular auto-generated form model**.   
   
  
Also notice, when we click the **"Save"** button, the **"department"**property along with the selected option value is logged to the console in browser developer tools.   
   
  
**How to have one of the select list option selected by default**  
  
If we include **"selected"** attribute on one of the options of the select list, we expect that option to be selected by default when the form initially loads. In the example below, we have included the "selected" attribute on the **"IT"** option, but when the form reloads, the **"IT"**option is not selected.  
  
<option value="3" selected>IT</option>  
  
If you remove the **"ngModel"** directive from the select list, then the the **"IT"** option gets selected as expected. Notice the **"ngModel"** directive is removed from the select list.

<div class="form-group">

  <label for="department">Department</label>

  <select id="department" name="department" class="form-control">

    <option value="1">Help Desk</option>

    <option value="2">HR</option>

    <option value="3" selected>IT</option>

    <option value="4">Paroll</option>

  </select>

</div>

In Angular, we use **"ngModel"** directive for two-way data binding. So the moment we put it back in place the **"selected"** attribute does not work. To make it work include **"department"** property in the component class and initialise it with one of the option value which you want to have selected by default. In our case, we want the **"IT"** option to be selected by default. The **"IT"** option value is **"3"**. So, I have initialised **"department"** property with a value of **'3'**  
  
department = '3'  
  
At this point you will have **"IT"** option selected by default when the form loads. Now, even if we remove the **"selected"** attribute from the "IT" option, it is still selected by default when the form loads. This is because of the two-way data binding that we get with **"ngModel"** directive.   
  
**How to disable a select list :**To disable a select element, use the **disabled**attribute

<select id="department" name="department" [(ngModel)]="department"

        class="form-control" disabled>

Another important point to keep in mind. By default, d**isabled form controls are not included in the Angular auto generated form model**. Since, the "department" select element is disabled, it will not be included in the Angular generated form model.   
  
In our form, we do not want the select element to be disabled, so please remove the **disabled**attribute. Also, we do not want any option to be selected by default, so remove the **"department"** property from the component class.  
  
In our next video, we will discuss, **how to get the select list options from the component class**, instead of having them **hard-coded in the HTML.**

# Angular select options from array

**Suggested Videos**  
[Part 7 - Angular radio button checked by default](https://www.youtube.com/watch?v=er8FD9V3FEU) | [Text](http://csharp-video-tutorials.blogspot.com/2018/01/angular-radio-button-checked-by-default.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/01/angular-radio-button-checked-by-default_2.html)   
[Part 8 - Bootstrap checkbox in angular](https://www.youtube.com/watch?v=3qfzuZBoThI) | [Text](http://csharp-video-tutorials.blogspot.com/2018/01/bootstrap-checkbox-in-angular.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/01/bootstrap-checkbox-in-angular-slides.html)   
[Part 9 - Angular bootstrap select list](https://www.youtube.com/watch?v=HMK4P_jx0y8) | [Text](http://csharp-video-tutorials.blogspot.com/2018/01/angular-bootstrap-select-list.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/01/angular-bootstrap-select-list-slides.html)   
  
In this video, we will discuss, how to get the select list options from an array in the component class, instead of having them hard-coded in the HTML. This is continuation to [Part 9](https://www.youtube.com/watch?v=HMK4P_jx0y8&index=9&list=PL6n9fhu94yhXwcl3a6rIfAI7QmGYIkfK5). So, please watch [Part 9](https://www.youtube.com/watch?v=HMK4P_jx0y8&index=9&list=PL6n9fhu94yhXwcl3a6rIfAI7QmGYIkfK5) from [Angular CRUD tutorial](https://www.youtube.com/playlist?list=PL6n9fhu94yhXwcl3a6rIfAI7QmGYIkfK5) before proceeding.   
  
  
**Step 1 :** Create the **Department**class.    
  
Add a TypeScript file to the models folder. Name it department.model.ts. Copy and paste the following code. Notice the Department class has 2 properties - id and name of the department.   
  
export class Department {  
    id: number;  
    name: string;  
}   
  
  
**Step 2 :** Import the Department class    
  
Include the following import statement in create-employee.component.ts file  
import { Department } from '../models/department.model';  
  
**Step 3 :**Include the following array of departments in CreateEmployeeComponent class in create-employee.component.ts file  
  
departments: Department[] = [  
  { id: 1, name: 'Help Desk' },  
  { id: 2, name: 'HR' },  
  { id: 3, name: 'IT' },  
  { id: 4, name: 'Payroll' }  
];  
  
**Please note :**The "Department" type is not required for the application to work, but it adds great value during development. Using it provides us intellisense, error checking and type saftey.   
  
**Step 4 :**In create-employee.component.html file, modify the HTML that displays the "Department" dropdownlist as shown below.

<div class="form-group">

  <label for="department">Department</label>

  <select id="department" name="department" [(ngModel)]="department"

          class="form-control">

    <option \*ngFor="let dept of departments" [value]="dept.id">

      {{dept.name}}

    </option>

  </select>

</div>

**Code explanation :**

* On the "option" element we are using ngFor structural directive to loop over the array of departments we have in the "departments" property of the component class
* For each "Department" object in the "departments" array, we get an option.
* The option value is the **department id** and the display text is the **department name**
* Notice the square brackets around the [value] property. This is property binding in Angular. We discussed property binding in detail in [Part 9](https://www.youtube.com/watch?v=RGYfTx9AAQA&list=PL6n9fhu94yhWqGD8BuKuX-VTKqlNBj-m6&index=9) of [Angular 2 tutorial](https://www.youtube.com/playlist?list=PL6n9fhu94yhWqGD8BuKuX-VTKqlNBj-m6). If you remove the square brackets the value for each option will be the literal text "dept.id" instead of the department id (1 or 2 or 3 etc.)
* To display the deprtment name we are using **interpolation**. We discussed interpolation in [Part 8](https://www.youtube.com/watch?v=2FkkAqFWRk8&list=PL6n9fhu94yhWqGD8BuKuX-VTKqlNBj-m6&index=8) of [Angular 2 tutorial](https://www.youtube.com/playlist?list=PL6n9fhu94yhWqGD8BuKuX-VTKqlNBj-m6).
* 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.

At this point, when we select a department, the respective department id is included in the Angular generated form model. Along the same lines, when we click the "Save" button the respective department id is logged to the console.  
  
**Please note :**It is important that we include the ngFor directive on the element that we want to be repeated. In our case we want an option element for each department we have in the array. So we included the **ngFor**directive on the option element. If we instead include the **ngFor**directive on the "div" element that has the bootstrap "form-group" class as shown below.

<div class="form-group" \*ngFor="let dept of departments">

  <label for="department">Department</label>

  <select id="department" name="department" [(ngModel)]="department"

          class="form-control">

    <option [value]="dept.id">

      {{dept.name}}

    </option>

  </select>

</div>

We get 4 department dropdownlists. That is one for each department in the array. So it is important we include the ngFor directive on the right element.   


# Angular datepicker tutorial

**Suggested Videos**  
[Part 8 - Bootstrap checkbox in angular](https://www.youtube.com/watch?v=3qfzuZBoThI) | [Text](http://csharp-video-tutorials.blogspot.com/2018/01/bootstrap-checkbox-in-angular.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/01/bootstrap-checkbox-in-angular-slides.html)  
[Part 9 - Angular bootstrap select list](https://www.youtube.com/watch?v=HMK4P_jx0y8) | [Text](http://csharp-video-tutorials.blogspot.com/2018/01/angular-bootstrap-select-list.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/01/angular-bootstrap-select-list-slides.html)   
[Part 10 - Angular select options from array](https://www.youtube.com/watch?v=4fKa13TTn7E) | [Text](http://csharp-video-tutorials.blogspot.com/2018/01/angular-select-options-from-array.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/01/angular-select-options-from-array-slides.html)  
  
**In this video we will discuss**

* Why is not a good practice to use the browser built-in DatePicker control
* Installing ngx-bootstrap
* Using ngx-bootstrap datepicker in Angular

**Why is not a good practice to use the browser built-in DatePicker control :**This is because the implementation of datepicker is different from browser vendor to vendor. This means our end users may have different experience depending on the browser they use. Let us understand this with an example.   
  
  
On our **"Create Employee"** form we want to capture **Date of Birth** of an employee. **Datepicker**control is very useful in capturing dates from users. When we use the HTML5 input type date, the browser automatically displays it's built-in datepicker control. Include the following piece of HTML on "create-employee.component.html" file just below the "Department" field HTML

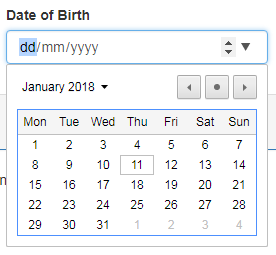
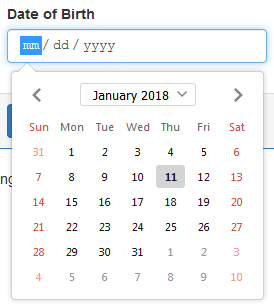
<div class="form-group">

  <label for="dateOfBirth">Date of Birth</label>

  <input id="dateOfBirth" name="dateOfBirth" [(ngModel)]="dateOfBirth"

          type="date" class="form-control" />

</div>

Notice we have set the input element type to date. At this point if we run the project and navigate to http://localhost:4200/create in Google chrome, we see the date-picker as shown below.   
   
  
Now, if we navigate to the same url in firefox, we see a date-picker control that is very different from the date-picker control that is on Google chrome browser.   
   
  
So, this means our end users have different experience depending on the browser they use. What we want here is consistency. There are many third party Date-picker controls that we can use, to provide consistent experience to our end users. ngx-bootstrap datepicker control is one of them.   
  
Please refer to the UI components section on the following page, to see the list of all third party UI components that we can use in Angular  
<https://angular.io/resources>  
  
**Installing ngx-bootstrap :**The following are the steps to install ngx-bootstrap   
  
**Step 1 :** Execute the following command to npm install ngx-bootstrap  
npm install ngx-bootstrap --save  
  
**Step 2 :**If you do not have Bootstrap installed, please install it using the following npm command. If you are following along we have already installed bootstrap in Part 1 of this Angular CRUD tutorial. So I am not going to execute this command again.  
npm install bootstrap@3 --save   
  
**Please note :**We are usng Bootstrap 3. We can also use Bootstrap 4 with ngx-bootstrap. Please refer to the documentation available at the following link on how to use Bootstrap 4 with ngx-bootstrap.  
<https://valor-software.com/ngx-bootstrap/#/getting-started>  
  
**Step 3 :**Once Bootstrap is installed, open .angular-cli.json file and specify the path to the Bootstrap stylesheet (bootstrap.min.css) in the styles property as shown below. Again, we have already done this in [Part 1](https://www.youtube.com/watch?v=JYPyy-hvjYc) of [Angular CRUD tutorial](https://www.youtube.com/watch?v=JYPyy-hvjYc&list=PL6n9fhu94yhXwcl3a6rIfAI7QmGYIkfK5).

"styles": [

  "../node\_modules/bootstrap/dist/css/bootstrap.min.css",

  "styles.css"

]

**Using ngx-bootstrap datepicker in Angular :** The following are the steps to use ngx-bootstrap datepicker in Angular  
  
**Step 1 :**In app.module.ts file, include the following import statement to import BsDatepickerModule   
import { BsDatepickerModule } from 'ngx-bootstrap/datepicker';  
  
Also, include BsDatepickerModule in the imports array of @NgModule decorator as shown below  
@NgModule({  
  imports: [BsDatepickerModule.forRoot(),...]  
})  
  
**Step 2 :**In "create-employee.component.html" file, make the following 2 changes to the HTML that displays the "Date of Birth" field

* Change the "type" attribute value from "date" to "text"
* Include "bsDatepicker" directive on the input element

<div class="form-group">

  <label for="dateOfBirth">Date of Birth</label>

  <input id="dateOfBirth" name="dateOfBirth" [(ngModel)]="dateOfBirth"

          type="text" bsDatepicker class="form-control" />

</div>

**Step 3 :**Include a reference to the bs-datepicker.css file in .angular-cli.json file.

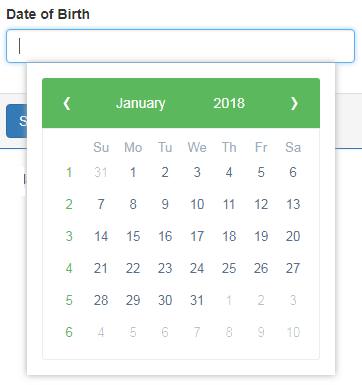
"styles": [

  "../node\_modules/bootstrap/dist/css/bootstrap.min.css",

  "../node\_modules/ngx-bootstrap/datepicker/bs-datepicker.css",

  "styles.css"

]

At this point when you view the page in Google chrome or Firefox, you get the same datepicker and hence the same experience.   
   
  
When we select a date from the date-picker control, the "Date of Birth" textbox is automatically populated with the selected date and it is also captured by the angular generated form model.  
  
**With this datepicker control, it is also very easy to capture a date range.** For example, you have an open job role, and you want to capture a date range for accepting CV's, we can very easily do this. All we have to do is use bsDaterangepicker directive instead of bsDatepicker directive on the input element as shown below.

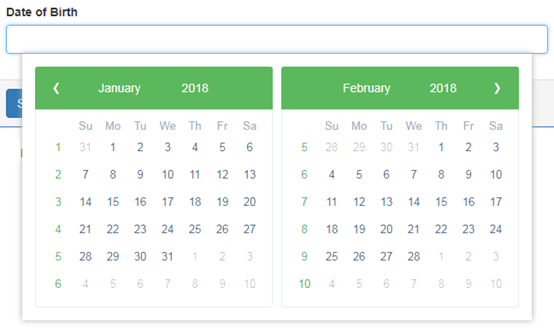
<div class="form-group">

  <label for="dateOfBirth">Date of Birth</label>

  <input id="dateOfBirth" name="dateOfBirth" [(ngModel)]="dateOfBirth"

          type="text" bsDaterangepicker class="form-control" />

</div>

The above simple change, will display Daterange picker as shown below. When we select a date range, the corresponding input field is automatically populated with the selected date range and it is also captured by the angular generated form model.   


* At the moment, the Datepicker is using the **default green theme**. We want to change it to **dark-blue theme**, so it matches with the rest of the form.
* The date is captured in the textbox in mm/dd/yyyy format. We want to change it to dd/mm/yyyy format
* At the moment there is no default date. We want to set a default date
* The input element is spanning across the entire width of the form. We want to limit it's width

Datepicker is a highly configurable component. We will discuss how to do all of the above in our next video.

# Customise ngx-bootstrap datepicker

**Suggested Videos**  
[Part 9 - Angular bootstrap select list](https://www.youtube.com/watch?v=HMK4P_jx0y8) | [Text](http://csharp-video-tutorials.blogspot.com/2018/01/angular-bootstrap-select-list.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/01/angular-bootstrap-select-list-slides.html)  
[Part 10 - Angular select options from array](https://www.youtube.com/watch?v=4fKa13TTn7E) | [Text](http://csharp-video-tutorials.blogspot.com/2018/01/angular-select-options-from-array.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/01/angular-select-options-from-array-slides.html)   
[Part 11 - Angular datepicker tutorial](https://www.youtube.com/watch?v=edaN6iCcqP4) | [Text](http://csharp-video-tutorials.blogspot.com/2018/01/angular-datepicker-tutorial.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/01/angular-datepicker-tutorial-slides.html)  
  
In this video we will discuss **customising the ngx-bootstrap datepicker**component with an example. This is continuation to [Part 11](https://www.youtube.com/watch?v=edaN6iCcqP4). Please watch [Part 11](https://www.youtube.com/watch?v=edaN6iCcqP4) from [Angular CRUD tutorial](https://www.youtube.com/playlist?list=PL6n9fhu94yhXwcl3a6rIfAI7QmGYIkfK5) before proceeding.   
  
  
At the moment we have date range picker. Use bsDatepicker directive instead of bsDaterangepicker directive on the input element so we can capture single date i.e the Date of Birth of the employee.

<div class="form-group">

  <label for="dateOfBirth">Date of Birth</label>

  <input id="dateOfBirth" name="dateOfBirth" [(ngModel)]="dateOfBirth"

          class="form-control" type="text" bsDatepicker  />

</div>

**Changing ngx-bootstrap datepicker theme :**At the moment, the Datepicker is using the default green theme. We want to change it to dark-blue theme, so it matches with the rest of the form. As of this recording ngx-bootstrap datepicker component has the following 6 color schemes.

1. theme-default
2. theme-green
3. theme-blue
4. theme-dark-blue
5. theme-red
6. theme-orange

We can change the default colour-scheme, by manipulating containerClass property in bsConfig object. Here are the steps.

**Step 1 :**Make the following changes in CreateEmployeeComponent class (i.e create-employee.component.ts file)

// Import BsDatepickerConfig type. This is the Config object for datepicker. Using this

// config object we can set minDate, maxDate, whether to show/hide week numbers and

// change the color theme using the containerClass property.

import { BsDatepickerConfig } from 'ngx-bootstrap/datepicker';

// In the CreateEmployeeComponent class make the following changes

export class CreateEmployeeComponent implements OnInit {

  // create a property of type Partial<BsDatepickerConfig>

  datePickerConfig: Partial<BsDatepickerConfig>;

  // In the constructor set containerClass property to the preferred theme

  constructor() {

    this.datePickerConfig = Object.assign({}, { containerClass: 'theme-dark-blue' });

   }

   // Rest of the code...

}

**Please note :**  
We are using the TypeScript partial type here to set only the "containerClass" property of BsDatepickerConfig object. To learn more about the partial type please refer to the following article.  
<https://www.typescriptlang.org/docs/handbook/release-notes/typescript-2-1.html>  
  
**Object.assign()** copies the property values from one or more source objects to a target object. The target object is the first parameter and the rest are the sources. Object.assign() is useful for merging objects or cloning them shallowly.  
  
**Step 2 :**In the view template (i.e in create-employee.component.html file) bind the "datePickerConfig" property in the component class we created above in Step 1, to the bsConfig input property.

<div class="form-group">

  <label for="dateOfBirth">Date of Birth</label>

  <input id="dateOfBirth" name="dateOfBirth" [(ngModel)]="dateOfBirth"

          class="form-control" type="text" bsDatepicker [bsConfig]="datePickerConfig" />

</div>

At this point, you should see the Datepicker using the **dark-blue** theme colour as shown below.   
   
  
**Showing or hiding week numbers :**By default, the weeknumber are displayed. If you want to hide them, all you have to do is set "showWeekNumbers" boolean property to false in the config object as shown below.

constructor() {

  this.datePickerConfig = Object.assign({},

    {

      containerClass: 'theme-dark-blue',

      showWeekNumbers: false

    });

}

You can find all the properties of the config object at the following page.   
<https://github.com/valor-software/ngx-bootstrap/blob/development/src/datepicker/bs-datepicker.config.ts>  
  
Along the same lines we can also set the **min and max dates**. Please note that the month numbers start from 0 and not 1. So for January it is 0, February it is 1, so on and so forth.

constructor() {

  this.datePickerConfig = Object.assign({},

    {

      containerClass: 'theme-dark-blue',

      showWeekNumbers: true,

      minDate: new Date(2018, 0, 1),

      maxDate: new Date(2018, 11, 31),

    });

}

To change the **date format**, use dateInputFormat property of the config object.

constructor() {

  this.datePickerConfig = Object.assign({},

    {

      containerClass: 'theme-dark-blue',

      showWeekNumbers: true,

      minDate: new Date(2018, 0, 1),

      maxDate: new Date(2018, 11, 31),

      dateInputFormat: 'DD/MM/YYYY'

    });

}

To set a default date, create a property (dateOfBirth) in the component class and set it to the default value you want. Since we are using 2 way databinding, the defualt date is displayed in the corresponding input field when them form loads. In this case we have set default date to January 30, 2018.  
  
dateOfBirth: Date = new Date(2018, 0, 30);  
  
At the moment, the "Date of Birth" input element is spanning across the entire width of the form. There are sevral options to limit it's width. One option is to use the Bootstrap row and grid classes (Example: col-md-4, col-md-5, etc...)

<div class="row">

  <div class="form-group col-md-4">

    <label for="dateOfBirth">Date of Birth</label>

    <input id="dateOfBirth"  name="dateOfBirth" [(ngModel)]="dateOfBirth"

           class="form-control" type="text" bsDatepicker

           [bsConfig]="datePickerConfig" />

  </div>

</div>

To control the placement of the Datepicker use the placement property. The allowed values are "top" | "bottom" | "left" | "right". The default is "bottom".  
  
For our form we do not want a default date to be set. So please remove the dateOfBirth property from the component class. We also do not want minDate and maxDate, so delete these properties as well from the datePickerConfig object. Also delete, showWeekNumbers property as it is set to true by default. This means our datePickerConfig object in the constructor has just 2 properties (dateInputFormat and containerClass)

constructor() {

  this.datePickerConfig = Object.assign({},

    {

      containerClass: 'theme-dark-blue',

      dateInputFormat: 'DD/MM/YYYY'

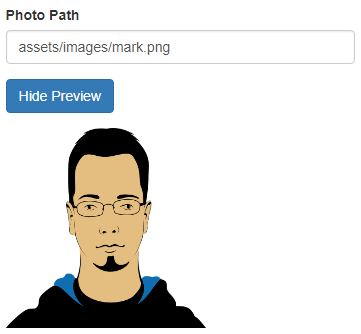
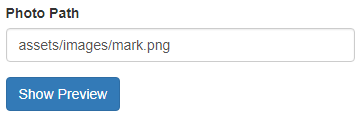
    });

}

# Angular ngif directive

**Suggested Videos**  
[Part 10 - Angular select options from array](https://www.youtube.com/watch?v=4fKa13TTn7E) | [Text](http://csharp-video-tutorials.blogspot.com/2018/01/angular-select-options-from-array.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/01/angular-select-options-from-array-slides.html)  
[Part 11 - Angular datepicker tutorial](https://www.youtube.com/watch?v=edaN6iCcqP4) | [Text](http://csharp-video-tutorials.blogspot.com/2018/01/angular-datepicker-tutorial.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/01/angular-datepicker-tutorial-slides.html)   
[Part 12 - Customise ngx-bootstrap datepicker](https://www.youtube.com/watch?v=R_QcssTIEr0) | [Text](http://csharp-video-tutorials.blogspot.com/2018/01/customise-ngx-bootstrap-datepicker.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/01/customise-ngx-bootstrap-datepicker_18.html)  
  
**In this video we will discuss**

* Angular ngIf structural directive with an example
* How to prevent a button from submitting form

**Here is what we want to do :**When the **Create Employee form**first loads, we want to display a field to enter "Photo Path" and "Show Preview" button   
  
   
  
  
For now, we will assume employee photo is already available in the **assets/images**folder. We will discuss uploading files in a later video in this series.    
  
Once the user has typed the photo path in the respective field, and when they click **"Show Preview"** button, we want to display the photo and the text on the button should change to **"Hide Preview"**.    
  
   
  
At this point when the employee clicks **"Hide Preview"** button, the photo should be hidden and the text on the button should change again back to **"Show Preview"**.   
  
   
  
**Here are the steps to achieve this**   
  
**Step 1 :** First include an input field for capturing employee photo path. As we have set both the **name**property and **ngModel**directive to **photoPath**, Angular generated form model will create a property with name **"photoPath"** and keeps track of what is typed in the photoPath textbox.

<div class="form-group">

  <label for="photoPath">Photo Path</label>

  <input id="photoPath" type="text" class="form-control"

          name="photoPath" [(ngModel)]="photoPath">

</div>

**Step 2 :** Include image element to preview the employee photo. Notice we have set height and width to 200 pixles. Also notice we are binding the img element src property to the photoPath property.

<div class="form-group">

  <img [src]="photoPath" height="200" width="200" />

</div>

With the above 2 changes in place, view the page in the browser and launch browser developer tools. On the console tab, you will see the following error. This is because, when the form loads, photoPath property is null and we have bound it to the src property of the img element.   
Failed to load resource: the server responded with a status of 404 (Not Found)   
  
At this point, as you start to type in the **"Photo Path"** textbox, you will see a 404 error logged to the console every time you type a character. This is because every time a character is typed, angular tries to bind the src property of the image element to the photoPath property, Since we have not completed typing the full valid photo path, Angular is not able to find the image and it logs a 404 error to the console. Once we complete typing the valid photo path, the photo is displayed.   
  
   
  
**Step 3 :**We do not want to render the image element when the form first loads. So create a boolen property with name previewPhoto in the CreateEmployeeComponent class and initialise it to false.  
  
**previewPhoto = false;**   
  
**Step 4 :**In the view template (i.e in create-employee.component.html) file, include \*ngIf structural directive on the image element. Notice the expression assigned to \*ngIf directive. It is the boolean property (previewPhoto) we created in the component class. If the value of the expression is truthy then the image element is rendered in the DOM otherwise it is not. Since we have initialised previewPhoto with false, the image element will not be rendered when the form is initially loaded.  
  
<img [src]="photoPath" height="200" width="200" \*ngIf="previewPhoto"/>   
  
At this point, view the page in the browser and launch browser developer tools. On the console tab, you will not see any errors on the initial form load. Also, when you start to type in the Photo Path field, you do not see any 404 errors in spite of having the img element bound to photoPath property. This is because the **\*ngIf**structural directive prevented the img element from being added to the DOM as it's value is falsy.    
  
**Step 5 :**Now we need to include a button to Show and Hide Image Preview. In the view template, include the following HTML.

<div class="form-group">

  <button (click)="togglePhotoPreview()" class="btn btn-primary">

    {{previewPhoto ? "Hide " : "Show " }} Preview

  </button>

</div>

**Code explanation :**

* On the button click, we are calling **"togglePhotoPreview()"** method. This is event binding. We discussed Event Binding in [Part 14](https://www.youtube.com/watch?v=g8ofqqT_UAM&index=14&list=PL6n9fhu94yhWqGD8BuKuX-VTKqlNBj-m6) of [Angular 2 tutorial](https://www.youtube.com/playlist?list=PL6n9fhu94yhWqGD8BuKuX-VTKqlNBj-m6).
* We have not created **togglePhotoPreview()**method. We will create it in out next step.
* We are using the Bootstrap btn and btn-primary classes for styling
* We are using interpolation to dynamically change the button text.

**Step 6 :**In the component class, create **togglePhotoPreview()** method. Notice this method toggles the value of **previewPhoto**property.   
  
togglePhotoPreview() {  
  this.previewPhoto = !this.previewPhoto;  
}

* At this point, view the page in the browser and launch browser developer tools. Type a valid photo path and click **"Show Preview"** button.
* The image will be displayed and the text on the button changes to **"Hide Privew"** as expected.
* If you look on the console tab, you will see that the Angular generated form model is logged to the console. We did not expect this to happen.
* 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.

<div class="form-group">

  <button type="button" (click)="togglePhotoPreview()" class="btn btn-primary">

    {{previewPhoto ? "Hide " : "Show " }} Preview

  </button>

</div>

# Angular disable browser validation

**Suggested Videos**  
[Part 11 - Angular datepicker tutorial](https://www.youtube.com/watch?v=edaN6iCcqP4) | [Text](http://csharp-video-tutorials.blogspot.com/2018/01/angular-datepicker-tutorial.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/01/angular-datepicker-tutorial-slides.html)  
[Part 12 - Customise ngx-bootstrap datepicker](https://www.youtube.com/watch?v=R_QcssTIEr0) | [Text](http://csharp-video-tutorials.blogspot.com/2018/01/customise-ngx-bootstrap-datepicker.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/01/customise-ngx-bootstrap-datepicker_18.html)   
[Part 13 - Angular ngif directive](https://www.youtube.com/watch?v=wG4E-FI_Fvs) | [Text](http://csharp-video-tutorials.blogspot.com/2018/01/angular-ngif-directive.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/01/angular-ngif-directive-slides.html)  
  
**In this video we will discuss the following**

* By default Angular 4 and later versions disable browser native validation. How to enable browser validation using ngNativeValidate directive
* How to explicitly disable the native browser validation using the novalidate attribute if you are using Angular 2.
* Why is it better to disable browser built-in validation and use Angular to validate instead

**To understand browser native validation,**

1. On the **"Create Employee"** view template, include required attribute on FullName input field
2. Navigate to the "Create Employee" form and launch browser developer tools.
3. Do not type anything in the "Full Name" input field click the "Save" button.
4. Notice we do not get any validation, in spite of having required attribute on the "Full Name" input field.
5. This is because by default, **Angular 4** and later versions disable browser validation by including novalidate attribute on the form tag.
6. To confirm this, click on the **"Elements"** tab in the browser developer tools and you will see "novalidate" attribute on the form tag
7. If you want to enable browser validation, include ngNativeValidate directive on the form tag in create-employee.component.html file

<form #employeeForm="ngForm" ngNativeValidate

      (ngSubmit)="saveEmployee(employeeForm)">

1. At this point, if you click the "Save" button without typing anything in the Full Name field, the native browser validation kicks in and you will see "Please fill in this field" validation error. At the moment I am using Google Chrome browser.

# Angular form validation

**Suggested Videos**  
[Part 12 - Customise ngx-bootstrap datepicker](https://www.youtube.com/watch?v=R_QcssTIEr0) | [Text](http://csharp-video-tutorials.blogspot.com/2018/01/customise-ngx-bootstrap-datepicker.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/01/customise-ngx-bootstrap-datepicker_18.html)  
[Part 13 - Angular ngif directive](https://www.youtube.com/watch?v=wG4E-FI_Fvs) | [Text](http://csharp-video-tutorials.blogspot.com/2018/01/angular-ngif-directive.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/01/angular-ngif-directive-slides.html)   
[Part 14 - Angular disable browser validation](https://www.youtube.com/watch?v=Xy562Awk8oM) | [Text](http://csharp-video-tutorials.blogspot.com/2018/01/angular-disable-browser-validation.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/01/angular-disable-browser-validation_25.html)  
  
In this video and in the next few videos we will discuss **Form Validation in Angular**with examples. Along the way we will discuss validating textboxes, check boxes, radio buttons, dropdownlists etc. We will also discuss, how to fix one of the common error that we get when exporting NgModel into a local variable. The error that we get is, **cannot assign to a reference or variable.** We will discuss what causes this error and how to fix it.   
  
  
**To understand validation in Angular**, we need to understand the following 3 sets of properties in Angular.

|  |  |  |
| --- | --- | --- |
| **touched**  **untouched** | **pristine**  **dirty** | **valid**  **invalid** |

These 6 properties are available at each individual form control level and also at the form level. Let's look at these properties in action at an individual form control level. Consider the following "Full Name" field on our CreateEmployee page.

<input id="fullName" required type="text" class="form-control" name="fullName"

       [(ngModel)]="fullName" #fullName="ngModel">

**Please note :**

* We have made the **Full Name** input field required by including requiredattribute on the input field. required is HTML 5 attribute. This attribute specifies that a field is required. Besides required, there are other HTML 5 validation attributes like maxlength, pattern, min, max etc. The following page has the HTML 5 validation attributes list.<https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5/Constraint_validation>
* We will discuss most of these attributes with examples, in our upcoming videos. Angular uses these HTML 5 validation attributes, for validating form fields and displaying meaningful error messages to the end user.
* 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.
* The other way to fix this error is by using our own model instead of using the Angular auto generated form model. Notice we have our own employee model in **employee.model.ts** file in models folder. We will discuss using our own employee model in our upcoming videos. For now let's continue discussing the validation properties provided by Angular.

Include the following HTML table, just after the Full Name field in create-employee.component.html. Notice we are using the local template variable **fullNameControl** to access the 6 validation properties provided by Angular

<table border=1 style="border-collapse:collapse; font-family:Arial; table-layout: fixed">

    <tr style="background-color:rgb(170, 120, 12); font-weight: bold">

      <td colspan="3" style="padding:3px; white-space:nowrap; width:100%">

        <h4>Full Name Field</h4>

      </td>

    </tr>

    <tr style="background-color:rgb(212, 149, 13); font-weight: bold">

      <td style="padding:10px; white-space:nowrap; width:33%">

        <div>touched : {{ fullNameControl.touched }}</div>

        <div>untouched : {{ fullNameControl.untouched }}</div>

      </td>

      <td style="padding:10px; white-space:nowrap; width:33%">

        <div>pristine : {{ fullNameControl.pristine }}</div>

        <div>dirty : {{ fullNameControl.dirty }}</div>

      </td>

      <td style="padding:10px; white-space:nowrap; width:33%">

        <div>valid : {{ fullNameControl.valid }}</div>

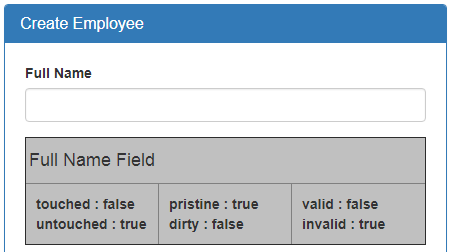
        <div>invalid : {{ fullNameControl.invalid }}</div>

      </td>

    </tr>

  </table>

**At this point, view the page in the browser and notice the following**

* touched is false and untouched is true as we did not touch the Full Name field yet. Notice when we click in the Full Name field, touched is still false and untouched is still true. These 2 properties change when the field loses focus. So when we tab out of the control, notice both the properties change as expected.   
  
* Notice the pristine and dirty properties. pristine means the form control value has not been changed and dirty means the value has been changed. Notice when we type something in the Full Name field the properties change as expected. Even after we delete everything we typed, the dirty property remains true because we have changed the value in the form control.
* Finally notice valid and invalid properties. Since we have required attribute on the Full Name field, valid is false, if we do not have anything typed in the field. The moment we type something, valid is true and invalid is false. If we delete everything we have typed, valid becomes false and invalid becomes true as expected.

We also have these same 6 properties at the form level. Notice the form tag, we are already exporting ngForm to a local template variable - employeeForm.

<form #employeeForm="ngForm" (ngSubmit)="saveEmployee(employeeForm)">

We can use this template reference variable **employeeForm** to access the validation properties at the form level. Copy and paste the following HTML in create-employee.component.html

<table border=1 style="border-collapse:collapse; font-family:Arial; table-layout: fixed">

  <tr style="background-color:silver; font-weight: bold">

    <td colspan="3" style="padding:3px; white-space:nowrap; width:100%">

      <h4>Employee Form</h4>

    </td>

  </tr>

  <tr style="background-color:silver; font-weight: bold">

    <td style="padding:10px; white-space:nowrap; width:33%">

      <div>touched : {{ employeeForm.touched }}</div>

      <div>untouched : {{ employeeForm.untouched }}</div>

    </td>

    <td style="padding:10px; white-space:nowrap; width:33%">

      <div>pristine : {{ employeeForm.pristine }}</div>

      <div>dirty : {{ employeeForm.dirty }}</div>

    </td>

    <td style="padding:10px; white-space:nowrap; width:33%">

      <div>valid : {{ employeeForm.valid }}</div>

      <div>invalid : {{ employeeForm.invalid }}</div>

    </td>

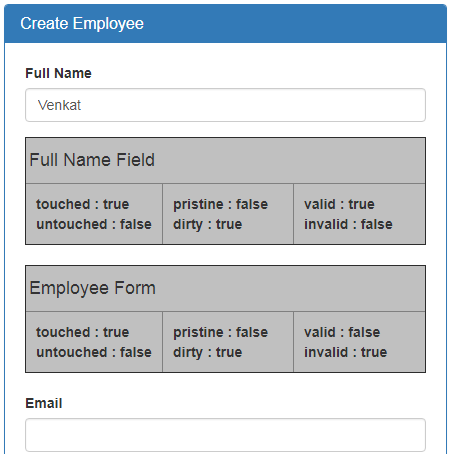
  </tr>

</table>

Also, make the Email input field required by placing the required attribute

<input id="email" required type="text" class="form-control"

       name="email" [(ngModel)]="email">

**Save the changes and view the page in the browser.**   


* Notice when we type "Venkat" in the Full Name input field,
* touched property both at the input field level and form level is set to true
* untouched property both at the input field level and form level is set to false
* Same is true for dirty and pristine properties
* However, notice valid property of the Full Name field is true but the form valid property is false. This is because, Emal field is also required and we did not enter anything in the email field, so email field is invalid and as a result the form is invalid.
* The moment we type something in the email field, the form valid property turns true.
* So this brings us to an important conclusion - If all the form fields are valid, then the form is valid. If any of the form field is invalid the form is also invalid.
* Along the same lines, if any of the form field is touched, the form is also marked touched and if any of the form field is dirty the form is also marked dirty.

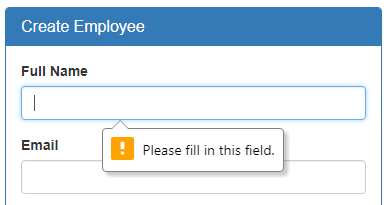
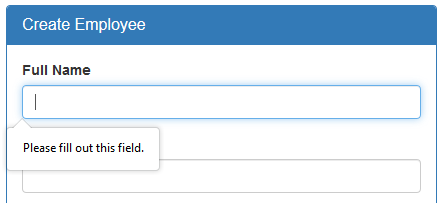
**So to check if a form field is valid**

* Include the HTML 5 validation attributes such as required for example on the input field you want to validate.
* Export ngModel directive to local template variable
* Finally use the template reference variable to access the validation properties like touched, dirty, valid etc.

**Along the same lines, to check if the form is valid**

* Export ngForm directive to a local template reference variable
* Then use the template reference variable to access the validation properties at the form level.

At the moment, we are not displaying any validation error messages to the user. We will discuss how to do this in our next video.   
  
On our form, we do not want to display the validation properties and their values. So to keep out form clean, please remove the HTML from the form, that displays the validation properties and their values.

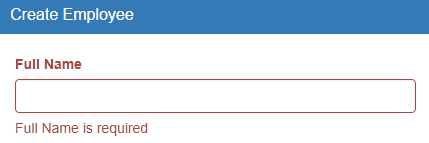
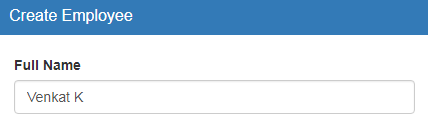
1. 
2. Now if you view the same page in a different browser like **Firefox** for example, you will have a different experience. The browser validation is displayed in Firefox as shown below.   
     
   
3. Because of this inconsistency it is better to disable browser native validation and use Angular instead to validate form fields. We will discuss validaing form fields using Angular in our upcoming videos.
4. By default, browser built-in validation is disabled if you are using Angular 4 or any later version.
5. At the moment, we are using Angular version 5 and want to keep browser validation disabled, so remove the ngNativeValidate directive from the form tag.
6. If you are using Angular 2, you will have to explicitly disable browser validation by using novalidate attribute on the form tag.

**Why is it better to disable browser built-in validation and use Angular to validate instead**  
Different browser vendors implement browser validation differently and as a result, the end users have different experience depending on the browser they use. Because of this inconsistency it is better to disable browser native validation and use Angular instead to validate form fields.    
  
**Next video :** We will discuss validating form fields using Angular

# Displaying angular form validation error messages

**Suggested Videos**  
[Part 13 - Angular ngif directive](https://www.youtube.com/watch?v=wG4E-FI_Fvs) | [Text](http://csharp-video-tutorials.blogspot.com/2018/01/angular-ngif-directive.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/01/angular-ngif-directive-slides.html)  
[Part 14 - Angular disable browser validation](https://www.youtube.com/watch?v=Xy562Awk8oM) | [Text](http://csharp-video-tutorials.blogspot.com/2018/01/angular-disable-browser-validation.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/01/angular-disable-browser-validation_25.html)   
[Part 15 - Angular form validation](https://www.youtube.com/watch?v=d8XONHXTv_4) | [Text](http://csharp-video-tutorials.blogspot.com/2018/01/angular-form-validation.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/01/angular-form-validation-slides.html)  
  
**In this video we will discuss**

* How to display validation error messages to the user
* Style the error messages using Bootstrap
* How to disable Submit button if the form is not valid

This is continuation to [Part 15](https://www.youtube.com/watch?v=d8XONHXTv_4). Please watch [Part 15](https://www.youtube.com/watch?v=d8XONHXTv_4) from [Angular CRUD tutorial](https://www.youtube.com/playlist?list=PL6n9fhu94yhXwcl3a6rIfAI7QmGYIkfK5) before proceeding.   
  
  
Here is what we want to do. If the **"Full Name" field is not valid**we want to style the input field with a red border. The associated label text should also turn red and "Full Name is required" validation error message should be displayed.   
  
   
  
Once we type something in the Full Name field, and when it becomes valid, the validation message and the red broder should disappear and also the label text should return to it's normal black colourcolour.   
  
   
  
We will be using the Bootstrap framework for styling validation error messages. If you are new to Bootstrap, please check out our [Bootstrap tutorial by clicking here](https://www.youtube.com/playlist?list=PL6n9fhu94yhXd4xnk-j5FGhHjUv1LsF0V).   
  
We discussed Bootstrap form validation states in [Part 23](https://www.youtube.com/watch?v=HvUC-o35gzg&index=23&list=PL6n9fhu94yhXd4xnk-j5FGhHjUv1LsF0V) of [Bootstrap tutorial](https://www.youtube.com/playlist?list=PL6n9fhu94yhXd4xnk-j5FGhHjUv1LsF0V). We will use the following Bootstrap classes for styling validation error messages.

* has-error
* control-label
* help-block

Modify the "Full name" input filed as shown below.

<div class="form-group" [class.has-error]="fullNameControl.invalid">

  <label for="fullName" class="control-label">Full Name</label>

  <input id="fullName" required type="text" class="form-control" name="fullName"

         [(ngModel)]="fullName" #fullNameControl="ngModel">

  <span class="help-block" \*ngIf="fullNameControl.invalid">

    Full Name is required

  </span>

</div>

**Code explanation :**

* [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.
* \*ngIf="fullNameControl.invalid". The \*ngIf structural directive on the span element adds or removes the validation error message depending on the invalid property value. If the invalid property is true, then the validation error message is displayed, otherwise it is removed. Also, notice we are using the Bootstrap **help-block**class on the span element for styling.

At this point, save the changes and view the page in the browser. Notice when the form initially loads, we see the validation error message Full Name is required and it is also styled as expected. As we soon as we start typing, the error goes away. When we delete everything that we have typed, the error appears again. So, it's working as expected.

Let's enhance this a bit more. **Some users does not like to see the validation error messages, even before they had the opportunity to touch the form field.** So what we want to do is,

* Do not display any validation error messages when the form is initially loaded.
* When the user touches the field, and if he leaves the field without typing in the value, then we want to display the validation error message.

This is easy. You might have already guessed we could use touched property to achieve this. So modify the Full Name field HTML as shown below. With this change, **the validation error message is displayed only when the Full Name field is invalid and touched.**

<div class="form-group"

     [class.has-error]="fullNameControl.invalid && fullNameControl.touched">

  <label for="fullName" class="control-label">Full Name</label>

  <input id="fullName" required type="text" class="form-control" name="fullName"

         [(ngModel)]="fullName" #fullNameControl="ngModel">

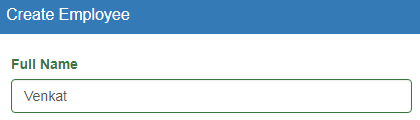
  <span class="help-block"

        \*ngIf="fullNameControl.invalid && fullNameControl.touched">

    Full Name is required

  </span>

</div>

To take this to the next level, we can style a valid field with a different colour. Here is what I mean.   
  


* When the form first loads, the Full Name and it's label are black in colour and the validation error message is not displayed
* When the user touches the field and leaves it without typing anything, the colour changes to red and the validation error message is displayed
* 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 as shown below. As you can see, the **has-success**class is added when valid property is true and it is removed when it is false.

<div class="form-group"

     [class.has-error]="fullNameControl.invalid && fullNameControl.touched"

     [class.has-success]="fullNameControl.valid">

As you can see these angular validation properties (valid, touched, dirty etc.) provide lot of power and flexibility when validating form fields and displaying validation error messages.  
  
**How to disable Submit button if the form is not valid :** 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>

# Model binding in angular template driven forms

**Suggested Videos**  
[Part 14 - Angular disable browser validation](https://www.youtube.com/watch?v=Xy562Awk8oM) | [Text](http://csharp-video-tutorials.blogspot.com/2018/01/angular-disable-browser-validation.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/01/angular-disable-browser-validation_25.html)  
[Part 15 - Angular form validation](https://www.youtube.com/watch?v=d8XONHXTv_4) | [Text](http://csharp-video-tutorials.blogspot.com/2018/01/angular-form-validation.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/01/angular-form-validation-slides.html)   
[Part 16 - Displaying angular form validation error messages](https://www.youtube.com/watch?v=Vh6estB9kqk) | [Text](http://csharp-video-tutorials.blogspot.com/2018/01/displaying-angular-form-validation.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/01/displaying-angular-form-validation_31.html)  
  
**In this video we will discuss**

* Binding Angular form to our own model class
* We will also discuss, how to fix one of the common error that we get when exporting ngModel into a local variable. The error that we get is, cannot assign to a reference or variable.

At the moment, in **CreateEmployeeComponent**we are using the Angular Auto-generated form model. Instead of using the Angular generated form model, we can use our model class.   
  
  
In **employee.model.ts** file in the **models**folder, we have Employee class. We want to use this class as the model when creating a new employee. Here are the steps.   
  
**Step 1 :** In create-employee.component.ts file, import the Employee model  
import { Employee } from '../models/employee.model';   
  
**Step 2 :** In CreateEmployeeComponent class, include **employee**property. Notice we have set the type to Employee and initialised all properties with NULL value.   
  
export class CreateEmployeeComponent implements OnInit {  
  employee: Employee = {  
    id: null,  
    name: null,  
    gender: null,  
    contactPreference: null,  
    phoneNumber: null,  
    email: null,  
    dateOfBirth: null,  
    department: null,  
    isActive: null,  
    photoPath: null  
  };   
  
**Step 3 :** In the view template, bind the ngModel directive of an input field to it's corresponding property on the employee object. The employee property we created in Step 2 returns an employee object, which is the model for our form.   
  
For example, bind ngModel directive on the email input field to the email property on the employee object.  
[(ngModel)]="employee.email"   
  
Except fullName, bind the ngModel directive of the rest of the input fields with the corresponding properties on the employee object.   
  
In the employee class we do not have **fullName**property. we have **name**instead. On the view template, the corresponding input field name is fullName. To keep things consistent let's change fullName to name on the label and the input field as shown below.

<div class="form-group" [class.has-error]="name.invalid && name.touched">

  <label for="name" class="control-label">Name</label>

  <input id="name" required type="text" class="form-control" name="name"

         [(ngModel)]="name" #name="ngModel">

  <span class="help-block" \*ngIf="name.invalid && name.touched">

    Name is required

  </span>

</div>

At this point, if you view the page in the browser, you will see the following error.  
**Cannot assign to a reference or variable**   
  
We get this error because, Angular generated form model creates **name** property and we are also creating a local template variable with the same name by exporting **ngModel**to **#name**. 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 ~~name~~. So if we change #name="ngModel" to #nameControl="noModel" the error goes away. We discussed this in detail in [Part 15](https://www.youtube.com/watch?v=d8XONHXTv_4) of [Angular CRUD tutorial](https://www.youtube.com/playlist?list=PL6n9fhu94yhXwcl3a6rIfAI7QmGYIkfK5).   
  
The other way to fix this error is by using our own model. Using the **ngModel**directive, bind the **name**property of the employee object to the **name**input field

[(ngModel)]="employee.name"

At this point, if you view the page in the browser and notice the error is gone and all the properties in the Angular generated form model are NULL as expected.   
  
To see our own employee model, include the following code in the view template file (create-employee.component.html)

Angular Generated Forom Model : {{employeeForm.value | json}}

<br/>

<br/>

Our Employee Model : {{ employee | json}}

At this point, on the browser we should see both - Angular generated form model and our own employee model. Notice as we change the values in the input fields, the respective properties in both the models are updated as expected.   
  
At the moment, when we click the "Save" button, we are logging the **employeeForm.value**to the console. We instead want to log our employee model object. To do this   
In the view template, pass the employee object to the saveEmployee() method.

<form #employeeForm="ngForm" (ngSubmit)="saveEmployee(employee)">

Modify saveEmployee() method in create-employee.component.ts file as shown below.   
  
saveEmployee(newEmployee: Employee): void {  
  console.log(newEmployee);  
}   
  
At this point, when we click the Save button, the employee object is logged to the console as expected.

# Angular email validation example

**Suggested Videos**  
[Part 15 - Angular form validation](https://www.youtube.com/watch?v=d8XONHXTv_4) | [Text](http://csharp-video-tutorials.blogspot.com/2018/01/angular-form-validation.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/01/angular-form-validation-slides.html)  
[Part 16 - Displaying angular form validation error messages](https://www.youtube.com/watch?v=Vh6estB9kqk) | [Text](http://csharp-video-tutorials.blogspot.com/2018/01/displaying-angular-form-validation.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/01/displaying-angular-form-validation_31.html)   
[Part 17 - Model binding in angular template driven forms](https://www.youtube.com/watch?v=H9CekX45hjU) | [Text](http://csharp-video-tutorials.blogspot.com/2018/02/model-binding-in-angular-template.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/02/model-binding-in-angular-template_5.html)  
  
**In this video we will discuss**

* Validating Email form field in Angular
* Using multiple validators on a single input field
* Angular safe navigation operator

**Email validation in Angular :**There are 2 different ways to validate email form field in Angular. We can either use **pattern validator**or **email validator**. Email validator is introduced in Angular 4. So if you are using Angular 4 or later version you may use email validator or pattern validator to validate email. If you are using Angular 2, then your only choice is to use Pattern validator.   
  
  
In this video we will discuss using the Angular built-in **Email validator**and in our next video we will discuss using the **Pattern validator**.    
  
**Consider the following HTML. Notice we are using Bootstrap classes for styling.**

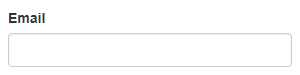
<div class="form-group">

  <label for="email">Email</label>

  <input id="email" type="text" class="form-control" name="email"

          [(ngModel)]="employee.email">

</div>

The above HTML would produce the following Email input field   
   
  
We want to validate this email input field for 2 things

* Email is required and
* Valid email must be provided

To make email, a required field modify the HTML as shown below

<div class="form-group" [class.has-error]="email.invalid && email.touched">

  <label for="email" class="control-label">Email</label>

  <input id="email" required type="text" class="form-control" name="email"

          [(ngModel)]="employee.email" #email="ngModel">

  <span class="help-block" \*ngIf="email.invalid && email.touched">

    Email is required

  </span>

</div>

**Code Explanation :**

* [class.has-error]="email.invalid && email.touched". This is class binding in angular. If the email field is touched and invalid, then the Bootstrap class has-error is added to the div element, else the class is removed.
* On the label that displays "Email" text, we applied control-label Bootstrap class. This class turns the label text to red if there is a validation error.
* \*ngIf="email.invalid && email.touched". Notice the \*ngIf structural directive on the span element. If the email field is touched and invalid the span element is added to the DOM, else it is removed. The Bootstrap help-block class on the span element is for styling.

At this point, if you touch the email field and leave it without typing in anything, you will see the validation error message "Email is required"

   
  
We also want to make sure the user enters a valid email. If someone types ABC, that is not a valid email. Angular 4 has built-in email validator, that we can use to validate if the user has entered a valid email. Here are the steps.  
  
**Step 1 :**On the email input field, place the **email** directive

<input id="email" required email

**Step 2 :**Use the following HTML, to display the validation error message. 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 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.

<span class="help-block" \*ngIf="email.errors?.email && email.touched">

  Email is Invalid

</span>

Here is the complete HTML that makes the email filed required and also checks if the email has a valid format

<div class="form-group" [class.has-error]="email.invalid && email.touched">

  <label for="email" class="control-label">Email</label>

  <input id="email" required email type="text" class="form-control" name="email"

          [(ngModel)]="employee.email" #email="ngModel">

  <span class="help-block" \*ngIf="email.errors?.required && email.touched">

    Email is required

  </span>

  <span class="help-block" \*ngIf="email.errors?.email && email.touched">

    Email is Invalid

  </span>

</div>

As of this recording, email validator provided by Angular does not allow null or empty values. When we leave the email field empty, the email validator is still fired. This is wrong. Checking NULL and empty values should be the job of the required validator. The following is the work around.  
  
Bind email directive to a boolean expression. The email validator is only added when the email field value is not an empty string. This ensures that, when we type something in the email field, the email validator is attached to the input field and it validates if the email format is valid or not.   
  
<input id="email" required [email]="employee.email!==''"  
  
**Please note :**Do not forget to initialise the email property in the employee object to an empty string.  
  
**Next video :**Using pattern validator to validate email in Angular

# Angular regular expression validation

**Suggested Videos**  
[Part 16 - Displaying angular form validation error messages](https://www.youtube.com/watch?v=Vh6estB9kqk) | [Text](http://csharp-video-tutorials.blogspot.com/2018/01/displaying-angular-form-validation.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/01/displaying-angular-form-validation_31.html)  
[Part 17 - Model binding in angular template driven forms](https://www.youtube.com/watch?v=H9CekX45hjU) | [Text](http://csharp-video-tutorials.blogspot.com/2018/02/model-binding-in-angular-template.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/02/model-binding-in-angular-template_5.html)   
[Part 18 - Angular email validation example](https://www.youtube.com/watch?v=DNqZ7Du_64Y) | [Text](http://csharp-video-tutorials.blogspot.com/2018/02/angular-email-validation-example.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/02/angular-email-validation-example-slides.html)  
  
In this video we will discuss using **pattern validator in angular** to meet most of your application complex validation requirements.    
  
  
**With the pattern validator we use a regular expression**. Regular expressions are extremely useful when you want to validate if a given string conforms to a specified pattern.   
  
For example, you can use regular expressions to check if a given email conforms to a a valid email format. Similarly you can also check if provided postcode conforms to a specific country postcode format.   
  
Apart from checking conformity with a pattern, they can also be used to extract sub-strings from a given input string.  
  
  
**To validate** if the provided email has a valid email pattern we can use the pattern validator in angular. To use the pattern validator use the pattern attribute along with the regular expression on the input field you want to validate.

<input pattern="^[a-zA-Z0-9\_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$" type="text"

       name="email" [(ngModel)]="employee.email" #email="ngModel">

It is easy to learn regular expressions. Initially they may appear complicated, but if you get the basics right it is very easy to understand them. However, you can also find the commonly used regular expressions on the internet. For example, if you want to find a regular expression to validate email address, simply search the internet with the following string  
Regular expression for email validation  
  
**Use the following HTML**, to display the validation error message. If the email is invalid, angular attaches pattern key to the errors collection. On the other hand, if the email field is valid, the key pattern will not be in the errors collection. The question mark here is called the **safe navigation operator**. We discussed this operator in detail in our previous video. If you are new to this operator, please check out our previous video.

<span class="help-block" \*ngIf="email.errors?.pattern && email.touched">

  Email is Invalid

</span>

The following example, shows both **required**and **pattern**validators on the Email input field.

<div class="form-group" [class.has-error]="email.invalid && email.touched">

  <label for="email" class="control-label">Email</label>

  <input required pattern="^[a-zA-Z0-9\_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$"

         id="email" type="text" class="form-control" name="email"

         [(ngModel)]="employee.email" #email="ngModel">

  <span class="help-block" \*ngIf="email.errors?.required && email.touched">

    Email is required

  </span>

  <span class="help-block" \*ngIf="email.errors?.pattern && email.touched">

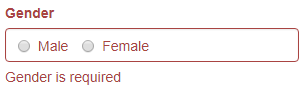
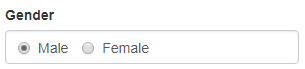
    Email is Invalid

  </span>

</div>

Let's take this pattern validation to the next level. I want to validate emails against a specific domain. For example pragimtech.com is the only valid domain that I want to allow. Any other domain should be considered invalid. This can be very easily achieved with the following regular expression.  
  
^[a-zA-Z0-9\_.+-]+@(?:(?:[a-zA-Z0-9-]+\.)?[a-zA-Z]+\.)?(pragimtech)\.com$

# Angular radio button validation

**Suggested Videos**  
[Part 17 - Model binding in angular template driven forms](https://www.youtube.com/watch?v=H9CekX45hjU) | [Text](http://csharp-video-tutorials.blogspot.com/2018/02/model-binding-in-angular-template.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/02/model-binding-in-angular-template_5.html)  
[Part 18 - Angular email validation example](https://www.youtube.com/watch?v=DNqZ7Du_64Y) | [Text](http://csharp-video-tutorials.blogspot.com/2018/02/angular-email-validation-example.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/02/angular-email-validation-example-slides.html)  
[Part 19 - Angular regular expression validation](https://www.youtube.com/watch?v=V8GVKAVkTVc) | [Text](http://csharp-video-tutorials.blogspot.com/2018/02/angular-regular-expression-validation.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/02/angular-regular-expression-validation_8.html)  
  
In this video we will discuss **radio button validation in Angular with example**.    
  
  
**Example :** Gender is a required field. If one of the gender radio button is not checked, we want to validate and display "Gender is required" validation error message.   
  
   
  
  
As soon as one of the "Gender" radio button is selected, the validation error message should disappear.   
  
   
  
Here is the HTML that makes this possible.

<div class="form-group" [class.has-error]="gender.invalid">

  <label class="control-label">Gender</label>

  <div class="form-control">

    <label class="radio-inline">

      <input type="radio" name="gender" required #gender="ngModel"

             value="male" [(ngModel)]="employee.gender"> Male

    </label>

    <label class="radio-inline">

      <input type="radio" name="gender" required #gender="ngModel"

             value="female" [(ngModel)]="employee.gender"> Female

    </label>

  </div>

  <span class="help-block" \*ngIf="gender.invalid">

    Gender is required

  </span>

</div>

**Code Explanation :**

* Notice we have required attribute on both the radio buttons (Male and Female). This attribute makes the "Gender" field required.
* #gender="ngModel". This creates a template reference variable. We can now this variable (gender) to check if the field is invalid. Notice #gender="ngModel" is placed on the both the radio buttons.
* [class.has-error]="gender.invalid". This class binding adds the has-errorbootstrap css class when the field is invalid and removes it when the field is valid. This class is used for styling the validation error messages.
* On the label element that displays the static text "Gender" we have "control-label" class. This class turns the text "Gender" to red when there is a validation error.
* \*ngIf="gender.invalid". Notice the \*ngIf structural directive on the span element. If the gender field is invalid the span element is added to the DOM, else it is removed. The Bootstrap help-block class on the span element is for styling.

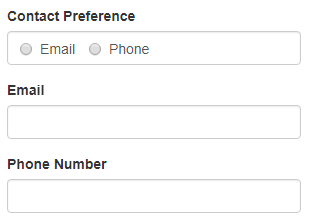
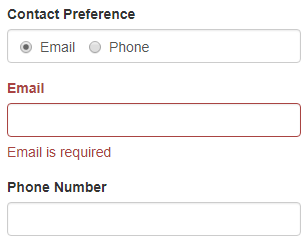
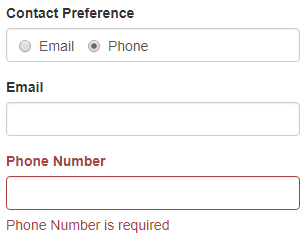
The span element that displays the validation error message can also be coded as shown below. Notice, instead of using "gender.invalid" as the expression for \*ngIf, we are using "gender.errors?.required". When the required validation fails, Angular attaches the required key to the errors collection property of the gender field. The key is removed if the field passes validation. So we can check for the existence of this key, to control the display of the validation error message.

<span class="help-block" \*ngIf="gender.errors?.required">

  Gender is required

</span>

# Add required attribute dynamically in angular

**Suggested Videos**  
[Part 18 - Angular email validation example](https://www.youtube.com/watch?v=DNqZ7Du_64Y) | [Text](http://csharp-video-tutorials.blogspot.com/2018/02/angular-email-validation-example.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/02/angular-email-validation-example-slides.html)  
[Part 19 - Angular regular expression validation](https://www.youtube.com/watch?v=V8GVKAVkTVc) | [Text](http://csharp-video-tutorials.blogspot.com/2018/02/angular-regular-expression-validation.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/02/angular-regular-expression-validation_8.html)   
[Part 20 - Angular radio button validation](https://www.youtube.com/watch?v=6QUkIS6vvEI) | [Text](http://csharp-video-tutorials.blogspot.com/2018/02/angular-radio-button-validation.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/02/angular-radio-button-validation-slides.html)  
  
In this video we will discuss, how to **add required attribute dynamically** in template driven forms. In our upcoming videos we will discuss how to do the same in reactive forms.   
  
  
**Example :**Consider the following 3 fields.   
  
   
  
  
If "Email" is selected as the "Contact Preference", then "Email" input field is required.   
  
   
  
If "Phone" is selected as the "Contact Preference", then "Phone" input field is required.   
  
   
  
**Contact Preference radio buttons HTML :**

<div class="form-group" [class.has-error]="contactPreference.invalid">

  <label class="control-label">Contact Preference</label>

  <div class="form-control">

    <label class="radio-inline">

      <input type="radio" required #contactPreference="ngModel"name="contactPreference"

              value="email" [(ngModel)]="employee.contactPreference"> Email

    </label>

    <label class="radio-inline">

      <input type="radio" required #contactPreference="ngModel"name="contactPreference"

              value="phone" [(ngModel)]="employee.contactPreference"> Phone

    </label>

  </div>

  <span class="help-block" \*ngIf="contactPreference.errors?.required">

      Contact Preference is required

  </span>

  <!-- Delete the below line after you see

      the selected value in the browser -->

  Contact Preference Selected Value : {{ contactPreference.value }}

</div>

**Code Explanation :**

* We made the contact preference field required, by including required attribute on both email and phone radio buttons.
* At this point view the page in the browser and notice the selected radio button value. When email radio button is selected we see the selected value as email, and when phone radio button is selected, we see the selected value as phone.
* We will use this selected contact preference radio button value (contactPreference.value) to dynamically add or remove required attribute to phone and email input fields.

**Email input field HTML :**

<div class="form-group" [class.has-error]="email.invalid">

  <label for="email" class="control-label">Email</label>

  <input id="email" [required]="contactPreference.value=='email'"

          type="text" class="form-control"

          pattern="^[a-zA-Z0-9\_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$"

          [(ngModel)]="employee.email" #email="ngModel" name="email">

  <span class="help-block" \*ngIf="email.errors?.required">

    Email is required

  </span>

  <span class="help-block" \*ngIf="email.errors?.pattern && email.touched">

    Email is Invalid

  </span>

</div>

**Code explanation :**

* Notice the required attribute. We assigned it a boolean expression (contactPreference.value=='email').
* 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.

**Phone Number input field HTML :**Notice the boolean expression assigned to the required attribute. This boolean expression dynamically adds or removes required attribute to the phone number field depending on whether phone contact preference radio button is selected or not.

<div class="form-group" [class.has-error]="phoneNumber.invalid">

  <label for="phoneNumber" class="control-label">Phone Number</label>

  <input id="phoneNumber" [required]="contactPreference.value=='phone'"

          #phoneNumber="ngModel" class="form-control" type="text"

          name="phoneNumber" [(ngModel)]="employee.phoneNumber">

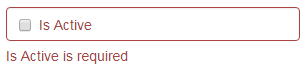
  <span class="help-block" \*ngIf="phoneNumber.errors?.required">

    Phone Number is required

  </span>

</div>

# Angular checkbox validation

**Suggested Videos**  
[Part 19 - Angular regular expression validation](https://www.youtube.com/watch?v=V8GVKAVkTVc) | [Text](http://csharp-video-tutorials.blogspot.com/2018/02/angular-regular-expression-validation.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/02/angular-regular-expression-validation_8.html)  
[Part 20 - Angular radio button validation](https://www.youtube.com/watch?v=6QUkIS6vvEI) | [Text](http://csharp-video-tutorials.blogspot.com/2018/02/angular-radio-button-validation.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/02/angular-radio-button-validation-slides.html)   
[Part 21 - Add required attribute dynamically in angular](https://www.youtube.com/watch?v=Qi6tV2M2QfQ) | [Text](http://csharp-video-tutorials.blogspot.com/2018/02/add-required-attribute-dynamically-in.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/02/add-required-attribute-dynamically-in_14.html)  
  
In this video we will discuss **checkbox validation in Angular**with example.    
  
  
We want to make the following **"Is Active" check box a required field**. If the checkbox is not checked, we want to validate and display "Is Active is required" validation error message. As soon as the checkbox is checked, the validation error message should disappear.   
  
   
  
  
**Consider the following HTML**

<div class="form-group" [class.has-error]="isActive.invalid && isActive.touched">

  <div class="form-control">

    <label class="checkbox-inline control-label">

      <input type="checkbox" required name="isActive"

             #isActive="ngModel" [(ngModel)]="employee.isActive">

      Is Active

    </label>

  </div>

  <span class="help-block"

        \*ngIf="isActive.errors?.required && isActive.touched">

    Is Active is required

  </span>

</div>

**Code Explanation :**

* The required attribute makes "Is Active" field required.
* #isActive="ngModel". This creates a template reference variable. We can now this variable (isActive) to check if the field is invalid, touched, dirty etc.
* [class.has-error]="isActive.invalid && isActive.touched". This class binding adds the has-error bootstrap css class when the field is invalid and touched and removes it when the field is valid. This class is used for styling the validation error messages.
* On the label element that displays the static text "Is Active" we have "control-label" class. This class turns the text "Is Active" to red when there is a validation error.
* \*ngIf="isActive.errors?.required && isActive.touched". Notice the \*ngIf structural directive on the span element. If the "Is Active" field fails required validation and touched, the span element is added to the DOM, else it is removed. The Bootstrap help-block class on the span element is for styling.

**At this point,**

* If you tab into the checkbox control and leave it, without checking it, you will see the validation error message
* If you select the checkbox box, the error goes away
* If you unselecet the checkbox, the required validation error appears again

This implementation of the checkbox validation is useful, when you want to force the user to select a checkbox. For example, on many web sites, you might have seen a checkbox with the following text. Only when you agree by checking that checkbox, you will be able to proceed. Otherwise you will have to cancel that specific action.

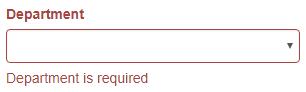
**I Agree to the terms and conditions**  
  
**What if the employee is terminated or resigned?** In that case we do not want the checkbox to be checked. But at the moment, the required validator is forcing us to have the checkbox checked. To fix this modify the required attribute as shown below. Notice, we are binding a boolean expression to the required attribute. If the expression is true the required validator is attached, otherwise it is removed.  
  
[required]="employee.isActive==null"   
  
**With this change**

* When the form first loads, isActive property on the employee object is null. So the required attribute is attached to the checkbox.
* If we tab into the checkbox and levae it without selecting it, we see the required validation error message as expected
* If we select the checkbox box, the error goes away
* If we unselecet the checkbox, notice we don't get the required validation. This is because, when the checkbox is unchecked, the value of isActive property on the employee object is false and not NULL. So the boolean expression bound to the required attribute returns false. Hence the required attribute is removed from the checkbox field and we do not see the required validation error.

At the moment, the user interface is confusing. If the employee you are creating is not active, you have to first check the (Is Active) checkbox and then un-check it. To make this less confusing there 2 options for us.

* Remove the required validator on the (Is Active) checkbox, and treat NULL as false.
* Use 2 radio buttons (Yes or No), instead of a single checkbox.

# Angular select list validation

**Suggested Videos**  
[Part 20 - Angular radio button validation](https://www.youtube.com/watch?v=6QUkIS6vvEI) | [Text](http://csharp-video-tutorials.blogspot.com/2018/02/angular-radio-button-validation.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/02/angular-radio-button-validation-slides.html)  
[Part 21 - Add required attribute dynamically in angular](https://www.youtube.com/watch?v=Qi6tV2M2QfQ) | [Text](http://csharp-video-tutorials.blogspot.com/2018/02/add-required-attribute-dynamically-in.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/02/add-required-attribute-dynamically-in_14.html)   
[Part 22 - Angular checkbox validation](https://www.youtube.com/watch?v=qgMtJKFcOpw) | [Text](http://csharp-video-tutorials.blogspot.com/2018/02/angular-checkbox-validation.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/02/angular-checkbox-validation-slides.html)  
  
In this video, we will discuss **Dropdown list validation in Angular** with example.    
  
  
**Example :** We want to make "Department" Dropdownlist a required field. If a department is not selected, we want to validate and display "Department is required"validation error message. As soon as a department is checked, the validation error message should disappear.   
  
   
  
  
**Consider the following HTML :**

<div class="form-group"

     [class.has-error]="department.touched && department.invalid">

  <label for="department" class="control-label">Department</label>

  <select id="department" required #department="ngModel"

          name="department" [(ngModel)]="employee.department"

          class="form-control">

    <option \*ngFor="let dept of departments" [value]="dept.id">

      {{dept.name}}

    </option>

  </select>

  <span class="help-block"

        \*ngIf="department.touched && department.invalid">

    Department is required

  </span>

</div>

**Code Explanation :**

* The required attribute makes "Department" field required.
* #department="ngModel". This creates a template reference variable. We can now this variable (department) to check if the field is invalid, touched, dirty etc.
* [class.has-error]="department.touched && department.invalid". This class binding adds the has-error bootstrap css class when the field is invalid and touched and removes it when the field is valid. This class is used for styling the validation error messages.
* On the label element that displays the static text "Department" we have "control-label" class. This class turns the text "Department" to red when there is a validation error.
* \*ngIf="department.touched && department.invalid". Notice the \*ngIf structural directive on the span element. If the "Department" field fails required validation and touched, the span element is added to the DOM, else it is removed. The Bootstrap help-block class on the span element is for styling.

At this point, the dropdown list validation works as expected. However, in most of the real world applications, you might see one of the following options as the first option in a dropdown list.

* Please select
* Select Department
* etc...

Modify the HTML to include "Select Department" as the first option. Notice the value of this option is set to**'-1'**, to indicate that it is not a valid department selection. The change is highlighted in YELLOW.

<div class="form-group"

     [class.has-error]="department.touched && department.invalid">

  <label for="department" class="control-label">

    Department

  </label>

  <select id="department" required #department="ngModel" name="department"

          [(ngModel)]="employee.department" class="form-control">

    <option value="-1">Select Department</option>

    <option \*ngFor="let dept of departments" [value]="dept.id">

      {{dept.name}}

    </option>

  </select>

  <span class="help-block"

        \*ngIf="department.touched && department.invalid">

    Department is required

  </span>

</div>

In the component class (create-employee.component.ts), initialise department property with a value of **'-1'**. This will ensure that, when the "Department" dropdownlist is loaded, the first default option 'Select Department' is selected.  
  
employee: Employee = {  
  id: null,  
  name: null,  
  gender: null,  
  contactPreference: null,  
  phoneNumber: null,  
  email: '',  
  dateOfBirth: null,  
  department: '-1',  
  isActive: null,  
  photoPath: null  
};  
  
At this point, view the page in the browser. The dropdownlist REQUIRED validation does not work as expected. The default first option, 'Select Department' is treated as a valid department selection. We will discuss, how to fix this in our next video.

# angular value vs ngvalue

**Suggested Videos**  
[Part 21 - Add required attribute dynamically in angular](https://www.youtube.com/watch?v=Qi6tV2M2QfQ) | [Text](http://csharp-video-tutorials.blogspot.com/2018/02/add-required-attribute-dynamically-in.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/02/add-required-attribute-dynamically-in_14.html)  
[Part 22 - Angular checkbox validation](https://www.youtube.com/watch?v=qgMtJKFcOpw) | [Text](http://csharp-video-tutorials.blogspot.com/2018/02/angular-checkbox-validation.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/02/angular-checkbox-validation-slides.html)   
[Part 23 - Angular select list validation](https://www.youtube.com/watch?v=TLUP1LhW5bo) | [Text](http://csharp-video-tutorials.blogspot.com/2018/02/angular-select-list-validation.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/02/angular-select-list-validation-slides.html)  
  
In this video we will discuss how to validate a select element if it has a custom option like one of the following.

* Please Select
* Select Department
* Etc...

Consider the following example :

<div class="form-group"

     [class.has-error]="department.touched && department.invalid">

  <label for="department" class="control-label">Department</label>

  <select required #department="ngModel" name="department"

          [(ngModel)]="employee.department" id="department"

          class="form-control">

    <option [ngValue]="null">Select Department</option>

    <option \*ngFor="let dept of departments" [value]="dept.id">

      {{dept.name}}

    </option>

  </select>

  <span class="help-block"

        \*ngIf="department.touched && department.errors?.required">

    Department is required

  </span>

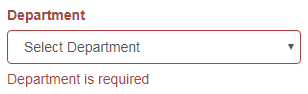
</div>

**Code explanation :**

* <option [ngValue]="null">Select Department</option>. Notice we are using ngValue instead of value. If you use value, null is treated as a string and not as a null. Hence the required validation does not work. Along with using ngValue, also make sure you set the department property on the employee model object to null.
* <option \*ngFor="let dept of departments" [value]="dept.id">{{dept.name}}</option>. Here we are using value instead of ngValue, because we just want the selected department id as a string. If you want the department object itself instead of just the department id string, then use ngValue.
* <option \*ngFor="let dept of departments" [ngValue]="dept">{{dept.name}}</option>. In this example we are using ngValue and binding it to the dept object. If we select a department now, we get the selected department object.  
  "department": { "id": 3, "name": "IT" }
* Use the disabled attribute, if you do not want the user to be able to select the "Select Department" option.   
  <option **disabled**[ngValue]="null">Select Department</option>

**Please note :**  The built-in required validator will only work with the SELECT element, if the default option value is null.  
<option disabled [ngValue]=**"null"**>Select Department</option>   
  
In a real world application, most of the time we load the SELECT list options from a database table. In some case we may also load the default option (like SELECT DEPARTMENT etc) also from the database. In this case the default option value may not be NULL. Depending on your use case it could be -1, or SELECT or something else. So in scenarios like this, built-in required validator does not work with the SELECT element. To make it work, we have to implement our own custom required validator. We will discuss how to do this in our next video.

# Angular custom validator example template driven forms

**Suggested Videos**  
[Part 22 - Angular checkbox validation](https://www.youtube.com/watch?v=qgMtJKFcOpw) | [Text](http://csharp-video-tutorials.blogspot.com/2018/02/angular-checkbox-validation.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/02/angular-checkbox-validation-slides.html)  
[Part 23 - Angular select list validation](https://www.youtube.com/watch?v=TLUP1LhW5bo) | [Text](http://csharp-video-tutorials.blogspot.com/2018/02/angular-select-list-validation.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/02/angular-select-list-validation-slides.html)   
[Part 24 - Angular value vs ngValue](https://www.youtube.com/watch?v=dyif1Xy9GY8) | [Text](http://csharp-video-tutorials.blogspot.com/2018/02/angular-value-vs-ngvalue.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/02/angular-value-vs-ngvalue-slides.html)  
  
In this video, we will discuss creating a **custom validator in angular with an example**. We will make this validator reusable and configurable. Along the way, we will also learn how to create a **custom directive in angular**.   
  
  
**Example :**When you have a default option like **"Select Department"** in a SELECT list, the REQUIRED validation does not work, if the default option value is anything else other than NULL. So if the default option value is not null, then that default option is also treated as a valid selection, and we do not get to see the required validation error. If "Select Department" is selected, we want the validation to fail, and display "Department is required" validation error message. To make this work the way we want, we implement a custom validator.    
  
   
  
  
**Create a custom Directive**  
To use a custom validator in template driven forms, we have to create the validator as a directive. Once we have the directive created, we can then use that directive as an attribute on the select element that we want to validate. This is going to be a configurable and reusable validator. We can use it with any SELECT list in an angular application. So create a **"shared"** folder in the **"app"** folder. In the "shared" folder create a file with name "select-required-validator.directive.ts". Copy and paste the folllowing code.

import { Validator, AbstractControl, NG\_VALIDATORS } from '@angular/forms';

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

@Directive({

    selector: '[appSelectValidator]',

    providers: [

        {

            provide: NG\_VALIDATORS,

            useExisting: SelectRequiredValidatorDirective,

            multi: true

        }]

})

export class SelectRequiredValidatorDirective implements Validator {

    validate(control: AbstractControl): { [key: string]: any } | null {

        return control.value === '-1' ? { 'defaultSelected': true } : null;

    }

}

**Code Exaplanation :**  
Since we are creating a directive, we decorate the class with **@Directive**decorator   
  
**NG\_VALIDATORS** is a collection of validators. It already contains all the built-in validators like required, pattern etc. Before we can use our custom validator we have to add it to the list of validators by adding it to NG\_VALIDATORS token. To specify that we want to add our validator to the list of validators, we set **multi** property to true   
  
providers: [  
    {  
        provide: NG\_VALIDATORS,  
        useExisting: SelectRequiredValidatorDirective,  
        multi: true  
    }]   
  
Implement **Validator** interface as we are creating a custom validator  
export class SelectRequiredValidatorDirective implements Validator  
  
Since we are implementing validator interface, we have to provide implementation for the interface **validate()** method. This method has one input parameter and it's type is **AbstractControl**. AbstractControl extends both **FormControl** and **FormGroup**. In some cases you may want to validate a Formgroup instead of a single FormControl. So to cater for both scenarios, the parent type - AbstractControl is specified. This function returns an object if the validation fails or null if the validation succeeds. The object that is returned when the validation fails contains a key/value pair. The key is a string and the value can be anything.  
validate(control: AbstractControl): { [key: string]: any } | null   
  
If the selected value in the SELECT list is the default value (-1), then we return an object with key 'defaultSelected' and value of true to indicate that the validation has failed. Otherwise we return NULL to indicate validation succeeded. In the HTML we can use the "defaultSelected" key to display the validation error specific to this custom validator.  
return control.value === '-1' ? { 'defaultSelected': true } : null;  
  
**Import the custom directive in a module where you want to use it.**  
  
At the moment we only have one module - Root module. So in app.module.ts file include the following import statement  
import { SelectRequiredValidatorDirective } from './shared/select-required-validator.directive';  
  
Also include SelectRequiredValidatorDirective in the declarations array of the NgModule() decorator  
  
**Using the custom required validator on the SELECT element**  
  
Modify the "Department" SELECT element in create-employee.component.html file as shown below.

<div class="form-group"

     [class.has-error]="department.touched && department.errors?.defaultSelected">

  <label for="department" class="control-label">Department</label>

  <select id="department" #department="ngModel" name="department"

          [(ngModel)]="employee.department" appSelectValidator

          class="form-control">

    <option value="-1">Select Department</option>

    <option \*ngFor="let dept of departments" [value]="dept.id">

      {{dept.name}}

    </option>

  </select>

  <span class="help-block"

        \*ngIf="department.touched && department.errors?.defaultSelected">

    Department is required

  </span>

</div>

**Code Explanation :**

* [class.has-error]="department.touched && department.errors?.defaultSelected". Notice, in this conditional class binding we are using the key "defaultSelected" to style the Department SELECT element and it's label when there is a validation error. This key is set by the required custom validator when the validation fails.
* \*ngIf="department.touched && department.errors?.defaultSelected". Notice here also we are using the key "defaultSelected" to display the validation error message.
* appSelectValidator is the selector we gave for our custom validation directive and we are using it as an attribute on the SELECT list that we want to validate.

At the moment, we can only use this custom validator with a SELECT list whose default option value is -1. We will discuss how to make this custom validator configurable and reusable in our next video.

# Angular select list required custom validator

**Suggested Videos**  
[Part 23 - Angular select list validation](https://www.youtube.com/watch?v=TLUP1LhW5bo) | [Text](http://csharp-video-tutorials.blogspot.com/2018/02/angular-select-list-validation.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/02/angular-select-list-validation-slides.html)  
[Part 24 - Angular value vs ngValue](https://www.youtube.com/watch?v=dyif1Xy9GY8) | [Text](http://csharp-video-tutorials.blogspot.com/2018/02/angular-value-vs-ngvalue.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/02/angular-value-vs-ngvalue-slides.html)   
[Part 25 - Angular custom validator example template driven forms](https://www.youtube.com/watch?v=2AAUf32pKy8) | [Text](http://csharp-video-tutorials.blogspot.com/2018/02/angular-custom-validator-example_27.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/02/angular-custom-validator-example_81.html)  
  
In this video we will discuss, **how to make the select list custom required validator configurable and reusable**. This is continuation to [Part 25](https://www.youtube.com/watch?v=2AAUf32pKy8). Please watch [Part 25](https://www.youtube.com/watch?v=2AAUf32pKy8) from [Angular CRUD tutorial](https://www.youtube.com/playlist?list=PL6n9fhu94yhXwcl3a6rIfAI7QmGYIkfK5) before proceeding.   
  
  
Here is the SELECT list **custom required validator**we implemented in our previous video

import { Validator, AbstractControl, NG\_VALIDATORS } from '@angular/forms';

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

@Directive({

    selector: '[appSelectValidator]',

    providers: [{

        provide: NG\_VALIDATORS,

        useExisting: SelectRequiredValidatorDirective,

        multi: true

    }]

})

export class SelectRequiredValidatorDirective implements Validator {

    validate(control: AbstractControl): { [key: string]: any } | null {

        return control.value === '-1' ? { 'defaultSelected': true } : null;

    }

}

**Consider this line of code :**Notice we have hard-coded the default option value '-1'. Because of this hard-coded value, we will not be able to reuse this validator with another SELECT list if it has a different default option value other than '-1'.    
  
return control.value === '-1' ? { 'defaultSelected': true } : null;  
  
To make this custom validator reusable, we want to be able to do pass the default option value from the template to our custom validator as shown below. Notice we are using our custom validator selector and passing it the default option value. In this case we are passing -101. If you have another SELECT list, and if it's default option value is -1, you simply pass that value.  
<select appSelectValidator="-101" #department="ngModel" ....>  
  
For this to work, we have to create a corresponding input property in the custom validator class. Modify SelectRequiredValidatorDirective as shown below. The changes are commented and self-explanatory

import { Validator, AbstractControl, NG\_VALIDATORS } from '@angular/forms';

// Import input from @angular/core package

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

@Directive({

    selector: '[appSelectValidator]',

    providers: [{

        provide: NG\_VALIDATORS,

        useExisting: SelectRequiredValidatorDirective,

        multi: true

    }]

})

export class SelectRequiredValidatorDirective implements Validator {

    // Create input property to receive the

    // specified default option value

    @Input() appSelectValidator: string;

    validate(control: AbstractControl): { [key: string]: any } | null {

        // Remove the hard-coded value and use the input property instead

        return control.value === this.appSelectValidator ?

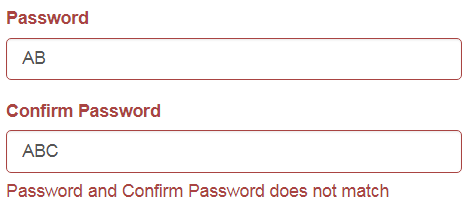
                                    { 'defaultSelected': true } : null;

    }

}

**Please note :**Since this is a directive input property, the input property name and the selector name must match.   
@Input() appSelectValidator: string;  
  
For some reason if you do not like the input property name, you can use an alias as shown below.   
@Input('appSelectValidator') defaultValue: string;  
  
We are now able to specify the default option value using the directive input property. This makes our custom validator configurable and reusable. We can now use this custom required validator to validate any SELECT list in our Angular application.

# Angular password and confirm password validation

**Suggested Videos**  
[Part 24 - Angular value vs ngValue](https://www.youtube.com/watch?v=dyif1Xy9GY8) | [Text](http://csharp-video-tutorials.blogspot.com/2018/02/angular-value-vs-ngvalue.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/02/angular-value-vs-ngvalue-slides.html)  
[Part 25 - Angular custom validator example template driven forms](https://www.youtube.com/watch?v=2AAUf32pKy8) | [Text](http://csharp-video-tutorials.blogspot.com/2018/02/angular-custom-validator-example_27.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/02/angular-custom-validator-example_81.html)   
[Part 26 - Angular select list required custom validator](https://www.youtube.com/watch?v=BjsaaUNw4lk) | [Text](http://csharp-video-tutorials.blogspot.com/2018/02/angular-select-list-required-custom.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/02/angular-select-list-required-custom_28.html)  
  
In this video we will discuss how to **compare password and confirm password fields**and validate if they are equal. If they are not equal we want to display "Password and Confirm Password does not match" validation error.   
  
   
  
  
This is also commonly called as **cross field validation in angular**. We cannot use any of the buil-in angular validators to perform cross-field validation. So let's create a custom validator. To use a custom validator in template driven forms, we create the validator as a directive. We discussed creating custom validators and directives in Parts [25](https://www.youtube.com/watch?v=2AAUf32pKy8&list=PL6n9fhu94yhXwcl3a6rIfAI7QmGYIkfK5&index=25&t=0s) and [26](https://www.youtube.com/watch?v=BjsaaUNw4lk&list=PL6n9fhu94yhXwcl3a6rIfAI7QmGYIkfK5&index=26&t=0s) of [Angular CRUD tutorial](https://www.youtube.com/playlist?list=PL6n9fhu94yhXwcl3a6rIfAI7QmGYIkfK5). If you are new to these concepts, please check out those videos.   
  
We will make this custom validator a reusable validator, so we could use it to compare any 2 input fields for equality. For example, we can use this same custom validator to compare if EMAIL and CONFIRM EMAIL fields are equal.   
  
  
**Create a custom Directive**  
Add a new TypeScript file to the "shared" folder. Name it confirm-equal-validator.directive.ts.Copy and paste the folllowing code.

import { Validator, AbstractControl, NG\_VALIDATORS } from '@angular/forms';

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

@Directive({

    selector: '[appConfirmEqualValidator]',

    providers: [{

        provide: NG\_VALIDATORS,

        useExisting: ConfirmEqualValidatorDirective,

        multi: true

    }]

})

export class ConfirmEqualValidatorDirective implements Validator {

    @Input() appConfirmEqualValidator: string;

    validate(control: AbstractControl): { [key: string]: any } | null {

        const controlToCompare = control.parent.get(this.appConfirmEqualValidator);

        if (controlToCompare && controlToCompare.value !== control.value) {

            return { 'notEqual': true };

        }

        return null;

    }

}

**Code Exaplanation :**  
Since we are creating a directive, we decorate the class with @Directive decorator  
  
This selector will be used as a directive on one of the 2 input fields that we want to compare for equality. In our case we will use it on the **Confirm Password** field.  
selector: '[appConfirmEqualValidator]',  
  
**NG\_VALIDATORS** is a collection of validators. It contains all the built-in validators like required, pattern etc. Before we can use our custom validator we have to add it to the list of validators by adding it to NG\_VALIDATORS token. To specify that we want to add our validator to the list of validators, we set multi property to true  
  
providers: [{  
    provide: NG\_VALIDATORS,  
    useExisting: ConfirmEqualValidatorDirective,  
    multi: true  
}]  
  
Implement Validator interface as we are creating a custom validator  
export class ConfirmEqualValidatorDirective implements Validator  
  
Since our custom validator class is implementing validator interface, we have to provide implementation for the interface validate() method. This method has one input parameter and it's type is AbstractControl. AbstractControl extends both FormControl and FormGroup. In some case you may want to validate a Formgroup instead of a single FormControl. So to cater for both scenarios, the parent type - AbstractControl is specified. This function returns an object if the validation fails or null if the validation succeeds. The object that is returned when the validation fails contains a key/value pair. The key is a string and the value can be anything.  
validate(control: AbstractControl): { [key: string]: any } | null  
  
The following line creates an input property. Since this is a directive input property, the input property name and the selector name must match.   
@Input() appConfirmEqualValidator: string;  
  
We will use this custom directive (appConfirmEqualValidator), as an attribute either on the PASSWORD field or CONFIRM PASSWORD FIELD. If we use this on the  CONFIRM PASSWORD field, we will also pass the field that we want to compare with. In this case the PASSWORD field.   
  
So the input property that we have created above receives the control that we want to compare CONFIRM PASSWORD field with. This input property prevents the need to hard code the name of the control that we want to compare with. Hence it makes our custom validator reusable. We can use it to compare any 2 input fields for equality.  
<input name="confirmPassword" appConfirmEqualValidator="password"

* In the validate() method implementation, we first retrieve the control that we want to compare CONFIRM PASSWORD field with. That field in our case is the PASSWORD field.
* Both PASSWORD and CONFIRM PASSWORD fields are siblings. So to get the PASSWORD field, we go one level up from the CONFIRM PASSWORD field using the parent property. The parent property returns the root FormGroup.
* On the root FormGroup we call the get() method passing it, the input property. The input property receives the name of the PASSWORD field
* Finally we check if the PASSWORD and CONFIRM PASSWORD filed values are equal. If they are equal, we return NULL indication validation succeeded otherwise we return an object with key=notEqual and value=true.
* In the HTML we can use this key (notEqual) to display the relevant validation error message

validate(control: AbstractControl): { [key: string]: any } | null {

    const controlToCompare = control.parent.get(this.appConfirmEqualValidator);

    if (controlToCompare && controlToCompare.value !== control.value) {

        return { 'notEqual': true };

    }

    return null;

}

**Import the custom directive in a module where you want to use it.**  
  
At the moment we only have one module - Root module. So in app.module.ts file include the following import statement  
import { ConfirmEqualValidatorDirective } from './shared/confirm-equal-validator.directive';  
  
Also include ConfirmEqualValidatorDirective in the declarations array of the NgModule() decorator  
  
**Using the custom validator**   
  
Include the following HTML for Password and Confirm Password fields in create-employee.component.html file as shown below.

<div class="form-group"

     [class.has-error]="password.touched && password.invalid">

  <label for="password" class="control-label">Password</label>

  <input id="password" required type="text" class="form-control"

         name="password" [(ngModel)]="employee.password"

         #password="ngModel">

  <span class="help-block"

        \*ngIf="password.touched && password.errors?.required">

    Password is required

  </span>

</div>

<div class="form-group"

     [class.has-error]="confirmPassword.touched && confirmPassword.invalid">

  <label for="confirmPassword" class="control-label">Confirm Password</label>

  <input name="confirmPassword" appConfirmEqualValidator="password" required

         id="confirmPassword" type="text" class="form-control"

         [(ngModel)]="employee.confirmPassword" #confirmPassword="ngModel">

  <span class="help-block"

        \*ngIf="confirmPassword.touched && confirmPassword.errors?.required">

    Confirm Password is required

  </span>

  <span class="help-block"

        \*ngIf="confirmPassword.touched && confirmPassword.errors?.notEqual &&

          !confirmPassword.errors?.required">

    Password and Confirm Password does not match

  </span>

</div>

Notice on the CONFIRM PASSWORD field we are using our custom directive and passing it the name of the control (PASSWORD) that we want to compare with.  
<input name="confirmPassword" appConfirmEqualValidator="password"  
  
Notice the expression of \*ngIf structural directive. We are using the key (notEqual) set by our custom validator to display the validation error message.

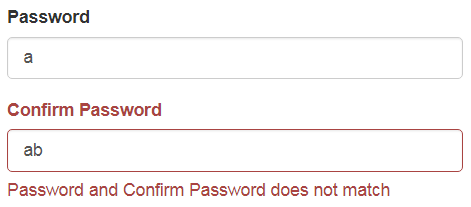
<span class="help-block"

      \*ngIf="confirmPassword.touched && confirmPassword.errors?.notEqual &&

        !confirmPassword.errors?.required">

  Password and Confirm Password does not match

</span>

**At the moment there are 2 problems with our custom validator**  
When the validation fails only the CONFIRM PASSWORD field is styled with red border and not the PASSWORD field. We want the password field also to have the red border.   
  
   
  
If you first change PASSWORD field and then the CONFIRM PASSWORD field, the validation works as expected. Now if you go back and change the PASSWORD field, the validation will not be triggered and you will not see the validation error even if the passwords do not match.   
  
**We will discuss why this is happening and how to fix it in our next video.**

# Angular trigger validation manually

**Suggested Videos**  
[Part 25 - Angular custom validator example template driven forms](https://www.youtube.com/watch?v=2AAUf32pKy8) | [Text](http://csharp-video-tutorials.blogspot.com/2018/02/angular-custom-validator-example_27.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/02/angular-custom-validator-example_81.html)  
[Part 26 - Angular select list required custom validator](https://www.youtube.com/watch?v=BjsaaUNw4lk) | [Text](http://csharp-video-tutorials.blogspot.com/2018/02/angular-select-list-required-custom.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/02/angular-select-list-required-custom_28.html)   
[Part 27 - Angular password and confirm password validation](https://www.youtube.com/watch?v=YhazkQd59Hk) | [Text](http://csharp-video-tutorials.blogspot.com/2018/03/angular-password-and-confirm-password.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/03/angular-password-and-confirm-password_6.html)  
  
**In this video we will discuss**

1. How to add or remove validation styles to a group of elements in Angular
2. How to trigger validation manually in Angular using the updateValueAndValidity() function

This is continuation to [Part 27](https://www.youtube.com/watch?v=YhazkQd59Hk). Please watch [Part 27](https://www.youtube.com/watch?v=YhazkQd59Hk) from [Angular CRUD tutorial](https://www.youtube.com/playlist?list=PL6n9fhu94yhXwcl3a6rIfAI7QmGYIkfK5) before proceeding.   
  
**How to add and remove validation styles to a group of elements in Angular :** Use the **ngModelGroup**directive and group the elements. Now we can add or remove validation styles from the group. This in turn adds or removes the validation styles from all the elements in that group.   
  
  
In our case, we want to **group password and confirm password fields** to be able to control their validation styles. Notice in the example below, both password and confirm password fields are grouped using the **ngModelGroup**directive. The bootstrap validation class **has-error** is conditionally added or removed from the group.

<div ngModelGroup="passwordGroup"

     [class.has-error]="confirmPassword.touched && confirmPassword.invalid">

  <div "passwordFieldDiv"> ...

  </div>

  <div "confirmPasswordFieldDiv"> ...

  </div>

</div>

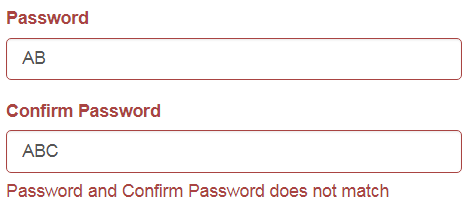
**Use of updateValueAndValidity() function :** At the moment we have a problem with confirm password field validation. It does not work in one of the scenarios. Here is the scenario.   
  
If you first change PASSWORD field and then the CONFIRM PASSWORD field, the validation works as expected. Now if you go back and change the PASSWORD field, the validation will not be triggered and you will not see the validation error even if the passwords do not match.   
  
This is because our custom validation directive is applied on the confirm password filed but not on the password. So our custom validation is triggered only when the confirm password field is changed and not when the password field is changed. To make this work, even when the password field is changed, we have to tell confirm password field to run it's validation when password field is changed.   
  
So the obvious question that comes to our mind is, **how to tell the confirm password field to run it's validation?**  
Well updateValueAndValidity() function comes to the rescue. When this method is called on a control, that control's validation logic is executed again. Notice the event binding in the example below. The **change**event of the password field triggers a call to confirm password field's updateValueAndValidity() function. This in turn runs the confirm password field validation.

<input name="password"

      (change)="confirmPassword.control.updateValueAndValidity()" …>

The **change**event is fired only after the form control has lost focus. The **input**event is fired as the user changes the value. So if you want the validation to trigger as the user is changing the value, use the **input**event instead of **change**event.

# Angular form group validation

**Suggested Videos**  
[Part 26 - Angular select list required custom validator](https://www.youtube.com/watch?v=BjsaaUNw4lk) | [Text](http://csharp-video-tutorials.blogspot.com/2018/02/angular-select-list-required-custom.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/02/angular-select-list-required-custom_28.html)  
[Part 27 - Angular password and confirm password validation](https://www.youtube.com/watch?v=YhazkQd59Hk) | [Text](http://csharp-video-tutorials.blogspot.com/2018/03/angular-password-and-confirm-password.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/03/angular-password-and-confirm-password_6.html)   
[Part 28 - Angular trigger validation manually](https://www.youtube.com/watch?v=lOAb-D_gYHE) | [Text](http://csharp-video-tutorials.blogspot.com/2018/03/angular-trigger-validation-manually.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/03/angular-trigger-validation-manually_7.html)  
  
In this video we will discuss how to validate a group of form controls in Angular. This is continuation to [Part 28](https://www.youtube.com/watch?v=lOAb-D_gYHE). Please watch [Part 28](https://www.youtube.com/watch?v=lOAb-D_gYHE) from [Angular CRUD tutorial](https://www.youtube.com/playlist?list=PL6n9fhu94yhXwcl3a6rIfAI7QmGYIkfK5) before proceeding.   
  
  
In our previous video we discussed one way to implement confirm password validation. There is another way as well. We will discuss that in this video.   
  
  
If the password and confirm password are not equal we want to validate and display "Password and Confirm Password does not match" validation error as shown below.   
  
   
  
**HTML used in the demo**

<div ngModelGroup="passwordGroup" #passwordGroup="ngModelGroup"

      appConfirmEqualValidator [class.has-error]="passwordGroup.errors?.notEqual

      && !confirmPassword.errors?.required">

  <div class="form-group"

        [class.has-error]="password.touched && password.invalid">

    <label for="password" class="control-label">Password</label>

    <input name="password" required type="text" class="form-control"

            [(ngModel)]="employee.password" #password="ngModel">

    <span class="help-block"

          \*ngIf="password.touched && password.errors?.required">

      Password is required

    </span>

  </div>

  <div class="form-group"

        [class.has-error]="confirmPassword.touched && confirmPassword.invalid">

    <label for="confirmPassword" class="control-label">Confirm Password</label>

    <input name="confirmPassword" required type="text" class="form-control"

            [(ngModel)]="employee.confirmPassword" #confirmPassword="ngModel">

    <span class="help-block"

          \*ngIf="confirmPassword.touched && confirmPassword.errors?.required">

      Confirm Password is required

    </span>

    <span class="help-block" \*ngIf="confirmPassword.touched &&

          passwordGroup.errors?.notEqual && !confirmPassword.errors?.required">

      Password and Confirm Password does not match

    </span>

  </div>

</div>

**Custom Validator Code :**

import { Validator, NG\_VALIDATORS, AbstractControl } from '@angular/forms';

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

@Directive({

    selector: '[appConfirmEqualValidator]',

    providers: [{

        provide: NG\_VALIDATORS,

        useExisting: ConfirmEqualValidatorDirective,

        multi: true

    }]

})

export class ConfirmEqualValidatorDirective implements Validator {

    validate(passwordGroup: AbstractControl): { [key: string]: any } | null {

        const passwordField = passwordGroup.get('password');

        const confirmPasswordField = passwordGroup.get('confirmPassword');

        if (passwordField && confirmPasswordField &&

            passwordField.value !== confirmPasswordField.value) {

            return { 'notEqual': true };

        }

        return null;

    }

}

**NgModelGroup Directive**

* Use to create a sub-group within a form
* Useful to validate a sub-group of elements on the form
* Useful to group properties of the form model in to a nested object
* The name of the ngModelGroup will become the key for the nested object in the form model
* The ngModelGroup directive can only be used as a child of NgForm directive

# Angular 5 services tutorial

**Suggested Videos**  
[Part 27 - Angular password and confirm password validation](https://www.youtube.com/watch?v=YhazkQd59Hk) | [Text](http://csharp-video-tutorials.blogspot.com/2018/03/angular-password-and-confirm-password.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/03/angular-password-and-confirm-password_6.html)  
[Part 28 - Angular trigger validation manually](https://www.youtube.com/watch?v=lOAb-D_gYHE) | [Text](http://csharp-video-tutorials.blogspot.com/2018/03/angular-trigger-validation-manually.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/03/angular-trigger-validation-manually_7.html)   
[Part 29 - Angular form group validation](https://www.youtube.com/watch?v=lhl3w5rH7A8) | [Text](http://csharp-video-tutorials.blogspot.com/2018/03/angular-form-group-validation.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/03/angular-form-group-validation-slides.html)  
  
In this video we will discuss **creating a service in angular**. Whether you are using Angular 2, Angular 4 or Angular 5, the steps to create a service are the same. We discussed the basics of Angular services and why we need services in detail in [Parts 25](https://www.youtube.com/watch?v=qAeh4jZTjCk&t=0s&list=PL6n9fhu94yhWqGD8BuKuX-VTKqlNBj-m6&index=25), [27](https://www.youtube.com/watch?v=DETRfgZGuk8&t=0s&list=PL6n9fhu94yhWqGD8BuKuX-VTKqlNBj-m6&index=27) and [34](https://www.youtube.com/watch?v=UA0hWHohSYQ&t=0s&list=PL6n9fhu94yhWqGD8BuKuX-VTKqlNBj-m6&index=34) of [Angular 2 tutorial](https://www.youtube.com/playlist?list=PL6n9fhu94yhWqGD8BuKuX-VTKqlNBj-m6). If you are new to services in Angular, please check out those videos by [clicking here](https://www.youtube.com/playlist?list=PL6n9fhu94yhWqGD8BuKuX-VTKqlNBj-m6).   
  
  
**Creating a service**   
  
Add a new TypeScript file to the **"employees"** folder and name it **employee.service.ts**. Copy and paste the following code. At the moment we have the data hard-coded in the service method. In a later video we will discuss retrieving data from a remote server using HTTP.  

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

import { Employee } from '../models/employee.model';

// The @Injectable() decorator is used to inject other dependencies

// into this service. As our service does not have any dependencies

// at the moment, we may remove the @Injectable() decorator and the

// service works exactly the same way. However, Angular recomends

// to always use @Injectable() decorator to ensures consistency

@Injectable()

export class EmployeeService {

    private listEmployees: Employee[] = [

        {

            id: 1,

            name: 'Mark',

            gender: 'Male',

            contactPreference: 'Email',

            email: 'mark@pragimtech.com',

            dateOfBirth: new Date('10/25/1988'),

            department: 'IT',

            isActive: true,

            photoPath: 'assets/images/mark.png'

        },

        {

            id: 2,

            name: 'Mary',

            gender: 'Female',

            contactPreference: 'Phone',

            phoneNumber: 2345978640,

            dateOfBirth: new Date('11/20/1979'),

            department: 'HR',

            isActive: true,

            photoPath: 'assets/images/mary.png'

        },

        {

            id: 3,

            name: 'John',

            gender: 'Male',

            contactPreference: 'Phone',

            phoneNumber: 5432978640,

            dateOfBirth: new Date('3/25/1976'),

            department: 'IT',

            isActive: false,

            photoPath: 'assets/images/john.png'

        },

    ];

    getEmployees(): Employee[] {

        return this.listEmployees;

    }

}

**Registering the service**   
  
**A service in angular can be registered in a component or in a module**. When a service is registered at a component level, then that service is available only to that component and any of it's children. It's not available to the other components.    
  
One the other hand, if we register a service at a module level, then that service is registered with the root injector and available to all the components in our application. We want our EmployeeService to be available in several components (like employee list component, edit component etc). So let's register it in the root module - AppModule.   
  
We discussed dependency injection and injectors in [Parts 32](https://www.youtube.com/watch?v=jWODteEGQmw&t=0s&list=PL6n9fhu94yhWqGD8BuKuX-VTKqlNBj-m6&index=32), [33](https://www.youtube.com/watch?v=EzioZ9cww08&t=0s&list=PL6n9fhu94yhWqGD8BuKuX-VTKqlNBj-m6&index=33), [35](https://www.youtube.com/watch?v=dH1EWRoEcMo&t=0s&list=PL6n9fhu94yhWqGD8BuKuX-VTKqlNBj-m6&index=35) and [36](https://www.youtube.com/watch?v=qy-0yD7hDVA&t=0s&list=PL6n9fhu94yhWqGD8BuKuX-VTKqlNBj-m6&index=36) in [Angular 2 tutorial](https://www.youtube.com/playlist?list=PL6n9fhu94yhWqGD8BuKuX-VTKqlNBj-m6). If you are new to injectors and dependency injection in Angular, please check out those videos by [clicking here](https://www.youtube.com/playlist?list=PL6n9fhu94yhWqGD8BuKuX-VTKqlNBj-m6).   
  
Include the following import statement to import EmployeeService in app.module.ts file  
import { EmployeeService } from './employees/employee.service';   
  
Also make sure to include EmployeeService in the providers array of @NgModuledecorator.   
  
**Injecting and using the service**   
  
We need the employee service we created above in ListEmployeesComponent. So let's import and use the Employee service in ListEmployeesComponent. Modify the code in list-employees.component.ts file as shown below.

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

import { Employee } from '../models/employee.model';

// Import EmployeeService

import { EmployeeService } from './employee.service';

@Component({

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

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

})

export class ListEmployeesComponent implements OnInit {

  employees: Employee[];

  // Inject EmployeeService using the constructor

  // The private variable \_employeeService which points to

  // EmployeeService singelton instance is then available

  // throughout the class and can be accessed using this keyword

  constructor(private \_employeeService: EmployeeService) { }

  // Call the getEmployees() service method of EmployeeService

  // using the private variable \_employeeService

  ngOnInit() {

    this.employees = this.\_employeeService.getEmployees();

  }

}

**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

In our next video, we will discuss creating a new Employee.

# Create operation in angular

**Suggested Videos**  
[Part 28 - Angular trigger validation manually](https://www.youtube.com/watch?v=lOAb-D_gYHE) | [Text](http://csharp-video-tutorials.blogspot.com/2018/03/angular-trigger-validation-manually.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/03/angular-trigger-validation-manually_7.html)  
[Part 29 - Angular form group validation](https://www.youtube.com/watch?v=lhl3w5rH7A8) | [Text](http://csharp-video-tutorials.blogspot.com/2018/03/angular-form-group-validation.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/03/angular-form-group-validation-slides.html)   
[Part 30 - Angular 5 services tutorial](https://www.youtube.com/watch?v=4lSvgj8ohAI) | [Text](http://csharp-video-tutorials.blogspot.com/2018/03/angular-5-services-tutorial.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/03/angular-5-services-tutorial-slides.html)  
  
In this video we will discuss creating a new employee using the Create Employee form. This is continuation to [Part 30](https://www.youtube.com/watch?v=4lSvgj8ohAI&index=30&list=PL6n9fhu94yhXwcl3a6rIfAI7QmGYIkfK5&t=0s). Please watch [Part 30](https://www.youtube.com/watch?v=4lSvgj8ohAI&index=30&list=PL6n9fhu94yhXwcl3a6rIfAI7QmGYIkfK5&t=0s) from [Angular CRUD tutorial](https://www.youtube.com/playlist?list=PL6n9fhu94yhXwcl3a6rIfAI7QmGYIkfK5) before proceeding.   
  
  
**Modify EmployeeService to include save() method**   
  
Include the following save() method in EmployeeService in **employee.service.ts** file. At the moment, the save() method just pushes the employee object into the employees array. In a later video, we will discuss saving the employee to a database table by calling a web service over http.   
  
  
save(employee: Employee) {  
    this.listEmployees.push(employee);  
}   
  
**Injecting and using EmployeeService in CreateEmployeeComponent**  
  
Make the following changes in **create-employee.component.ts** file   
  
Include the following import statement to import EmployeeService  
import { EmployeeService } from './employee.service';  
  
Also, import Angular Router service. We will use this service's navigate() method to redirect the user to the employee list page after saving the employee.  
import { Router } from '@angular/router';  
  
Inject EmployeeService and Router service into the CreateEmployeeComponent using it's constructor.  
constructor(private \_employeeService: EmployeeService,  
            private \_router: Router)  
  
**Modify saveEmployee() method** as shown below. Notice the saveEmployee() method calls the EmployeeService save() method passing it the employee object we want to save. Wait a minute, the employee object has all it's properties initialised to NULL. How do we get the values from the controls on the form to the properties of this employee object.  
  
Well, we do not have to write any code to keep employee object properties and the form control values in sync. Angular's two way data-binding does that for us automatically.  
  
Finally, we redirect the user to the "list" route using the **navigate()** method of the angular **Router**service.  
  
saveEmployee(): void {  
  this.\_employeeService.save(this.employee);  
  this.\_router.navigate(['list']);  
}  
  
Also, modify the call to saveEmployee() method in create-employee.component.html file as shown below. Notice we removed the employee object that was being passed as a paramter.

<form #employeeForm="ngForm" (ngSubmit)="saveEmployee()">

At the moment, there is a small issue with data on the employee list page. Notice for the existing employees the department name is displayed where as for the new employeees department id is displayed. We will discuss how to fix this in our next video.

# Angular switch case example

**Suggested Videos**  
[Part 29 - Angular form group validation](https://www.youtube.com/watch?v=lhl3w5rH7A8) | [Text](http://csharp-video-tutorials.blogspot.com/2018/03/angular-form-group-validation.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/03/angular-form-group-validation-slides.html)  
[Part 30 - Angular 5 services tutorial](https://www.youtube.com/watch?v=4lSvgj8ohAI) | [Text](http://csharp-video-tutorials.blogspot.com/2018/03/angular-5-services-tutorial.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/03/angular-5-services-tutorial-slides.html)   
[Part 31 - Create operation in angular](https://www.youtube.com/watch?v=CUAHJxWGia0) | [Text](http://csharp-video-tutorials.blogspot.com/2018/03/create-operation-in-angular.html) | [Slides](http://csharp-video-tutorials.blogspot.com/2018/03/create-operation-in-angular-slides.html)  
  
In this video, we will discuss **angular ngSwitch directive**. Let us understand switch case with an example. Switch case in angular is a combination of 3 directives 

* ngSwitch directive
* ngSwitchCase directive
* ngSwitchDefault directive

Consider the following **department**data. Depending on the employee's department id we want to display department name.   
  
departments: Department[] = [  
  { id: 1, name: 'Help Desk' },  
  { id: 2, name: 'HR' },  
  { id: 3, name: 'IT' },  
  { id: 4, name: 'Payroll' }  
];   
  
  
**ngSwitch example** 

<div [ngSwitch]="employee.department">

  <span \*ngSwitchCase="1"> Help Desk </span>

  <span \*ngSwitchCase="2"> HR </span>

  <span \*ngSwitchCase="3"> IT </span>

  <span \*ngSwitchCase="4"> Payroll </span>

  <span \*ngSwitchDefault> N/A </span>

</div>

**Code Explanation** 

* [ngSwitch]="employee.department". The expression (employee.department) returns department id (1, 2 , 3 etc)
* <span \*ngSwitchCase="1"> Help Desk </span>. If the department id is 1, then this switchcase is executed and it displays Help Desk
* <span \*ngSwitchDefault> N/A </span>. The default switch case is executed if the department id is not 1, 2, 3 and 4
* Jus like ngIf and ngFor, the directives ngSwicthCase and ngSwitchDefaultare also structural directives, hence they have an asterisk in front of them
* If multiple ngSwitchCases match the switch expression value, then all those ngSwitchCases are displayed