**Angular Template-driven Forms**

**is one of the two ways of building forms in Angular. In this tutorial, we will learn how to build a simple Template-driven forms example app. First, we build a simple HTML form using a few form elements. The (ngForm) directive will convert it to the Template-driven form and create the top-level FormGroup control.**

**🡪, we use the ngModel directive to create the FormControl instance for each of the HTML form elements.**

**Later, we will learn how to submit the form data to the component class. We will also learn how to initialize or reset the form data and use the data binding to access the data in the component class.**

**.**

**Table of Contents**

* [**What is Template-driven form?**](https://www.tektutorialshub.com/angular/angular-template-driven-forms/#what-is-template-driven-form)
* [**Create the Example Application**](https://www.tektutorialshub.com/angular/angular-template-driven-forms/#create-the-example-application)
  + [**Import FormsModule**](https://www.tektutorialshub.com/angular/angular-template-driven-forms/#import-formsmodule)
  + [**HTML Form**](https://www.tektutorialshub.com/angular/angular-template-driven-forms/#html-form)
  + [**ngForm**](https://www.tektutorialshub.com/angular/angular-template-driven-forms/#ngform)
  + [**FormControl**](https://www.tektutorialshub.com/angular/angular-template-driven-forms/#formcontrol)
  + [**Submit Form**](https://www.tektutorialshub.com/angular/angular-template-driven-forms/#submit-form)
  + [**Final Template**](https://www.tektutorialshub.com/angular/angular-template-driven-forms/#final-template)
  + [**Receive Form Data**](https://www.tektutorialshub.com/angular/angular-template-driven-forms/#receive-form-data)
* [**Local Variable**](https://www.tektutorialshub.com/angular/angular-template-driven-forms/#local-variable)
  + [**ngForm**](https://www.tektutorialshub.com/angular/angular-template-driven-forms/#ngform-2)
  + [**FormControl**](https://www.tektutorialshub.com/angular/angular-template-driven-forms/#formcontrol-2)
* [**Nested FormGroup**](https://www.tektutorialshub.com/angular/angular-template-driven-forms/#nested-formgroup)
* [**Setting the Initial Value**](https://www.tektutorialshub.com/angular/angular-template-driven-forms/#setting-the-initial-value)
* [**Validating the Form**](https://www.tektutorialshub.com/angular/angular-template-driven-forms/#validating-the-form)
* [**Summary**](https://www.tektutorialshub.com/angular/angular-template-driven-forms/#summary)

**What is Template-driven form?**

**In Template Driven Forms we specify behaviors/validations using directives and attributes in our template**

***and let it work behind the scenes. All things happen in Templates hence very little code is required in the component class.***

**The Template-driven forms**

1. **The form is set up using ngForm directive**
2. **controls are set up using the ngModel directive**
3. **ngModel also provides the two-way data binding**
4. **The Validations are configured in the template via directives**

**Template-driven forms are**

1. **Contains little code in the component class**
2. **Easier to set up**

**While they are**

1. **Difficult to add controls dynamically**
2. **Unit testing is a challenge**

**Create the Example Application**

**Use ng new to create a new application**

|  |  |
| --- | --- |
|  | **ng new tdf  --routing=true --style=css** |

**Run ng serve and verify if everything is installed correctly.**

**Import FormsModule**

**To work with Template-driven forms, we must import the FormsModule.**

**We usually import it in root module or in a**[**shared module**](https://www.tektutorialshub.com/angular/angular-folder-structure-best-practices/#shared-module)**. The FormsModule contains all the form directives and constructs for working with forms**

**Open the app.module.ts and**

**add the import { FormsModule } from '@angular/forms'; to it.**

**And also add the FormsModule to the *imports metadata property array***

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16**  **17**  **18**  **19**  **20**  **21**  **22** | **import { BrowserModule } from '@angular/platform-browser';**  **import { NgModule } from '@angular/core';**  **import { FormsModule } from '@angular/forms';        //import FormsModule**    **import { AppRoutingModule } from './app-routing.module';**  **import { AppComponent } from './app.component';**    **@NgModule({**  **declarations: [**  **AppComponent**  **],**  **imports: [**  **BrowserModule,**  **AppRoutingModule,**  **FormsModule                    //Add in Imports Array**  **],**  **providers: [],**  **bootstrap: [AppComponent]**  **})**  **export class AppModule { }** |

**HTML Form**

**The first task is to build the template. The following is a regular HTML form. We enclose it in a <form> tag. We have included two text input (FirstName & LastName), a email (email), a radio button (gender), a checkbox (isMarried), and a select list (country). These are form elements.**

|  |  |
| --- | --- |
|  | **<form>**    **<p>**  **<label for="firstname">First Name</label>**  **<input type="text" id="firstname" name="firstname">**  **</p>**    **<p>**  **<label for="lastname">Last Name</label>**  **<input type="text" id="lastname" name="lastname">**  **</p>**    **<p>**  **<label for="email">Email </label>**  **<input type="text" id="email" name="email">**  **</p>**    **<p>**  **<label for="gender">Geneder</label>**  **<input type="radio" value="male" id="gender" name="gender"> Male**  **<input type="radio" value="female" id="gender" name="gender"> Female**  **</p>**    **<p>**  **<label for="isMarried">Married</label>**  **<input type="checkbox" id="isMarried" name="isMarried">**  **</p>**    **<p>**  **<label for="country">country </label>**  **<select name="country" id="country">**  **<option selected="" value=""></option>**  **<option [ngValue]="c.id" \*ngFor="let c of countryList">**  **{{c.name}}**  **</option>**  **</select>**  **</p>**    **<p>**  **<button type="submit">Submit</button>**  **</p></form>** |

**Component Class**

|  |  |
| --- | --- |
|  | **import { Component } from '@angular/core';**    **@Component({**  **selector: 'app-root',**  **templateUrl: './app.component.html',**  **styleUrls: ['./app.component.css']**  **})**  **export class AppComponent {**  **title = 'Template driven forms';**    **countryList:country[] = [**  **new country("1", "India"),**  **new country('2', 'USA'),**  **new country('3', 'England')**  **];**  **}**    **export class country {**  **id:string;**  **name:string;**    **constructor(id:string, name:string) {**  **this.id=id;**  **this.name=name;**  **}**  **}** |

**ngForm**

**Once, we have a form with few form elements, the angular automatically converts it into a Template-driven form. This is done by the ngForm directive.**

**The ngForm directive is what makes the Angular template-driven forms work. But we do not need to add it explicitly. Angular adds it automatically**

**When we include FormsModule, the Angular is going to look out for any <form> tag in our HTML template. Angular does this via ngForm**[**directive**](https://www.tektutorialshub.com/angular/angular-directives/)**.**

**ngForm directive automatically detects the <form> tag and automatically binds to it. You do not have to do anything on your part to invoke and bind the ngForm directive.**

**The ngForm does the following**

1. **Binds itself to the <Form> directive**
2. **Creates a top-level FormGroup instance**
3. **CreatesFormControl instance for each of child control, which has ngModel directive.**
4. **CreatesFormGroup instance for each of the  NgModelGroup directive.**

**We can export the ngForm instance into a local template variable using ngForm as the key (ex: #contactForm="ngForm"). This allows us to access the many properties and methods of ngForm using the template variable contactForm**

**Hence, update the form element as shown below.**

|  |  |
| --- | --- |
| **1**  **2**  **3** | **<form #contactForm="ngForm">** |

**FormControl**

**The FormControl is the basic building block of the**[**Angular Forms**](https://www.tektutorialshub.com/angular/angular-forms-fundamentals/)**. It represents a single input field in an**[**Angular form**](https://www.tektutorialshub.com/angular/angular-forms-fundamentals/)**. The**[**Angular Forms Module**](https://www.tektutorialshub.com/angular/angular-forms-fundamentals/#angular-forms-module)**binds the input element to a FormControl. We use the FormControl instance to track the**

**value, user interaction and validation status of an individual form element. Each individual Form element is a FormControl**

**We have six form elements in our HTML template. They are firstName, lastname, email, gender, isMarried & country. We need to bind them to FormControl instance. We do this by using the ngModel directive. Add the ngModel directive to each control as shown below.**

|  |  |
| --- | --- |
| **1**  **2**  **3** | **<input type="text" name="firstname" ngModel>** |

**ngModel will use the name attribute to create the FormControl instance for each of the Form field it is attached.**

**Submit Form**

**Now have the template ready, except for the final piece i.e submitting data to the component.**

**We use the ngSubmit event, to submit the form data to the component class. We use the**[**event binding**](https://www.tektutorialshub.com/angular/angular-data-binding/)**(parentheses) to bind ngSubmit to OnSubmit method in the component class. When the user clicks on the submit button, the ngSubmit event will fire**

|  |  |
| --- | --- |
| **1**  **2**  **3** | **<form #contactForm="ngForm" (ngSubmit)="onSubmit(contactForm)">** |

**We are passing the local template variable contactForm in onSubmit method. contactForm holds the reference to the ngForm directive. We can use this in our component class to extract the data from the form fields.**

**Final Template**

**Our final template is as shown below.**

|  |  |
| --- | --- |
|  | **<form #contactForm="ngForm" (ngSubmit)="onSubmit(contactForm)">**    **<p>**  **<label for="firstname">First Name</label>**  **<input type="text" name="firstname" ngModel>**  **</p>**    **<p>**  **<label for="lastname">Last Name</label>**  **<input type="text" name="lastname" ngModel>**  **</p>**    **<p>**  **<label for="email">Email </label>**  **<input type="text" id="email" name="email" ngModel>**  **</p>**    **<p>**  **<label for="gender">Geneder</label>**  **<input type="radio" value="male" name="gender" ngModel> Male**  **<input type="radio" value="female" name="gender" ngModel> Female**  **</p>**    **<p>**  **<label for="isMarried">Married</label>**  **<input type="checkbox" name="isMarried" ngModel>**  **</p>**    **<select name="country" ngModel>**  **<option [ngValue]="c.id" \*ngFor="let c of countryList">**  **{{c.name}}**  **</option>**  **</select>**    **<p>**  **<button type="submit">Submit</button>**  **</p>**    **</form>** How does ngModel work? Notice that each form control has the ngModel directive applied to it. This directive will bind to the corresponding HTML element, in this case the two input fields first name and password.  The ngModel directive will keep track of the value typed in by the user with each key pressed, and it will also keep track of the validity state of that particular form control only.  The ngForm parent directive will then interact with all its child ngModel directives, and build a model of the whole form, with all its field values and validity states. How does ngSubmit work? The form will trigger the component method onSubmit() on submission, but the submit button is only enabled if both required fields are filled in.  The component class where onSubmit() is defined will then get access to the latest data via the user member variable.  Notice that the submission of this form will not trigger a backend HTTP POST request, like in the case of a plain HTTP form submit.  The ngSubmit directive will ensure that this submission does not occur, and instead that the onSubmit() method gets called.  The ngSubmit directive allows us to access the native form submission event if we need to, via $event. But other than that, it works just like if we would have made the submit button a plain button (without type=submit) and added it a click handler instead.  But all of this is only a small part of what is going on here. |

**Receive Form Data**

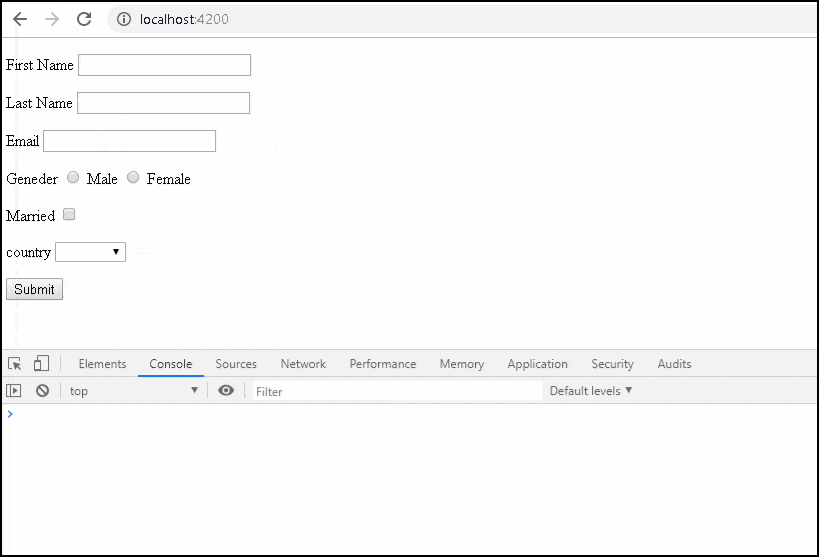
**We need to receive the data in component class from our form. To do this we need to create the onSubmit method in our component class. The submit method receives the reference to the ngForm directive, which we named is as contactForm. The contactForm exposes the value method which returns the form fields as a Json object.**

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5** | **onSubmit(contactForm) {**  **console.log(contactForm.value);**  **}** |

**You can print the value to the console using the console.log(contactForm.value)**

**Run the code now and enter some data into the form. Open the Developer Console in your browser and check the output, when you submit the data.**

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8** | **country: "1"**  **firstname: "Sachin"**  **email:"sachin@gmail.com"**  **gender: "male"**  **isMarried: true**  **lastname: "Tendulkar"** |

**Angular template-driven forms in Action**

**Local Variable**

**We can assign the ngForm,FormControl or FormGroup instance to a template local variable. This allows us to check the status of the form like whether the form is valid, submitted, and value of the form elements, etc**

**ngForm**

**We have access to the ngForm instance via the local template variable #contactForm.**

|  |  |
| --- | --- |
| **1**  **2**  **3** | **<form #contactForm="ngForm" (ngSubmit)="onSubmit(contactForm)">** |

**Now, we can make use of some of the properties & methods to know the status of form. For Example**

|  |  |
| --- | --- |
|  | **<p>**  **<button type="submit">Submit</button>**  **</p>**    **<pre>Value : {{contactForm.value | json }} </pre>**  **<pre>Valid : {{contactForm.valid}} </pre>**  **<pre>Touched : {{contactForm.touched  }} </pre>**  **<pre>Submitted : {{contactForm.submitted  }} </pre>** |

**value: The value property returns the object containing the value of every FormControl  
valid: Returns true if the form is Valid else returns false.  
touched: True if the user has entered a value in at least in one field.  
submitted: Returns true if the form is submitted. else false.**

**FormControl**

**Similarly, we can also get access to the FormControl instance by assigning the ngModel to a local variable as shown below**

|  |  |
| --- | --- |
| **1**  **2**  **3** | **<input type="text" name="firstname" #fname="ngModel" ngModel>** |

**Now, the variable #fname holds the reference to the firstname FormControl. We can then access the properties of FormControl like value, valid, isvalid, tocuhed etc**

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11** | **<p>**  **<label for="firstname">First Name </label>**  **<input type="text" name="firstname" #fname="ngModel" ngModel>**  **</p>**    **<pre>Value    : {{fname.value}} </pre>**  **<pre>valid    : {{fname.valid}} </pre>**  **<pre>invalid  : {{fname.invalid}} </pre>**  **<pre>touched  : {{fname.touched}} </pre** |

**value: Returns the current value of the control  
valid: Returns true if the value is Valid else false  
invalid: True if the value is invalid else false  
touched: Returns true if the value is entered in the element**

**Nested FormGroup**

**The FormGroup is a collection of FormControl. It can also contain other FormGroup's.**

**The ngForm directive creates the top Level FormGroup behind the scene, when we use the <Form> directive.**

|  |  |
| --- | --- |
| **1**  **2**  **3** | **<form #contactForm="ngForm" (ngSubmit)="onSubmit(contactForm)">** |

**We can add new [FormGroup](https://www.tektutorialshub.com/angular/formgroup-in-angular/) using the ngModelGroup directive. Let us add street, city & Pincode form controls and group them under the address FormGroup**

**All you need to do is to enclose the fields inside a div element with ngModelGroup directive applied on it as shown below**

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16**  **17**  **18**  **19** | **<div ngModelGroup="address">**    **<p>**  **<label for="city">City</label>**  **<input type="text" name="city" ngModel>**  **</p>**    **<p>**  **<label for="street">Street</label>**  **<input type="text" name="street" ngModel>**  **</p>**  **<p>**  **<label for="pincode">Pin Code</label>**  **<input type="text" name="pincode" ngModel>**  **</p>**    **</div>** |

**Run the App and submit. The resultant object is as shown below.**

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15** | **Value : {**  **"firstname": "Sachin",**  **"lastname": "Tendulkar",**  **"email":"sachin@gmail.com"**  **"gender": "male",**  **"isMarried": true,**  **"country": "1",**  **"address": {**  **"city": "Mumbai",**  **"street": "Fashin Street",**  **"pincode": "400600"**  **}**  **}** |

**Setting the Initial Value**

**The form is usually pre-filled with some default data. In the case of editing, we have to show the user the current data..**

**Validating the Form**

**Validating the form is another important task. We have covered it in Validation in template-driven form tutorial.**

**Summary**

**Angular Template-driven Forms is simpler compared to the reactive forms. The FormsModule is imported first. Then we create the HTML form. The Angular detects the <form> tag and converts the form to the Angular Form. ngModel directive added to each form element, which converts them to FormControl. Finally, submit event is subscribed via event binding.**

## Template-driven Form Validation

Angular is actually tracking three separate form field states for us and applying the following CSS classes to both the form and its controls:

* ng-touched or ng-untouched
* ng-valid or ng-invalid
* ng-pristine or ng-dirty

All of these CSS class pairs are mutually exclusive, and they are very useful for styling form error states, both at the individual form control level but also at the level of the whole form.

Validations in [Template-driven forms](https://www.tektutorialshub.com/angular/angular-template-driven-forms/) are provided by the Validation directives.

The [Angular Forms Module](https://www.tektutorialshub.com/angular/angular-forms-fundamentals/) comes with several built-in validators. You can also create your own custom Validator.

## Template

Consider the following template-driven form. It has firstname, lastname, email, gender & istoc form fields.

|  |  |
| --- | --- |
|  | <form #contactForm="ngForm" (ngSubmit)="onSubmit(contactForm)">      <p>      <label for="firstname">First Name </label>      <input type="text" id="firstname" name="firstname" [(ngModel)]="contact.firstname">    </p>      <p>      <label for="lastname">Last Name </label>      <input type="text" id="lastname" name="lastname" [(ngModel)]="contact.lastname">    </p>      <p>      <label for="email">email </label>      <input type="text" id="email" name="email" [(ngModel)]="contact.email">    </p>      <p>      <label for="gender">Geneder </label>      <input type="radio" value="male" id="gender" name="gender" [(ngModel)]="contact.gender"> Male      <input type="radio" value="female" id="gender" name="gender" [(ngModel)]="contact.gender"> Female    </p>      <p>      <label for="isToc">Accept TOC</label>      <input type="checkbox" id="isToc" name="isToc" [(ngModel)]="contact.isToc">    </p>      <p>      <button type="submit">Submit</button>    </p>    </form> |

## ****Component Class****

|  |  |
| --- | --- |
|  | import { Component, ViewChild, ElementRef, OnInit } from '@angular/core';  import { NgForm } from '@angular/forms';    @Component({    selector: 'app-root',    templateUrl: './app.component.html',    styleUrls: ['./app.component.css']  })  export class AppComponent implements OnInit  {    title = 'Template driven forms';      @ViewChild('contactForm',null) contactForm: NgForm;      contact:contact;      ngOnInit() {        this.contact = {        firstname:"",        lastname:"",        gender:"male",        isToc:true,        email:"",      };      }      onSubmit() {      console.log(this.contactForm.value);    }    }    export class contact {    firstname:string;    lastname:string;    gender:string;    isToc:boolean;    email:string;  } |

## Disabling the Browser validation

First, we need to disable browser validator interfering with the Angular validator.

To do that we need to add novalidate attribute on **<form>** element as shown below

|  |  |
| --- | --- |
|  | <form #contactForm="ngForm" (ngSubmit)="onSubmit(contactForm)" novalidate> |

## Built-in Validators

The Built-in validators use the HTML5 validation attributes like required, minlength, maxlength & pattern. Angular interprets these validation attributes and add the validator functions to [FormControl](https://www.tektutorialshub.com/angular/formcontrol-in-angular/) instance.

## Adding in Built-in Validators

### Required Validation

The required validator returns true only if the form control has non-empty value entered. Let us add this validator to all fields

|  |  |
| --- | --- |
|  | **<input type="text" id="firstname" name="firstname" required [(ngModel)]="contact.firstname">** |

### 

### Minlength Validation

This Validator requires the control value must not have less number of characters than the value specified in the validator.

For Example, minlength validator ensures that the firstname value has at least 10 characters.

|  |  |
| --- | --- |
|  | <input type="text" id="firstname" name="firstname" required minlength="10" [(ngModel)]="contact.firstname"> |

### Maxlength Validation

This Validator requires that the number of characters must not exceed the value of the attribute.

|  |  |
| --- | --- |
|  | <input type="text" id="lastname" name="lastname" required maxlength="15" [(ngModel)]="contact.lastname"> |

### Pattern Validation

This Validator requires that the control value must match the regex pattern provided in the attribute. For example, the pattern ^[a-zA-Z]+$ ensures that the only letters are allowed (even spaces are not allowed). Let us apply this pattern to the lastName

|  |  |
| --- | --- |
|  | <input type="text" id="lastname" name="lastname" required maxlength="15"      pattern="^[a-zA-Z]+$" [(ngModel)]="contact.lastname"> |

### Email Validation

This Validator requires that the control value must be a valid email address. We apply this to the email field

|  |  |
| --- | --- |
|  | <input type="text" id="email" name="email" required email [(ngModel)]="contact.email"> |

## Disable Submit button

Now, we have successfully added the validators. You will notice that the click submit button still submits the form.

We need to disable the submit button if our form is not valid.

Angular forms module keep track of the state of our form and each of its form elements. These states are exposed to the user through FormGroup, FormArray & FormControl objects.

We get the reference to the top-level FormGroup instance by creating a template variable and bind it to ngForm. We have already done it when we had added the #contactForm="ngForm" in our form tag.

The [FormGroup](https://www.tektutorialshub.com/angular/formgroup-in-angular/) has a valid property, which is set to true if all of its child controls are valid. We use it to set the disabled attribute of the submit button.

|  |  |
| --- | --- |
|  | <button type="submit" [disabled]="!contactForm.valid">Submit</button> |

So long as contactForm.valid remains false, the submit button remains disabled.

## Displaying the Validation/Error messages

We need to provide a short and meaningful error message to the user.

Angular creates a FormControl for each and every field, which has ngModel directive applied. The FormControl exposes the state of form element like valid, dirty, touched, etc.

There are two ways in which you can get the reference to the FormControl.

One way is to use the **contactForm variable.** Like #contactForm

We can use the

(1) **contactForm.controls.firstname.valid**

 to find out if the firstname is valid.

(2)The other way is to create a new local variable for each FormControl For Example,

the following firstname="ngModel" creates the firstname variable with the FormControl instance.

|  |  |
| --- | --- |
|  | <input type="text" id="firstname" name="firstname" required minlength="10"              #firstname="ngModel" [(ngModel)]="contact.firstname"> |

Now, we have a reference to the firstname FormControl instance, we can check its status. We use the valid property to check if the firstname has any errors.

valid: returns either invalid status or null which means a valid status

|  |  |
| --- | --- |
|  | <div \*ngIf="!firstname?.valid && (firstname?.dirty || firstname?.touched)">     Invalid First Name  </div> |

### Why check dirty and touched?

We do not want the application to display the error when the form is displayed for the first time. We want to display errors only after the user has attempted to change the value.

The dirty & touched properties help us do that.

**dirty:** A control is dirty if the user has changed the value in the UI.  
**touched:** A control is touched if the user has triggered a blur event on it.

### Error message

The error message ” “Invalid First Name” ” is not helpful. The firstname has **two validators. required and minlength**

Any errors generated by the failing validation is updated in the errors object. The errors object returns the error object or null if there are no errors.

|  |  |
| --- | --- |
|  | <div \*ngIf="!firstname?.valid && (firstname?.dirty || firstname?.touched)">    Invalid First Name    <div \*ngIf="firstname.errors.required">       First Name is required    </div>    <div \*ngIf="firstname.errors.minlength">      First Name Minimum Length is {{firstname.errors.minlength?.requiredLength}}    </div>  </div> |

Note that the minlength validators return the {{firstname.errors.minlength?.requiredLength}}, which we use the display the error message.

## Final Template

|  |  |
| --- | --- |
|  | <form #contactForm="ngForm" (ngSubmit)="onSubmit(contactForm)" novalidate>      <p>      <label for="firstname">First Name </label>    <input type="text" id="firstname" name="firstname" required minlength="10" #firstname="ngModel"        [(ngModel)]="contact.firstname">    </p>    <div \*ngIf="!firstname?.valid && (firstname?.dirty || firstname?.touched)" class="error">      <div \*ngIf="firstname.errors.required">        First Name is required      </div>      <div \*ngIf="firstname.errors.minlength">        First Name Minimum Length is {{firstname.errors.minlength?.requiredLength}}      </div>    </div>      <p>      <label for="lastname">Last Name </label>      <input type="text" id="lastname" name="lastname" required maxlength="15" #lastname="ngModel"              pattern="^[a-zA-Z]+$"  [(ngModel)]="contact.lastname">    </p>    <div \*ngIf="!lastname?.valid && (lastname?.dirty || lastname?.touched)" class="error">      <div \*ngIf="lastname.errors.required">        Last Name is required      </div>      <div \*ngIf="lastname.errors.maxlength">        Last Name Minimum Length is {{lastname.errors.maxlength?.requiredLength}}      </div>      <div \*ngIf="lastname.errors.pattern">        Only characters are allowed      </div>    </div>          <p>      <label for="email">email </label>      <input type="text" id="email" name="email" required email #email="ngModel" [(ngModel)]="contact.email">    </p>    <div \*ngIf="!email?.valid && (email?.dirty || email?.touched)" class="error">      <div \*ngIf="email.errors.required">        Email is required      </div>      <div \*ngIf="email.errors.email">        Invalid Email Address      </div>    </div>      <p>      <label for="gender">Geneder </label>      <input type="radio" value="male" id="gender" name="gender" #gender="ngModel" required [(ngModel)]="contact.gender">      Male      <input type="radio" value="female" id="gender" name="gender" #gender="ngModel" required        [(ngModel)]="contact.gender"> Female    </p>    <div \*ngIf="!gender?.valid && (gender?.dirty || gender?.touched)" class="error">      <div \*ngIf="gender.errors.required">        Gender is required      </div>    </div>        <p>      <label for="isToc">Accept TOC</label>      <input type="checkbox" id="isToc" name="isToc" required #isToc="ngModel" [(ngModel)]="contact.isToc">    </p>    <div \*ngIf="!isToc?.valid && (isToc?.dirty || isToc?.touched)" class="error">      <div \*ngIf="isToc.errors.required">        Please accept the TOC      </div>    </div>      <p>      <button type="submit" [disabled]="!contactForm.valid">Submit</button>    </p>      <p>{{contactForm.valid}} </p>    </form> |

## Summary

Angular template-driven form validation uses the directives known as validators. The validators handle form validations and display validation messages. The Angular comes up with several built-in validators for this purpose. They are minlength, maxlength, email, pattern, required, etc.

## ****DropDownList in Angular Template Driven Forms****

In this article, I am going to discuss **DropDownList in Angular Template Driven Forms** in detail. Please read our previous article as it is a continuation part to that article where we discussed [**Checkbox in Angular Template Driven Forms**](https://dotnettutorials.net/lesson/checkbox-angular-template-driven-forms/). At the end of this article, you will understand what is DropDownList and when and how to use DropDownList in Angular Template Driven Forms.

##### **What is a DropDownList?**

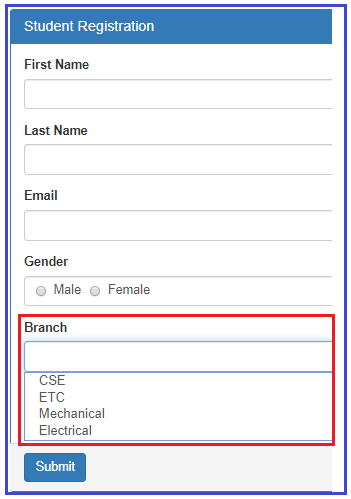
A DropDownList is an HTML Element which is nothing but a collection of list items from which it will allows the user to select a single list item. Depending on your business requirement you may either hard code the values or you may retrieve the values from a database table.

In this article, I am going to discuss both the approaches. First, we will discuss creating the DropDownList using the hard-coded value then we will see how to create the DropDownList with the values coming from a component.

##### **Example to understand DropDownList in Angular Template Driven Forms:**

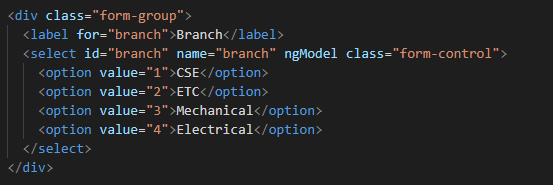
Let us understand how to create and use **DropDownList**in Angular Template Driven Forms with an example. We are going to work with the same example that we worked in our previous article.

Now, we want to include Branches dropdownlist in the student registration form as shown in the below image. When the user select a particular branch from the dropdownlist and click on the “Submit” button, then we want to display the selected drop down list value on the console.



##### **How to create the dropdownlist in angular using template driven forms?**

Please have a look at the following code which will create a Drop Down List with the required items.



##### **Code Explanation**

As shown in the code, we have hard coded the drop down list options in the HTML. Notice each option also has a corresponding value attribute and its value is the branch 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 our upcoming article.

##### **The complete code of app.component.html:**

Following is the complete code of **app.component.htm**l file.

**<br/>**

**<div** class="container"**>**

**<div** class="row"**>**

**<div** class="form-bg"**>**

**<form** #studentForm="ngForm" (ngSubmit)="RegisterStudent(studentForm)" **>**

**<div** class="panel panel-primary"**>**

**<div** class="panel-heading"**>**

**<h3** class="panel-title"**>**Student Registration**</h3>**

**</div>**

**<div** class="panel-body"**>**

**<div** class="form-group"**>**

**<label** for="firstName"**>**First Name**</label>**

**<input** id="firstName" type="text" class="form-control"

name="firstName" ngModel**>**

**</div>**

**<div** class="form-group"**>**

**<label** for="lastName"**>**Last Name**</label>**

**<input** id="lastName" type="text" class="form-control"

name="lastName" ngModel**>**

**</div>**

**<div** class="form-group"**>**

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

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

name="email" ngModel**>**

**</div>**

**<div** class="form-group"**>**

**<label>**Gender**</label>**

**<div** class="form-control"**>**

**<label** class="radio-inline"**>**

**<input** type="radio" name="gender" value="male" ngModel**>**

Male

**</label>**

**<label** class="radio-inline"**>**

**<input** type="radio" name="gender" value="female" ngModel**>**

Female

**</label>**

**</div>**

**</div>**

**<div** class="form-group"**>**

**<label** for="branch"**>**Branch**</label>**

**<select** id="branch" name="branch" ngModel class="form-control"**>**

**<option** value="1"**>**CSE**</option>**

**<option** value="2"**>**ETC**</option>**

**<option** value="3"**>**Mechanical**</option>**

**<option** value="4"**>**Electrical**</option>**

**</select>**

**</div>**

**<div** class="form-group"**>**

**<div** class="form-control"**>**

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

**<input** type="checkbox" name="isAccept" ngModel**>**Accept Terms & Conditions

**</label>**

**</div>**

**</div>**

**</div>**

**<div** class="panel-footer"**>**

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

**</div>**

**</div>**

**</form>**

**</div>**

**</div>**

**</div>**

##### **Modifying app.component.ts file:**

We want to log the posted form values into the console tab. So, modify the **app.component.ts**file as shown below.

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

**import** **{** NgForm **}** from '@angular/forms'

@Component**({**

selector: 'app-root',

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

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

**})**

**export** **class** AppComponent **{**

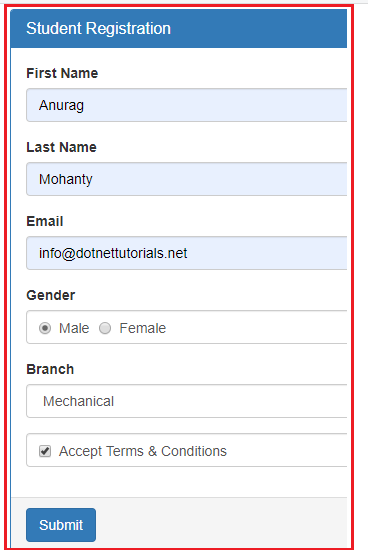
RegisterStudent**(**studentForm: NgForm**)**: **void** **{**

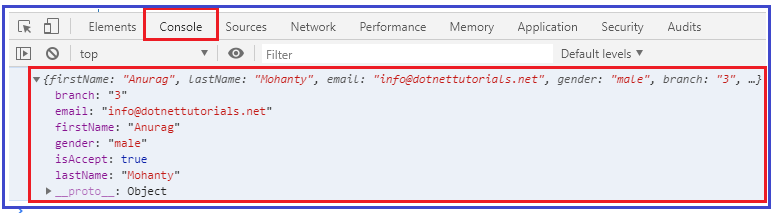
console.log**(**studentForm.value**)**;

**}**

**}**

Save the changes and browse the application, then open browser developers tool by pressing F12 key and click on the console tab. Fill the form, select one value from the drop down list and click on the submit button and you should see the posted form values in the console tab as shown in the below image.

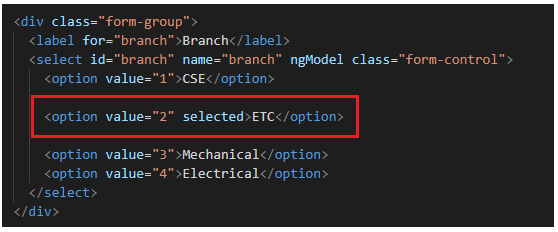




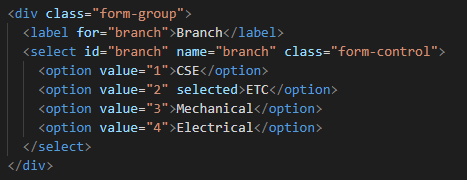
##### **How to have one of the dropdownlist item selected by default in Angular?**

As we know when working with real-time applications, many a times we need to provide one option to be selected in the drop down list by default when the form load. And we normally do this by adding the **selected**attribute on one of the option of drop down list.

If we include the **selected**attribute on the **dropdownlist**, then we may expect that option or item to be selected by default. But in angular template driven forms, that will not work. Lets include the “**selected**” attribute on the ETC branch option to verify this. So. Modify the Drop Down List HTML code as shown below.



With the above changes in place, now browse the application and you will see that the ETC department is not selected by default when the page load.



As we already discussed, we use the “ngModel” directive in angular for two-way data binding. So when we put the ngModel directive back into the control then the “**selected**” attribute will not work on the drop down list or select list. If we remove the ngModel directive from the control then **selected**attribute work but two way data binding will not work.

##### **How to make it works?**

In order to make it work, what we need to do is, we need to add a property lets say “BranchId” in the component class and initialize its value with the branch value that you want to be selected when the page load. As we want ETC branch to be selected by default and as its value is 1, so, we need to BranchId property and initialize its value to 2 in the AppComponent class. So, modify the **app.component.ts** file as shown below.

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

**import** **{** NgForm **}** from '@angular/forms'

@Component**({**

selector: 'app-root',

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

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

**})**

**export** **class** AppComponent **{**

BranchId = "2";

RegisterStudent**(**studentForm: NgForm**)**: **void** **{**

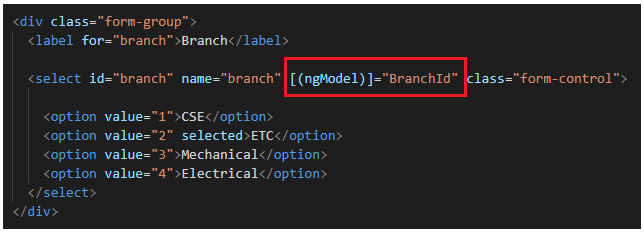
console.log**(**studentForm.value**)**;

**}**

**}**

##### **Modifying the app.component.html file**

Next, we include to include the ngModel directive and bind it with the component property BranchId. To do so, modify the Select List code in the **app.component.html** file as shown below.



With the above changes in place, now if you browse the application, then you should see the ETC branch is selected by default in the Branch Drop Down List when the form loads .

Now, even if you remove the “selected” attribute from the Option, then also it will work i.e. it will select the ETC by default. This is possible because of the two-way data binding which is provided by angular. In our example, we do not want the ETC to be selected by default, so we remove the “selected” attribute and the “BranchId” property from the component class and ngModel directive.

##### **How to disable a Drop Down List in Angular Template Driven Forms?**

In order to disable a drop down list in Angular Template Driven Form, we need use the disabled attribute on the select element as shown below.

**<select id=”branch” name=”branch” class=”form-control” ngModel disabled>**

With the above changes, now it is not possible to select any item from the drop down list. As we already discussed, by default, the disabled form controls are not included in the Angular auto generated form model.

In our example, we do not want the dropdownlist to be disabled, so please remove the disabled attribute from the select element.

##### **How to get the select list options from the component class?**

As of now we have hard-coded the select list options in HTML itself. In most of the real-time applications, you will get this data from a database. So, modify the **app.component.ts** file as shown below. Here, we created one property called **Branches**which will return the list of items that we want to show in the drop down list. It has two properties id and name.

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

**import** **{** NgForm **}** from '@angular/forms'

@Component**({**

selector: 'app-root',

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

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

**})**

**export** **class** AppComponent **{**

RegisterStudent**(**studentForm: NgForm**)**: **void** **{**

console.log**(**studentForm.value**)**;

**}**

Branches: **any[]** = **[**

**{** id: 1, name: 'CSE' **}**,

**{** id: 2, name: 'ETC' **}**,

**{** id: 3, name: 'Mechanical' **}**,

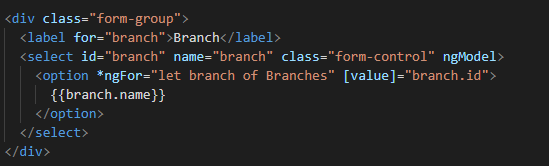
**{** id: 4, name: 'Electrical' **}**

**]**;

**}**

##### **Modifying the app.component.html file:**

Next we need to modify the drop down list code in the app.component.html file as shown below.



##### **Code explanation:**

As you can see on the “option” element we are using ngFor structural directive to loop through all the branches that we have in the “Branches” property of the component class.

For each “Branch” store in the “Branches” array, we get an option. The option value is the Branch id and the display text is the Branch name.

Please have a look at the square brackets around the [value] property. This is property binding in Angular. If you remove the square brackets the value for each option will be the literal text “branch.id” instead of the branch id (1 or 2 or 3, etc.). To display the Branch name we are using angular interpolation.

##### **The complete code of app.component.html:**

**<br/>**

**<div** class="container"**>**

**<div** class="row"**>**

**<div** class="form-bg"**>**

**<form** #studentForm="ngForm" (ngSubmit)="RegisterStudent(studentForm)" **>**

**<div** class="panel panel-primary"**>**

**<div** class="panel-heading"**>**

**<h3** class="panel-title"**>**Student Registration**</h3>**

**</div>**

**<div** class="panel-body"**>**

**<div** class="form-group"**>**

**<label** for="firstName"**>**First Name**</label>**

**<input** id="firstName" type="text" class="form-control"

name="firstName" ngModel**>**

**</div>**

**<div** class="form-group"**>**

**<label** for="lastName"**>**Last Name**</label>**

**<input** id="lastName" type="text" class="form-control"

name="lastName" ngModel**>**

**</div>**

**<div** class="form-group"**>**

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

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

name="email" ngModel**>**

**</div>**

**<div** class="form-group"**>**

**<label>**Gender**</label>**

**<div** class="form-control"**>**

**<label** class="radio-inline"**>**

**<input** type="radio" name="gender" value="male" ngModel**>**

Male

**</label>**

**<label** class="radio-inline"**>**

**<input** type="radio" name="gender" value="female" ngModel**>**

Female

**</label>**

**</div>**

**</div>**

**<div** class="form-group"**>**

**<label** for="branch"**>**Branch**</label>**

**<select** id="branch" name="branch" class="form-control" ngModel**>**

**<option** \*ngFor="let branch of Branches" [value]="branch.id"**>**

{{branch.name}}

**</option>**

**</select>**

**</div>**

**<div** class="form-group"**>**

**<div** class="form-control"**>**

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

**<input** type="checkbox" name="isAccept" ngModel**>**Accept Terms & Conditions

**</label>**

**</div>**

**</div>**

**</div>**

**<div** class="panel-footer"**>**

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

**</div>**

**</div>**

**</form>**

**</div>**

**</div>**

**</div>**

## ****Checkbox in Angular Template Driven Forms****

In this article, I am going to discuss **Checkbox in Angular Template Driven Forms** in detail. Please read our previous article as it is a continuation part to that article where we discussed [**Radio Buttons in Angular Template Driven Forms**](https://dotnettutorials.net/lesson/radio-buttons-in-angular-template-driven-forms/). At the end of this article, you will understand what are Checkbox controls and when and how to use Checkbox in Angular Applications.

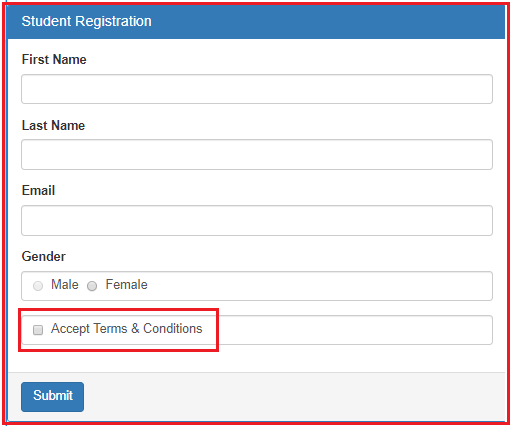
##### **What is a Checkbox?**

A **Checkbox** is an HTML element which allows the users to select multiple options from the available options. For example, in most of the website when you are filling a form you may find a checkbox for terms and conditions which needs to be accept in order to submit the form.

##### **Example to understand Checkbox in Angular Template Driven Forms:**

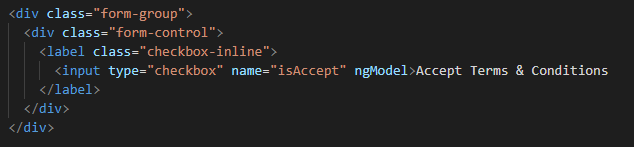
Let us understand how to create and use **Checkbox** in Angular Template Driven Forms. We are going to work with the same example that we worked in our previous article.

Now, we want to include “Accept Terms & Conditions” checkbox in the student registration form as shown in the below image. When we select the Accept Terms & Conditions checkbox and when we click the “Submit” button, we want to display the selected value of the checkbox in the console. Here, if the use select the checkbox then true will be logged into the console else false will be logged into the console.



##### **How to create check box in angular template driven forms?**

Please have a look at the below code which will create Accept Terms and Conditions checkbox.



##### **Code Explanation**

In the above code, we set the name attribute of the input element checkbox to isAccept. We have not set the value property here. This is because its value can be true of false. If the checkbox is checked or selected then the value is true else the value is false.

##### **The complete code of app.component.html:**

Following is the complete code of **app.component.htm**l file.

**<br/>**

**<div** class="container"**>**

**<div** class="row"**>**

**<div** class="form-bg"**>**

**<form** #studentForm="ngForm" (ngSubmit)="RegisterStudent(studentForm)" **>**

**<div** class="panel panel-primary"**>**

**<div** class="panel-heading"**>**

**<h3** class="panel-title"**>**Student Registration**</h3>**

**</div>**

**<div** class="panel-body"**>**

**<div** class="form-group"**>**

**<label** for="firstName"**>**First Name**</label>**

**<input** id="firstName" type="text" class="form-control"

name="firstName" ngModel**>**

**</div>**

**<div** class="form-group"**>**

**<label** for="lastName"**>**Last Name**</label>**

**<input** id="lastName" type="text" class="form-control"

name="lastName" ngModel**>**

**</div>**

**<div** class="form-group"**>**

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

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

name="email" ngModel**>**

**</div>**

**<div** class="form-group"**>**

**<label>**Gender**</label>**

**<div** class="form-control"**>**

**<label** class="radio-inline"**>**

**<input** type="radio" name="gender" value="male" ngModel**>**

Male

**</label>**

**<label** class="radio-inline"**>**

**<input** type="radio" name="gender" value="female" ngModel**>**

Female

**</label>**

**</div>**

**</div>**

**<div** class="form-group"**>**

**<div** class="form-control"**>**

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

**<input** type="checkbox" name="isAccept" ngModel**>**Accept Terms & Conditions

**</label>**

**</div>**

**</div>**

**</div>**

**<div** class="panel-footer"**>**

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

**</div>**

**</div>**

**</form>**

**</div>**

**</div>**

**</div>**

##### **Modifying app.component.ts file:**

We want to log the posted form values into the console tab. So, modify the **app.component.ts**file as shown below.

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

**import** **{** NgForm **}** from '@angular/forms'

@Component**({**

selector: 'app-root',

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

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

**})**

**export** **class** AppComponent **{**

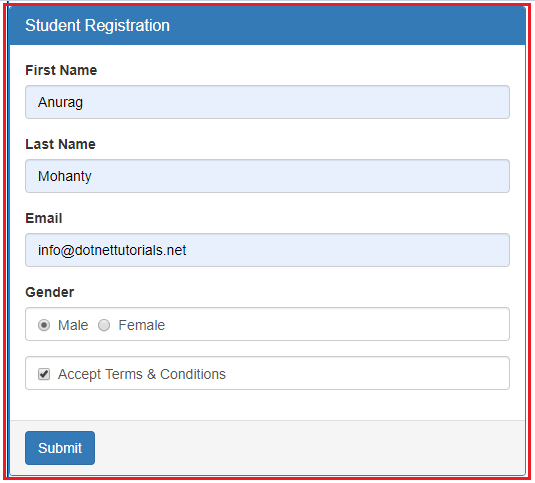
RegisterStudent**(**studentForm: NgForm**)**: **void** **{**

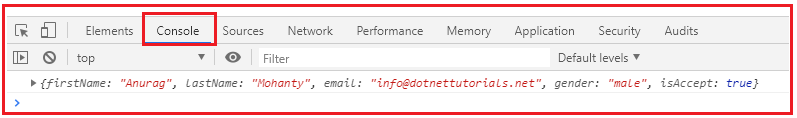
console.log**(**studentForm.value**)**;

**}**

**}**

Save the changes and browse the application, then open browser developers tool by pressing F12 key and click on the console tab. Fill the form and click on the submit button and you should see the posted form values in the console tab as shown in the below image.





##### **How to get a radio checkbox checked by default in Angular?**

As we know when working with real-time applications, many a times we need to provide the checkbox to be checked by default when the form load initially and we normally do this by adding the checked attribute of the checkbox.

If we include the checked attribute on the checkbox, then we may expect that the checkbox to be checked by default when the page load. But in angular template driven forms, you will not get that default checked when the page loads initially.So, lets include the “checked” attribute on the checkbox and verify this. So. Modify the Checkbox HTML code as shown below.

**<input type=”checkbox” name=”isAccept” ngModel checked>Accept Terms & Conditions**

With the above changes in place, now browse the application and you will see the the checkbox is not checked by default when the page load.

However, if we remove the “ngModel” directive from the checkbox as shown below, then you will see that the checkbox is checked when the form is load.

**<input type=”checkbox” name=”isAccept” checked>Accept Terms & Conditions**

As we already discussed, in Angular, we generally use the “ngModel” directive for two-way data binding. So when we put the ngModel directive back into the control then the “checked” attribute will not work on the checkbox. If we remove the ngModel then checked attributes work but two way data binding will not work.

##### **How to make it works?**

In order to make it work, we need to include a property lets say “isAccept” in the component class and initialize its value to true. So.modify the app.component.ts file as shown below.

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

**import** **{** NgForm **}** from '@angular/forms'

@Component**({**

selector: 'app-root',

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

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

**})**

**export** **class** AppComponent **{**

isAccept = **true**;

RegisterStudent**(**studentForm: NgForm**)**: **void** **{**

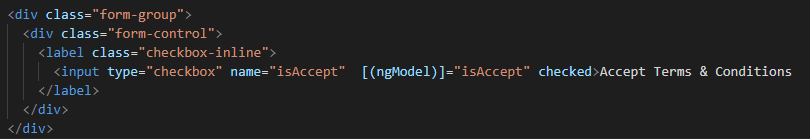
console.log**(**studentForm.value**)**;

**}**

**}**

##### **Modifying the app.component.html**

Now we include to include the ngModel directive and bind it with the component property isAccept. To do so, modify the checkbox HTML code in the **app.component.html** file as shown below.



With the above changes in place, now browse the application, you should see the Accept Terms & Conditions checkbox is checked by default when the form loads initially.

Now, even if you remove the “checked” attribute from the checkbox, then it still checked by default when the form loads. This is possible because of the Angular two-way data binding. In our example, we do not want the checkbox to be checked by default, so we remove the “checked” attribute and the “isAccept” property from the component class and ngModel directive.

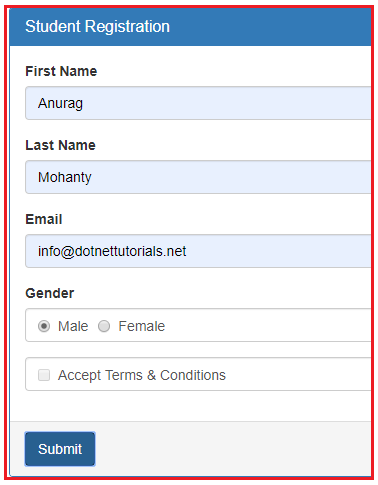
##### **How to disable a Checkbox in Angular Template Driven Forms?**

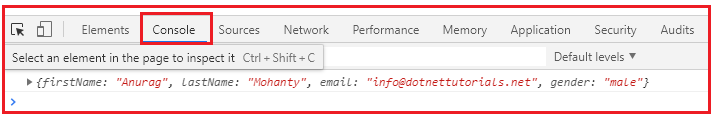
In order to disable a checkbox in Angular Template Driven Form, we need use the disabled attribute on the checkbox as shown below.

**<input type=”checkbox” name=”isAccept” ngModel disabled>Accept Terms & Conditions**

With the above changes, now it is not possible to select the checkbox. The most important point that you need to remember is, by default, the disabled form controls are not included in the Angular auto generated form model.

Since, the isAccept checkbox is disabled, it will not be included in the Angular generated form model. So, after the filling when you click on the button, in the console tab, you will not find the isAceept value as shown in the below image.





In our example, we do not want the checkbox to be disabled, so please remove the disabled attribute from the checkbox.