Angular Developer 5

example code

Exam ?

Angular Interview Questions & Answers

Which questions?

5, 6, 7, 8, 9, 10, 12, 13, 14, 15, 16, 21, 22, 24, 25, 26, 27, 28, 29, 30, 60, 62, 79

Forms

Two approaches

- Template driven
- Reactive

Template driven

- Html / template based structure
- More natural (traditional) way
- Quick and simple
- Less versatile and scalable

Reactive forms

- Cool
- Not today

Template driven forms

Super simple 'form'

```ts
// app.component.ts
export class AppComponent {
  public userInput: string;

  public onClick(): void {
      console.log(this.userInput);
  }
}
```

```html
<!-- app.component.html -->
<input type="text"
       [(ngModel)]="userInput"/>
<button [disabled]="userInput.length < 3"
        (click)="onClick()">
    click
</button>
```

Built in HTML validation

```
<!-- app.component.html -->
<input type="text"
       pattern="(\w.+\s).+"
       maxlength="30"
       minlength="3"
       [(ngModel)]="userInput"/>
<!-- How to block the button? -->
<button [disabled]="??????"
        (click)="onClick()">
    click
</button>
```

TEMPLATE REFERENCE VARIABLES

Way to assign DOM elements to variables

Lets assign some elements

```
1 <!-- app.component.html -->
2 <div class="column-3" #myDiv>
3 </div>
```

Simply tag any element with
`#variableName`

and you have element in variable :)

What for?

```
1 <!-- app.component.html -->
2 <input type="text" #myInput/>
3 <button (click)="greet(myInput.value)">
4     click
5 </button>
```

Custom component

```ts
// some.component.ts
export class SomeComponent {
  public doSomething () { ... };
}
```

```html
<!-- app.component.html -->
<some-component #myComponent>
  </some-component>
<button (click)="myComponent.doSomething()">
  click
</button>
```

TS code support?

```html
<!-- app.component.html -->
<input type="text" #myInput/>
```

```typescript
// app.component.ts
import { ElementRef, ViewChild } from '@angular/core';
export class AppComponent {
  @ViewChild('myInput') myInput: ElementRef;
  public greet() {
    const name = this.myInput.nativeElement.value;
    alert(name);
  }
}
```

TS code support?

```
<!-- app.component.html -->
<some-component #myComponent>
  </some-component>
```

```
// app.component.ts
import { ElementRef, ViewChild } from '@angular/core';
export class AppComponent {
  @ViewChild('myComponent') myComponent: SomeComponent;
  public logic() {
    this.myComponent.doSomething();
  }
}
```

TEMPLATE VARIABLES

- Way to assign values to variables in templates
- Created by adding #name on DOM element

```
<div #myDiv></div>
```

- Accessible from component class with @ViewChild decorator

```
@ViewChild('myDiv') div: ElementRef;
```

- By default: references to elements on which reside
- Can hold directives *exportAs* value

Template variables and directives

ngModel

```
<!-- app.component.html -->
<input type="text"
       #myInputEl <!-- element -->
       #myInputModel="ngModel"<!-- ngModel instance -->
       required
       [(ngModel)]="userInput"/>
<button [disabled]="myInput.valid === false"
        (click)="onClick()">
    click
</button>
```

It is possible to assign directives to our variables

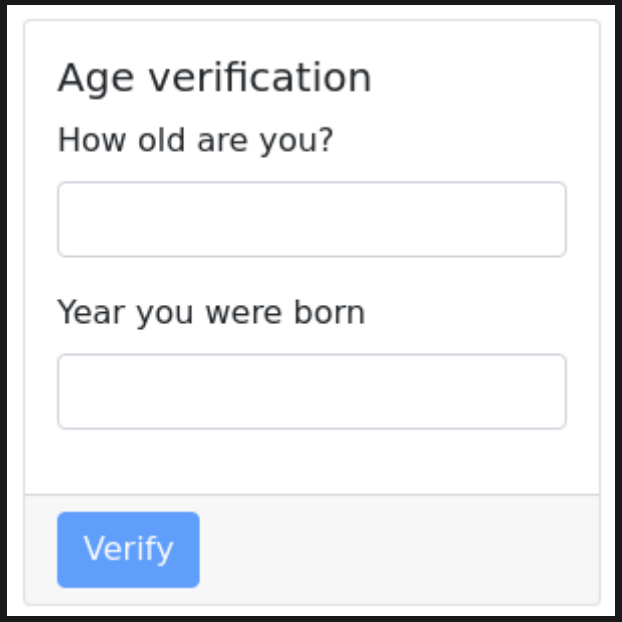
```
#variableName="directiveName"
```

The most commonly used directives

- ngModel
- ngForm

back to FORMS

Age verification



```ts
// app.component.ts
export class AppComponent {
  public age: number;
  public verify() {
    this.age >= 18 ? alert('OK') : alert('NOT OK');
  }
}
```

```html
<!-- app.component.html -->
<label>Your age</label>
<input type='number' [(ngModel)]="age"
   #ageModel="ngModel"
   required/>
<button [disabled]="!ageModel.valid"
    (click)="verify()">Verify
</button>
```

```
<!-- app.component.html -->
<label>Your age</label>
<input type='number' [(ngModel)]="age"
   #ageModel="ngModel"
   required/>
<label>Year you were born</label>
<select [(ngModel)]="year" #yearModel="ngModel"
    required>
    <option *ngFor="let year of years"
        [value]="year">{{ year }}
    </option>
</select>
<button [disabled]="!ageModel.valid || !yearModel.valid"
    (click)="verify()">Verify
</button>
```

What if we add more inputs?

```
1 <button [disabled]="!input.valid && input2.valid && another.
2     (click)="verify()">Verify
3 </button>
```

not enough screen space...

```
<input/>
<input/>
<select>
  <option></option>
</select>
```

```
<form>
  <input/>
  <input/>
  <select>
    <option></option>
  </select>
</form>
```

```
<form #ageForm="ngForm">
  <input type='number' name="age"
     [(ngModel)]="age"
     #ageModel="ngModel"
     required/>
  <select name="year"
      [(ngModel)]="year"
      #yearModel="ngModel"
      required>
      <option *ngFor="let year of years"
          [value]="year">{{ year }}</option>
  </select>
  <button [disabled]="!ageForm.valid"
      (click)="verify()">Verify
  </button>
```

Lets simplify it

```
<form #ageForm="ngForm">
  <input type='number' name="age"
     [(ngModel)]="age"
     required/>
  <select name="year"
      [(ngModel)]="year"
      required>
      <option *ngFor="let year of years"
          [value]="year">{{ year }}</option>
  </select>
  <button [disabled]="!ageForm.valid"
      (click)="verify()">Verify
  </button>
</form>
```

MORE!

```
<form #ageForm="ngForm"
  (submit)="verify(ageForm.form)">
  <input type='number' name="age"
     ngModel
     required/>
  <select name="year"
      ngModel
      required>
      <option *ngFor="let year of years"
          [value]="year">{{ year }}</option>
  </select>
  <button [disabled]="!ageForm.valid">
    Verify
  </button>
</form>
```

```
export class AppComponent {

  public verify(form: FormGroup) {
    const age = form.value.age;
    const year = form.value.year;
    ...
  }
}
```

Visual Validation

Displaying error messages

```
<input type="text"
       ngModel
       #myInput="ngModel"
       required/>
<p *ngIf="myInput.invalid">
  This field is required
</p>
```

Validation classes

```
<form class="ng-untouched ng-pristine ng-invalid">
  <input type="number" name="age"
         class="ng-untouched ng-pristine ng-invalid" />
  <select name="year"
          class="ng-untouched ng-pristine ng-invalid">
  ...
```

```
/* app.component.css */
.ng-touched.ng-invalid {
  border: 1px solid red;
}
```

Instant debugging

```
<form #ageForm="ngForm" #myForm
    (submit)="verify(ageForm.form)">
    <input type='number' name="age"
           #ageEl #ageModel="ngModel"
           ngModel
           required/><br/>
    Classes: {{ ageEl.className }}<br/>
    Errors:  {{ ageModel.errors | json }} <br/>
    <button [disabled]="!ageForm.valid">
        Verify
    </button>
</form>
```

Template Driven Form

- Based on **ngModel** directive tied to DOM elements

```
<input type="text" ngModel name="email"/>
<input type="checkbox" [(ngModel)]="accepted" name="accepted"/>
```

- With **ngForm** on *form* element for proper grouping

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

- Supports browser/html built in validations
  - pattern, minlength, required...
- Reflects controls statuses with set of css classes
  - ng-touched, ng-invalid

Also possible

- custom, async or cross validators

Rather tricky

- Nested forms, dynamic forms

BUT

this means that its time for **Reactive forms**

ATTENTION

problem

```
error NG8002: Can't bind to 'ngModel'
since it isn't a known property of 'input'.
```

solution

```
// app.module.ts
import {FormsModule} from '@angular/forms';

@NgModule({
 ...
 imports: [
   ...
   FormsModule
 ],
}
```