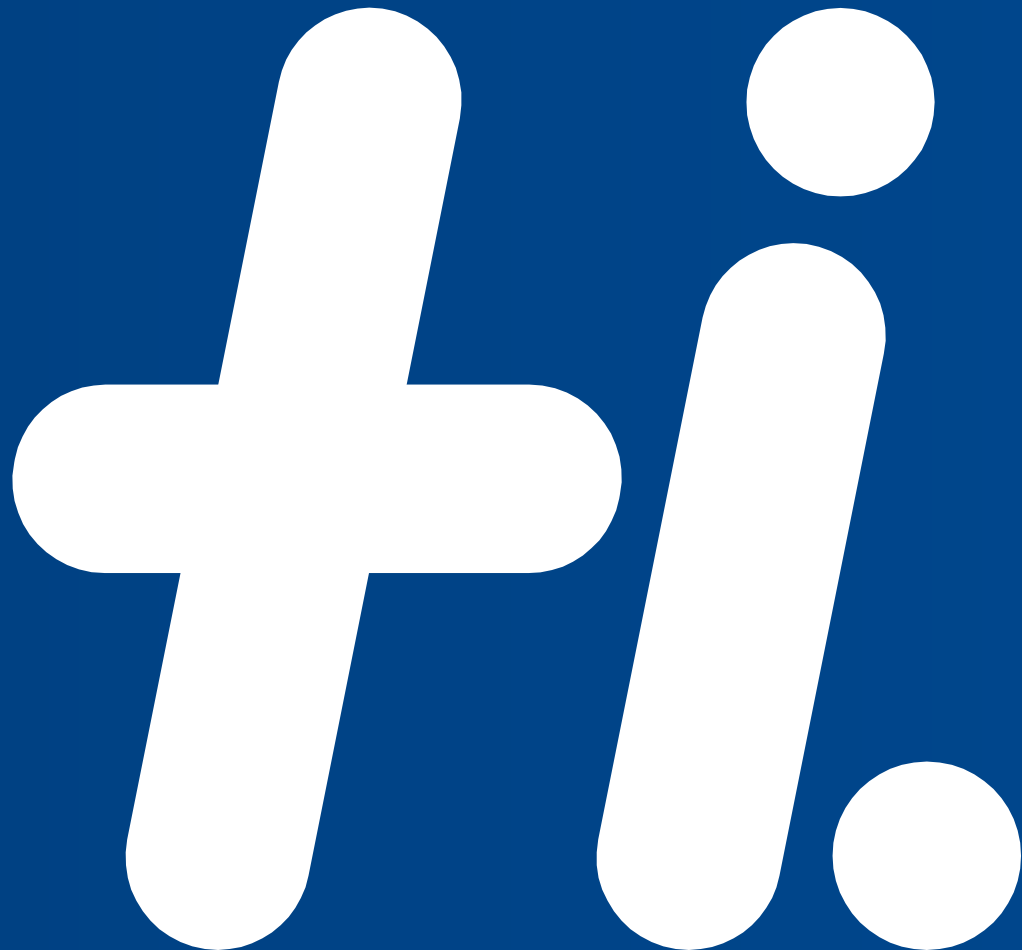
HEXAWARE

Angular – Forms & Validation

Session Objective

- To understand and workout the base concept of Forms and its types with validation.
- Template driven Form
- Reactive Form
- Validations
- Built-in Directives

Forms & Validation



Angular Forms

- Angular provides us with two types of forms:
  - Template Driven Forms
  - Reactive Forms

Template Driven Form

Template Driven Form Cont…

- Create controls on the component template and bind data using ngModel.
- With these, we dont create controls, form objects, or write code to work with pushing and pulling of data between component class and template
- In template driven forms, there is very little code for validations in the component class, and they're asynchronous.

Template Driven Form Cont…

- For activating the template-driven form, import **FormsModule.**
- All the elements have “name” attribute which will be used to identify property.

```
<input type="text" name="name" ngModel />
```


Template Driven Form Cont…

- **FormsModule** will detect a form element and **ngForm** automatically.
- To register the input elements to our **ngForm**, include **ngModel** for all elements.

Template Driven Form Cont…

NgForm offers two functionalities:

- Retrieving all the registered controls values.
- Retrieving the complete state of all the controls.

To attach the ngForm to Form, include the below code.

```
<form #validateForm="ngForm">
 ..
</form>
```

ngForm allows to access the form controls using **validateForm.value**

Template Driven Form Cont…

- To handle a submit of form “ngSubmit” is used.
- ngSubmit will do the same thing that is done by onSubmit
- Passing the form name to the ngSubmit function we can get the handler of the submitted form in component.

```
<form #validateForm="ngForm" (ngSubmit)="validate(validateForm)">
```


Template Driven Form Cont…

- Create a “validate” method in AppComponent

```
validate (validateForm: NgForm) {
 console.log('Successfully Submitted');
 console.log(validateForm);
}
```


Form control state Cont…

State	Class if true	Class if false
The control has been visited.	ng-touched	ng-untouched
The control's value has changed.	ng-dirty	ng-pristine
The control's value is valid.	ng-valid	ng-invalid

Form control state Cont…

Dirty & Pristine

- Dirty is true if the user has changed the value of the control.
  - <pre>Dirty? {{email.dirty }}</pre>
- This would be true if the user hasn’t changed the value, and false if the user has.
  - <pre>Pristine? {{ myform.controls.email.pristine }}</pre>

Form control state Cont…

Touched & Untouched

- A controls is said to be touched if the the user focused on the control and then focused on something else. For example by clicking into the control and then pressing tab or clicking on another control in the form.
- The difference between touched and dirty is that with touched the user doesn’t need to actually change the value of the input control.

Form control state Cont…

- Touched is true of the field has been touched by the user, otherwise it’s false.
- The opposite of touched is the property untouched.
  - <pre>Touched? {{email.touched }}</pre>

Form control state

Cont…

Valid & Invalid

- Valid is true if the field doesn’t have any validators or if all the validators are passing.
  - <pre>Valid? {{email.valid }}</pre>
- This would be true if the control was invalid and false if it was valid.
  - <pre>Invalid? {{email.invalid }}</pre>

Reactive Form

Reactive Forms

- Create form controls as trees of objects in the component class and bind them to the native form controls in the template.
- All validations and the creation of form controls are written in the component class.
- In Reactive Forms, all validations and changes in the state of native forms are synchronous
- Reactive form is created when the data model is immutable and usually mapped to a database.

Reactive Forms Cont..

Angular forms building blocks:

- FormControl
- FormGroup
- FormArray

Reactive Forms Cont..

HEXAWARE

FormControl

It tracks the value and validity status of an angular form control. It matches to a HTML form control like an input.

Training Name:<input type="text" formControlName="tname">

tname:new FormControl(null,Validators.required)

Reactive Forms Cont..

FormGroup

- It tracks the value and validity state of a **FormBuilder** instance group.
- It aggregates the values of each child **FormControl** into one object, using the name of each form control as the key.

Reactive Forms Cont..

```
fbform=new FormGroup({
tname:new FormControl(null,Validators.required),
fb:new
FormControl(null,[Validators.required,Validators.minLength
(4),Validators.maxLength(8)])
})
```


Reactive Forms Cont..

HEXAWARE

FormBuilder

- Form Builder is a helper class that creates **FormGroup, FormControl** and **FormArray** instances.
- It basically reduces the repetition and clutter by handling details of form control creation for you.

Reactive Forms Cont..

- All of them should be imported from the @**angular/forms** module.
- import { Validators, FormBuilder, FormGroup, FormControl } from '@angular/forms‘;
- Add **ReactiveFormsModule** in Module.ts file

Reactive Forms Cont..

```
<form [formGroup]="fbform" (ngSubmit)="fbFun()">
Training Name:<input type="text" formControlName="tname">
<div *ngIf="fbform.get('tname').invalid && fbform.get('tname').touched">
Training name is required
</div><br>
Feedback:<input type="text" formControlName="fb">
<div *ngIf="fbform.get('fb').invalid && fbform.get('fb').touched">
Feedback is required with min 4 char
</div><br>
<input type="submit" value="submit">
</form>
{{fbform.value|json}}
```


Reactive Forms Cont..

```
export class FeedbackComponent implements OnInit {
fbform=new FormGroup({
tname:new FormControl(null,Validators.required),
fb:new
FormControl(null,[Validators.required,Validators.minLength(4),Validators.max
Length(8)])
})
constructor() { }
ngOnInit() { }
fbFun(){
console.log("feedback submitted");
}}
```


Built-in Directives

Built-in Directives

- Directives are components without a view.
- They are components without a template

NgFor Cont…

- NgFor is a structural directive, meaning that it changes the structure of the DOM.
- It’s point is to repeat a given HTML template once for each value in an array, each time passing it the array value as context for string interpolation or binding

NgFor Cont…

Example:

app.component.html

```
<div *ngFor="let person of people">

  {{ person.name }}   </div>
```

app.component.ts

```
 people: any[] = [
{"name": Thomas"},
{ "name": “Sam" },
];
```


NgFor Cont…

Index

- To get the index of the item in the array
- Add another variable to ngFor expression and make it equal to index

NgFor Cont…

```
<ul>
<li *ngFor="let person of people; let i = index">
    {{ i + 1 }} - {{ person.name }}
 </li> </ul>
```

Note:
Create another variable called i and make it equal to the special keyword index.
We can use the variable i just like we can use the variable person in our template.

NgIf

Cont…

- The NgIf directive is used to display or remove an element based on a condition.
- If the condition is false the element in the directive is attached to it will be removed from the DOM.

```
*ngIf="<condition>"
```

NgIf - Example

App.component.html	App.component.ts
<ul *ngFor="let person of people"> <li *ngIf="person.age < 30"> {{ person.name }} {{ person.age }} 	people: any[] = [{"name": “Thomas", "age":35}, { "name": “Sam","age":32}, { "name": “Hendry","age": 21 },];

NgStyle Cont…

- The NgStyle directive lets to set a given DOM elements style properties.
- One way to set styles is by using the NgStyle directive and assigning it an object literal,
  - <div [ngStyle]="{'background-color':'green'}"></
- This sets the background color of the div to green.

NgStyle Cont…

- **ngStyle** becomes much more useful when the value is dynamic.

```
<div [ngStyle]="{'background-color':person.country === ‘India' ? 'green'
: 'red'  }"></</div>
```


NgSwitch Cont…

- NgSwitch is a directive which is bound to an expression.
- NgSwitch is used to display one element tree from a set of many element trees based on some condition.

NgSwitch Cont…

- It uses three keywords as follows,
- **ngSwitch**: We bind an expression to it that returns the switch value. It uses property binding.
- **ngSwitchCase**: Defines the element for matched value. We need to prefix it with asterisk (*).
- **ngSwitchDefault**: Defines the default element when there is no match. We need to prefix it with asterisk (*).

NgSwitch Cont…

```
<ul [ngSwitch]="person">
<li *ngSwitchCase="'Mohan'">Hello Mohan</li>
<li *ngSwitchCase="'Sohan'">Hello Sohan</li>
<li *ngSwitchCase="'Vijay'">Hello Vijay</li>
<li *ngSwitchDefault>Bye Bye</li>
 </ul>
```


NgSwitch with NgFor and NgClass Cont…

- Code snippet 1using NgSwitch with NgFor and NgClass. NgSwitch using interpolation {{ }}

```
<div *ngFor="let id of ids">
Id is {{id}}
<div ngSwitch="{{id%2}}">
 <div *ngSwitchCase="'0'" [ngClass]="'one'">I am Even.</div>
<div *ngSwitchCase="'1'" [ngClass]="'two'">I am Odd.</div>
<div *ngSwitchDefault>Nothing Found.</div>
</div> </div>
```


NgSwitch with NgFor and NgClass

- Here ids is an array defined in component which has numbers.
- For each iteration we are getting array element in the variable id.
- Using this variable in ngSwitch and dividing it by 2.
- For 0 value the element with CSS class .one will be executed and for 1 the element with CSS class .two will be executed.

HEXAWARE

Innovative Services

Passionate Employees

Delighted Customers

Thank you

www.hexaware.com