Using pipes

date

This is used to convert the input string to date format.

Syntax

Propertyvalue | date:”dateformat”

Parameters

dateformat − This is the date format the input string should be converted to.

Result

The property value will be converted to date format.

Using pipes

```
import { Component } from '@angular/core';
@Component ({
  selector: 'my-app',
  templateUrl: 'app/app.component.html'
})

export class AppComponent {
  newdate = new Date(2016, 3, 15);
}
```

Using pipes :

app/app.component.ts

```
import { Component } from '@angular/core';
@Component ({
  selector: 'my-app',
  templateUrl: 'app/app.component.html'
})
export class AppComponent {
  Name1: string = 'Angular2';
  appList: string[] = ["Binding", "Display", "Services"];
}
```

Using pipes :

app/app.component.html

```
<div>
  The name is {{Name1}}<br>
  The first Topic is {{appList[0] | lowercase}}<br>
  The second Topic is {{appList[1] | lowercase}}<br>
  The third Topic is {{appList[2]| lowercase}}<br>
</div>
```

Using pipes :

app/app.component.html

```
<div>
  The name is {{Name1}}<br>
  The first Topic is {{appList[0] | uppercase}}<br>
  The second Topic is {{appList[1] | uppercase}}<br>
  The third Topic is {{appList[2]| uppercase}}<br>
</div>
```

Using pipes : slice

This is used to slice a piece of data from the input string.
Propertyvalue | slice:start:end

Parameters

start – This is the starting position from where the slice should start.

end – This is the starting position from where the slice should end.

Result

The property value will be sliced based on the start and end positions.

Using pipes :SLICE

app/app.component.ts

```
import { Component } from '@angular/core';
@Component ({
  selector: 'my-app',
  templateUrl: 'app/app.component.html'
})
export class AppComponent {
  Name1: string = 'Angular2';
  appList: string[] = ["Binding", "Display", "Services"];
}
```

Using pipes :SLICE

```
<div>
  The name is {{Name1}}<br>
  The first Topic is {{appList[0] | slice:1:2}}<br>
  The second Topic is {{appList[1] | slice:1:3}}<br>
  The third Topic is {{appList[2]| slice:2:3}}<br>
</div>
```

Using pipes :currency

This is used to convert the input string to currency

format

Propertyvalue | currency

Result

The property value will be converted to currency format.

ngModel: Binding

```
import { Component } from '@angular/core';
import { FormsModule } from '@angular/forms';
@Component({
  selector: 'app-hello-world',
  template: `
    <div>
      <label>Name:</label>
      <input type="text" [(ngModel)]="name">
      <h1>Hello {{name}}!</h1>
    </div>`
})
export class HelloWorldComponent {
  name = 'World';
}
```

DATA BINDING

Text

We use two-way data binding with ngModel to bind to the textValue property on the component class. Two-way binding allows us to use the property to set the initial value of an <input>, and then have the user's changes flow back to the property. We can also change the property value in the code and have these changes reflected in the <input>.

DATA BINDING : text.component.ts

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-text-box',
  template: `
       <h1>Text ({{textValue}})</h1>
       <input #textbox type="text" [(ngModel)]="textValue" required>
       <button (click)="logText(textbox.value)">Update Log</button>
       <button (click)="textValue=''">Clear</button>
       <h2>Template Reference Variable</h2>
       Type: '{{textbox.type}}', required: '{{textbox.hasAttribute('required')}}',
       upper: '{{textbox.value.toUpperCase()}}'

       <h2>Log <button (click)="log=''">Clear</button></h2>
       <pre>{{log}}</pre>`
})
export class TextComponent {

  textValue = 'initial value';
  log = '';

  logText(value: string): void {
    this.log += `Text changed to '${value}'\n`;
  }
}
```

ngModel: Binding

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

@Component({
  selector: 'app-two-way-binding',
  template: `
    <h1>
      {{person.name.forename}} {{person.name.surname}}
      ({{person.address.street}}, {{person.address.city}}, {{person.address.country}})
    </h1>
    <p>
      <input [value]="person.name.forename"
            (keyup)="person.name.forename=$event.target.value">
```

ngModel: Two Way Binding

```
 <input [value]="person.name.surname"
       (input)="person.name.surname=$event.target.value">
</p>
<p>
 Street: <input [ngModel]="person.address.street"
            (ngModelChange)="person.address.street=$event">
</p>
<p>
 City: <input [(ngModel)]="person.address.city">
</p>
<p>
 Country: <input bindon-ngModel="person.address.country">
</p>
<p>
```

ngModel: Binding

```
    <button (click)="reset()">Reset</button>
   </p>
   <pre>
      {{person | json:2}}
   </pre>`
})
export class TwoWayBindingComponent {

  person: any;

  constructor() {
   this.reset();
  }
```

ngModel: Binding

```
    reset(): void {
   this.person = {
     name: {
       forename: 'John',
       surname: 'Doe'
     },
     address: {
       street: 'Lexington Avenue',
       city: 'New York',
       country: 'USA'
     }
   };
  }
}
```

ngModel: Binding

The Application

Here is the output from this component.

Doe (Lexington Avenue, New York, USA)

Doe

Street: Lexington Avenue

City: New York

Country: USA

Reset


```
{
  "name": {
    "forename": "",
    "surname": "Doe"
  },
  "address": {
    "street": "Lexington Avenue",
    "city": "New York",
    "country": "USA"
  }
}
```

ngModel: Binding-TEXTAREA

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-text-area',
  template: `
      <h1>Text Area</h1>
      <textarea ref-textarea [(ngModel)]="textValue"
   rows="4"></textarea><br/>
      <button (click)="logText(textarea.value)">Update Log</button>
      <button (click)="textValue=''">Clear</button>
```

ngModel: Binding-TEXTAREA

```
 <h2>Log <button (click)="log=''">Clear</button></h2>
       <pre>{{log}}</pre>`
})
export class TextAreaComponent {

  textValue = 'initial value';
  log = '';

  logText(value: string): void {
    this.log += `Text changed to '${value}'\n`;
  }
}
```

ngModel: Two Way Binding-TEXTAREA

- textarea behaves in a similar way to the textbox. Again, we use two-way data binding with ngModel to bind to the textValue property on the component class to keep user and programmatic changes in sync.
- This time we use the *canonical* form to specify the template reference variable but we could just as easily used #textarea instead.

ngModel: Binding-TEXTAREA

ngModel: Two Way Binding-CHECKBOX

- The change event is triggered when the user clicks on a checkbox. We use this event to pass a template reference variable into a function to log user action. We can determine whether or not the checkbox has been selected using the checked property of the argument.
- We also use the checked property to highlight the selected checkbox labels using class bindings. This uses the styles property to set the text color to OrangeRed for all the selected checkboxes.

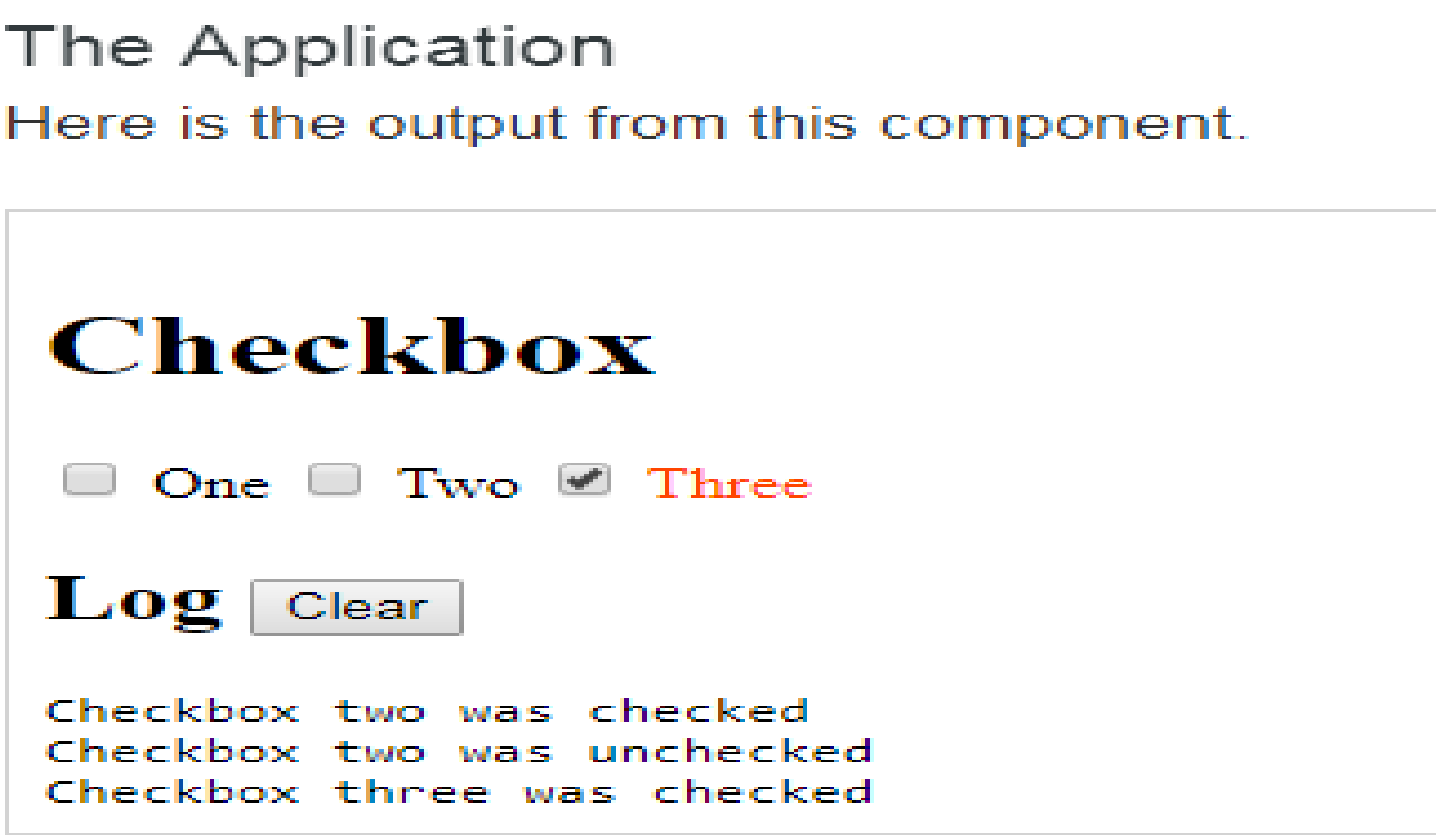
ngModel: Binding-CHECKBOX

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-checkbox',
  template: `
      <h1>Checkbox</h1>
      <p>
        <label [class.selected]="cb1.checked">
          <input #cb1 type="checkbox" value="one" (change)="logCheckbox(cb1)"> One
        </label>
        <label [class.selected]="cb2.checked">
          <input #cb2 type="checkbox" value="two" (change)="logCheckbox(cb2)"> Two
        </label>
        <label [class.selected]="cb3.checked">
          <input #cb3 type="checkbox" value="three" (change)="logCheckbox(cb3)"> Three
        </label>
      </p>
```

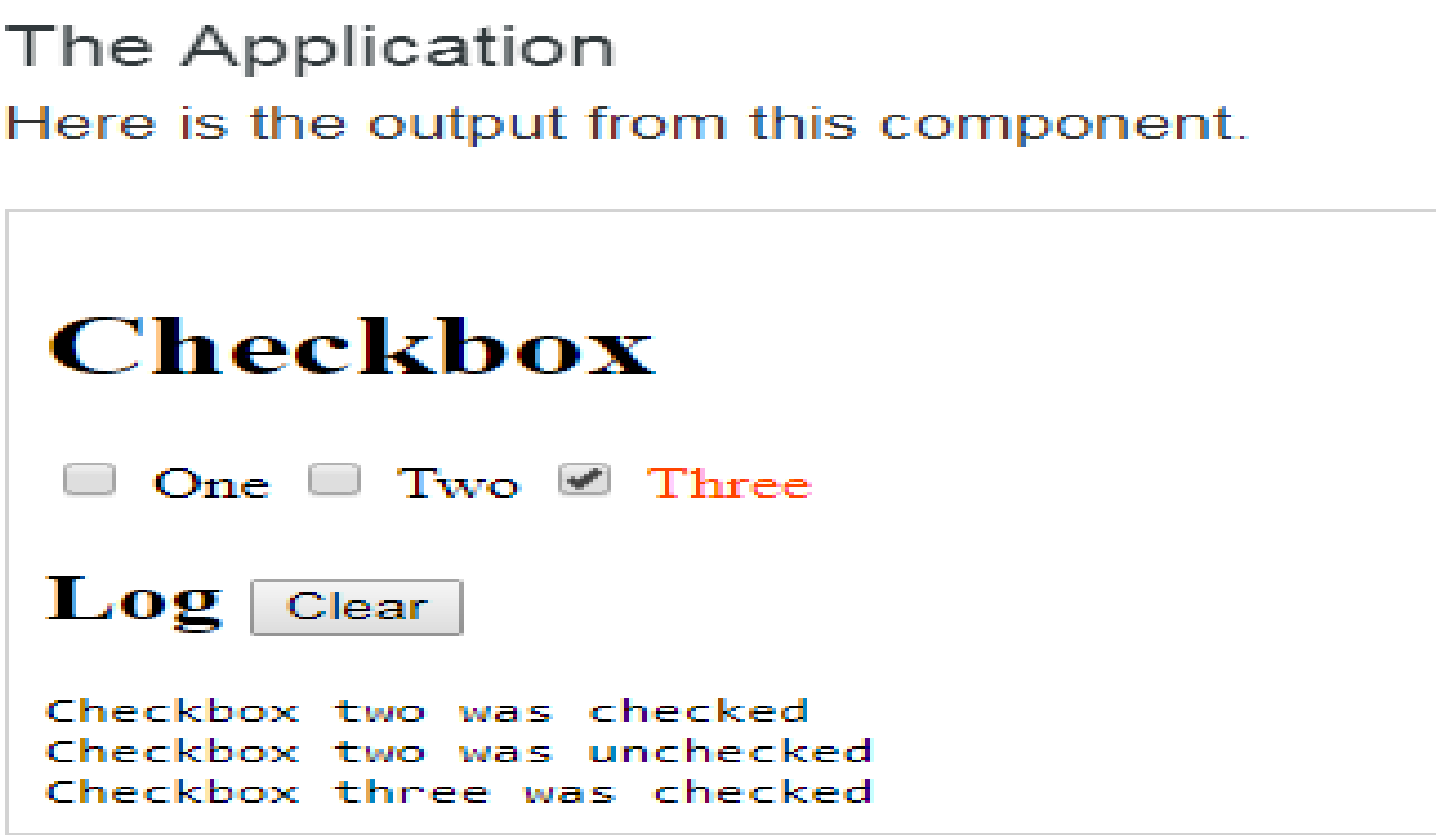
ngModel: Binding-CHECKBOX

```
 <h2>Log <button (click)="log=''">Clear</button></h2>
      <pre>{{log}}</pre>`,
  styles: ['.selected {color: OrangeRed;}']
})
export class CheckboxComponent {

  log = '';

  logCheckbox(element: HTMLInputElement): void {
    this.log += `Checkbox ${element.value} was ${element.checked ? '' :
   'un'}checked\n`;
  }
}
```

ngModel: Binding-CHECKBOX



ngModel: Binding-CHECKBOX



ngModel: Binding-RADIO

- The radio field behaves in a similar way to the checkbox. We use the change event on the <input> element to execute a template statement which passes a template reference variable into a component method.
- The value property property of the HTMLInputElement argument is used to log user's selection. This value property corresponds to the value attribute of the HTML <input> element.

ngModel: Two Way Binding-RADIO

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

@Component({
 selector: 'app-radio',
 template: `
      <h1>Radio</h1>
      <p>
        <label [class.selected]="r1.checked">
          <input #r1 type="radio" name="r" value="one" (change)="logRadio(r1)"> One
        </label>
        <label [class.selected]="r2.checked">
          <input #r2 type="radio" name="r" value="two" (change)="logRadio(r2)"> Two
        </label>
        <label [class.selected]="r3.checked">
          <input #r3 type="radio" name="r" value="three" (change)="logRadio(r3)"> Three
        </label>
      </p>
```

ngModel: Binding-RADIO

```
      <h2>Log <button (click)="log=''">Clear</button></h2>
      <pre>{{log}}</pre>`,
  styles: ['.selected {color: OrangeRed;}']
})
export class RadioComponent {

  log = '';

  logRadio(element: HTMLInputElement): void {
    this.log += `Radio ${element.value} was selected\n`;
  }
}
```

ngModel: Binding-RADIO

ngModel: Binding-Drop-down List

- We use an array to populate the drop-down with a list of values. The built-in ngFor directive comes in handy here to loop through the array and set up the option values.
- [(ngModel)] sets up two-way data binding on the select element. This will ensure that the drop-down list will display the current item when it is opened. When a new value is chosen from the list, two-way binding also ensures that the current property on the component is automatically updated.
- An additional binding to the change event of the select element logs the user action by calling a method in the template statement.

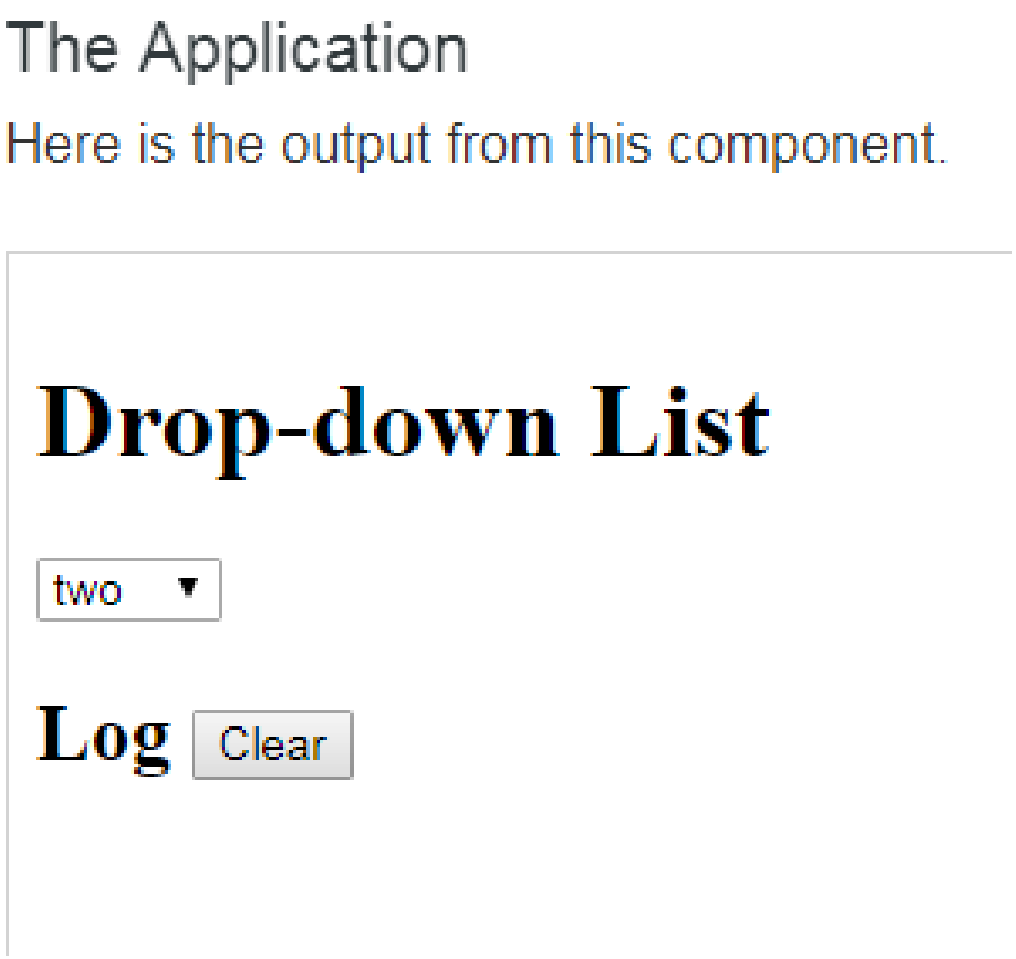
ngModel: Binding-Drop-down List

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

@Component({
  selector: 'app-drop-down',
  template: `
      <h1>Drop-down List</h1>
      <select #select [(ngModel)]="current" (change)="logDropdown(select.value)">
         <option *ngFor="let item of list" [value]="item.id">{{item.name}}</option>
      </select>

      <h2>Log <button (click)="log=''">Clear</button></h2>
      <pre>{{log}}</pre>`
})
```

ngModel: Binding-Drop-down List

```
export class DropDownComponent {

  list: any = [
    {id: 1, name: 'one'},
    {id: 2, name: 'two'},
    {id: 3, name: 'three'}
  ];
  current = 2;
  log = '';

  logDropdown(id: number): void {
    const NAME = this.list.find((item: any) => item.id === +id).name;
    this.log += `Value ${NAME} was selected\n`;
  }
}
```

ngModel: Binding-Drop-down List



Property binding

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-property',
  template: `
    <h1 [textContent]="'Name: ' + person.name"></h1>

    <button (click)="person = male" [disabled]="person.sex=='m'">Male</button>
    <button (click)="person = female" [disabled]="person.sex=='f'">Female</button>
```

Property binding

```
 <p><img [src]="person.photo" [alt]="person.name" [title]="person.name"></p>
   <p [hidden]="!person.rating">
    Rating: <span [innerHTML]="'&#10032;'.repeat(person.rating)"></span>
   </p>`
})
export class PropertyComponent {
  female = {
   name: 'Turanga Leela',
   sex: 'f',
   rating: 4,
   photo: 'assets/images/leela.jpg'
  };
```

Property binding

```
    male = {
    name: 'Philip J. Fry',
    sex: 'm',
    photo: 'assets/images/fry.jpg'
  };
  person: any = this.female;
}
```

Property binding

The Application

Here is the output from this component.

Name: Turanga Leela

Male | Female

Rating: ☆☆☆☆☆

Property binding

- Property binding means that all sorts of previous built-in directives in Angular 1 go away

Ng1 Directive	Functionality	Replaced with
ng-disabled	Disables the control – used on inputs, selects, textareas	[disabled]="expression"
ng-src/ng-href	Replacement for src/href properties since interpolation didn't play well with them	[src]="expression" [href]="expression"
ng-show/ng-hidden	Hides/shows an element based on the truthiness of an expression	[hidden]="expression"

Property binding

Text

- We use two-way data binding with ngModel to bind to the textValue property on the component class.
- Two-way binding allows us to use the property to set the initial value of an <input>, and then have the user's changes flow back to the property. We can also change the property value in the code and have these changes reflected in the <input>.

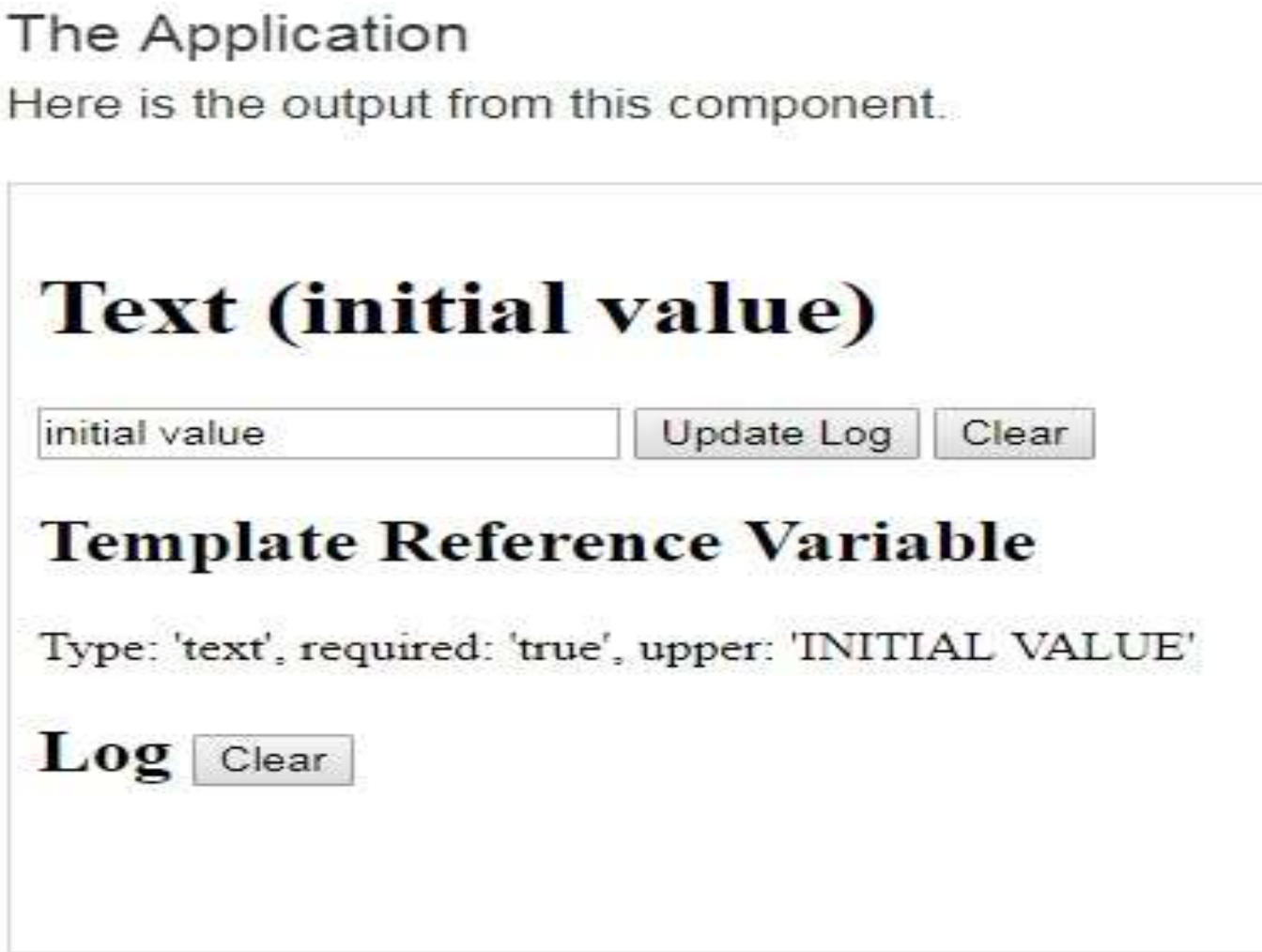
Property binding

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

@Component({
  selector: 'app-text-box',
  template: `
      <h1>Text ({{textValue}})</h1>
      <input #textbox type="text" [(ngModel)]="textValue" required>
      <button (click)="logText(textbox.value)">Update Log</button>
      <button (click)="textValue=''">Clear</button>

      <h2>Template Reference Variable</h2>
      Type: '{{textbox.type}}', required: '{{textbox.hasAttribute('required')}}',
      upper: '{{textbox.value.toUpperCase()}}'
```

Property binding

```
 <h2>Log <button (click)="log=''">Clear</button></h2>
      <pre>{{log}}</pre>`
})
export class TextComponent {

  textValue = 'initial value';
  log = '';

  logText(value: string): void {
    this.log += `Text changed to '${value}'\n`;
  }
}
```

Property binding



What is Angular Module?

Angular apps are modular and Angular has its own modularity system called Angular modules or NgModules.
Every Angular app should have at least one module class, the root module.
We bootstrap that module to launch the application.

Angular 2 NgModules contains imports, declarations, and bootstrap.

imports - other modules whose exported classes are needed by component templates declared in this module.
declarations - the view classes that belong to this module. Angular has three kinds of view classes: components, directives, and pipes.
bootstrap - the main application view, called the root component, that hosts all other app views. Only the root module should set this bootstrap property.

What is Angular Module?

```
import {NgModule} from '@angular/core'
import {BrowserModule} from '@angular/platform-browser'
import {AppComponent} from './app.component'

  @NgModule({
imports: [BrowserModule],
declarations: [AppComponent],
bootstrap: [AppComponent]
})
export class AppModule{
}
```

ANGULAR TEMPLATE FORM-DEMO

#App Setup

Here's our file structure:

```
|- app/
|- app.component.html
|- app.component.ts
|- app.module.ts
|- main.ts
|- user.interface.ts
|- index.html
|- styles.css
|- tsconfig.json
```

Demo

```
// user.interface.ts

export interface User {
name: string; // required with minimum 5 characters
address: {
street: string; // required
postcode: string;
}
}
```

// app.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms';

   import { AppComponent } from './app.component';

   @NgModule({
imports: [ BrowserModule, FormsModule ], // import forms module here
declarations: [ AppComponent ],
bootstrap: [ AppComponent ]
})

   export class AppModule { }
```

App.component.ts

```
import { Component, OnInit } from '@angular/core';
import { User } from './user.interface';
  @Component({
//moduleId: module.id,
selector: 'my-app',
templateUrl: 'app.component.html',
})
export class AppComponent implements OnInit {
```

App.component.ts

```
public user: User; // our model
    ngOnInit() {       // we will initialize our form here
this.user = {
name: '',
address: {
street: '',
postcode: '8000' // set default value to 8000
}
};
}
save(model: User, isValid: boolean) {
// check if model is valid
// if valid, call API to save customer
console.log(model, isValid);
}
}
```

app.component.html

```
<!-- app.component.html -->
...
<form #f="ngForm" novalidate (ngSubmit)="save(f.value, f.valid)">
<!-- we will place our fields here -->
<!--name-->
<div>
<label>Name</label>
<!--bind name to ngModel, it's required with minimum 5 characters-->
<input type="text" name="name" [(ngModel)]="user.name" #name="ngModel" required minlength
   ="5">
<!--show error only when field is not valid & it's dirty or form submited-->
<small [hidden]="name.valid || (name.pristine && !f.submitted)">
Name is required (minimum 5 characters).
</small>
</div>
```

app.component.html

```
<!--adrress group-->
<div ngModelGroup="address">
<!--street-->
<div>
<label>Street</label>
<input type="text" name="street" [(ngModel)]="user.address.street" #street="ngModel"
required>
<small [hidden]="street.valid || (street.pristine && !f.submitted)" class="text-danger">
Street is required.
</small>
</div>
<!--postcode-->
<div>
<label>Post code</label>
<input type="text" name="postcode" [(ngModel)]="user.address.postcode">
</div>
</div>
<button type="submit">Submit</button>
</form>
```

Index.html

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>TestProject</title>
  <base href="/">

  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <my-app></my-app>
</body>
</html>
```

Validation rules to the Form Object.

<!-- app.component.html -->

```
<div class="container">
  <h1>
    Welcome to {{title}}!!
  </h1>
  <form [formGroup]="angForm" novalidate>
      <div class="form-group">
          <label>Name:</label>
          <input class="form-control" formControlName="name" type="text">
      </div>
      <div *ngIf="angForm.controls['name'].invalid && (angForm.controls['name'].dirty || angForm.controls['name'].touched)" class="alert alert-
    danger">
          <div *ngIf="angForm.controls['name'].errors.required">
            Name is required.
          </div>
```

Validation rules to the Form Object.

```
</div>
      <div class="form-group">
        <label>Address:</label>
        <input class="form-control" formControlName="address" type="text">
      </div>
      <div *ngIf="angForm.controls['address'].invalid && (angForm.controls['address'].dirty || angForm.controls['address'].touched)" class="alert alert-danger">
        <div *ngIf="angForm.controls['email'].errors.required">
          email is required.
        </div>
      </div>
      <button type="submit"
        [disabled]="angForm.pristine || angForm.invalid" class="btn btn-success">
        Save
      </button>
 </form>
 <br />
 <p>Form value: {{ angForm.value | json }}</p>
 <p>Form status: {{ angForm.status | json }}</p>
</div>
```

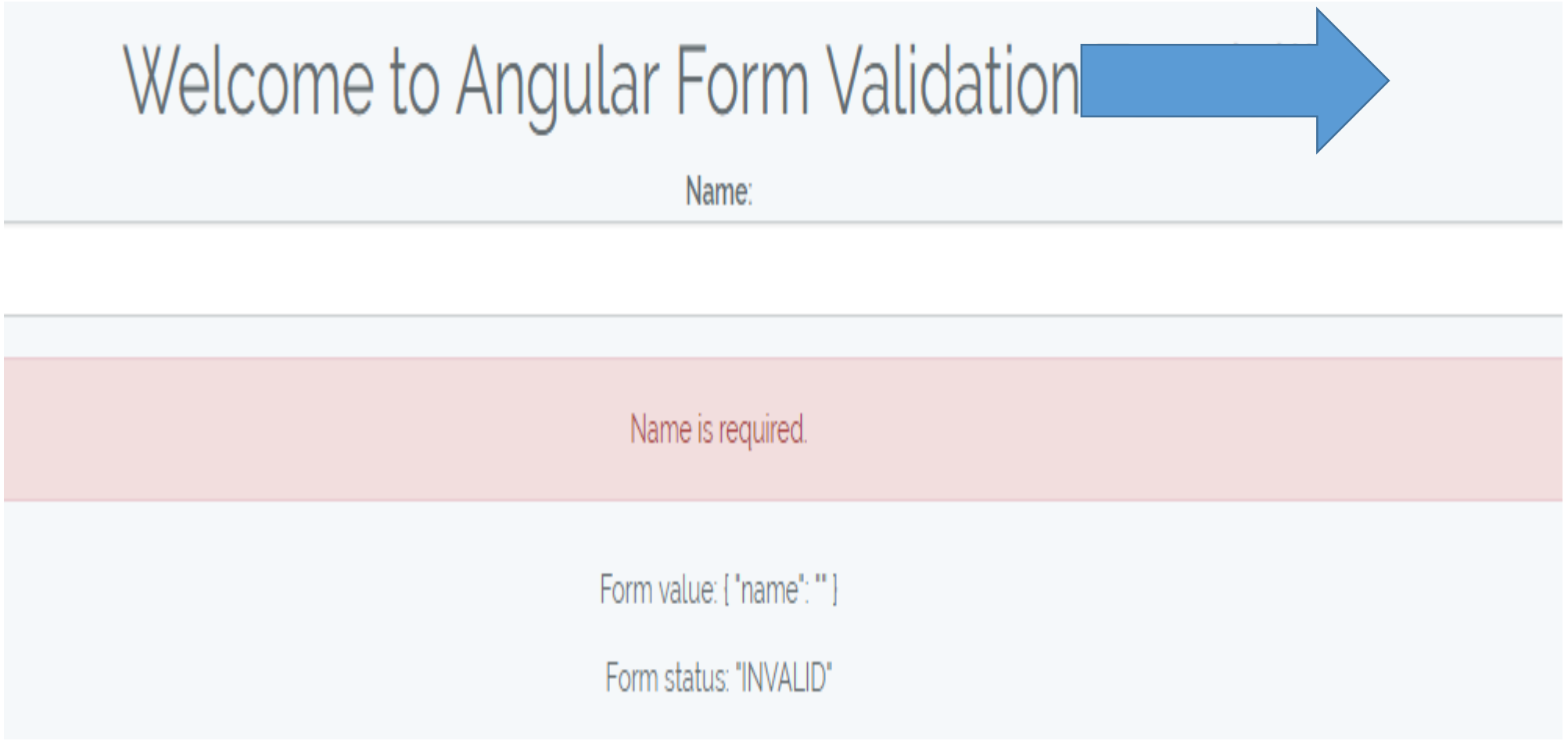
app.component.ts file looks like this.

```
import { Component } from '@angular/core';
import { FormGroup,  FormBuilder,  Validators } from '@angular/forms';
@Component({
 selector: 'app-root',
 templateUrl: './app.component.html',
 styleUrls: ['./app.component.css']
})
export class AppComponent {
 title = 'Angular Form Validation;
  angForm: FormGroup;
  constructor(private fb: FormBuilder) {
   this.createForm();
 }
  createForm() {
   this.angForm = this.fb.group({
     name: ['', Validators.required ],
     address: ['', Validators.required ]
   });
 }
}
```

app.module.ts file looks like this.

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { ReactiveFormsModule } from '@angular/forms';

    import { AppComponent } from './app.component';

    @NgModule({
declarations: [
AppComponent
],
imports: [
BrowserModule, ReactiveFormsModule
],
providers: [],
bootstrap: [AppComponent]
})
export class AppModule { }
```

Validation rules to the Form Object.



ANGULAR ROUTING AND SINGLE PAGE APPLICATION

SPA BENEFITS

- ❑ Runs faster
- ❑ Gives better UX
- ❑ Uses less network bandwidth

What is Angular Router?

- Angular Router is an official Angular routing library, written and maintained by the Angular Core Team.
- It’s a JavaScript router implementation that’s designed to work with Angular and is packaged as @angular/router.
- First of all, Angular Router takes care of the duties of a JavaScript router:
- it activates all required Angular components to compose a page when a user navigates to a certain URL
- it lets users navigate from one page to another without page reload
- it updates the browser’s history so the user can use the *back* and *forward* buttons when navigating back and forth between pages.

ANGULAR 2

HOW TO CREATE ANGULAR2 ROUTING/SPA

Angular 2 Router?

Step 1: create project

-ng new web-router

-ng serve --open

Step 2: Create 2 Components

-ng generate component about

-ng g c services

-ng serve --open

Step 3: Create an app.router.ts file

create a route module to store our routing information specifically. Place it within the /app folder.

app.router.ts

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

import { AppComponent } from './app.component';
import { AboutComponent } from './about/about.component';
import { ServicesComponent } from './services/services.component';
```

app.router.ts

```
export const router: Routes = [
    { path: '', redirectTo: 'about', pathMatch: 'full' },
    { path: 'about', component: AboutComponent },
    { path: 'services', component: ServicesComponent }
];

export const routes: ModuleWithProviders =
   RouterModule.forRoot(router);
```

app.router.ts

In a nutshell, we're importing the necessary routing members on the first 2 lines. Then we're importing the components that currently exist within our application.

After that, we're defining a constant that contains the routes that we wish to create. "path" defines the URL; so "about"

becomes **http://someurl.com/about** -- then we specify the component associated with that view.

app.module.ts

Step 3: Import the router to app.module.ts

```
import { routes } from './app.router';
import { AppComponent } from './app.component';
import { AboutComponent } from './about/about.component';
import { ServicesComponent } from './services/services.component';
@NgModule({
declarations: [
AppComponent,
AboutComponent,
ServicesComponent
],
imports: [
BrowserModule,
routes
],
providers: [],
bootstrap: [AppComponent]
})
export class AppModule { }
```

app.component.html

Step 4:

Now within app.component.html, we add:

```
<div>
<router-outlet></router-outlet>
</div>
<div>
<ul class="nav navbar-nav">
<li>
<a routerLink="about">About</a>
</li>
<li>
<a routerLink="services">Services</a>
</li>
</ul>
</div>
```

Angular Router

Step 6: Ensure <base href="/"> is in index.html

If you used the angular-cli to create the project you're working with, it will automatically add this in the head tag. If it's not there, add it:

<base href="/">