|  |
| --- |
|  |
| Angular 8 |
|  |
| Angular – Template Forms, Event Handling, Two Way Data Binding, Validation |

**TechBrain Express**

December 9, 2019

Authored by: Ashu

Template Forms, Event Handling, Two Way Data Binding, Validation

* **What is Form?**
* **Types of Form?**
* **Explore Template Form**
* **Demonstrate event handling and two way binding**
* **Validate the Form**

**FOLLOW: D:\backupdt20092019\_beforeformat\Revature\Angular\RevatureAngular\revaturebookstorelogin**

**What is Form?**

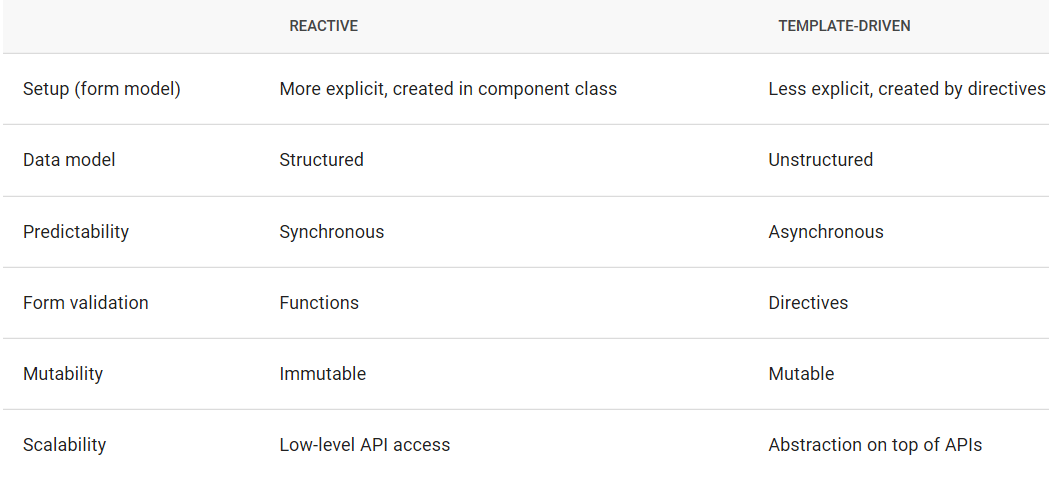
1. Applications use forms to enable users to log in, to update a profile, to enter sensitive information, and to perform many other data-entry tasks.

2. Capture user input events from the view, validate the user input, create a form model and data model to Cupdate, and provide a way to track changes.

**Types of Form?**

Angular provides two different approaches to handling user input through forms:

1. Reactive
   1. More scalable, reusable, and testable.
2. Template-driven
   1. Useful for adding a simple form to an app, such as an email list signup form.
   2. If you have very basic form requirements and logic that can be managed solely in the template, use template-driven forms.

****

**Common**

* [FormControl](https://angular.io/api/forms/FormControl) tracks the value and validation status of an individual form control.
* [FormGroup](https://angular.io/api/forms/FormGroup) tracks the same values and status for a collection of form controls.
* [FormArray](https://angular.io/api/forms/FormArray) tracks the same values and status for an array of form controls.
* [ControlValueAccessor](https://angular.io/api/forms/ControlValueAccessor) creates a bridge between Angular [FormControl](https://angular.io/api/forms/FormControl) instances and native DOM elements.

**Explore Template Form**

Reference: <https://angular.io/guide/forms-overview>

Reactive and template-driven forms both use a form model to track value changes between Angular forms and form input elements.

**SOURCE OF TRUTH**

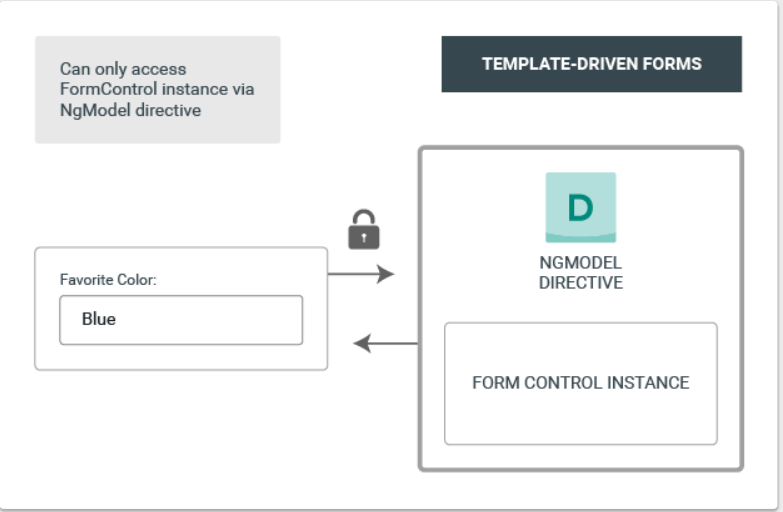
The source of truth provides the value and status of the form element at a given point in time.

**FORM CONTROL**

[FormControl](https://angular.io/api/forms/FormControl) tracks the value and validation status of an individual form control.

### Form Setup

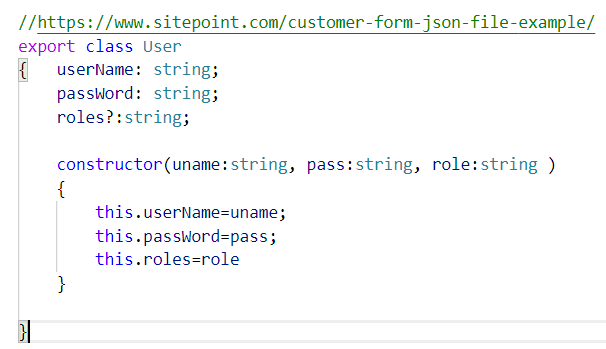
In template-driven forms, the source of truth is the template.



1. The template-driven form directive [NgModel](https://angular.io/api/forms/NgModel) is responsible for creating and managing the [FormControl](https://angular.io/api/forms/FormControl) instance for a given form element
2. It's less explicit, but you no longer have direct control over the form model.

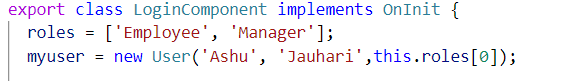
[**https://angular.io/guide/forms#ngForm**](https://angular.io/guide/forms#ngForm)

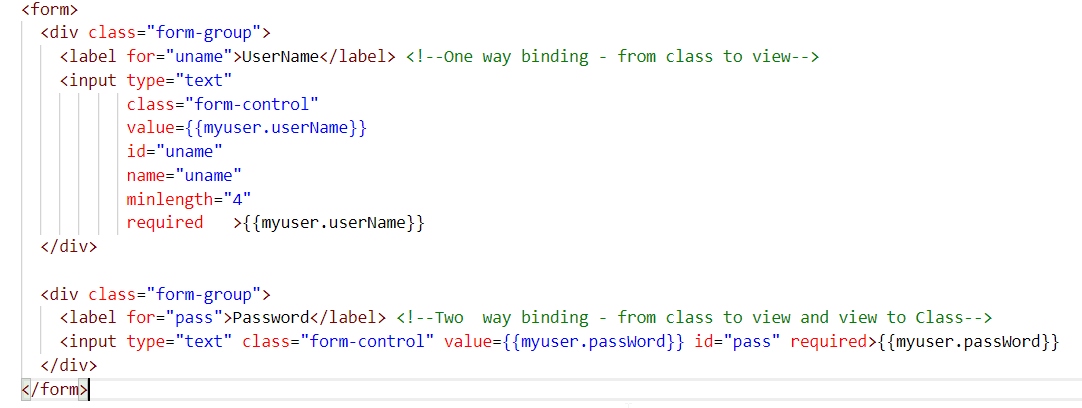
**Lab: Create a class app/model/User.ts**

****

**Lab: Create a template based Login page using login component and set routing for it.**

**Create an object of user class and set username as Ashu**

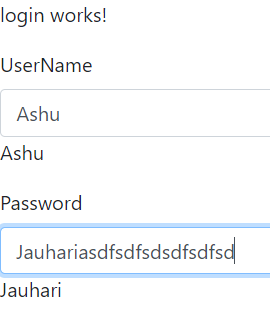
****

****

**Review above code-**

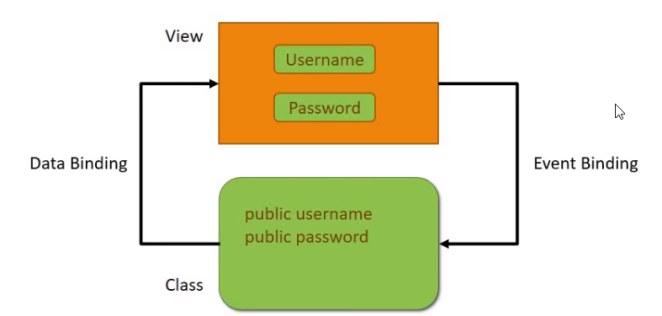
In above code, username is bound to form control and its values is set as ‘Ashu’, and value displayed.

**Observation:** Updating value in text box does not change the displayed value.

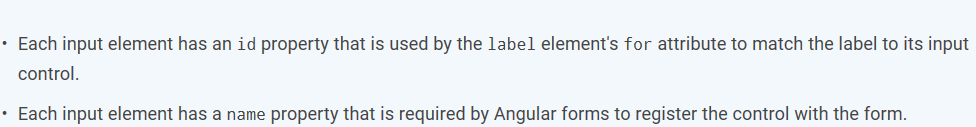
****

**Here,**  the value myuser.username is bound to form element in other words, class is bound to view but not the reverse.

This is called “ONE WAY BINDING (Data Binding).



# Demonstrate One Way Binding / Two Way Binding



Lab: Update login form so that in case we make any change in form element it should reflected in username variable.

1. Set ngModel

<form >

<div class="form-group">

<label for="uname">UserName</label> <!--One way binding - from class to view-->

<!--ONE WAY BINDING-->

<input type="text"

class="form-control"

value={{myuser.userName}}

id="uname"

name="uname"

minlength="4"

required >{{myuser.userName}}

</div>

<!--TWO WAY BINDING-->

<div class="form-group">

<label for="pass">Password</label> <!--Two way binding - from class to view and view to Class-->

<input type="text"

class="form-control"

id="pass"

name="pass"

[(ngModel)]="myuser.passWord"

required>{{myuser.passWord}}

</div>

</form

1. Add formmodule to app.module.ts

import { FormsModule } from '@angular/forms';

imports: [

BrowserModule,

NgbModule,

AppRoutingModule,

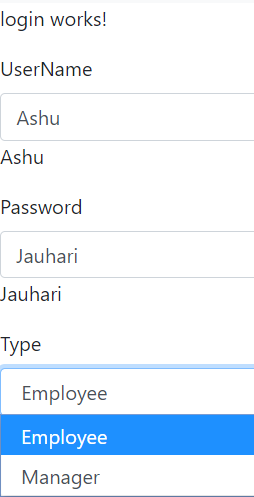
FormsModule,

RouterModule.forRoot(appRoutes) ],

## Lab: Add two roles in Login.

1. Add role property in login.ts file
2. export class LoginComponent implements OnInit {

roles = ['Employee', 'Manager'];

1. Update login.html
2. 

<form >

<div class="form-group">

<label for="uname">UserName</label> <!--One way binding - from class to view-->

<!--ONE WAY BINDING-->

<input type="text"

class="form-control"

value={{myuser.userName}}

id="uname"

name="uname"

minlength="4"

required >{{myuser.userName}}

</div>

<!--TWO WAY BINDING-->

<div class="form-group">

<label for="pass">Password</label> <!--Two way binding - from class to view and view to Class-->

<input type="text"

class="form-control"

id="pass"

name="pass"

[(ngModel)]="myuser.passWord"

required>{{myuser.passWord}}

</div>

<div class="form-group">

<label for="roles">Type</label>

<select #myrole class="form-control" id="" required>

<option \*ngFor="let role of roles" [value]="role">{{role}}</option>

</select> {{myrole.value}}

</div>

</form>

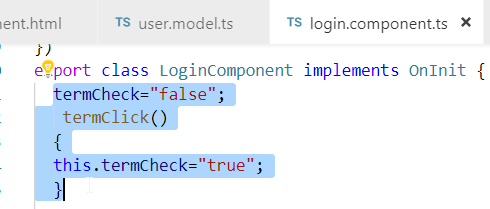
**Note:**  Mandatory to specify name of element.

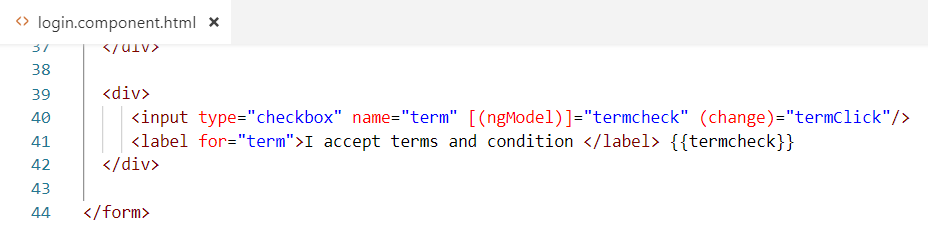
**Demonstrate event Handling**

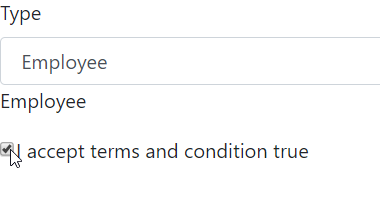
1. Event binding is the simplest (or the most proper) way of listening to events in a component template.
2. Information flows from elements in a component template to the corresponding component’s class.
3. With event binding, you don’t need to give the target element an identifier in order to access and attach your listeners to it because you are dealing with the target event and the target element directly in the template.



## Lab: Create a checkbox with label “I accept term and condition” and display true and false according beside label in login page.



1. 



Note: <button (click)="handleClick()">Save</button>

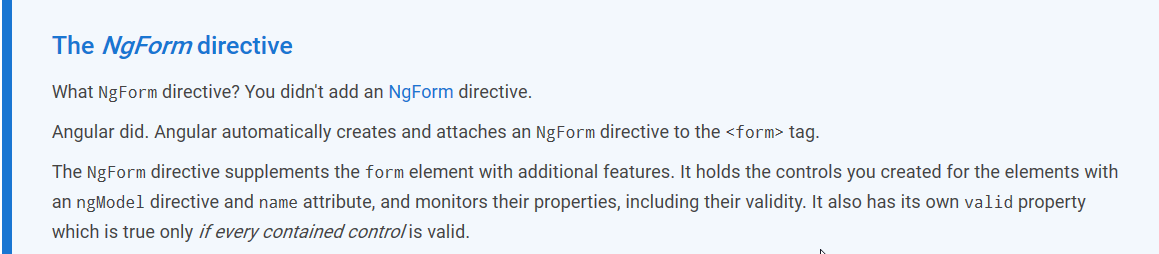
## Limitation of Event Binding

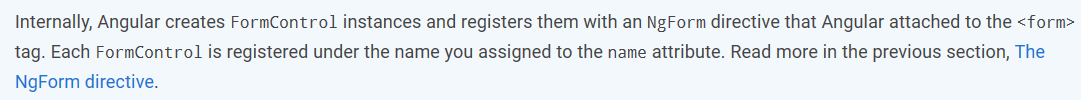
1. You cannot listen to events fired outside the components template.
2. You cannot listen to events on your component’s host element, which wraps the component’s template.
3. If a DOM event is fired in the child, the parent component will hear it as the event propagates up through.
4. One thing to note is that only DOM events bubble up like this. Custom Angular events fired by the [EventEmitter](https://angular.io/api/core/EventEmitter) do not bubble up.
5. Event binding is that you cannot dynamically add or remove the listener once event binding is set on an element in the component template.  The bindings will be active once the element gets rendered into the DOM, and remain active until it’s removed from the DOM. This means, for event binding, it is Angular that controls when a listener starts or stops listening to their target events.

**Exception** - Even though you cannot listen to events outside of the component’s template, you can listen to events on global elements such as window, document, and body with event binding.

## Lab: Create a submit button to submit login form and display submitted values

Note: To submit form you can call a method for fomr processing using ngSubmit directive as well ngForm





1. Update Login form to add submit button and call ngSubmit. **login.component.html**

<form (ngSubmit)="onSubmit()" #loginForm="ngForm" >

{{diagnostic}}

<div class="form-group">

<label for="uname">UserName</label> <!--One way binding - from class to view-->

<!--ONE WAY BINDING-->

<input type="text"

class="form-control"

value={{myuser.userName}}

id="uname"

name="uname"

minlength="4"

required >{{myuser.userName}}

</div>

<!--TWO WAY BINDING-->

<div class="form-group">

<label for="pass">Password</label> <!--Two way binding - from class to view and view to Class-->

<input type="text"

class="form-control"

id="pass"

name="pass"

[(ngModel)]="myuser.passWord"

required>{{myuser.passWord}}

</div>

<div class="form-group">

<label for="roles">Type</label>

<select #myrole class="form-control" id="" required>

<option \*ngFor="let role of roles" [value]="role">{{role}}</option>

</select> {{myrole.value}}

</div>

<div>

<input type="checkbox" name="term" [(ngModel)]="termcheck" (change)="termClick"/>

<label for="term">I accept terms and condition </label> {{termcheck}}

</div>

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

</form>

1. Define diagnostic and onsubmit() in login.component.ts

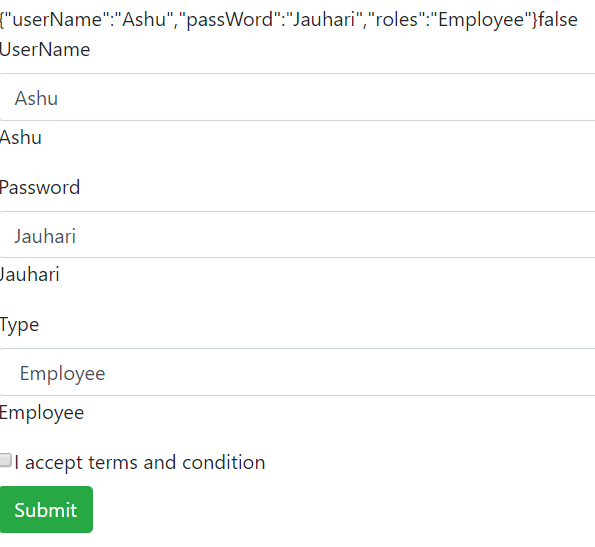
get diagnostic() { return JSON.stringify(this.myuser) + this.submitted; }

//Handle Submittion of Form

submitted = false;

onSubmit() { this.submitted = true; }

constructor() { }



# Toggle Form region

## Lab: Display a section to display entered data on submission using login.componenet.html and should allow editing on clicking edit button in login page.

<!-- Display on submission-->

<!--Display the form section on submission otherwise hide it -->

<div [hidden]="!submitted">

<h2>You submitted the following:</h2>

<div class="row">

<div class="col-xs-3">UserName</div>

<div class="col-xs-9">{{ myuser.userName }}</div>

</div>

<div class="row">

<div class="col-xs-3">Password</div>

<div class="col-xs-9">{{ myuser.passWord }}</div>

</div>

<div class="row">

<div class="col-xs-3">Role</div>

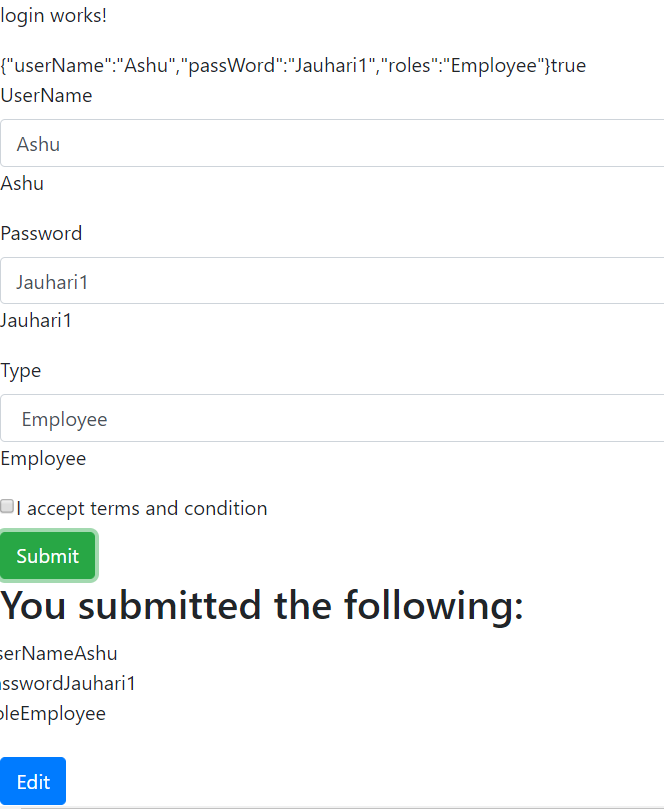
<div class="col-xs-9">{{ myuser.roles }}</div>

</div>

<br>

<button class="btn btn-primary" (click)="submitted=false">Edit</button>

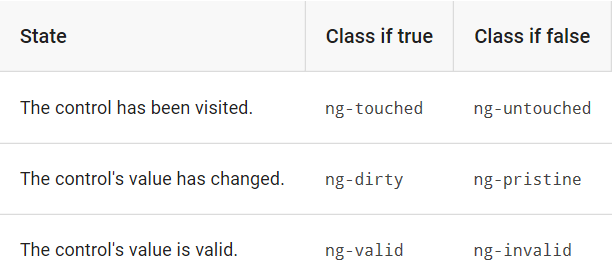
</div>



# Form Validation

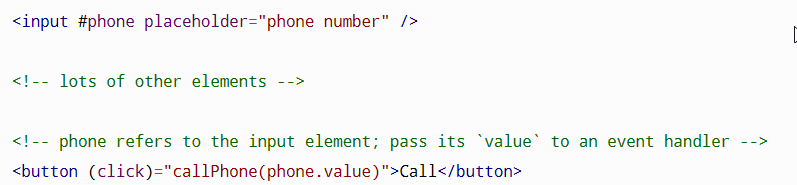
Using [ngModel](https://angular.io/api/forms/NgModel) in a form gives you more than just two-way data binding. It also tells you if the user touched the control, if the value changed, or if the value became invalid.

**Directives to apply validation**

****

**Template Reference Variable**  plays a vital role in referring DOM Element within template. Use the hash symbol (#) to declare a reference variable. To refer dom element using template is name of:

**template reference variable . value**

**Eg: Eg: **

with [NgForm](https://angular.io/api/forms/NgForm), itemForm is a reference to the [NgForm](https://angular.io/api/forms/NgForm) directive with the ability to track the value and validity of every control in the form.

The native <form> element doesn't have a form property, but the [NgForm](https://angular.io/api/forms/NgForm) directive does, which allows disabling the submit button if the itemForm.form.valid is invalid and passing the entire form control tree to the parent component's onSubmit() method.

Lab: Demonstrate the ng-touch, ng-dirty, ng-valid in password field. –Login.component.html

<!--TWO WAY BINDING-->

<div class="form-group">

<label for="pass">Password</label> <!--Two way binding - from class to view and view to Class-->

<input type="text"

class="form-control"

id="pass"

name="pass"

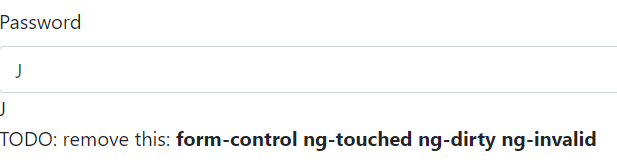
[(ngModel)]="myuser.passWord"

minlength="4" #passtemp

required>{{myuser.passWord}}

<br>TODO: remove this: <b>{{passtemp.className}}</b>

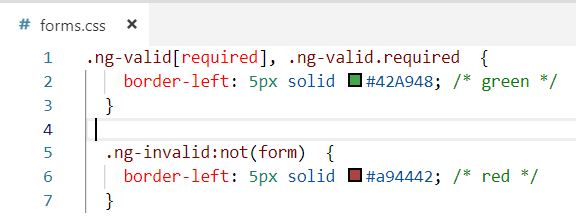
</div>



## Lab: 1.Add visual effect on ng-pristine/dirty/ ng-invalid/valid on password field in login.component.html

## 2.Show or hide the errors

Step 1: Create src/assets/forms.css



Step 2: Add following link



Step 3: Validate password

<!--TWO WAY BINDING-->

<div class="form-group">

<label for="pass">Password</label> <!--Two way binding - from class to view and view to Class-->

<input type="text"

class="form-control"

id="pass"

name="pass"

[(ngModel)]="myuser.passWord"

minlength="4" #passtemp="ngModel"

required>{{myuser.passWord}}

<br>TODO: remove this: <b>{{passtemp.className}}</b>

<div [hidden]="passtemp.valid || passtemp.pristine"

class="alert alert-danger">

Password is required

</div>

</div>

Note: if only #passtemp is given without “ngModel” it password will retain on page.

## Lab: Validate username for minimum 4 character and required validation

<div class="form-group">

<label for="uname">UserName</label> <!--One way binding - from class to view-->

<!--ONE WAY BINDING-->

<input type="text"

class="form-control"

[(ngModel)]=myuser.userName

id="uname"

name="uname"

minlength="4"

#uname1="ngModel"

required >{{myuser.userName}}

</div>

<div \*ngIf="uname1.invalid && (uname1.dirty ||uname1.touched)"

class="alert alert-danger">

<div \*ngIf="uname1.errors.required"> Name is required. </div>

<div \*ngIf="uname1.errors.minlength"> Name must be at least 4 characters long. </div>

</div>