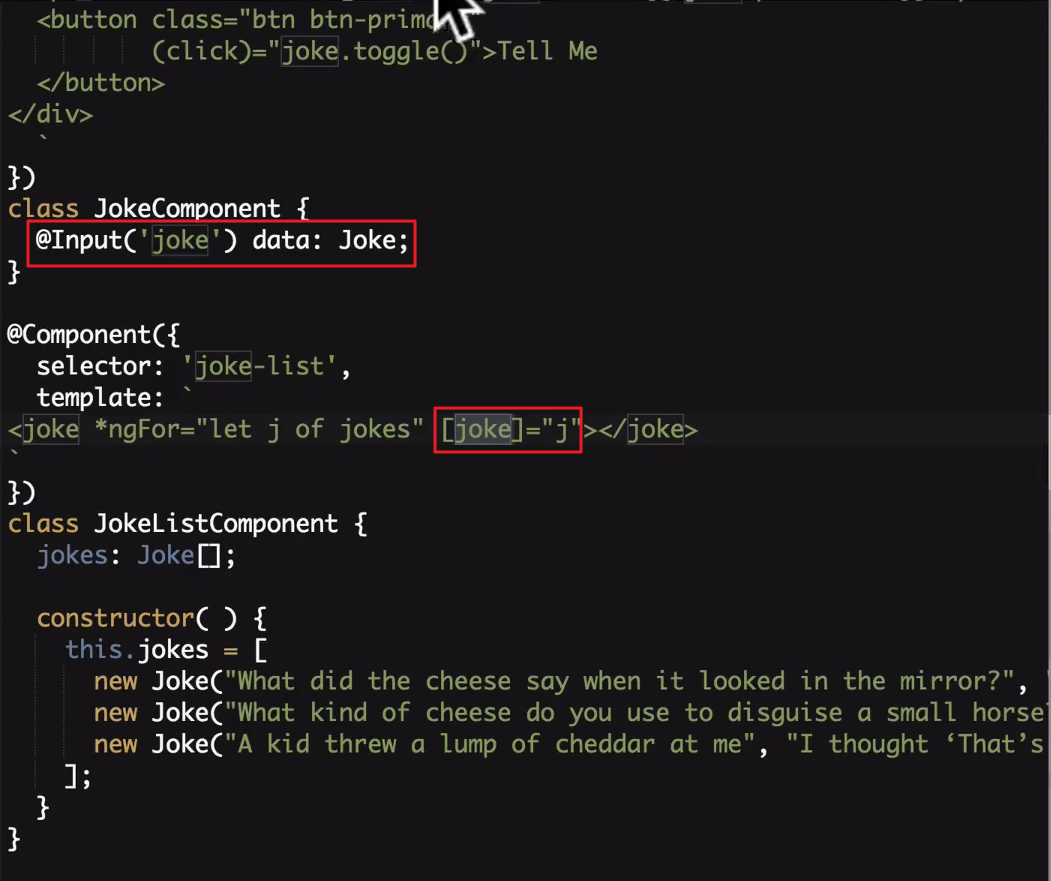
<https://codecraft.tv/courses/angular/>

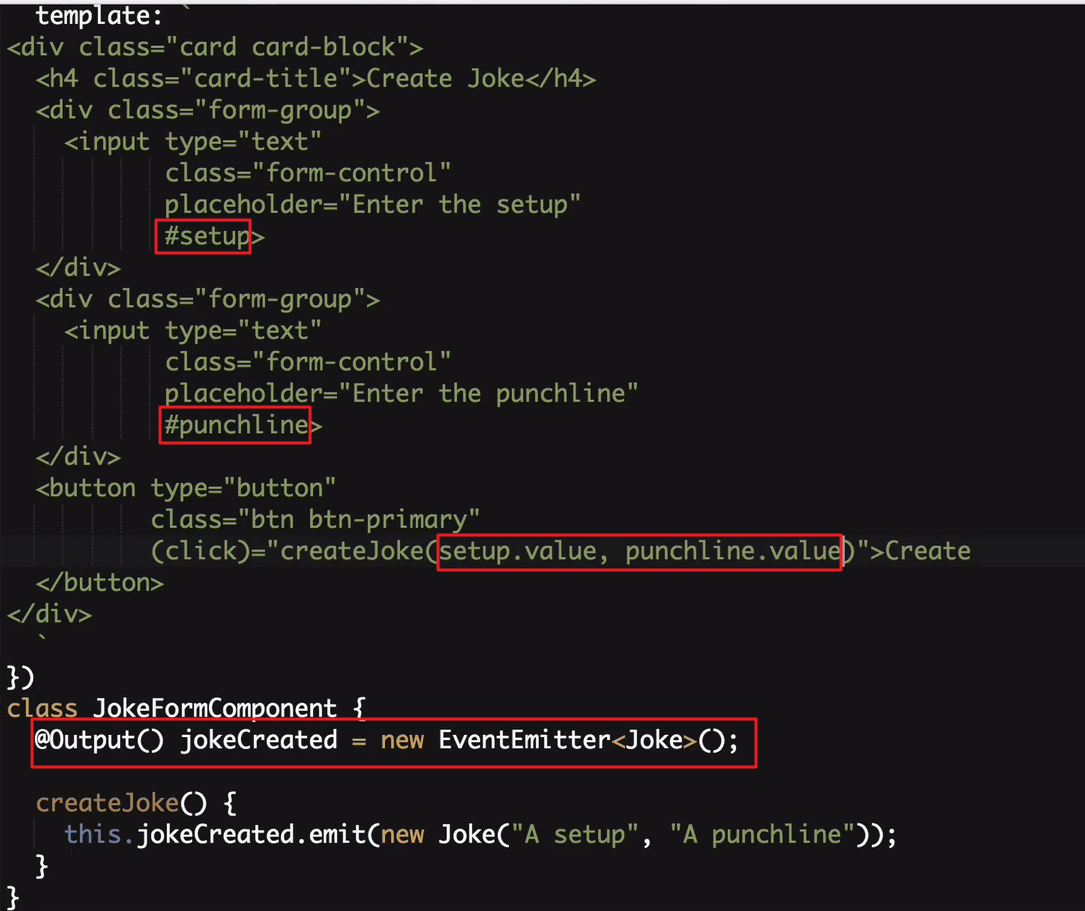
[https://github.com/codecraftpro/](https://github.com/codecraftpro/angular2-sample-code/blob/master/2.es6-typescript/10.class-interface/script.ts)

**QuickStart**

@Input

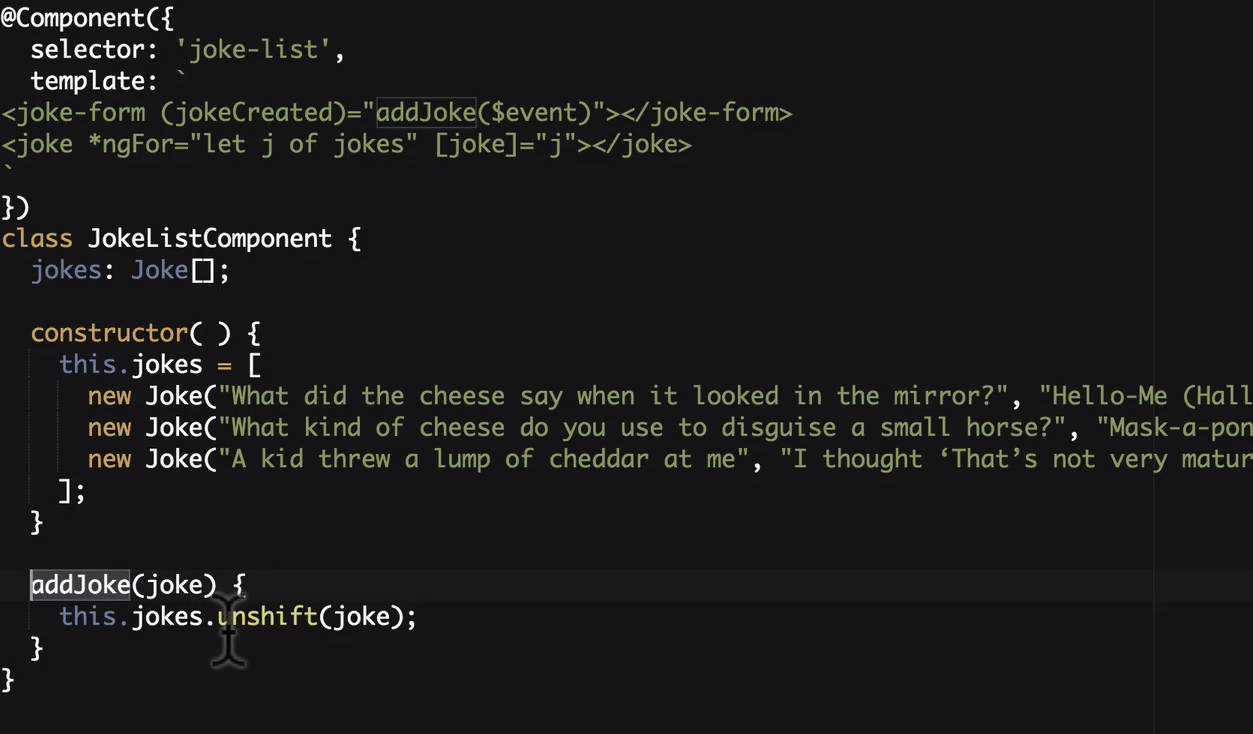


템플릿 안쪽에서 참조



EventEmitter로 컴포넌트간의 데이터 전송



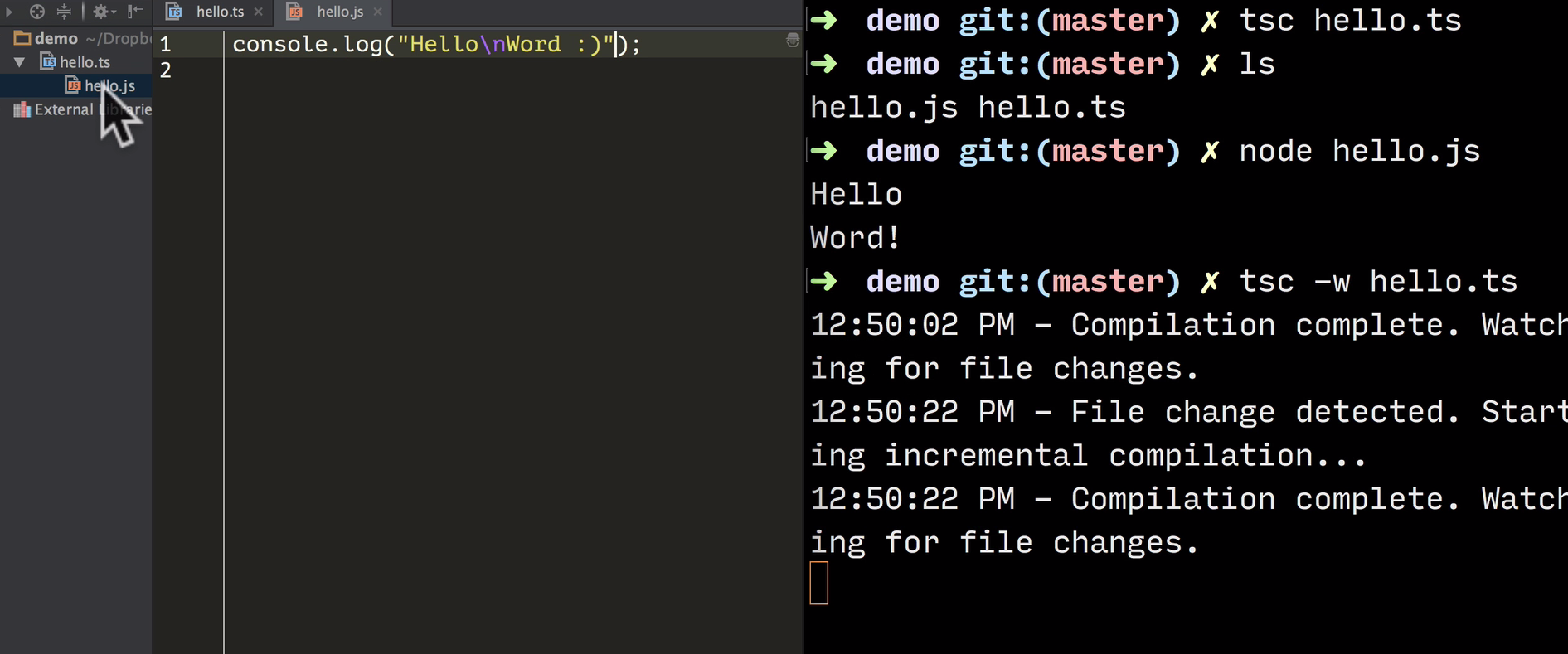


<http://plnkr.co/edit/b0F6Dhb40Hm5zfiamAix?p=preview>

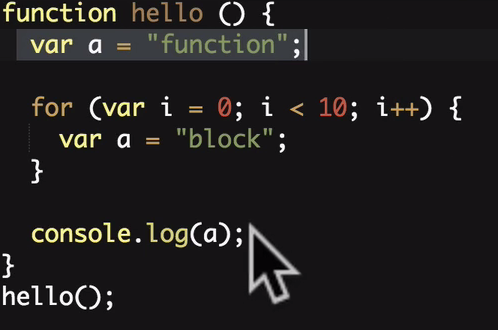
|  |
| --- |
| **import** {platformBrowserDynamic} from '@angular/platform-browser-dynamic'; **import** {Component, NgModule, Input, Output, EventEmitter} from '@angular/core'; **import** {BrowserModule} from '@angular/platform-browser';  **class** Joke {  **public** setup: string;  **public** punchline: string;  **public** hide: boolean;   constructor(setup: string, punchline: string) {  **this**.setup = setup;  **this**.punchline = punchline;  **this**.hide = **true**;  }   toggle() {  **this**.hide = !**this**.hide;  } }   @Component({  selector: 'joke-form',  template: ` <div class="card card-block">  <h4 class="card-title">Create Joke</h4>  <div class="form-group">  <input type="text"  class="form-control"  placeholder="Enter the setup"  #setup>  </div>  <div class="form-group">  <input type="text"  class="form-control"  placeholder="Enter the punchline"  #punchline>  </div>  <button type="button"  class="btn btn-primary"  (click)="createJoke(setup.value, punchline.value)">Create  </button> </div>  ` }) **class** JokeFormComponent {  @Output() jokeCreated = **new** EventEmitter<Joke>();   createJoke(setup: string, punchline: string) {  **this**.jokeCreated.emit(**new** Joke(setup, punchline));  } }  @Component({  selector: 'joke',  template: ` <div class="card card-block">  <h4 class="card-title">  {{data.setup}}  </h4>  <p class="card-text"  [hidden]="data.hide">{{data.punchline}}</p>  <a (click)="data.toggle()"  class="btn btn-warning">Tell Me  </a>  <a (click)="deleteItem()"  class="btn btn-danger">Delete  </a>  </div>  ` }) **class** JokeComponent {  @Input('joke') data: Joke;  @Output() jokeDeleted = **new** EventEmitter<Joke>();   deleteItem() {  **this**.jokeDeleted.emit(**this**.data);  } }  @Component({  selector: 'joke-list',  template: ` <joke-form (jokeCreated)="addJoke($event)"></joke-form> <joke \*ngFor="let j of jokes" [joke]="j" (jokeDeleted)="deleteJoke($event)"></joke>  ` }) **class** JokeListComponent {  jokes: Joke[];   constructor() {  **this**.jokes = [  **new** Joke("What did the cheese say when it looked in the mirror?", "Hello-me (Halloumi)"),  **new** Joke("What kind of cheese do you use to disguise a small horse?", "Mask-a-pony (Mascarpone)"),  **new** Joke("A kid threw a lump of cheddar at me", "I thought ‘That’s not very mature’"),  ];  }   addJoke(joke) {  **this**.jokes.unshift(joke);  }   deleteJoke(joke) {  **let** indexToDelete = **this**.jokes.indexOf(joke);  **if** (indexToDelete !== -1) {  **this**.jokes.splice(indexToDelete,1);  }  } }  @Component({  selector: 'app',  template: ` <joke-list></joke-list>  ` }) **class** AppComponent { }  @NgModule({  imports: [BrowserModule],  declarations: [  AppComponent,  JokeComponent,  JokeListComponent,  JokeFormComponent  ],  bootstrap: [AppComponent] }) **export class** AppModule { }  platformBrowserDynamic().bootstrapModule(AppModule); |

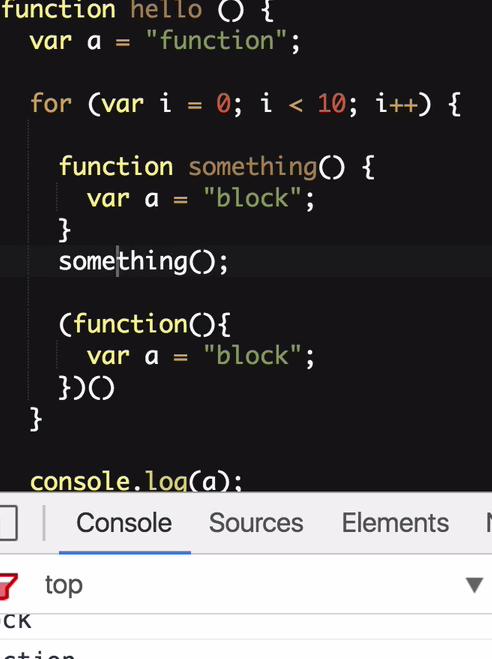
**E6 & TypeScript**

타입스크립트 컴파일 실행



**var**

되기때문에

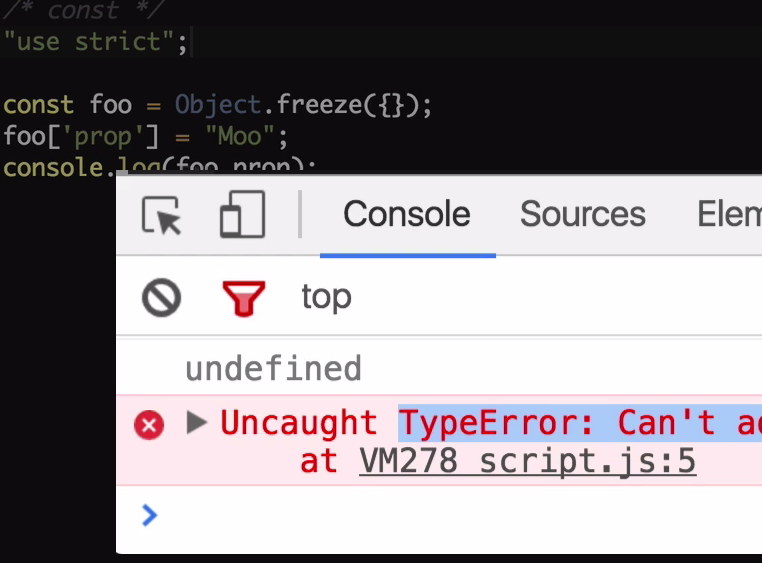
처럼 처리를 해야됐다.

let 은 안쪾에 세이프하게 해준다.

**const**

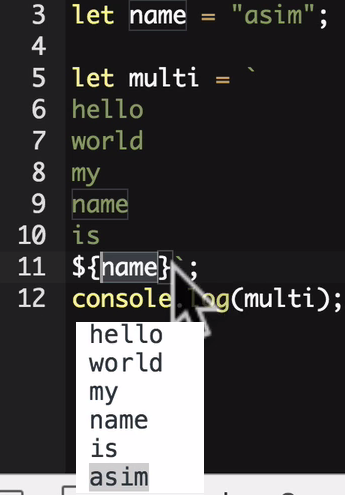
무조건 초기값이 있어야하며 값을 변경할수 없다.

블럭안에서만 효율성이 있다.



오브젝트 안쪽내용까지 다 수정못하게 하려면 Object.freeze로 초기값을 해도된다.

**template string**



**Fat Arrow Function**

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  | | |

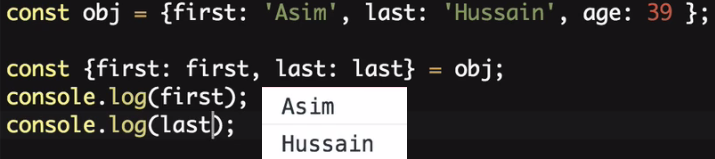
|  |  |
| --- | --- |
|  |  |

호출자의 기준으로 this가 지정되는것을 => 형식으로 쓴다면 그것또한 해제시켜준다 예를보자

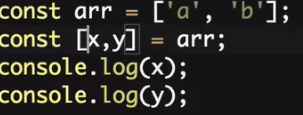
|  |  |
| --- | --- |
| 문제발생 | 기존처럼 문제해결 |
| => 로 문제해결 | |

**Destructuring**

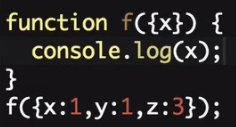
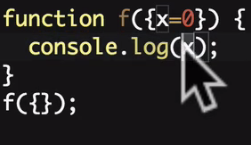
오브젝트 형을 바로 변수로받기







파라미터로 받기

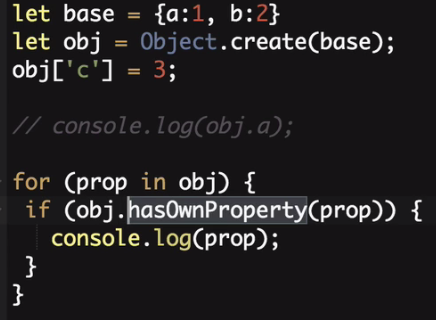
 

**For of**

기존

|  |  |
| --- | --- |
|  |  |
| of |  |

다른 Object와 프로퍼티 비교하기

 c 하나 나옴

**Map & Set**

|  |  |
| --- | --- |
|  |  |
| map.clear() , map.keys(), map.values(), map.entries() |  |

|  |  |
| --- | --- |
|  |  |
| set.values(), set.size, set.clear(), set.add() |  |

**Promises**

|  |  |
| --- | --- |
| 기존 |  |
|  |  |

**Class & Interface**

|  |  |
| --- | --- |
|  |  |

|  |
| --- |
| **interface** Human {  firstName: **string**;  lastName: **string**;  name?: Function;  isLate?(time: Date): Function; }  **class** Person **implements** Human {  **constructor**(**public** firstName, **public** lastName) {  }   **public** name() {  **return** `${**this**.firstName} ${**this**.lastName}`;  }   **protected** whoAreYou() {  **return** `Hi i'm ${**this**.name()}`;  } }  **class** Student **extends** Person {  **constructor**(**public** firstName, **public** lastName, **public** course) {  **super**(firstName, lastName);  }   whoAreYou() {  **return** `${**super**.whoAreYou()} and i'm studying ${**this**.course}`;  } }  **let** asim = **new** Student("Asim", "Hussain", "typescript"); console.log(asim.whoAreYou()); |

**Decorator**

선언시에 가로채서 변환을 해준다.

|  |
| --- |
| **function** Student(config) {  **return function** (target) {  Object.defineProperty(target.prototype, 'course', {value: () => config.course})  } }    @Student({  course: "angular3" }) **class** Person {  **constructor**(**private** firstName, **private** lastName) {  }   **public** name() {  **return** `${**this**.firstName} ${**this**.lastName}`;  }   **protected** whoAreYou() {  **return** `Hi i'm ${**this**.name()}`;  } }  **let** asim = **new** Person("Asim", "Hussain"); //noinspection TypeScriptUnresolvedFunction console.log(asim.course());  tsc --experimentalDecorators decorator.ts  node decorator.js |

**Module**

export 로 다른 모듈에서 내자신을 import가능하도록한다.

|  |  |
| --- | --- |
| script.ts | utils.ts |
| **import** \* **as** utils **from** './utils'; console.log(utils.square(4)); utils.cow();  import {square, cow} from "./utils"  import square from "./utils"  import {square as sqr, cow} from "./utils" | **export function** square(x) {  **return** Math.pow(x, 2) }  **export function** cow() {  console.log("Mooooo!!!") }  export {square: square, cow: cow} 처럼해도됨 |

**Types**

|  |
| --- |
| "use strict";  // Core **let** decimal: **number** = 6; **let** done: **boolean** = **false**; **let** color: **string** = "blue"; **let** list: **number**[] = [1, 2, 3]; **let** list2: Array<**number**> = [1, 2, 3];  // Function **let** fun: Function = () => console.log("Hello"); **function** returnNumber(): **number** {  **return** 1; }  // Void **function** returnNothing(): **void** {  console.log("Moo"); }  // Enum **enum** Direction {  *Up*,  *Down*,  *Left*,  *Right* } **let** go: Direction; go = Direction.*Up*;   // Class **class** Person { } **let** person: Person; **let** people: Person[];  // Any **let** notsure: **any** = 1; notsure = "hello"; // This is fine since we don't do type checking with any  // Type Assertion **let** value: **any** = "Asim Hussain"; **let** length: **number** = (<**string**>value).length;   // Generics **class** Audio { } **class** Video { }  **class** Post<T> {  content: T; }  **let** audioPost: Post<Audio>; **let** videoPost: Post<Video>; |

npm install -g typings

typings install jquery --save --source dt --global

typings install ionic --save --source dt --global

**Angular CLI**

|  |  |
| --- | --- |
| npm install -g angular-cli  ng -v  ng new newProject  ng server  -----  ng generate component Header    ng g component LoginButton  ng g directive My  ng g pipe My  ng g servcice My  ng g class MyClass  ng g interface MyInterface  ng g enum MyEnum  -----  ng build    ng build --dev  또는  ./node\_modules/@angular/cli/bin/ng build -w -op ../webapp --dev  운영시  ng build --prod |  |

npm install moment --save

npm install @types/moment --save

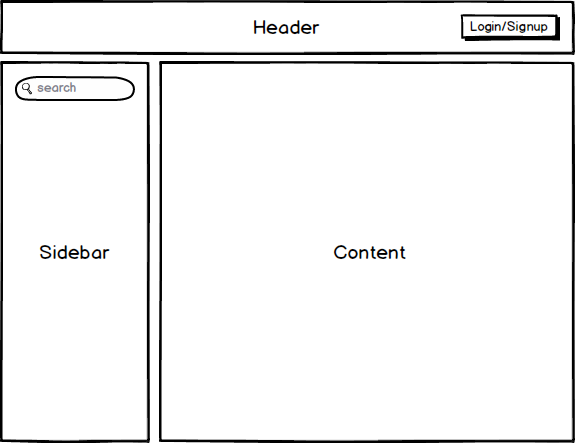
npm install bootstrap@next

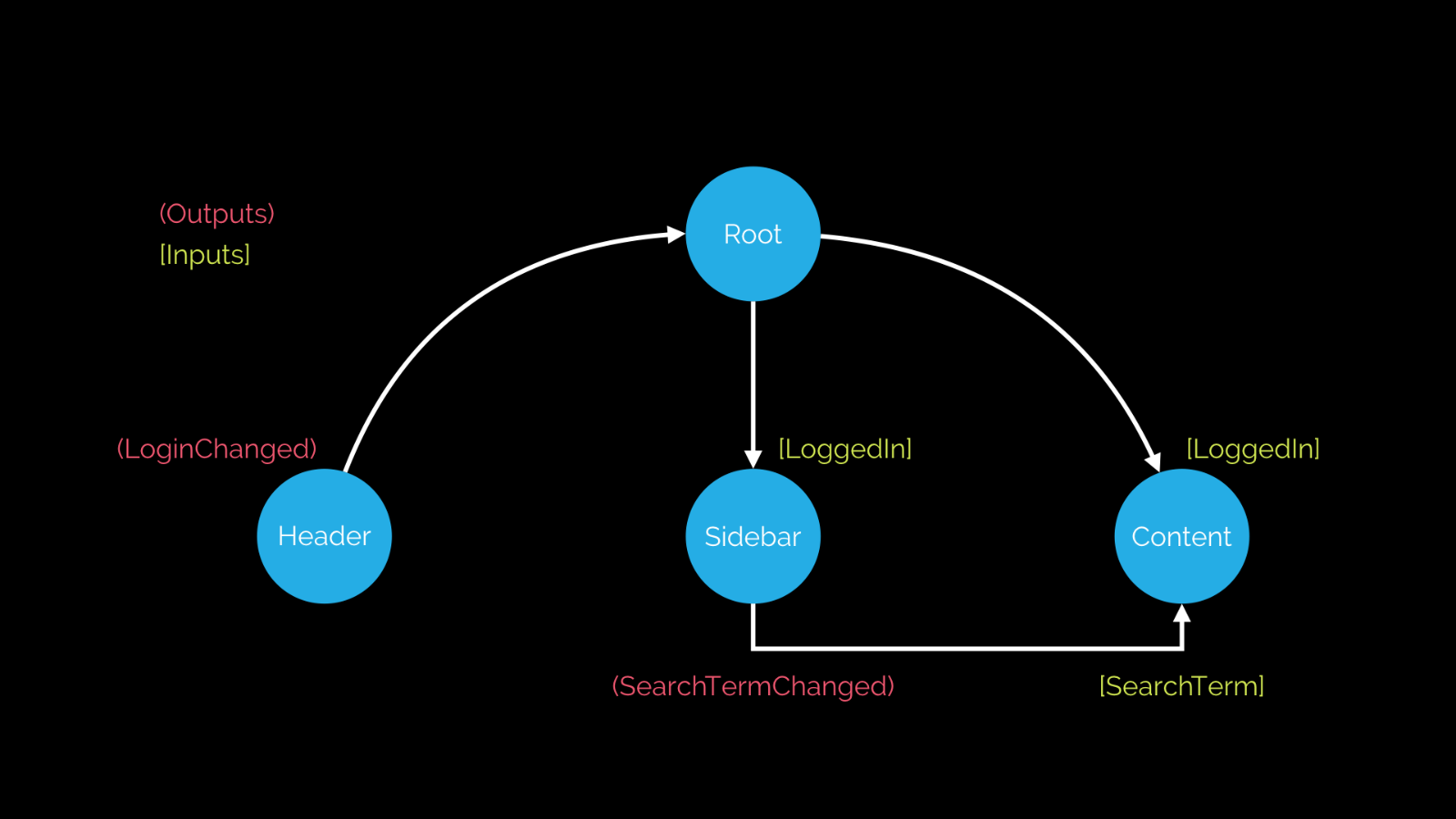
ng test

ng help

**Component**

Architecting with Components





<header (loginChanged)="loggedIn = $event"></header>

<sidebar (searchTermChanged)="searchTerm = $event"></sidebar>

<content [searchTerm]="searchTerm"></content>

**Templates, Styles & View Encapsulation**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| @Component({  selector: 'joke-form',  templateUrl: 'joke-form-component.html',  styleUrls: [  'joke-form-component.css'  ],  encapsulation: ViewEncapsulation.Emulated  // encapsulation: ViewEncapsulation.Native  // encapsulation: ViewEncapsulation.None  }) **class** JokeFormComponent {  @Output() jokeCreated = **new** EventEmitter<Joke>();   createJoke(setup: **string**, punchline: **string**) {  **this**.jokeCreated.emit(**new** Joke(setup, punchline));  } }  encapsulation: ViewEncapsulation.Emulated // encapsulation: ViewEncapsulation.Native // encapsulation: ViewEncapsulation.None   |  |  | | --- | --- | | ViewEncapsulation.None | encapsulated none screen  encapsulated none html | | ViewEncapsulation.Native |  | | ViewEncapsulation.Emulated | 기본값  encapsulation emulated html  encapsulation emulated css | |  |  | |

**Content Projection**

|  |
| --- |
| @Component({  selector: 'joke',  template: ` <div class="card card-block">  <h4 class="card-title">  <ng-content select=".setup"></ng-content>  </h4>  <p class="card-text"  [hidden]="data.hide">  <ng-content select=".punchline"></ng-content>  </p>  <a class="btn btn-primary"  (click)="data.toggle()">Tell Me  </a> </div> ` }) **class** JokeComponent {  @Input('joke') data: Joke; }  @Component({  selector: 'joke-list',  template: ` <joke-form (jokeCreated)="addJoke($event)"></joke-form> <joke \*ngFor="let j of jokes" [joke]="j">  <span class="setup">{{ j.setup }}?</span>  <h1 class="punchline">{{ j.punchline }}</h1> </joke>  ` }) **class** JokeListComponent {  jokes: Joke[];   **constructor**() {  **this**.jokes = [  **new** Joke("What did the cheese say when it looked in the mirror", "Hello-me (Halloumi)"),  **new** Joke("What kind of cheese do you use to disguise a small horse", "Mask-a-pony (Mascarpone)"),  **new** Joke("A kid threw a lump of cheddar at me", "I thought ‘That’s not very mature’"),  ];  }   addJoke(joke) {  **this**.jokes.unshift(joke);  } } |

**Lifecycle Hooks 라이프 사이클**

|  |
| --- |
| @Component({  selector: 'joke',  template: ` <div class="card card-block">  <h4 class="card-title">  <ng-content select=".setup"></ng-content>  </h4>  <p class="card-text"  [hidden]="data.hide">  <ng-content select=".punchline"></ng-content>  </p>  <a class="btn btn-primary"  (click)="data.toggle()">Tell Me  </a> </div> ` }) **class** JokeComponent **implements** OnChanges,  OnInit,  DoCheck,  AfterContentInit,  AfterContentChecked,  AfterViewInit,  AfterViewChecked,  OnDestroy {  @Input('joke') data: Joke;   **constructor**() {  console.log(`new - data is ${**this**.data}`);  }   ngOnChanges(changes: SimpleChanges) {  console.log(`ngOnChanges - data is ${**this**.data}`);   **for** (**let** key **in** changes) {  console.log(`${key} changed.  Current: ${changes[key].currentValue}.  Previous: ${changes[key].previousValue}`);  }  }  ngOnInit() {  console.log(`ngOnInit - data is ${**this**.data}`);  }  ngDoCheck() {  console.log("ngDoCheck")  }  ngAfterContentInit() {  console.log("ngAfterContentInit");  }  ngAfterContentChecked() {  console.log("ngAfterContentChecked");  }  ngAfterViewInit() {  console.log("ngAfterViewInit");  }  ngAfterViewChecked() {  console.log("ngAfterViewChecked");  }  ngOnDestroy() {  console.log("ngOnDestroy");  } }  Lifecycle Hooks |

**ViewChildren & ContentChildren**

|  |
| --- |
| @Component({  selector: 'joke',  template: ` <div class="card card-block">  <h4 class="card-title">  <ng-content select=".setup"></ng-content>  </h4>  <p class="card-text"  [hidden]="data.hide">  <ng-content select=".punchline"></ng-content>  </p>  <a class="btn btn-primary"  (click)="data.toggle()">Tell Me  </a> </div> ` }) **class** JokeComponent {   @Input('joke') data: Joke; }  @Component({  selector: 'joke-list',  template: ` <h4 #header>View Jokes</h4> <joke \*ngFor="let j of jokes" [joke]="j">  <span class="setup">{{ j.setup }}?</span>  <h1 class="punchline">{{ j.punchline }}</h1> </joke> <h4>Content Jokes</h4> <ng-content></ng-content> ` }) **class** JokeListComponent **implements** OnInit,  AfterContentInit,  AfterViewInit {   jokes: Joke[] = [  **new** Joke("What did the cheese say when it looked in the mirror", "Hello-me (Halloumi)"),  **new** Joke("What kind of cheese do you use to disguise a small horse", "Mask-a-pony (Mascarpone)")  ];   @ViewChild(JokeComponent) jokeViewChild: JokeComponent;  @ViewChildren(JokeComponent) jokeViewChildren: QueryList<JokeComponent>;  @ViewChild("header") headerEl: ElementRef;  @ContentChild(JokeComponent) jokeContentChild: JokeComponent;   **constructor**() {  console.log(`new - jokeViewChild is ${**this**.jokeViewChild}`);  console.log(`new - jokeContentChild is ${**this**.jokeContentChild}`);  }   ngAfterContentInit() {  console.log(`ngAfterContentInit - jokeContentChild is ${**this**.jokeContentChild}`);  }   ngAfterViewInit() {  console.log(`ngAfterViewInit - jokeViewChild is ${**this**.jokeViewChild}`);   **let** jokes: JokeComponent[] = **this**.jokeViewChildren.toArray();  console.log(jokes);   console.log(`ngAfterViewInit - headerEl is ${**this**.headerEl}`);  **this**.headerEl.nativeElement.textContent = "Best Joke Machine";  } } |

**Built-in**

**NgFor**

|  |
| --- |
| **import** {NgModule, Component} **from** '@angular/core'; **import** {BrowserModule} **from** '@angular/platform-browser'; **import** {platformBrowserDynamic} **from** '@angular/platform-browser-dynamic';   @Component({  selector: 'ngfor-example',  template: `<h4>NgFor</h4> <ul>  <li \*ngFor="let person of people; let i = index">  {{ i + 1 }} - {{ person.name }}  </li> </ul>  ` }) **class** NgForExampleComponent {  people: **any**[] = [  {  "name": "Douglas Pace"  },  {  "name": "Mcleod Mueller"  },  {  "name": "Day Meyers"  },  {  "name": "Aguirre Ellis"  },  {  "name": "Cook Tyson"  }  ]; }   @Component({  selector: 'ngfor-grouped-example',  template: `<h4>NgFor (grouped)</h4> <ul \*ngFor="let group of peopleByCountry">  <li>{{ group.country }}</li>  <ul>  <li \*ngFor="let person of group.people">  {{ person.name }}  </li>  </ul> </ul>  ` }) **class** NgForGroupedExampleComponent {   peopleByCountry: **any**[] = [  {  'country': 'UK',  'people': [  {  "name": "Douglas Pace"  },  {  "name": "Mcleod Mueller"  },  ]  },  {  'country': 'US',  'people': [  {  "name": "Day Meyers"  },  {  "name": "Aguirre Ellis"  },  {  "name": "Cook Tyson"  }  ]  }  ]; }  @Component({  selector: 'directives-app',  template: `  <ngfor-grouped-example></ngfor-grouped-example>  <ngfor-example></ngfor-example>  ` }) **class** DirectivesAppComponent { }   @NgModule({  imports: [BrowserModule],  declarations: [  NgForExampleComponent,  NgForGroupedExampleComponent,  DirectivesAppComponent],  bootstrap: [DirectivesAppComponent], }) **class** AppModule {  }  platformBrowserDynamic().bootstrapModule(AppModule);  NgFor Index  NgFor Grouped |

**NgIf & NgSwitch**

|  |
| --- |
| **import** {NgModule, Component} **from** '@angular/core'; **import** {BrowserModule} **from** '@angular/platform-browser'; **import** {platformBrowserDynamic} **from** '@angular/platform-browser-dynamic';  @Component({  selector: 'ngif-example',  template: ` <h4>NgIf</h4> <ul \*ngFor="let person of people">  <li \*ngIf="person.age < 30">  {{ person.name }} ({{ person.age }})  </li> </ul> ` }) **class** NgIfExampleComponent {   people: **any**[] = [  {  "name": "Douglas Pace",  "age": 35  },  {  "name": "Mcleod Mueller",  "age": 32  },  {  "name": "Day Meyers",  "age": 21  },  {  "name": "Aguirre Ellis",  "age": 34  },  {  "name": "Cook Tyson",  "age": 32  }  ]; }   @Component({  selector: 'ngswitch-example',  template: `<h4>NgSwitch</h4> <ul \*ngFor="let person of people"  [ngSwitch]="person.country">   <li \*ngSwitchCase="'UK'"  class="text-success">  {{ person.name }} ({{ person.country }})  </li>  <li \*ngSwitchCase="'USA'"  class="text-primary">  {{ person.name }} ({{ person.country }})  </li>  <li \*ngSwitchCase="'HK'"  class="text-danger">  {{ person.name }} ({{ person.country }})  </li>  <li \*ngSwitchDefault  class="text-warning">  {{ person.name }} ({{ person.country }})  </li> </ul>` }) **class** NgSwitchExampleComponent {   people: **any**[] = [  {  "name": "Douglas Pace",  "age": 35,  "country": 'MARS'  },  {  "name": "Mcleod Mueller",  "age": 32,  "country": 'USA'  },  {  "name": "Day Meyers",  "age": 21,  "country": 'HK'  },  {  "name": "Aguirre Ellis",  "age": 34,  "country": 'UK'  },  {  "name": "Cook Tyson",  "age": 32,  "country": 'USA'  }  ]; }   @Component({  selector: 'directives-app',  template: `  <ngswitch-example></ngswitch-example>  <ngif-example></ngif-example>  ` }) **class** DirectivesAppComponent { }   @NgModule({  imports: [BrowserModule],  declarations: [  NgIfExampleComponent,  NgSwitchExampleComponent,  DirectivesAppComponent],  bootstrap: [DirectivesAppComponent] }) **class** AppModule {  }  platformBrowserDynamic().bootstrapModule(AppModule); |

**NgStyle & NgClass**

|  |
| --- |
| **import** {NgModule, Component} **from** '@angular/core'; **import** {BrowserModule} **from** '@angular/platform-browser'; **import** {platformBrowserDynamic} **from** '@angular/platform-browser-dynamic';  @Component({  selector: 'ngstyle-example',  template: `<h4>NgStyle</h4> <ul \*ngFor="let person of people">  <li [ngStyle]="{'font-size.px':24}"  [style.color]="getColor(person.country)">  {{ person.name }} ({{ person.country }})  </li> </ul>  ` }) **class** NgStyleExampleComponent {   getColor(country) {  **switch** (country) {  **case** 'UK':  **return** 'green';  **case** 'USA':  **return** 'blue';  **case** 'HK':  **return** 'red';  }  }   people: **any**[] = [  {  "name": "Douglas Pace",  "country": 'UK'  },  {  "name": "Mcleod Mueller",  "country": 'USA'  },  {  "name": "Day Meyers",  "country": 'HK'  },  {  "name": "Aguirre Ellis",  "country": 'UK'  },  {  "name": "Cook Tyson",  "country": 'USA'  }  ]; }   @Component({  selector: 'ngclass-example',  template: `<h4>NgClass</h4> <ul \*ngFor="let person of people">  <li [ngClass]="{  'text-success':person.country === 'UK',  'text-primary':person.country === 'USA',  'text-danger':person.country === 'HK'  }">  {{ person.name }} ({{ person.country }})  </li> </ul>  <!-- <ul \*ngFor="let person of people">  <li [class.text-success]="person.country === 'UK'"  [class.text-primary]="person.country === 'USA'"  [class.text-danger]="person.country === 'HK'">  {{ person.name }} ({{ person.country }})  </li> </ul> -->  ` }) **class** NgClassExampleComponent {   people: **any**[] = [  {  "name": "Douglas Pace",  "age": 35,  "country": 'UK'  },  {  "name": "Mcleod Mueller",  "age": 32,  "country": 'USA'  },  {  "name": "Day Meyers",  "age": 21,  "country": 'HK'  },  {  "name": "Aguirre Ellis",  "age": 34,  "country": 'UK'  },  {  "name": "Cook Tyson",  "age": 32,  "country": 'USA'  }  ]; }   @Component({  selector: 'directives-app',  template: ` <ngclass-example></ngclass-example> <ngstyle-example></ngstyle-example>` }) **class** DirectivesAppComponent { }   @NgModule({  imports: [BrowserModule],  declarations: [  NgClassExampleComponent,  NgStyleExampleComponent,  DirectivesAppComponent],  bootstrap: [DirectivesAppComponent] }) **class** AppModule {  }  platformBrowserDynamic().bootstrapModule(AppModule); |

**NgNonBindable**

|  |
| --- |
| **import** {NgModule, Component} **from** '@angular/core'; **import** {BrowserModule} **from** '@angular/platform-browser'; **import** {platformBrowserDynamic} **from** '@angular/platform-browser-dynamic';  @Component({  selector: 'ngnonbindable-example',  template: `<h4>NgNonBindable</h4> <div>  To render the name variable we use this syntax  <pre ngNonBindable>{{ name }}</pre> </div>   ` }) **class** NgNonBindableExampleComponent { }  @Component({  selector: 'directives-app',  template: `<ngnonbindable-example></ngnonbindable-example>` }) **class** DirectivesAppComponent { }   @NgModule({  imports: [BrowserModule],  declarations: [NgNonBindableExampleComponent, DirectivesAppComponent],  bootstrap: [DirectivesAppComponent], }) **class** AppModule {  }  platformBrowserDynamic().bootstrapModule(AppModule);  NgNonBindable |

**Structural Directives**

|  |
| --- |
| **import** {platformBrowserDynamic} **from** '@angular/platform-browser-dynamic'; **import** {  Component,  Directive,  NgModule,  Input,  Output,  EventEmitter,  TemplateRef,  ViewContainerRef } **from** '@angular/core'; **import** {BrowserModule} **from** '@angular/platform-browser'; **import** {Directive, Input} **from** '@angular/core';  // // Domain Model //  **class** Joke {  **public** hide: **boolean**;   **constructor**(**public** setup: **string**, **public** punchline: **string**) {  **this**.hide = **true**;  }   toggle() {  **this**.hide = !**this**.hide;  } }  // // Structural Directives //  @Directive({  selector: '[ccIf]' }) **export class** CodeCraftIfDirective {  **constructor**(**private** templateRef: TemplateRef<**any**>,  **private** viewContainer: ViewContainerRef) {  }   @Input() **set** ccIf(condition: **boolean**) {  **if** (condition) {  **this**.viewContainer.createEmbeddedView(**this**.templateRef);  } **else** {  **this**.viewContainer.clear();  }  } }   @Directive({  selector: '[ccFor]' }) **export class** CodeCraftForOfDirective {  **constructor**(**private** templateRef: TemplateRef<**any**>,  **private** viewContainer: ViewContainerRef) {  }   @Input() **set** ccForOf(collection: **any**) {  **if** (condition) {  **this**.viewContainer.createEmbeddedView(**this**.templateRef);  } **else** {  **this**.viewContainer.clear();  }  } }   // // Components // //  @Component({  selector: 'joke',  template: ` <div class="card card-block">  <h4 class="card-title">  {{ data.setup }}  </h4>  <ng-template [ngIf]="!data.hide">  <p class="card-text">  {{ data.punchline }}  </p>   </ng-template>  <button class="btn btn-primary"  (click)="data.toggle()">Tell Me  </button> </div> ` }) **class** JokeComponent **implements** OnInit {  @Input('joke') data: Joke; }  @Component({  selector: 'joke-list',  template: ` <ng-template ngFor  let-j  [ngForOf]="jokes">  <joke [joke]="j"></joke> </ng-template> ` }) **class** JokeListComponent {  jokes: Joke[] = [];   **constructor**() {  **this**.jokes = [  **new** Joke("What did the cheese say when it looked in the mirror?", "Hello-me (Halloumi)"),  **new** Joke("What kind of cheese do you use to disguise a small horse?", "Mask-a-pony (Mascarpone)"),  **new** Joke("A kid threw a lump of cheddar at me", "I thought ‘That’s not very mature’"),  ];  } }   @Component({  selector: 'app',  template: ` <joke-list></joke-list> ` }) **class** AppComponent { }  // // Bootstrap //  @NgModule({  imports: [BrowserModule],  declarations: [  AppComponent,  JokeComponent,  JokeListComponent,  CodeCraftIfDirective  ],  bootstrap: [AppComponent] }) **export class** AppModule { }  platformBrowserDynamic().bootstrapModule(AppModule); |

**Custom Directives**

**Creating a custom directive**

|  |
| --- |
| **import** {platformBrowserDynamic} **from** '@angular/platform-browser-dynamic'; **import** {  Component,  Directive,  Renderer,  ElementRef,  NgModule,  Input,  Output,  EventEmitter } **from** '@angular/core'; **import** {BrowserModule} **from** '@angular/platform-browser';  **class** Joke {  **public** setup: **string**;  **public** punchline: **string**;  **public** hide: **boolean**;   **constructor**(setup: **string**, punchline: **string**) {  **this**.setup = setup;  **this**.punchline = punchline;  **this**.hide = **true**;  }   toggle() {  **this**.hide = !**this**.hide;  } }   @Directive({  selector: "[ccCardHover]" }) **class** CardHoverDirective {  **constructor**(**private** el: ElementRef,  **private** renderer: Renderer) {  //noinspection TypeScriptUnresolvedVariable,TypeScriptUnresolvedFunction  renderer.setElementStyle(el.nativeElement, 'backgroundColor', 'gray');  } }  @Component({  selector: 'joke',  template: ` <div class="card card-block" ccCardHover>  <h4 class="card-title">{{data.setup}}</h4>  <p class="card-text"  [hidden]="data.hide">{{data.punchline}}</p>  <button (click)="data.toggle()"  class="btn btn-primary">Tell Me  </button> </div>  ` }) **class** JokeComponent {  @Input('joke') data: Joke; }  @Component({  selector: 'joke-list',  template: ` <joke \*ngFor="let j of jokes" [joke]="j"></joke>  ` }) **class** JokeListComponent {  jokes: Joke[];   **constructor**() {  **this**.jokes = [  **new** Joke("What did the cheese say when it looked in the mirror?", "Hello-me (Halloumi)"),  **new** Joke("What kind of cheese do you use to disguise a small horse?", "Mask-a-pony (Mascarpone)"),  **new** Joke("A kid threw a lump of cheddar at me", "I thought ‘That’s not very mature’"),  ];  } }   @Component({  selector: 'app',  template: ` <joke-list></joke-list>  ` }) **class** AppComponent { }  @NgModule({  imports: [BrowserModule],  declarations: [  AppComponent,  JokeComponent,  JokeListComponent,  CardHoverDirective  ],  bootstrap: [AppComponent] }) **export class** AppModule { }  platformBrowserDynamic().bootstrapModule(AppModule);  Directive Gray |

**HostListener & HostBinding**

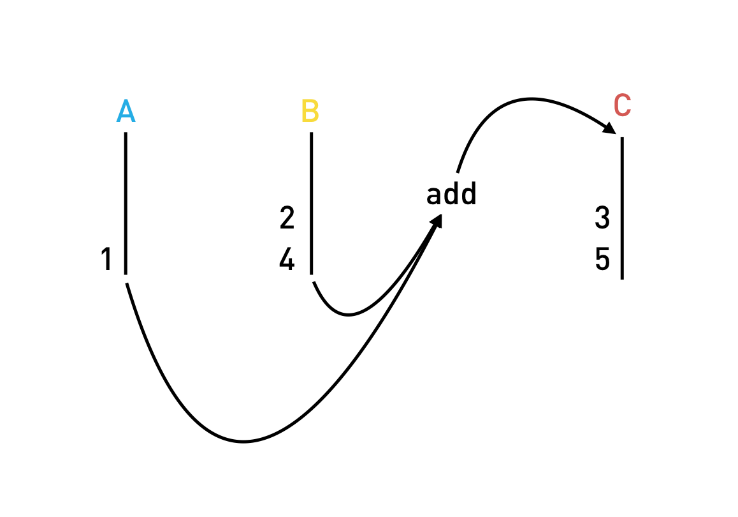
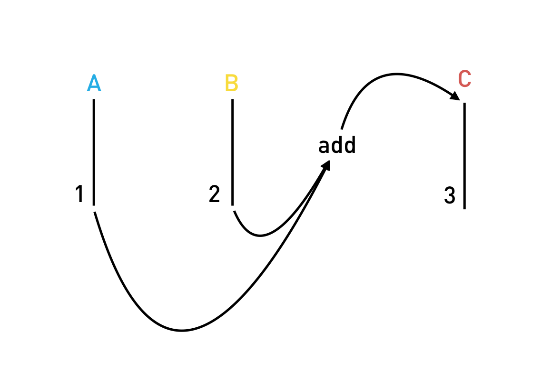
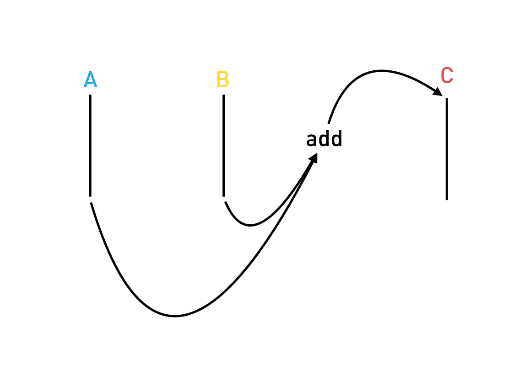
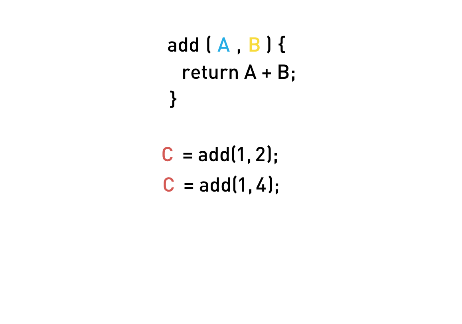
|  |
| --- |
| **import** {platformBrowserDynamic} **from** '@angular/platform-browser-dynamic'; **import** {  Component,  Directive,  Renderer,  HostListener,  HostBinding,  ElementRef,  NgModule,  Input,  Output,  EventEmitter } **from** '@angular/core'; **import** {BrowserModule} **from** '@angular/platform-browser';  **class** Joke {  **public** setup: **string**;  **public** punchline: **string**;  **public** hide: **boolean**;   **constructor**(setup: **string**, punchline: **string**) {  **this**.setup = setup;  **this**.punchline = punchline;  **this**.hide = **true**;  }   toggle() {  **this**.hide = !**this**.hide;  } }   @Directive({  selector: "[ccCardHover]" }) **class** CardHoverDirective {  @HostBinding('class.card-outline-primary')**private** ishovering: **boolean**;   **constructor**(**private** el: ElementRef,  **private** renderer: Renderer) {  // renderer.setElementStyle(el.nativeElement, 'backgroundColor', 'gray');  }   @HostListener('mouseover') onMouseOver() {  **let** part = **this**.el.nativeElement.querySelector('.card-text');  **this**.renderer.setElementStyle(part, 'display', 'block');  **this**.ishovering = **true**;  }   @HostListener('mouseout') onMouseOut() {  **let** part = **this**.el.nativeElement.querySelector('.card-text');  **this**.renderer.setElementStyle(part, 'display', 'none');  **this**.ishovering = **false**;  } }  @Component({  selector: 'joke',  template: ` <div class="card card-block" ccCardHover>  <h4 class="card-title">{{data.setup}}</h4>  <p class="card-text"  [style.display]="'none'">{{data.punchline}}</p> </div>  ` }) **class** JokeComponent {  @Input('joke') data: Joke; }  @Component({  selector: 'joke-list',  template: ` <joke \*ngFor="let j of jokes" [joke]="j"></joke>  ` }) **class** JokeListComponent {  jokes: Joke[];   **constructor**() {  **this**.jokes = [  **new** Joke("What did the cheese say when it looked in the mirror?", "Hello-me (Halloumi)"),  **new** Joke("What kind of cheese do you use to disguise a small horse?", "Mask-a-pony (Mascarpone)"),  **new** Joke("A kid threw a lump of cheddar at me", "I thought ‘That’s not very mature’"),  ];  } }   @Component({  selector: 'app',  template: ` <joke-list></joke-list>  ` }) **class** AppComponent { }  @NgModule({  imports: [BrowserModule],  declarations: [  AppComponent,  JokeComponent,  JokeListComponent,  CardHoverDirective  ],  bootstrap: [AppComponent] }) **export class** AppModule { }  platformBrowserDynamic().bootstrapModule(AppModule);  Directive HostBinding |

**Inputs & Configuration**

|  |
| --- |
| **import** {platformBrowserDynamic} **from** '@angular/platform-browser-dynamic'; **import** {  Component,  Directive,  Renderer,  HostListener,  HostBinding,  ElementRef,  NgModule,  Input,  Output,  EventEmitter } **from** '@angular/core'; **import** {BrowserModule} **from** '@angular/platform-browser';  **class** Joke {  **public** setup: **string**;  **public** punchline: **string**;  **public** hide: **boolean**;   **constructor**(setup: **string**, punchline: **string**) {  **this**.setup = setup;  **this**.punchline = punchline;  **this**.hide = **true**;  }   toggle() {  **this**.hide = !**this**.hide;  } }   @Directive({  selector: "[ccCardHover]" }) **class** CardHoverDirective {  @HostBinding('class.card-outline-primary') **private** ishovering: **boolean**;   @Input('ccCardHover') config: Object = {  querySelector: '.card-text'  };   **constructor**(**private** el: ElementRef,  **private** renderer: Renderer) {  // renderer.setElementStyle(el.nativeElement, 'backgroundColor', 'gray');  }   @HostListener('mouseover') onMouseOver() {  **let** part = **this**.el.nativeElement.querySelector(**this**.config.querySelector);  **this**.renderer.setElementStyle(part, 'display', 'block');  **this**.ishovering = **true**;  }   @HostListener('mouseout') onMouseOut() {  **let** part = **this**.el.nativeElement.querySelector(**this**.config.querySelector);  **this**.renderer.setElementStyle(part, 'display', 'none');  **this**.ishovering = **false**;  } }  @Component({  selector: 'joke',  template: ` <div class="card card-block"  [ccCardHover]="{querySelector:'.card-text'}">  <h4 class="card-title">{{data.setup}}</h4>  <p class="card-text"  [style.display]="'none'">{{data.punchline}}</p> </div>  ` }) **class** JokeComponent {  @Input('joke') data: Joke; }  @Component({  selector: 'joke-list',  template: ` <joke \*ngFor="let j of jokes" [joke]="j"></joke>  ` }) **class** JokeListComponent {  jokes: Joke[];   **constructor**() {  **this**.jokes = [  **new** Joke("What did the cheese say when it looked in the mirror?", "Hello-me (Halloumi)"),  **new** Joke("What kind of cheese do you use to disguise a small horse?", "Mask-a-pony (Mascarpone)"),  **new** Joke("A kid threw a lump of cheddar at me", "I thought ‘That’s not very mature’"),  ];  } }   @Component({  selector: 'app',  template: ` <joke-list></joke-list>  ` }) **class** AppComponent { }  @NgModule({  imports: [BrowserModule],  declarations: [  AppComponent,  JokeComponent,  JokeListComponent,  CardHoverDirective  ],  bootstrap: [AppComponent] }) **export class** AppModule { }  platformBrowserDynamic().bootstrapModule(AppModule); |

**Reactive Programming with RxJS**

**Streams & Reactive Programming**



**Observables & RxJS**

|  |
| --- |
| let obs = Rx.Observable  .interval(1000)  .take(3)  .map((v) => Date.now());  obs.subscribe(value => console.log("Subscriber: " + value)); |
| <!DOCTYPE html> <html>  <head>  <script src="https://cdnjs.cloudflare.com/ajax/libs/rxjs/4.1.0/rx.all.js"></script>  <script src="main.js"></script> </head>  <body> </body>  </html> |

|  |  |
| --- | --- |
| Subscriber: 0  Subscriber: 1  Subscriber: 2  Subscriber: 3  Subscriber: 4  Subscriber: 5  Subscriber: 6  Subscriber: 7  Subscriber: 8  Subscriber: 9  Subscriber: 10 | Subscriber: 0  Subscriber: 1  Subscriber: 2 |
| Subscriber: 1475506794287  Subscriber: 1475506795286  Subscriber: 1475506796285 | Map Marble |
|  |  |

[**http://reactivex.io**](http://reactivex.io)

**RxJS & Angular**

|  |
| --- |
| **import** {NgModule, Component} **from** '@angular/core'; **import** {BrowserModule} **from** '@angular/platform-browser'; **import** {platformBrowserDynamic} **from** '@angular/platform-browser-dynamic'; **import** {ReactiveFormsModule, FormGroup, FormControl, Validators, FormBuilder} **from** "@angular/forms"; **import** 'rxjs/Rx';  @Component({  selector: 'form-app',  template: `<form [formGroup]="form"  (ngSubmit)="onSubmit()">    <!-- Output comment -->  <div class="card card-block">  <pre class="card-text">{{ form.value.comment }}</pre>  </div>  <p class="small">{{ form.value.lastUpdateTS }}</p>   <!-- Comment text area -->  <div class="form-group">  <label for="comment">Comment</label>  <textarea class="form-control"  formControlName="comment"  rows="3"></textarea>  <small class="form-text text-muted">  <span>{{ 100 - form.value.comment.length }}</span> characters left  </small>  </div>   <!-- Name input -->  <div class="form-group">  <label for="name">Name</label>  <input type="text"  class="form-control"  formControlName="name"  placeholder="Enter name">  </div>   <!-- Email input -->  <div class="form-group">  <label for="email">Email address</label>  <input type="email"  class="form-control"  formControlName="email"  placeholder="Enter email">  <small class="form-text text-muted">  We'll never share your email with anyone else.  </small>  </div>   <button type="submit"  class="btn btn-primary"  [disabled]="!form.valid">Submit  </button> </form>   ` }) **class** FormAppComponent {  form: FormGroup;  comment = **new** FormControl("", Validators.required);  name = **new** FormControl("", Validators.required);  email = **new** FormControl("", [  Validators.required,  Validators.pattern("[^ @]\*@[^ @]\*")  ]);   /\* Observable Solution \*/  **constructor**(fb: FormBuilder) {  **this**.form = fb.group({  "comment": **this**.comment,  "name": **this**.name,  "email": **this**.email  });  this.form.valueChanges  .filter(data => this.form.valid)  .map(data => {  data.comment = data.comment.replace(/<(?:.|\n)\*?>/gm, '');  return data  })  .map(data => {  data.lastUpdateTS = new Date();  return data  })  .subscribe( data => console.log(JSON.stringify(data)));  }   /\* None Observable Solution \*/  // constructor(fb: FormBuilder) {  // this.form = fb.group({  // "comment": this.comment,  // "name": this.name,  // "email": this.email  // });  // this.form.valueChanges  // .subscribe( data => {  // if (this.form.valid) {  // data.comment = data.comment.replace(/<(?:.|\n)\*?>/gm, '');  // data.lastUpdateTS = new Date();  // console.log(JSON.stringify(data))  // }  // });  // }   onSubmit() {  console.log("Form submitted!");  } }  @Component({  selector: 'app',  template: ` <form-app></form-app>  ` }) **class** AppComponent { }   @NgModule({  imports: [BrowserModule, ReactiveFormsModule],  declarations: [AppComponent, FormAppComponent],  bootstrap: [AppComponent], }) **class** AppModule {  }  platformBrowserDynamic().bootstrapModule(AppModule);  form example |

**Pipes**

**Built-in Pipes**

|  |
| --- |
| **import** {NgModule, Component} **from** '@angular/core'; **import** {BrowserModule} **from** '@angular/platform-browser'; **import** {platformBrowserDynamic} **from** '@angular/platform-browser-dynamic';  @Component({  selector: 'pipe-builtins',  template: `<div class="card card-block">  <h4 class="card-title">Currency</h4>  <div class="card-text">  <p ngNonBindable>{{ 1234.56 | currency:'CAD' }}</p>  <p>{{ 1234.56 | currency:"CAD" }}</p>    <p ngNonBindable>{{ 1234.56 | currency:'CAD':'code' }}</p>  <p>{{ 1234.56 | currency:'CAD':'code'}}</p>    <p ngNonBindable>{{ 1234.56 | currency:'CAD':'symbol' }}</p>  <p>{{ 1234.56 | currency:'CAD':'symbol'}}</p>    <p ngNonBindable>{{ 1234.56 | currency:'CAD':'symbol-narrow' }}</p>  <p>{{ 1234.56 | currency:'CAD':'symbol-narrow'}}</p>   </div> </div>  <div class="card card-block">  <h4 class="card-title">Date</h4>  <div class="card-text">  <p ngNonBindable>{{ dateVal | date: 'shortTime' }}</p>  <p>{{ dateVal | date: 'shortTime' }}</p>   <p ngNonBindable>{{ dateVal | date:'fullDate' }}</p>  <p>{{ dateVal | date: 'fullDate' }}</p>   <p ngNonBindable>{{ dateVal | date: 'shortTime' }}</p>  <p>{{ dateVal | date: 'shortTime' }}</p>   <p ngNonBindable>{{ dateVal | date: 'd/M/y' }}</p>  <p>{{ dateVal | date: 'd/M/y' }}</p>  </div> </div>  <div class="card card-block">  <div class="card-text">  <h4 class="card-title">DecimalPipe</h4>  <p ngNonBindable>{{ 3.14159265 | number: '3.1-2' }}</p>  <p>{{ 3.14159265 | number: '3.1-2' }}</p>   <p ngNonBindable>{{ 3.14159265 | number: '1.4-4' }}</p>  <p>{{ 3.14159265 | number: '1.4-4' }}</p>  </div> </div>  <div class="card card-block">  <h4 class="card-title">JsonPipe</h4>  <div class="card-text">  <p ngNonBindable>{{ jsonVal }}</p>  <p>{{ jsonVal }}</p>    <p ngNonBindable>{{ jsonVal | json }}</p>  <p>{{ jsonVal | json }}</p>  </div> </div>   <div class="card card-block">  <h4 class="card-title">LowerCasePipe</h4>  <div class="card-text">  <p ngNonBindable>{{ 'ASIM' | lowercase }}</p>  <p>{{ 'ASIM' | lowercase }}</p>  </div> </div>  <div class="card card-block">  <h4 class="card-title">UpperCasePipe</h4>  <div class="card-text">  <p ngNonBindable>{{ 'asim' | uppercase }}</p>  <p>{{ 'asim' | uppercase }}</p>  </div> </div>  <div class="card card-block">  <h4 class="card-title">PercentPipe</h4>  <div class="card-text">  <p ngNonBindable>{{ 0.123456 | percent }}</p>  <p>{{ 0.123456 | percent }}</p>   <p ngNonBindable>{{ 0.123456 | percent: '2.1-2' }}</p>  <p>{{ 0.123456 | percent: '2.1-2' }}</p>   <p ngNonBindable>{{ 42 | percent: '10.4-4' }}</p>  <p>{{ 0.123456 | percent : "10.4-4" }}</p>  </div> </div>  <div class="card card-block">  <h4 class="card-title">SlicePipe</h4>  <div class="card-text">  <p ngNonBindable>{{ [1,2,3,4,5,6] | slice:1:3 }}</p>  <p>{{ [1,2,3,4,5,6] | slice:1:3 }}</p>   <p ngNonBindable>{{ [1,2,3,4,5,6] | slice:2 }}</p>  <p>{{ [1,2,3,4,5,6] | slice:2 }}</p>   <p ngNonBindable>{{ [1,2,3,4,5,6] | slice:2:-1 }}</p>  <p>{{ [1,2,3,4,5,6] | slice:2:-1 }}</p>   <pre ngNonBindable>  &lt;ul&gt;  &lt;li \*ngFor=&quot;let v of [1,2,3,4,5,6] | slice:2:-1&quot;&gt;  {{v}}  &lt;/li&gt; &lt;/ul&gt;  </pre>    <ul>  <li \*ngFor="let v of [1,2,3,4,5,6] | slice:2:-1">  {{v}}  </li>  </ul>  </div> </div>   ` }) **class** PipeBuiltinsComponent {  **private** dateVal: Date = **new** Date();  **private** jsonVal: Object = {moo: 'foo', goo: {too: 'new'}};  }  @Component({  selector: 'app',  template: ` <pipe-builtins></pipe-builtins>  ` }) **class** AppComponent { }   @NgModule({  imports: [BrowserModule],  declarations: [AppComponent,  PipeBuiltinsComponent  ],  bootstrap: [AppComponent], }) **class** AppModule {  }  platformBrowserDynamic().bootstrapModule(AppModule);  decimal pipe  date pipe  currency pipe  json pipe  lowercase pipe  uppercase pipe  percent pipe  slice pipe  <https://angular.io/docs/ts/latest/api/common/index/DatePipe-pipe.html> |

**Async Pipe**

|  |
| --- |
| **import** {NgModule, Component, OnDestroy} **from** '@angular/core'; **import** {BrowserModule} **from** '@angular/platform-browser'; **import** {platformBrowserDynamic} **from** '@angular/platform-browser-dynamic'; **import** { Observable } **from** 'rxjs/Rx'; @Component({  selector: 'async-pipe',  template: `  <div class="card card-block">  <h4 class="card-title">AsyncPipe</h4>    <p class="card-text" ngNonBindable>{{ promise | async }} </p>  <p class="card-text">{{ promise | async }} </p>   <p class="card-text" ngNonBindable>{{ observable | async }} </p>  <p class="card-text">{{ observable | async }}</p>    <p class="card-text" ngNonBindable>{{ observableData }} </p>  <p class="card-text">{{ observableData }}</p>   </div> ` }) **class** AsyncPipeComponent **implements** OnDestroy {  promise: Promise<**string**>;  observable: Observable<**number**>;  subscription: Object = **null**;  observableData: **number**;  **constructor**() {  **this**.promise = **this**.getPromise();  **this**.observable = **this**.getObservable();  **this**.subscribeObservable();  }  getObservable() {  **return** Observable  .interval(1000)  .take(10)  .map((v) => v \* v);  }  subscribeObservable() {// AsyncPipe subscribes to the observable automatically  **this**.subscription = **this**.getObservable()  .subscribe((v) => **this**.observableData = v);  }  getPromise() {  **return new** Promise((resolve, reject) => {  setTimeout(() => resolve("Promise complete!"), 3000);  });  }  ngOnDestroy() {// AsyncPipe unsubscribes from the observable automatically  **if** (**this**.subscription) {  **this**.subscription.unsubscribe();  }  } } @Component({  selector: 'app',  template: ` <async-pipe></async-pipe> ` }) **class** AppComponent {  imageUrl: **string** = ""; } @NgModule({  imports: [BrowserModule],  declarations: [AppComponent,  AsyncPipeComponent  ],  bootstrap: [AppComponent], }) **class** AppModule { } platformBrowserDynamic().bootstrapModule(AppModule); |

**Custom Pipes**

|  |
| --- |
| **import** {NgModule, Component, Pipe} from '@angular/core'; **import** {BrowserModule} from '@angular/platform-browser'; **import** {platformBrowserDynamic} from '@angular/platform-browser-dynamic'; **import** {Observable} from 'rxjs/Rx';   @Pipe({  name: "default" }) **class** DefaultPipe {  transform(value: string, fallback: string, forceHttps: boolean = **false**): string {  **let** image = "";  **if** (value) {  image = value;  } **else** {  image = fallback;  }   **if** (forceHttps) {  **if** (image.indexOf("https") == -1) {  image = image.replace("http", "https");  }  }   **return** image;  } }  @Component({  selector: 'app',  template: `  <img [src]="imageUrl | default:'http://s3.amazonaws.com/uifaces/faces/twitter/sillyleo/128.jpg':true"/>  ` }) **class** AppComponent {  imageUrl: string = ""; }   @NgModule({  imports: [BrowserModule],  declarations: [AppComponent,  DefaultPipe  ],  bootstrap: [AppComponent], }) **class** AppModule {  }  platformBrowserDynamic().bootstrapModule(AppModule);  default pipe part1 |

**Form**

**Model Driven Forms**

|  |
| --- |
| **import** {  NgModule,  Component,  Pipe,  OnInit } **from** '@angular/core'; **import** {  ReactiveFormsModule,  FormsModule,  FormGroup,  FormControl,  Validators,  FormBuilder } **from** '@angular/forms'; **import** {BrowserModule} **from** '@angular/platform-browser'; **import** {platformBrowserDynamic} **from** '@angular/platform-browser-dynamic';   @Component({  selector: 'model-form',  template: `<!--suppress ALL --> <form novalidate  [formGroup]="myform">   <fieldset formGroupName="name">  <div class="form-group">  <label>First Name</label>  <input type="text"  class="form-control"  formControlName="firstName">  </div>   <div class="form-group">  <label>Last Name</label>  <input type="text"  class="form-control"  formControlName="lastName">  </div>  </fieldset>    <div class="form-group">  <label>Email</label>  <input type="email"  class="form-control"  formControlName="email">  </div>   <div class="form-group">  <label>Password</label>  <input type="password"  class="form-control"  formControlName="password">  </div>   <div class="form-group">  <label>Language</label>  <select class="form-control"  formControlName="language">  <option value="">Please select a language</option>  <option \*ngFor="let lang of langs"  [value]="lang">{{lang}}  </option>  </select>  </div>   <pre>{{myform.value | json}}</pre> </form>  ` }) **class** ModelFormComponent **implements** OnInit {  langs: **string**[] = [  'English',  'French',  'German',  ];  myform: FormGroup;    ngOnInit() {  **this**.myform = **new** FormGroup({  name: **new** FormGroup({  firstName: **new** FormControl('', Validators.required),  lastName: **new** FormControl('', Validators.required),  }),  email: **new** FormControl('', [  Validators.required,  Validators.pattern("[^ @]\*@[^ @]\*")  ]),  password: **new** FormControl('', [  Validators.required,  Validators.minLength(8)  ]),  language: **new** FormControl()  });  } }   @Component({  selector: 'app',  template: `<model-form></model-form>` }) **class** AppComponent { }   @NgModule({  imports: [  BrowserModule,  FormsModule,  ReactiveFormsModule],  declarations: [  AppComponent,  ModelFormComponent  ],  bootstrap: [  AppComponent  ], }) **class** AppModule { }  platformBrowserDynamic().bootstrapModule(AppModule);  form setup |

**Model Driven Form Validation**

|  |
| --- |
| **import** {  NgModule,  Component,  Pipe,  OnInit } **from** '@angular/core'; **import** {  ReactiveFormsModule,  FormsModule,  FormGroup,  FormControl,  Validators,  FormBuilder } **from** '@angular/forms'; **import** {BrowserModule} **from** '@angular/platform-browser'; **import** {platformBrowserDynamic} **from** '@angular/platform-browser-dynamic';   @Component({  selector: 'model-form',  template: `<form novalidate  [formGroup]="myform">   <fieldset formGroupName="name">  <div class="form-group"  [ngClass]="{  'has-danger': firstName.invalid && (firstName.dirty || firstName.touched),  'has-success': firstName.valid && (firstName.dirty || firstName.touched)  }">  <label>First Name</label>  <input type="text"  class="form-control"  formControlName="firstName"  required>  <div class="form-control-feedback"  \*ngIf="firstName.errors && (firstName.dirty || firstName.touched)">  <p \*ngIf="firstName.errors.required">First Name is required</p>  </div>   <!--  <pre>Valid? {{ myform.controls.name.controls.firstName.valid }}</pre>  <pre>Dirty? {{ myform.controls.name.controls.firstName.dirty }}</pre>  -->  </div>   <div class="form-group"  [ngClass]="{  'has-danger': lastName.invalid && (lastName.dirty || lastName.touched),  'has-success': lastName.valid && (lastName.dirty || lastName.touched)  }">  <label>Last Name</label>  <input type="text"  class="form-control"  formControlName="lastName"  required>  <div class="form-control-feedback"  \*ngIf="lastName.errors && (lastName.dirty || lastName.touched)">  <p \*ngIf="lastName.errors.required">Last Name is required</p>  </div>  </div>  </fieldset>    <div class="form-group"  [ngClass]="{  'has-danger': email.invalid && (email.dirty || email.touched),  'has-success': email.valid && (email.dirty || email.touched)  }">  <label>Email</label>  <input type="email"  class="form-control"  formControlName="email"  required>  <div class="form-control-feedback"  \*ngIf="email.errors && (email.dirty || email.touched)">  <p \*ngIf="email.errors.required">Email is required</p>  <p \*ngIf="password.errors.pattern">The email address must contain at least the @ character</p>  </div>   <!--  <pre>Valid? {{ myform.controls.email.valid }}</pre>  <pre>Dirty? {{ myform.controls.email.dirty }}</pre>  -->   </div>   <div class="form-group"  [ngClass]="{  'has-danger': password.invalid && (password.dirty || password.touched),  'has-success': password.valid && (password.dirty || password.touched)  }">  <label>Password</label>  <input type="password"  class="form-control"  formControlName="password"  required>  <div class="form-control-feedback"  \*ngIf="password.errors && (password.dirty || password.touched)">  <p \*ngIf="password.errors.required">Password is required</p>  <p \*ngIf="password.errors.minlength">Password must be 8 characters long, we need another {{password.errors.minlength.requiredLength - password.errors.minlength.actualLength}} characters </p>  </div>  </div>   <!--  <pre>{{ password.errors | json }}</pre>  -->   <div class="form-group"  [ngClass]="{  'has-danger': language.invalid && (language.dirty || language.touched),  'has-success': language.valid && (language.dirty || language.touched)  }">  <label>Language</label>  <select class="form-control"  formControlName="language">  <option value="">Please select a language</option>  <option \*ngFor="let lang of langs"  [value]="lang">{{lang}}  </option>  </select>  </div>   <pre>{{myform.value | json}}</pre> </form>` }) **class** ModelFormComponent **implements** OnInit {  langs: **string**[] = [  'English',  'French',  'German',  ];  myform: FormGroup;  firstName: FormControl;  lastName: FormControl;  email: FormControl;  password: FormControl;  language: FormControl;    ngOnInit() {  **this**.createFormControls();  **this**.createForm();  }   createFormControls() {  **this**.firstName = **new** FormControl('', Validators.required);  **this**.lastName = **new** FormControl('', Validators.required);  **this**.email = **new** FormControl('', [  Validators.required,  Validators.pattern("[^ @]\*@[^ @]\*")  ]);  **this**.password = **new** FormControl('', [  Validators.required,  Validators.minLength(8)  ]);  **this**.language = **new** FormControl('');  }   createForm() {  **this**.myform = **new** FormGroup({  name: **new** FormGroup({  firstName: **this**.firstName,  lastName: **this**.lastName,  }),  email: **this**.email,  password: **this**.password,  language: **this**.language  });  } }   @Component({  selector: 'app',  template: `<model-form></model-form>` }) **class** AppComponent { }   @NgModule({  imports: [  BrowserModule,  FormsModule,  ReactiveFormsModule],  declarations: [  AppComponent,  ModelFormComponent  ],  bootstrap: [  AppComponent  ], }) **class** AppModule { }  platformBrowserDynamic().bootstrapModule(AppModule); |

**Submitting & Resetting**

|  |
| --- |
| **import** {  NgModule,  Component,  Pipe,  OnInit } **from** '@angular/core'; **import** {  ReactiveFormsModule,  FormsModule,  FormGroup,  FormControl,  Validators,  FormBuilder } **from** '@angular/forms'; **import** {BrowserModule} **from** '@angular/platform-browser'; **import** {platformBrowserDynamic} **from** '@angular/platform-browser-dynamic';   @Component({  selector: 'model-form',  template: `<form novalidate  [formGroup]="myform"  (ngSubmit)="onSubmit()">   <fieldset formGroupName="name">  <div class="form-group"  [ngClass]="{  'has-danger': firstName.invalid && (firstName.dirty || firstName.touched),  'has-success': firstName.valid && (firstName.dirty || firstName.touched)  }">  <label>First Name</label>  <input type="text"  class="form-control"  formControlName="firstName"  required>  <div class="form-control-feedback"  \*ngIf="firstName.errors && (firstName.dirty || firstName.touched)">  <p \*ngIf="firstName.errors.required">Last Name is required</p>  </div>   <!--  <pre>Valid? {{ myform.controls.name.controls.firstName.valid }}</pre>  <pre>Dirty? {{ myform.controls.name.controls.firstName.dirty }}</pre>  -->  </div>   <div class="form-group"  [ngClass]="{  'has-danger': lastName.invalid && (lastName.dirty || lastName.touched),  'has-success': lastName.valid && (lastName.dirty || lastName.touched)  }">  <label>Last Name</label>  <input type="text"  class="form-control"  formControlName="lastName"  required>  <div class="form-control-feedback"  \*ngIf="lastName.errors && (lastName.dirty || lastName.touched)">  <p \*ngIf="lastName.errors.required">Last Name is required</p>  </div>  </div>  </fieldset>    <div class="form-group"  [ngClass]="{  'has-danger': email.invalid && (email.dirty || email.touched),  'has-success': email.valid && (email.dirty || email.touched)  }">  <label>Email</label>  <input type="email"  class="form-control"  formControlName="email"  required>  <div class="form-control-feedback"  \*ngIf="email.errors && (email.dirty || email.touched)">  <p \*ngIf="email.errors.required">Email is required</p>  <p \*ngIf="password.errors.pattern">The email address must contain at least the @ character</p>  </div>   <!--  <pre>Valid? {{ myform.controls.email.valid }}</pre>  <pre>Dirty? {{ myform.controls.email.dirty }}</pre>  -->   </div>   <div class="form-group"  [ngClass]="{  'has-danger': password.invalid && (password.dirty || password.touched),  'has-success': password.valid && (password.dirty || password.touched)  }">  <label>Password</label>  <input type="password"  class="form-control"  formControlName="password"  required>  <div class="form-control-feedback"  \*ngIf="password.errors && (password.dirty || password.touched)">  <p \*ngIf="password.errors.required">Password is required</p>  <p \*ngIf="password.errors.minlength">Password must be 8 characters long, we need another {{password.errors.minlength.requiredLength - password.errors.minlength.actualLength}} characters </p>  </div>  </div>   <!--   <pre>{{ language.errors | json }}</pre>  -->   <div class="form-group"  [ngClass]="{  'has-danger': language.invalid && (language.dirty || language.touched),  'has-success': language.valid && (language.dirty || language.touched)  }">  <label>Language</label>  <select class="form-control"  formControlName="language">  <option value="">Please select a language</option>  <option \*ngFor="let lang of langs"  [value]="lang">{{lang}}  </option>  </select>  </div>   <button type="submit"  class="btn btn-primary">Submit  </button>   <pre>{{myform.value | json}}</pre> </form>` }) **class** ModelFormComponent **implements** OnInit {  langs: **string**[] = [  'English',  'French',  'German',  ];  myform: FormGroup;  firstName: FormControl;  lastName: FormControl;  email: FormControl;  password: FormControl;  language: FormControl;    ngOnInit() {  **this**.createFormControls();  **this**.createForm();  }   createFormControls() {  **this**.firstName = **new** FormControl('', Validators.required);  **this**.lastName = **new** FormControl('', Validators.required);  **this**.email = **new** FormControl('', [  Validators.required,  Validators.pattern("[^ @]\*@[^ @]\*")  ]);  **this**.password = **new** FormControl('', [  Validators.required,  Validators.minLength(8)  ]);  **this**.language = **new** FormControl('');  }   createForm() {  **this**.myform = **new** FormGroup({  name: **new** FormGroup({  firstName: **this**.firstName,  lastName: **this**.lastName,  }),  email: **this**.email,  password: **this**.password,  language: **this**.language  });  }   onSubmit() {  **if** (**this**.myform.valid) {  console.log("Form Submitted!");  **this**.myform.reset();  }  } }  @Component({  selector: 'app',  template: `<model-form></model-form>` }) **class** AppComponent { } @NgModule({  imports: [  BrowserModule,  FormsModule,  ReactiveFormsModule],  declarations: [  AppComponent,  ModelFormComponent  ],  bootstrap: [  AppComponent  ], }) **class** AppModule { }  platformBrowserDynamic().bootstrapModule(AppModule); |

**Reactive Model Form**

|  |
| --- |
| **import** {  NgModule,  Component,  OnInit } from '@angular/core'; **import** {  ReactiveFormsModule,  FormControl } from '@angular/forms'; **import** {BrowserModule} from '@angular/platform-browser'; **import** {platformBrowserDynamic} from '@angular/platform-browser-dynamic'; **import** 'rxjs/Rx';  @Component({  selector: 'reactive-model-form',  template: `<input type="search"  class="form-control"  placeholder="Please enter search term"  [formControl]="searchField"> <hr/> <ul>  <li \*ngFor="let search of searches">{{ search }}</li> </ul> ` }) **class** ReactiveModelFormComponent **implements** OnInit {   searchField: FormControl;  searches: string[] = [];   ngOnInit() {  **this**.searchField = **new** FormControl();  **this**.searchField.valueChanges  .debounceTime(400)  .distinctUntilChanged()  .subscribe(term => {  **this**.searches.push(term);  });  } }  @Component({  selector: 'app',  template: `<reactive-model-form></reactive-model-form>` }) **class** AppComponent { }   @NgModule({  imports: [  BrowserModule,  ReactiveFormsModule  ],  declarations: [  AppComponent,  ReactiveModelFormComponent  ],  bootstrap: [  AppComponent  ], }) **class** AppModule { }  platformBrowserDynamic().bootstrapModule(AppModule); |

**Template Driven Forms**

|  |
| --- |
| **import** {  NgModule,  Component,  OnInit,  ViewChild } **from** '@angular/core'; **import** {  FormsModule,  FormGroup,  FormControl } **from** '@angular/forms'; **import** {BrowserModule} **from** '@angular/platform-browser'; **import** {platformBrowserDynamic} **from** '@angular/platform-browser-dynamic';  **class** Signup {  **constructor**(**public** firstName: **string** = '',  **public** lastName: **string** = '',  **public** email: **string** = '',  **public** password: **string** = '',  **public** language: **string** = '') {  } }   @Component({  selector: 'template-form',  template: `<!--suppress ALL --> <form novalidate  (ngSubmit)="onSubmit()"  #f="ngForm">   <fieldset ngModelGroup="name">  <div class="form-group"  [ngClass]="{  'has-danger': firstName.invalid && (firstName.dirty || firstName.touched),  'has-success': firstName.valid && (firstName.dirty || firstName.touched)  }">  <label>First Name</label>  <input type="text"  class="form-control"  name="firstName"  [(ngModel)]="model.firstName"  required  #firstName="ngModel">  <div class="form-control-feedback"  \*ngIf="firstName.errors && (firstName.dirty || firstName.touched)">  <p \*ngIf="firstName.errors.required">First name is required</p>  </div>  </div>   <div class="form-group"  [ngClass]="{  'has-danger': lastName.invalid && (lastName.dirty || lastName.touched),  'has-success': lastName.valid && (lastName.dirty || lastName.touched)  }">  <label>Last Name</label>  <input type="text"  class="form-control"  name="lastName"  [(ngModel)]="model.lastName"  required  #lastName="ngModel">  <div class="form-control-feedback"  \*ngIf="lastName.errors && (lastName.dirty || lastName.touched)">  <p \*ngIf="lastName.errors.required">Last name is required</p>  </div>  </div>  </fieldset>    <div class="form-group"  [ngClass]="{  'has-danger': email.invalid && (email.dirty || email.touched),  'has-success': email.valid && (email.dirty || email.touched)  }">  <label>Email</label>  <input type="email"  class="form-control"  name="email"  [(ngModel)]="model.email"  required  pattern="[^ @]\*@[^ @]\*"  #email="ngModel">  <div class="form-control-feedback"  \*ngIf="email.errors && (email.dirty || email.touched)">  <p \*ngIf="email.errors.required">Email is required</p>  <p \*ngIf="email.errors.pattern">Email must contain at least the @ character</p>  </div>  </div>    <div class="form-group"  [ngClass]="{  'has-danger': password.invalid && (password.dirty || password.touched),  'has-success': password.valid && (password.dirty || password.touched)  }">  <label>Password</label>  <input type="password"  class="form-control"  name="password"  [(ngModel)]="model.password"  required  minlength="8"  #password="ngModel">  <div class="form-control-feedback"  \*ngIf="password.errors && (password.dirty || password.touched)">  <p \*ngIf="password.errors.required">Password is required</p>  <p \*ngIf="password.errors.minlength">Password must be at least 8 characters long</p>  </div>  </div>   <div class="form-group">  <label>Language</label>  <select class="form-control"  name="language"  [(ngModel)]="model.language">  <option value="">Please select a language</option>  <option \*ngFor="let lang of langs"  [value]="lang">{{lang}}  </option>  </select>  </div>   <button type="submit"  class="btn btn-primary"  [disabled]="f.invalid">Submit  </button>   <pre>{{f.value | json}}</pre> </form>  ` }) **class** TemplateFormComponent {   model: Signup = **new** Signup();  @ViewChild('f') form: **any**;   langs: **string**[] = [  'English',  'French',  'German',  ];   onSubmit() {  **if** (**this**.form.valid) {  console.log("Form Submitted!");  **this**.form.reset();  }  } }  @Component({  selector: 'app',  template: `<template-form></template-form>` }) **class** AppComponent { }   @NgModule({  imports: [  BrowserModule,  FormsModule  ],  declarations: [  AppComponent,  TemplateFormComponent  ],  bootstrap: [  AppComponent  ], }) **class** AppModule { }  platformBrowserDynamic().bootstrapModule(AppModule);  **이렇게 쓸수 있는것을** **할수있다** |

**Dependency Injection & Providers**

**Injectors**

|  |
| --- |
| **import** {ReflectiveInjector} **from** '@angular/core'; **import** {OpaqueToken} **from** '@angular/core'; // Simple Injector Example {  console.log("Simple Injector Example");  **class** MandrillService {  }  **class** SendGridService {  }   **let** injector = ReflectiveInjector.resolveAndCreate([  MandrillService,  SendGridService  ]);  **let** emailService = injector.get(MandrillService);  console.log(emailService);   // Injector Caching Example  {  console.log("Injector Caching Example");  **let** emailService1 = injector.get(MandrillService);  **let** emailService2 = injector.get(MandrillService);  console.log(emailService1 === emailService2); // true  }   // Injector Caching Caching State Sharing Example  {  console.log("Injector Caching Caching State Sharing Example");  **let** emailService1 = injector.get(MandrillService);  emailService1.foo = "moo";   **let** emailService2 = injector.get(MandrillService);  console.log(emailService2.foo); // moo  }  }  // Child Injector Forwards Request to Parent 1 {  console.log("Child Injector Forwards Request to Parent");  **class** EmailService {  }   **let** injector = ReflectiveInjector.resolveAndCreate([EmailService]);  **let** childInjector = injector.resolveAndCreateChild([EmailService]);   console.log(injector.get(EmailService) === childInjector.get(EmailService)); // false }  // Child Injector Forwards Request to Parent 2 {  console.log("Child Injector Forwards Request to Parent");  **class** EmailService {  }   **let** injector = ReflectiveInjector.resolveAndCreate([EmailService]);  **let** childInjector = injector.resolveAndCreateChild([]);   console.log(injector.get(EmailService) === childInjector.get(EmailService)); // true }  // Child Injector Returns Different Instance {  console.log("Child Injector Returns Different Instance");  **class** EmailService {  }  **class** PhoneService {  }   **let** injector = ReflectiveInjector.resolveAndCreate([EmailService]);  **let** childInjector = injector.resolveAndCreateChild([EmailService]);   console.log(injector.get(EmailService) === childInjector.get(EmailService)); } |

**Providers**

|  |
| --- |
| **import** {ReflectiveInjector} **from** '@angular/core'; **import** {OpaqueToken} **from** '@angular/core';   // Switching Dependencies Example {  console.log("Switching Dependencies Example");  **class** MandrillService {  }  **class** SendGridService {  }   **let** injector = ReflectiveInjector.resolveAndCreate([  {provide: "EmailService", useClass: MandrillService}  ]);   **let** emailService = injector.get("EmailService");  console.log(emailService); // new MandrillService()   {  **let** injector = ReflectiveInjector.resolveAndCreate([  {provide: "EmailService", useClass: SendGridService}  ]);   **let** emailService = injector.get("EmailService");  console.log(emailService); // new SendGridService()  } }  // useClass Provider {  console.log("useClass");  **class** EmailService {  }  **class** MandrillService **extends** EmailService {  }  **class** SendGridService **extends** EmailService {  }   **let** injector = ReflectiveInjector.resolveAndCreate([  {provide: EmailService, useClass: SendGridService}  ]);   **let** emailService = injector.get(EmailService);  console.log(emailService); }  // useExisting {  console.log("useExisting");  **class** MandrillService {  }  **class** SendGridService {  }  **class** GenericEmailService {  }   **let** injector = ReflectiveInjector.resolveAndCreate([  {provide: GenericEmailService, useClass: GenericEmailService},  {provide: MandrillService, useExisting: GenericEmailService},  {provide: SendGridService, useExisting: GenericEmailService}  ]);   **let** emailService1 = injector.get(SendGridService);  console.log(emailService1); // GenericEmailService {}  **let** emailService2 = injector.get(MandrillService);  console.log(emailService2); // GenericEmailService {}  **let** emailService3 = injector.get(GenericEmailService);  console.log(emailService3); // GenericEmailService {}  console.log(emailService1 === emailService2 && emailService2 === emailService3); // true }  // useValue {  console.log("useValue");  **let** injector = ReflectiveInjector.resolveAndCreate([  {  provide: "Config",  useValue: Object.freeze({  'APIKey': 'XYZ1234ABC',  'APISecret': '555-123-111'  })  }  ]);   **let** config = injector.get("Config");  console.log(config); // Object {APIKey: "XYZ1234ABC", APISecret: "555-123-111"} }  // useFactory {  console.log("useFactory");  **class** MandrillService {  }  **class** SendGridService {  }   **const** isProd = **true**;   **let** injector = ReflectiveInjector.resolveAndCreate([  {  provide: "EmailService",  useFactory: () => {  **if** (isProd) {  **return new** MandrillService();  } **else** {  **return new** SendGridService();  }  }  },  ]);   **let** emailService1 = injector.get("EmailService");  console.log(emailService1); // MandrillService {} } |

**Tokens**

|  |
| --- |
| **import** {ReflectiveInjector} from '@angular/core'; **import** {InjectionToken} from '@angular/core';  // String Token (Fail Case) Example {  console.log("String Token (Fail Case) Example");  **class** MandrillService {  }  **class** SendGridService {  }   **let** MandrillServiceToken = "EmailService";  **let** SendGridServiceToken = "EmailService";   **let** injector = ReflectiveInjector.resolveAndCreate([  {provide: SendGridServiceToken, useClass: SendGridService},  {provide: MandrillServiceToken, useClass: MandrillService},  ]);   **let** emailService1 = injector.get(SendGridServiceToken);  **let** emailService2 = injector.get(MandrillServiceToken);  console.log(emailService1 === emailService2); }  // OpaqueToken {  console.log("InjectionToken");  **class** MandrillService {  }  **class** SendGridService {  }   **const** MandrillServiceToken = **new** InjectionToken<string>("EmailService");  **const** SendGridServiceToken = **new** InjectionToken<string>("EmailService");   **let** injector = ReflectiveInjector.resolveAndCreate([  {provide: SendGridServiceToken, useClass: SendGridService},  {provide: MandrillServiceToken, useClass: MandrillService},  ]);   **let** emailService1 = injector.get(SendGridServiceToken);  **let** emailService2 = injector.get(MandrillServiceToken);  console.log(emailService1 === emailService2); // false } |

**Configuring Dependency Injection in Angular**

**@Injection @Injectable**

|  |
| --- |
| **import** {NgModule, Component, Injectable, Inject, TypeDecorator} **from** '@angular/core'; **import** {BrowserModule} **from** '@angular/platform-browser'; **import** {platformBrowserDynamic} **from** '@angular/platform-browser-dynamic';  **class** OtherService {  **constructor**() {  }; }  // This version doesn't work as Angular doesn't know it should be injecting otherService // class SimpleService { // otherService: OtherService; // constructor(otherService: OtherService) { // this.otherService = otherService; // }; // }  // This version works but we have to decorate every parameter to our constructor with @Inject // class SimpleService { // otherService: OtherService; // // constructor(@Inject(OtherService) otherService: OtherService) { // this.otherService = otherService; // }; // // }  // This works because @Injectable automatically injects every parameter to the constructor as long as that parameter has a type @Injectable() **class** SimpleService {  otherService: OtherService;   **constructor**(otherService: OtherService) {  **this**.otherService = otherService;  }; }  // This DOESN'T work because the otherService parameter doesn't have a type // @Injectable // class SimpleService { // otherService: OtherService; // // constructor(otherService: any) { // this.otherService = otherService; // }; // }  @Component({  selector: 'simple',  template: `<p>Simple is as simple does</p>`, }) **class** SimpleComponent {  **constructor**(**private** simpleService: SimpleService) {  } }   @Component({  selector: 'app',  template: '<simple></simple>' }) **class** AppComponent { }   @NgModule({  imports: [BrowserModule],  declarations: [AppComponent, SimpleComponent],  bootstrap: [AppComponent],  providers: [OtherService, SimpleService] }) **class** AppModule {  }  platformBrowserDynamic().bootstrapModule(AppModule);  injector tree |
| * declarations - 이 모듈에서 사용하는 뷰 클래스를 정의한다. Angular에는 컴포넌트, 디렉티브, 파이프 세 종류의 뷰 클래스가 있다. * exports - 다른 모듈이나 컴포넌트 템플릿에서 접근할 수 있도록 외부로 공개 선언한다. * import - export로 공개된 클래스를 다른 컴포넌트 템플릿의 this 모듈에 선언해서 사용할 때 사용한다. * providers - 전역에서 사용되는 서비스를 해당 객체에서 사용할 수 있도록 생성하고 지정한다. 프로바이더는 앱의 모든 곳에서 접근할 수 있다. * bootstrap - 루트 컴포넌트라고 하는 메인 어플리케이션의 뷰를 선언한다. bootstrap 항목은 루트 모듈에만 존재한다. |

**NgModule.providers vs Component.providers vs Component.viewProviders**

|  |
| --- |
| **import** { NgModule, Component, Injectable } **from** '@angular/core'; **import** { FormsModule } **from** '@angular/forms'; **import** { BrowserModule } **from** '@angular/platform-browser'; **import** { platformBrowserDynamic } **from** '@angular/platform-browser-dynamic';   **class** SimpleService {  value: **string**; }  @Component({  selector: 'child',  template: `  <div class="child">   <p>Child</p>  {{ service.value }} </div>  `,  styles: [`  .child {  background-color: #239CDE;  padding: 10px;  }  `],  // providers: [SimpleService] }) **class** ChildComponent {  **constructor**(**private** service: SimpleService) { } }  @Component({  selector: 'parent',  template: `  <div class="parent">   <p>Parent</p>  <form novalidate>  <div class="form-group">  <input type="text"  class="form-control"  name="value"  [(ngModel)]="service.value">  </div>  </form>  <ng-content></ng-content> </div>  `,  styles: [`  .parent {  background-color: #D1E751;  padding: 10px;  }  `],  viewProviders: [SimpleService ]  // providers: [SimpleService] }) **class** ParentComponent {  **constructor**(**private** service: SimpleService) { } }    @Component({  selector: 'app',  template: `  <div class="row">  <div class="col-xs-6">  <parent><child></child></parent>  </div>  <div class="col-xs-6">  <parent><child></child></parent>  </div> </div>  ` }) **class** AppComponent { }   @NgModule({  imports: [ BrowserModule, FormsModule ],  declarations: [ AppComponent, ParentComponent, ChildComponent ],  bootstrap: [ AppComponent ],  providers: [SimpleService] }) **class** AppModule {  }  platformBrowserDynamic().bootstrapModule(AppModule);  di example  ngmodule providerscomponent providers  component viewproviders |

**HTTP**

**Core HTTP API**

|  |
| --- |
| **import** {NgModule, Component} **from** '@angular/core'; **import** {HttpModule, Http, URLSearchParams, Headers, RequestOptions} **from** '@angular/http'; **import** {BrowserModule} **from** '@angular/platform-browser'; **import** {platformBrowserDynamic} **from** '@angular/platform-browser-dynamic'; // import 'rxjs/add/operator/toPromise'; **import** 'rxjs/Rx';   @Component({  selector: 'app',  template: `  <div class="row">  <div class="m-t-1">  <button class="btn btn-primary" (click)="doGET()">GET</button>  <button class="btn btn-primary" (click)="doPOST()">POST</button>  <button class="btn btn-primary" (click)="doPUT()">PUT</button>  <button class="btn btn-primary" (click)="doDELETE()">DELETE</button>  </div> </div>  <div class="row">  <div class="m-t-1">  <button class="btn btn-secondary" (click)="doGETAsPromise()">As Promise</button>  <button class="btn btn-secondary" (click)="doGETAsPromiseError()">Error as Promise</button>  <button class="btn btn-secondary" (click)="doGETAsObservableError()">Error as Observable</button>  </div> </div>  <div class="row">  <div class="m-t-1">  <button class="btn btn-danger" (click)="doGETWithHeaders()">With Headers</button>  </div> </div>  ` }) **class** AppComponent {  apiRoot: **string** = "http://httpbin.org";   **constructor**(**private** http: Http) {  }   doGET() {  console.log("GET");  **let** url = `${**this**.apiRoot}/get`;  **let** search = **new** URLSearchParams();  search.set('foo', 'moo');  search.set('limit', 25);  **this**.http.get(url, {search}).subscribe(res => console.log(res.json()));  }   doPOST() {  console.log("POST");  **let** url = `${**this**.apiRoot}/post`;  **let** search = **new** URLSearchParams();  search.set('foo', 'moo');  search.set('limit', 25);  **this**.http.post(url, {moo: "foo", goo: "loo"}, {search}).subscribe(res => console.log(res.json()));  }   doPUT() {  console.log("PUT");  **let** url = `${**this**.apiRoot}/put`;  **let** search = **new** URLSearchParams();  search.set('foo', 'moo');  search.set('limit', 25);  **this**.http.put(url, {moo: "foo", goo: "loo"}, {search}).subscribe(res => console.log(res.json()));  }   doDELETE() {  console.log("DELETE");  **let** url = `${**this**.apiRoot}/delete`;  **let** search = **new** URLSearchParams();  search.set('foo', 'moo');  search.set('limit', 25);  **this**.http.delete(url, {search}).subscribe(res => console.log(res.json()));  }   doGETAsPromise() {  console.log("GET AS PROMISE");  **let** url = `${**this**.apiRoot}/get`;  **this**.http.get(url)  .toPromise()  .then(res => console.log(res.json()));  }   doGETAsPromiseError() {  console.log("GET AS PROMISE ERROR");  **let** url = `${**this**.apiRoot}/post`;  **this**.http.get(url)  .toPromise()  .then(  res => console.log(res.json()),  msg => console.error(`Error: ${msg.status} ${msg.statusText}`)  );  }   doGETAsObservableError() {  console.log("GET AS OBSERVABLE ERROR");  **let** url = `${**this**.apiRoot}/post`;  **this**.http.get(url).subscribe(  res => console.log(res.json()),  msg => console.error(`Error: ${msg.status} ${msg.statusText}`)  );  }   doGETWithHeaders() {  console.log("GET WITH HEADERS");  **let** headers: Headers = **new** Headers();  headers.append('Authorization', btoa('username:password'));  **let** opts: RequestOptions = **new** RequestOptions();  opts.headers = headers;  **let** url = `${**this**.apiRoot}/get`;  **this**.http.get(url, opts).subscribe(  res => console.log(res.json()),  msg => console.error(`Error: ${msg.status} ${msg.statusText}`)  );  } }   @NgModule({  imports: [BrowserModule, HttpModule],  declarations: [AppComponent],  bootstrap: [AppComponent] }) **class** AppModule {  }  platformBrowserDynamic().bootstrapModule(AppModule);  application demo |

**HTTP Example with Promises**

|  |
| --- |
| **import** {NgModule, Component, Injectable} **from** '@angular/core'; **import** {BrowserModule} **from** '@angular/platform-browser'; **import** {platformBrowserDynamic} **from** '@angular/platform-browser-dynamic'; **import** {HttpModule, Http, Response} **from** '@angular/http'; **import** 'rxjs/Rx';  **class** SearchItem {  **constructor**(**public** track: **string**,  **public** artist: **string**,  **public** link: **string**,  **public** thumbnail: **string**,  **public** artistId: **string**) {  } }   @Injectable() **export class** SearchService {  apiRoot: **string** = 'https://itunes.apple.com/search';  results: SearchItem[];  loading: **boolean**;   **constructor**(**private** http: Http) {  **this**.results = [];  **this**.loading = **false**;  }   search(term: **string**) {  **let** promise = **new** Promise((resolve, reject) => {  **let** apiURL = `${**this**.apiRoot}?term=${term}&media=music&limit=20`;  **this**.http.get(apiURL)  .toPromise()  .then(  res => { // Success  **this**.results = res.json().results.map(item => {  **return new** SearchItem(  item.trackName,  item.artistName,  item.trackViewUrl,  item.artworkUrl30,  item.artistId  );  });  // this.results = res.json().results;  resolve();  },  msg => { // Error  reject(msg);  }  );  });  **return** promise;  } }   @Component({  selector: 'app',  template: `  <form class="form-inline">  <div class="form-group">  <input type="search"  class="form-control"  placeholder="Enter search string"  #search>  </div>  <button type="button" class="btn btn-primary" (click)="doSearch(search.value)">Search</button> </form>  <hr/>  <div class="text-center">  <p class="lead" \*ngIf="loading">Loading...</p> </div>  <ul class="list-group">  <li class="list-group-item"  \*ngFor="let track of itunes.results">  <img src="{{track.thumbnail}}">  <a target="\_blank"  href="{{track.link}}">{{ track.track }}  </a>  </li> </ul>  ` }) **class** AppComponent {  **private** loading: **boolean** = **false**;   constructor(**private** itunes: SearchService) {  }   doSearch(term: **string**) {  **this**.loading = **true**;  **this**.itunes.search(term).then(\_ => **this**.loading = **false**)  } }  @NgModule({  imports: [  BrowserModule,  HttpModule,  ],  declarations: [AppComponent],  bootstrap: [AppComponent],  providers: [SearchService] }) **class** AppModule { }  platformBrowserDynamic().bootstrapModule(AppModule);  http promise app |

**HTTP Example with Observables**

|  |
| --- |
| **import** {NgModule, Component, Injectable} **from** '@angular/core'; **import** {BrowserModule} **from** '@angular/platform-browser'; **import** {platformBrowserDynamic} **from** '@angular/platform-browser-dynamic'; **import** {HttpModule, Http, Response} **from** '@angular/http'; **import** {ReactiveFormsModule, FormControl, FormsModule} **from** '@angular/forms'; **import** {Observable} **from** 'rxjs'; **import** 'rxjs/add/operator/map'; **import** 'rxjs/add/operator/debounceTime'; **import** 'rxjs/add/operator/distinctUntilChanged'; **import** 'rxjs/add/operator/switchMap'; **import** 'rxjs/add/operator/do';  **class** SearchItem {  **constructor**(**public** track: **string**,  **public** artist: **string**,  **public** link: **string**,  **public** thumbnail: **string**,  **public** artistId: **string**) {  } }  @Injectable() **export class** SearchService {  apiRoot: **string** = 'https://itunes.apple.com/search';   **constructor**(**private** http: Http) {  }   search(term: **string**): Observable<SearchItem[]> {  **let** apiURL = `${**this**.apiRoot}?term=${term}&media=music&limit=20`;  **return this**.http.get(apiURL)  .map(res => {  **return** res.json().results.map(item => {  **return new** SearchItem(  item.trackName,  item.artistName,  item.trackViewUrl,  item.artworkUrl30,  item.artistId  );  });  });  } }  @Component({  selector: 'app',  template: ` <form class="form-inline">  <div class="form-group">  <input type="search"  class="form-control"  placeholder="Enter search string"  [formControl]="searchField">  </div> </form>  <div class="text-center">  <p class="lead" \*ngIf="loading">Loading...</p> </div>  <ul class="list-group">  <li class="list-group-item"  \*ngFor="let track of results | async">  <img src="{{track.thumbnail}}">  <a target="\_blank"  href="{{track.link}}">{{ track.track }}  </a>  </li> </ul>  ` }) **class** AppComponent {  **private** loading: **boolean** = **false**;  **private** results: Observable<SearchItem[]>;  **private** searchField: FormControl;   **constructor**(**private** itunes: SearchService) {  }   ngOnInit() {  **this**.searchField = **new** FormControl();  **this**.results = **this**.searchField.valueChanges  .debounceTime(400)  .distinctUntilChanged()  .do(\_ => **this**.loading = **true**)  .switchMap(term => **this**.itunes.search(term))  .do(\_ => **this**.loading = **false**)  }   doSearch(term: **string**) {  **this**.itunes.search(term)  } }  @NgModule({  imports: [  BrowserModule,  ReactiveFormsModule,  FormsModule,  HttpModule  ],  declarations: [AppComponent],  bootstrap: [AppComponent],  providers: [SearchService] }) **class** AppModule { }  platformBrowserDynamic().bootstrapModule(AppModule);  switch animation |

**JSONP Example with Observables**

|  |
| --- |
| **import** {NgModule, Component, Injectable} **from** '@angular/core'; **import** {BrowserModule} **from** '@angular/platform-browser'; **import** {platformBrowserDynamic} **from** '@angular/platform-browser-dynamic'; **import** {JsonpModule, Jsonp, Response} **from** '@angular/http'; **import** {ReactiveFormsModule, FormControl, FormsModule} **from** '@angular/forms'; **import** {Observable} **from** 'rxjs'; **import** 'rxjs/add/operator/map'; **import** 'rxjs/add/operator/debounceTime'; **import** 'rxjs/add/operator/distinctUntilChanged'; **import** 'rxjs/add/operator/switchMap'; **import** 'rxjs/add/operator/do';  **class** SearchItem {  **constructor**(**public** track: **string**,  **public** artist: **string**,  **public** link: **string**,  **public** thumbnail: **string**,  **public** artistId: **string**) {  } }  @Injectable() **export class** SearchService {  apiRoot: **string** = 'https://itunes.apple.com/search';   **constructor**(**private** jsonp: Jsonp) {  }   search(term: **string**) {  **let** apiURL = `${**this**.apiRoot}?term=${term}&media=music&limit=20&callback=JSONP\_CALLBACK`;  **return this**.jsonp.request(apiURL)  .map(res => {  **return** res.json().results.map(item => {  **return new** SearchItem(  item.trackName,  item.artistName,  item.trackViewUrl,  item.artworkUrl30,  item.artistId  );  });  });  } }  @Component({  selector: 'app',  template: ` <form class="form-inline">  <div class="form-group">  <input type="search"  class="form-control"  placeholder="Enter search string"  [formControl]="searchField">  </div> </form>  <div class="text-center">  <p class="lead" \*ngIf="loading">Loading...</p> </div>  <ul class="list-group">  <li class="list-group-item"  \*ngFor="let track of results | async">  <img src="{{track.thumbnail}}">  <a target="\_blank"  href="{{track.link}}">{{ track.track }}  </a>  </li> </ul>  ` }) **class** AppComponent {  **private** loading: **boolean** = **false**;  **private** results: Observable<SearchItem[]>;  **private** searchField: FormControl;   **constructor**(**private** itunes: SearchService) {  }   ngOnInit() {  **this**.searchField = **new** FormControl();  **this**.results = **this**.searchField.valueChanges  .debounceTime(400)  .distinctUntilChanged()  .do(\_ => **this**.loading = **true**)  .switchMap(term => **this**.itunes.search(term))  .do(\_ => **this**.loading = **false**)  } }  @NgModule({  imports: [  BrowserModule,  ReactiveFormsModule,  FormsModule,  JsonpModule  ],  declarations: [AppComponent],  bootstrap: [AppComponent],  providers: [SearchService] }) **class** AppModule { }  platformBrowserDynamic().bootstrapModule(AppModule); |

**Routing**

**Routing Strategies**

|  |
| --- |
| **import** {NgModule, Component, Injectable} **from** '@angular/core'; **import** {BrowserModule} **from** '@angular/platform-browser'; **import** {platformBrowserDynamic} **from** '@angular/platform-browser-dynamic'; **import** {JsonpModule, Jsonp, Response} **from** '@angular/http'; **import** {ReactiveFormsModule, FormControl, FormsModule} **from** '@angular/forms'; **import** {Routes, RouterModule} **from** "@angular/router"; **import** {Observable} **from** 'rxjs'; **import** 'rxjs/add/operator/toPromise';  **class** SearchItem {  **constructor**(**public** name: **string**,  **public** artist: **string**,  **public** link: **string**,  **public** thumbnail: **string**,  **public** artistId: **string**) {  } }  @Injectable() **class** SearchService {  apiRoot: **string** = 'https://itunes.apple.com/search';  results: SearchItem[];   **constructor**(**private** jsonp: Jsonp) {  **this**.results = [];  }   search(term: **string**) {  **return new** Promise((resolve, reject) => {  **this**.results = [];  **let** apiURL = `${**this**.apiRoot}?term=${term}&media=music&limit=20&callback=JSONP\_CALLBACK`;  **this**.jsonp.request(apiURL)  .toPromise()  .then(  res => { // Success  **this**.results = res.json().results.map(item => {  **return new** SearchItem(  item.trackName,  item.artistName,  item.trackViewUrl,  item.artworkUrl30,  item.artistId  );  });  resolve();  },  msg => { // Error  reject(msg);  }  );  });  } }   @Component({  selector: 'app-search',  template: `<form class="form-inline">  <div class="form-group">  <input type="search"  class="form-control"  placeholder="Enter search string"  #search>  </div>  <button type="button"  class="btn btn-primary"  (click)="doSearch(search.value)">  Search  </button> </form>  <hr />  <div class="text-center">  <p class="lead"  \*ngIf="loading">Loading...</p> </div>  <div class="list-group">  <a href="#"  class="list-group-item list-group-item-action"  \*ngFor="let track of itunes.results">  <img src="{{track.thumbnail}}">  {{ track.name }} <span class="text-muted">by</span> {{ track.artist }}  </a> </div>  ` }) **class** SearchComponent {  **private** loading: **boolean** = **false**;   **constructor**(**private** itunes: SearchService) {  }   doSearch(term: **string**) {  **this**.loading = **true**;  **this**.itunes.search(term).then(\_ => **this**.loading = **false**)  } }  @Component({  selector: 'app-home',  template: ` <div class="jumbotron">  <h1 class="display-3">iTunes Search App</h1> </div>  ` }) **class** HomeComponent { }  @Component({  selector: 'app-header',  template: ` <nav class="navbar navbar-light bg-faded">  <a class="navbar-brand" href="#">iTunes Search App</a>  <ul class="nav navbar-nav">  <li class="nav-item active">  <a class="nav-link" href="#">Home</a>  </li>  <li class="nav-item">  <a class="nav-link" href="#">Search</a>  </li>  </ul> </nav>  ` }) **class** HeaderComponent { }  @Component({  selector: 'app',  template: `  <app-header></app-header>  <div class="m-t-1">  **<router-outlet></router-outlet>**  </div>  ` }) **class** AppComponent { }  **const routes: Routes = [  {path: '', redirectTo: 'home', pathMatch: 'full'},  {path: 'find', redirectTo: 'search'},  {path: 'home', component: HomeComponent},  {path: 'search', component: SearchComponent},  {path: '\*\*', component: HomeComponent} ];**   @NgModule({  imports: [  BrowserModule,  ReactiveFormsModule,  FormsModule,  JsonpModule,  **RouterModule.forRoot(routes, {useHash: true})**  ],  declarations: [  AppComponent,  SearchComponent,  HomeComponent,  HeaderComponent  ],  bootstrap: [AppComponent],  providers: [SearchService] }) **class** AppModule { }  platformBrowserDynamic().bootstrapModule(AppModule);  app search app home |

**Navigation**

|  |
| --- |
| **import** {NgModule, Component, Injectable} from '@angular/core'; **import** {BrowserModule} from '@angular/platform-browser'; **import** {platformBrowserDynamic} from '@angular/platform-browser-dynamic'; **import** {JsonpModule, Jsonp, Response} from '@angular/http'; **import** {ReactiveFormsModule, FormControl, FormsModule} from '@angular/forms'; **import** {Routes, RouterModule, Router} from "@angular/router"; **import** {Observable} from 'rxjs'; **import** 'rxjs/add/operator/toPromise';  **class** SearchItem {  **constructor**(**public** name: string,  **public** artist: string,  **public** link: string,  **public** thumbnail: string,  **public** artistId: string) {  } }  @Injectable() **class** SearchService {  apiRoot: string = 'https://itunes.apple.com/search';  results: SearchItem[];   **constructor**(**private** jsonp: Jsonp) {  **this**.results = [];  }   search(term: string) {  **return new** Promise((resolve, reject) => {  **this**.results = [];  **let** apiURL = `${**this**.apiRoot}?term=${term}&media=music&limit=20&callback=JSONP\_CALLBACK`;  **this**.jsonp.request(apiURL)  .toPromise()  .then(  res => { // Success  **this**.results = res.json().results.map(item => {  **return new** SearchItem(  item.trackName,  item.artistName,  item.trackViewUrl,  item.artworkUrl30,  item.artistId  );  });  resolve();  },  msg => { // Error  reject(msg);  }  );  });  } }   @Component({  selector: 'app-search',  template: `<form class="form-inline">  <div class="form-group">  <input type="search"  class="form-control"  placeholder="Enter search string"  #search>  </div>  <button type="button"  class="btn btn-primary"  (click)="doSearch(search.value)">  Search  </button> </form>  <hr />  <div class="text-center">  <p class="lead"  \*ngIf="loading">Loading...</p> </div>  <div class="list-group">  <a href="#"  class="list-group-item list-group-item-action"  \*ngFor="let track of itunes.results">  <img src="{{track.thumbnail}}">  {{ track.name }} <span class="text-muted">by</span> {{ track.artist }}  </a> </div>  ` }) **class** SearchComponent {  **private** loading: boolean = **false**;   constructor(**private** itunes: SearchService) {  }   doSearch(term: string) {  **this**.loading = **true**;  **this**.itunes.search(term).then(\_ => **this**.loading = **false**)  } }  @Component({  selector: 'app-home',  template: ` <div class="jumbotron">  <h1 class="display-3">iTunes Search App</h1> </div>  ` }) **class** HomeComponent { }  @Component({  selector: 'app-header',  template: `<nav class="navbar navbar-light bg-faded">  <a class="navbar-brand"  [routerLink]="['home']">iTunes Search App  </a>  <ul class="nav navbar-nav">  <li class="nav-item"  **[routerLinkActive]="['active']"**>  <a class="nav-link"  [routerLink]="['home']">Home  </a>  </li>  <li class="nav-item"  **[routerLinkActive]="['active']**">  <a class="nav-link"  [routerLink]="['search']">Search  </a>  </li>  </ul> </nav>  ` }) **class** HeaderComponent {  constructor(**private** router: Router) {  }   goHome() {  **this.router.navigate(['']);**  }   goSearch() {  **this.router.navigate(['search']);**  } }  @Component({  selector: 'app',  template: `  <app-header></app-header>  <div class="m-t-1">  <router-outlet></router-outlet>  </div>  ` }) **class** AppComponent { }  **const** routes: Routes = [  {path: '', redirectTo: 'home', pathMatch: 'full'},  {path: 'find', redirectTo: 'search'},  {path: 'home', component: HomeComponent},  {path: 'search', component: SearchComponent},  {path: '\*\*', component: HomeComponent} ];   @NgModule({  imports: [  BrowserModule,  ReactiveFormsModule,  FormsModule,  JsonpModule,  RouterModule.forRoot(routes, {useHash: **true**})  ],  declarations: [  AppComponent,  SearchComponent,  HomeComponent,  HeaderComponent  ],  bootstrap: [AppComponent],  providers: [SearchService] }) **class** AppModule { }  platformBrowserDynamic().bootstrapModule(AppModule); |

**Parameterised Routes**

|  |
| --- |
| **import** {NgModule, Component, Injectable} **from** '@angular/core'; **import** {BrowserModule} **from** '@angular/platform-browser'; **import** {platformBrowserDynamic} **from** '@angular/platform-browser-dynamic'; **import** {JsonpModule, Jsonp, Response} **from** '@angular/http'; **import** {ReactiveFormsModule, FormControl, FormsModule} **from** '@angular/forms'; **import** {Routes, RouterModule} **from** "@angular/router"; **import** {Observable} **from** 'rxjs'; **import** 'rxjs/add/operator/toPromise'; **import** {Routes, RouterModule, Router, ActivatedRoute} **from** "@angular/router";  **class** SearchItem {  **constructor**(**public** name: **string**,  **public** artist: **string**,  **public** link: **string**,  **public** thumbnail: **string**,  **public** artistId: **string**) {  } }  @Injectable() **class** SearchService {  apiRoot: **string** = 'https://itunes.apple.com/search';  results: SearchItem[];   **constructor**(**private** jsonp: Jsonp) {  **this**.results = [];  }   search(term: **string**) {  **return new** Promise((resolve, reject) => {  **this**.results = [];  **let** apiURL = `${**this**.apiRoot}?term=${term}&media=music&limit=20&callback=JSONP\_CALLBACK`;  **this**.jsonp.request(apiURL)  .toPromise()  .then(  res => { // Success  **this**.results = res.json().results.map(item => {  **return new** SearchItem(  item.trackName,  item.artistName,  item.trackViewUrl,  item.artworkUrl30,  item.artistId  );  });  resolve();  },  msg => { // Error  reject(msg);  }  );  });  } }   @Component({  selector: 'app-search',  template: `<form class="form-inline">  <div class="form-group">  <input type="search"  class="form-control"  placeholder="Enter search string"  #search>  </div>  <button type="button"  class="btn btn-primary"  (click)="onSearch(search.value)">  Search  </button> </form>  <hr />  <div class="text-center">  <p class="lead"  \*ngIf="loading">Loading...</p> </div>  <div class="list-group">  <a href="#"  class="list-group-item list-group-item-action"  \*ngFor="let track of itunes.results">  <img src="{{track.thumbnail}}">  {{ track.name }} <span class="text-muted">by</span> {{ track.artist }}  </a> </div>  ` }) **class** SearchComponent {  **private** loading: **boolean** = **false**;   **constructor(private itunes: SearchService,  private route: ActivatedRoute,  private router: Router) {  this.route.params.subscribe(params => {  console.log(params);  if (params['term']) {  this.doSearch(params['term'])  }  });  }**   doSearch(term: **string**) {  **this**.loading = **true**;  **this**.itunes.search(term).then(\_ => **this**.loading = **false**)  }   **onSearch(term: string) {  this.router.navigate(['search', {term: term}]);  }** }  @Component({  selector: 'app-home',  template: ` <div class="jumbotron">  <h1 class="display-3">iTunes Search App</h1> </div>  ` }) **class** HomeComponent { }  @Component({  selector: 'app-header',  template: `<nav class="navbar navbar-light bg-faded">  <a class="navbar-brand"  [routerLink]="['home']">iTunes Search App  </a>  <ul class="nav navbar-nav">  <li class="nav-item"  [routerLinkActive]="['active']">  <a class="nav-link"  [routerLink]="['home']">Home  </a>  </li>  <li class="nav-item"  [routerLinkActive]="['active']">  <a class="nav-link"  [routerLink]="['search']">Search  </a>  </li>  </ul> </nav>  ` }) **class** HeaderComponent {  constructor(**private** router: Router) {  }   goHome() {  **this**.router.navigate(['']);  }   goSearch() {  **this**.router.navigate(['search']);  } }  @Component({  selector: 'app',  template: `  <app-header></app-header>  <div class="m-t-1">  <router-outlet></router-outlet>  </div>  ` }) **class** AppComponent { }  **const** routes: Routes = [  {path: '', redirectTo: 'home', pathMatch: 'full'},  {path: 'find', redirectTo: 'search'},  {path: 'home', component: HomeComponent}, **{path: 'search', component: SearchComponent}, //{path: 'search/:term', component: SearchComponent},**  {path: '\*\*', component: HomeComponent} ]; @NgModule({  imports: [  BrowserModule,  ReactiveFormsModule,  FormsModule,  JsonpModule,  RouterModule.forRoot(routes, {useHash: **true**})  ],  declarations: [  AppComponent,  SearchComponent,  HomeComponent,  HeaderComponent  ],  bootstrap: [AppComponent],  providers: [SearchService] }) **class** AppModule { }  platformBrowserDynamic().bootstrapModule(AppModule);  **this.router.navigate(['search', term);**  **search/:term 을 하면은 뒤쪽에 localhost:8080/#/search/keyword 처럼되고**  **this.router.navigate(['search', {term: term}]);**  **search로 한다면 뒤쪽에 localhost:8080/#/search;term=keyword 처럼된다.** |

**Nested Routes**

|  |
| --- |
| **import** {NgModule, Component, Injectable} **from** '@angular/core'; **import** {BrowserModule} **from** '@angular/platform-browser'; **import** {platformBrowserDynamic} **from** '@angular/platform-browser-dynamic'; **import** {JsonpModule, Jsonp, Response} **from** '@angular/http'; **import** {ReactiveFormsModule, FormControl, FormsModule} **from** '@angular/forms'; **import** {Routes, RouterModule} **from** "@angular/router"; **import** {Observable} **from** 'rxjs'; **import** 'rxjs/add/operator/toPromise'; **import** {Routes, RouterModule, Router, ActivatedRoute} **from** "@angular/router";  **class** SearchItem {  **constructor**(**public** name: **string**,  **public** artist: **string**,  **public** link: **string**,  **public** thumbnail: **string**,  **public** artistId: **string**) {  } }  @Injectable() **class** SearchService {  apiRoot: **string** = 'https://itunes.apple.com/search';  results: SearchItem[];   **constructor**(**private** jsonp: Jsonp) {  **this**.results = [];  }   search(term: **string**) {  **return new** Promise((resolve, reject) => {  **this**.results = [];  **let** apiURL = `${**this**.apiRoot}?term=${term}&media=music&limit=20&callback=JSONP\_CALLBACK`;  **this**.jsonp.request(apiURL)  .toPromise()  .then(  res => { // Success  **this**.results = res.json().results.map(item => {  **return new** SearchItem(  item.trackName,  item.artistName,  item.trackViewUrl,  item.artworkUrl30,  item.artistId  );  });  resolve();  },  msg => { // Error  reject(msg);  }  );  });  } }   @Component({  selector: 'app-search',  template: `<form class="form-inline">  <div class="form-group">  <input type="search"  class="form-control"  placeholder="Enter search string"  #search>  </div>  <button type="button"  class="btn btn-primary"  (click)="onSearch(search.value)">  Search  </button> </form>  <hr />  <div class="text-center">  <p class="lead"  \*ngIf="loading">Loading...</p> </div>  <div class="list-group">  <a **[routerLink]="['/artist', track.artistId]"**  class="list-group-item list-group-item-action"  \*ngFor="let track of itunes.results">  <img src="{{track.thumbnail}}">  {{ track.name }} <span class="text-muted">by</span> {{ track.artist }}  </a> </div>  ` }) **class** SearchComponent {  **private** loading: **boolean** = **false**;   **constructor**(**private** itunes: SearchService,  **private** route: ActivatedRoute,  **private** router: Router) {  **this**.route.params.subscribe(params => {  console.log(params);  **if** (params['term']) {  **this**.doSearch(params['term'])  }  });  }   doSearch(term: **string**) {  **this**.loading = **true**;  **this**.itunes.search(term).then(\_ => **this**.loading = **false**)  }   onSearch(term: **string**) {  **this**.router.navigate(['search', {term: term}]);  } }  @Component({  selector: 'app-home',  template: ` <div class="jumbotron">  <h1 class="display-3">iTunes Search App</h1> </div>  ` }) **class** HomeComponent { }  @Component({  selector: 'app-header',  template: `<nav class="navbar navbar-light bg-faded">  <a class="navbar-brand"  [routerLink]="['home']">iTunes Search App  </a>  <ul class="nav navbar-nav">  <li class="nav-item"  [routerLinkActive]="['active']">  <a class="nav-link"  [routerLink]="['home']">Home  </a>  </li>  <li class="nav-item"  [routerLinkActive]="['active']">  <a class="nav-link"  [routerLink]="['search']">Search  </a>  </li>  </ul> </nav>  ` }) **class** HeaderComponent {  **constructor**(**private** router: Router) {  }   goHome() {  **this**.router.navigate(['']);  }   goSearch() {  **this**.router.navigate(['search']);  } }  @Component({  selector: 'app-artist-track-list',  template: ` <ul class="list-group">  <li class="list-group-item"  \*ngFor="let track of tracks">  <img src="{{track.artworkUrl30}}">  <a target="\_blank"  href="{{track.trackViewUrl}}">{{ track.trackName }}  </a>  </li> </ul>  ` }) **class** ArtistTrackListComponent {  **private** tracks: **any**[];   **constructor**(**private** jsonp: Jsonp,  **private** route: ActivatedRoute) {  **this.route.parent.params.subscribe(params => {**  **this**.jsonp.request(`https://itunes.apple.com/lookup?id=${params['artistId']}&entity=song&callback=JSONP\_CALLBACK`)  .toPromise()  .then(res => {  console.log(res.json());  **this**.tracks = res.json().results.slice(1);  });  });  } }  @Component({  selector: 'app-artist-album-list',  template: `<ul class="list-group">  <li class="list-group-item"  \*ngFor="let album of albums">  <img src="{{album.artworkUrl60}}">  <a target="\_blank"  href="{{album.collectionViewUrl}}">{{ album.collectionName }}  </a>  </li> </ul>  ` }) **class** ArtistAlbumListComponent {  **private** albums: **any**[];   **constructor**(**private** jsonp: Jsonp,  **private** route: ActivatedRoute) {  **this**.route.parent.params.subscribe(params => {  **this**.jsonp.request(`https://itunes.apple.com/lookup?id=${params['artistId']}&entity=album&callback=JSONP\_CALLBACK`)  .toPromise()  .then(res => {  console.log(res.json());  **this**.albums = res.json().results.slice(1);  });  });  } }  @Component({  selector: 'app-artist',  template: `<div class="card">  <div class="card-block">  <h4>{{artist?.artistName}} <span class="tag tag-default">{{artist?.primaryGenreName}}</span></h4>  <hr />  <footer>  <ul class="nav nav-pills">  <li class="nav-item">  <a class="nav-link"  [routerLinkActive]="['active']"  **[routerLink]="['./tracks']"**>Tracks  </a>  </li>  <li class="nav-item">  <a class="nav-link"  [routerLinkActive]="['active']"  **[routerLink]="['./albums']"**>Albums  </a>  </li>  </ul>  </footer>  </div> </div>  <div class="m-t-1">  **<router-outlet></router-outlet>** </div>  ` }) **class** ArtistComponent {  **private** artist: **any**;   **constructor**(**private** jsonp: Jsonp,  **private** route: ActivatedRoute) {  **this**.route.params.subscribe(params => {  **this**.jsonp.request(`https://itunes.apple.com/lookup?id=${params['artistId']}&callback=JSONP\_CALLBACK`)  .toPromise()  .then(res => {  console.log(res.json());  **this**.artist = res.json().results[0];  console.log(**this**.artist);  });  });  } }  @Component({  selector: 'app',  template: `  <app-header></app-header>  <div class="m-t-1">  <router-outlet></router-outlet>  </div>  ` }) **class** AppComponent { }  **const** routes: Routes = [  {path: '', redirectTo: 'home', pathMatch: 'full'},  {path: 'find', redirectTo: 'search'},  {path: 'home', component: HomeComponent},  {path: 'search', component: SearchComponent},  **{  path: 'artist/:artistId',  component: ArtistComponent,  children: [  {path: '', redirectTo: 'tracks', pathMatch: 'full'},  {path: 'tracks', component: ArtistTrackListComponent},  {path: 'albums', component: ArtistAlbumListComponent},  ]  },**  {path: '\*\*', component: HomeComponent} ];   @NgModule({  imports: [  BrowserModule,  ReactiveFormsModule,  FormsModule,  JsonpModule,  RouterModule.forRoot(routes, {useHash: **true**})  ],  declarations: [  AppComponent,  SearchComponent,  HomeComponent,  HeaderComponent,  ArtistAlbumListComponent,  ArtistTrackListComponent,  ArtistComponent  ],  bootstrap: [AppComponent],  providers: [SearchService] }) **class** AppModule { }  platformBrowserDynamic().bootstrapModule(AppModule);  artist track listing artist album listing |

**Router Guards**

|  |
| --- |
| **import** {NgModule, Component, Injectable} **from** '@angular/core'; **import** {BrowserModule} **from** '@angular/platform-browser'; **import** {platformBrowserDynamic} **from** '@angular/platform-browser-dynamic'; **import** {JsonpModule, Jsonp, Response} **from** '@angular/http'; **import** {ReactiveFormsModule, FormControl, FormsModule} **from** '@angular/forms'; **import** {Routes, RouterModule} **from** "@angular/router"; **import** {Observable} **from** 'rxjs'; **import** 'rxjs/add/operator/toPromise'; **import** {  Routes,  RouterModule,  Router,  ActivatedRoute,  CanActivate,  CanActivateChild,  CanDeactivate,  ActivatedRouteSnapshot,  RouterStateSnapshot } **from** "@angular/router";   // Domain models  **class** SearchItem {  **constructor**(**public** name: **string**,  **public** artist: **string**,  **public** link: **string**,  **public** thumbnail: **string**,  **public** artistId: **string**) {  } }  // Services  @Injectable() **class** SearchService {  apiRoot: **string** = 'https://itunes.apple.com/search';  results: SearchItem[];   **constructor**(**private** jsonp: Jsonp) {  **this**.results = [];  }   search(term: **string**) {  **return new** Promise((resolve, reject) => {  **this**.results = [];  **let** apiURL = `${**this**.apiRoot}?term=${term}&media=music&limit=20&callback=JSONP\_CALLBACK`;  **this**.jsonp.request(apiURL)  .toPromise()  .then(  res => { // Success  **this**.results = res.json().results.map(item => {  **return new** SearchItem(  item.trackName,  item.artistName,  item.trackViewUrl,  item.artworkUrl30,  item.artistId  );  });  resolve();  },  msg => { // Error  reject(msg);  }  );  });  } }  @Injectable() **class** UserService {  isLoggedIn(): **boolean** {  **return true**; // Switch to `false` to make OnlyLoggedInUsersGuard work  } }  // Components  @Component({  selector: 'app-search',  template: `<form class="form-inline">  <div class="form-group">  <input type="search"  class="form-control"  placeholder="Enter search string"  #search>  </div>  <button type="button"  class="btn btn-primary"  (click)="onSearch(search.value)">  Search  </button> </form>  <hr />  <div class="text-center">  <p class="lead"  \*ngIf="loading">Loading...</p> </div>  <div class="list-group">  <a [routerLink]="['/artist', track.artistId]"  class="list-group-item list-group-item-action"  \*ngFor="let track of itunes.results">  <img src="{{track.thumbnail}}">  {{ track.name }} <span class="text-muted">by</span> {{ track.artist }}  </a> </div>  ` }) **class** SearchComponent {  **private** loading: **boolean** = **false**;   **constructor**(**private** itunes: SearchService,  **private** route: ActivatedRoute,  **private** router: Router) {  **this**.route.params.subscribe(params => {  console.log(params);  **if** (params['term']) {  **this**.doSearch(params['term'])  }  });  }   doSearch(term: **string**) {  **this**.loading = **true**;  **this**.itunes.search(term).then(\_ => **this**.loading = **false**)  }   onSearch(term: **string**) {  **this**.router.navigate(['search', {term: term}]);  }   canDeactivate() {  **return this**.itunes.results.length > 0;  } }  @Component({  selector: 'app-home',  template: ` <div class="jumbotron">  <h1 class="display-3">iTunes Search App</h1> </div>  ` }) **class** HomeComponent { }  @Component({  selector: 'app-header',  template: `<nav class="navbar navbar-light bg-faded">  <a class="navbar-brand"  [routerLink]="['home']">iTunes Search App  </a>  <ul class="nav navbar-nav">  <li class="nav-item"  [routerLinkActive]="['active']">  <a class="nav-link"  [routerLink]="['home']">Home  </a>  </li>  <li class="nav-item"  [routerLinkActive]="['active']">  <a class="nav-link"  [routerLink]="['search']">Search  </a>  </li>  </ul> </nav>  ` }) **class** HeaderComponent {  **constructor**(**private** router: Router) {  }   goHome() {  **this**.router.navigate(['']);  }   goSearch() {  **this**.router.navigate(['search']);  } }  @Component({  selector: 'app-artist-track-list',  template: ` <ul class="list-group">  <li class="list-group-item"  \*ngFor="let track of tracks">  <img src="{{track.artworkUrl30}}">  <a target="\_blank"  href="{{track.trackViewUrl}}">{{ track.trackName }}  </a>  </li> </ul>  ` }) **class** ArtistTrackListComponent {  **private** tracks: **any**[];   **constructor**(**private** jsonp: Jsonp,  **private** route: ActivatedRoute) {  **this**.route.parent.params.subscribe(params => {  **this**.jsonp.request(`https://itunes.apple.com/lookup?id=${params['artistId']}&entity=song&callback=JSONP\_CALLBACK`)  .toPromise()  .then(res => {  console.log(res.json());  **this**.tracks = res.json().results.slice(1);  });  });  } }  @Component({  selector: 'app-artist-album-list',  template: `<ul class="list-group">  <li class="list-group-item"  \*ngFor="let album of albums">  <img src="{{album.artworkUrl60}}">  <a target="\_blank"  href="{{album.collectionViewUrl}}">{{ album.collectionName }}  </a>  </li> </ul>  ` }) **class** ArtistAlbumListComponent {  **private** albums: **any**[];   constructor(**private** jsonp: Jsonp,  **private** route: ActivatedRoute) {  **this**.route.parent.params.subscribe(params => {  **this**.jsonp.request(`https://itunes.apple.com/lookup?id=${params['artistId']}&entity=album&callback=JSONP\_CALLBACK`)  .toPromise()  .then(res => {  console.log(res.json());  **this**.albums = res.json().results.slice(1);  });  });  } }  @Component({  selector: 'app-artist',  template: `<div class="card">  <div class="card-block">  <h4>{{artist?.artistName}} <span class="tag tag-default">{{artist?.primaryGenreName}}</span></h4>  <hr />  <footer>  <ul class="nav nav-pills">  <li class="nav-item">  <a class="nav-link"  [routerLinkActive]="['active']"  [routerLink]="['./tracks']">Tracks  </a>  </li>  <li class="nav-item">  <a class="nav-link"  [routerLinkActive]="['active']"  [routerLink]="['./albums']">Albums  </a>  </li>  </ul>  </footer>  </div> </div>  <div class="m-t-1">  <router-outlet></router-outlet> </div>  ` }) **class** ArtistComponent {  **private** artist: **any**;   constructor(**private** jsonp: Jsonp,  **private** route: ActivatedRoute) {  **this**.route.params.subscribe(params => {  **this**.jsonp.request(`https://itunes.apple.com/lookup?id=${params['artistId']}&callback=JSONP\_CALLBACK`)  .toPromise()  .then(res => {  console.log(res.json());  **this**.artist = res.json().results[0];  console.log(**this**.artist);  });  });  } }  @Component({  selector: 'app',  template: `  <app-header></app-header>  <div class="m-t-1">  <router-outlet></router-outlet>  </div>  ` }) **class** AppComponent { }  // Guards  **class** AlwaysAuthGuard **implements** CanActivate {  canActivate() {  console.log("AlwaysAuthGuard");  **return true**;  } }  @Injectable() **class** OnlyLoggedInUsersGuard **implements** CanActivate {  constructor(**private** userService: UserService) {  };   canActivate() {  console.log("OnlyLoggedInUsers");  **if** (**this**.userService.isLoggedIn()) {  **return true**;  } **else** {  window.alert("You don't have permission to view this page");  **return false**;  }  } }   **class** AlwaysAuthChildrenGuard **implements** CanActivateChild {  canActivateChild() {  console.log("AlwaysAuthChildrenGuard");  **return true**;  } }  **class** UnsearchedTermGuard **implements** CanDeactivate<SearchComponent> {  canDeactivate(component: SearchComponent,  route: ActivatedRouteSnapshot,  state: RouterStateSnapshot): **boolean** {  console.log("UnsearchedTermGuard");  console.log(state.url);  **return** component.canDeactivate() || window.confirm("Are you sure?");  } }  // Routes  **const** routes: Routes = [  {path: '', redirectTo: 'home', pathMatch: 'full'},  {path: 'find', redirectTo: 'search'},  {path: 'home', component: HomeComponent},  {path: 'search', component: SearchComponent, canDeactivate: [UnsearchedTermGuard]},  {  path: 'artist/:artistId',  component: ArtistComponent,  canActivate: [OnlyLoggedInUsersGuard, AlwaysAuthGuard],  canActivateChild: [AlwaysAuthChildrenGuard],  children: [  {path: '', redirectTo: 'tracks', pathMatch: 'full'},  {path: 'tracks', component: ArtistTrackListComponent},  {path: 'albums', component: ArtistAlbumListComponent},  ]  },  {path: '\*\*', component: HomeComponent} ];  // Bootstrap  @NgModule({  imports: [  BrowserModule,  ReactiveFormsModule,  FormsModule,  JsonpModule,  RouterModule.forRoot(routes, {useHash: **true**})  ],  declarations: [  AppComponent,  SearchComponent,  HomeComponent,  HeaderComponent,  ArtistAlbumListComponent,  ArtistTrackListComponent,  ArtistComponent  ],  bootstrap: [AppComponent],  providers: [  SearchService,  UserService,  OnlyLoggedInUsersGuard,  AlwaysAuthGuard,  AlwaysAuthChildrenGuard,  UnsearchedTermGuard  ] }) **class** AppModule { }  platformBrowserDynamic().bootstrapModule(AppModule); |

**UnitTesting**

**Jasmine & Karma**

[**http://plnkr.co/edit/8ApdkvletoEc4Q0maXSN?p=preview**](http://plnkr.co/edit/8ApdkvletoEc4Q0maXSN?p=preview)

[**https://jasmine.github.io**](https://jasmine.github.io)

[**https://jasmine.github.io/edge/introduction.html#section-Included\_Matchers**](https://jasmine.github.io/edge/introduction.html#section-Included_Matchers)

|  |
| --- |
| **Main.js**  **function** helloWorld() {  **return** 'Hello world!'; } |
| **Test.js**  describe('Hello world', () => {   let expected = "";   beforeEach(() => {  expected = "Hello world!";  });   afterEach(() => {  expected = "";  });   **it('says hello', () => {  expect(helloWorld())  .toEqual(expected);  });** }); |
| **Index.html**  <!-- Run application specs in a browser --> <!DOCTYPE html> <html> <head>  <title>Jasmine Running</title>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1">  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/jasmine/2.4.1/jasmine.css"> </head> <body>  <script src="https://cdnjs.cloudflare.com/ajax/libs/jasmine/2.4.1/jasmine.js"></script>  <script src="https://cdnjs.cloudflare.com/ajax/libs/jasmine/2.4.1/jasmine-html.js"></script>  <script src="https://cdnjs.cloudflare.com/ajax/libs/jasmine/2.4.1/boot.js"></script>  <script src="main.js"></script>  <script src="test.js"></script> </body>  </html> |
|  |

**Karma**

**Testing Classes & Pipes**

|  |
| --- |
| Auth.service.ts  **export class** AuthService {  isAuthenticated(): **boolean** {  **return** !!localStorage.getItem('token');  } } |
| Auth.service.spec.ts  **import** {AuthService} **from** './auth.service';  describe('Service: Auth', () => {   **let** service: AuthService;   beforeEach(() => {  service = **new** AuthService();  });   afterEach(() => {  service = **null**;  localStorage.removeItem('token');  });   it('should return true from isAuthenticated when there is a token', () => {  localStorage.setItem('token', '1234');  expect(service.isAuthenticated()).toBeTruthy();  });   it('should return false from isAuthenticated when there is no token', () => {  expect(service.isAuthenticated()).toBeFalsy();  });  }); |
|  |
| **Default.pipe.ts**  **import** {Pipe, PipeTransform} **from** '@angular/core';  @Pipe({  name: 'default' }) **export class** DefaultPipe **implements** PipeTransform {   transform(value: **string**, fallback: **string**, forceHttps: **boolean** = **false**): **string** {  **let** image = "";  **if** (value) {  image = value;  } **else** {  image = fallback;  }  **if** (forceHttps) {  **if** (image.indexOf("https") == -1) {  image = image.replace("http", "https");  }  }  **return** image;  } } |
| **default.pipe.spect.ts**  /\* tslint:disable:no-unused-variable \*/  **import** {DefaultPipe} **from** './default.pipe';  describe('Pipe: Default', () => {  **let** pipe: DefaultPipe;   beforeEach(() => {  pipe = **new** DefaultPipe();  });   it('providing no value returns fallback', () => {  expect(pipe.transform('', 'http://place-hold.it/300')).toBe('http://place-hold.it/300');  });   it('providing a value returns value', () => {  expect(pipe.transform('http://place-hold.it/300', 'fallback')).toBe('http://place-hold.it/300');  });   it('asking for https returns https', () => {  expect(pipe.transform('', 'http://place-hold.it/300', **true**)).toBe('https://place-hold.it/300');  }); }); |
| **Index.html**  <!-- Run application specs in a browser --> <!DOCTYPE html> <html> <head>  <script>document.write('<base href="' + document.location + '" />');</script>  <title>Sample App Specs</title>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1">  <link rel="stylesheet" href="styles.css">  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/jasmine/2.4.1/jasmine.css">  </head> <body>  <script src="https://unpkg.com/systemjs@0.19.39/dist/system.src.js"></script>   <script src="https://cdnjs.cloudflare.com/ajax/libs/jasmine/2.4.1/jasmine.js"></script>  <script src="https://cdnjs.cloudflare.com/ajax/libs/jasmine/2.4.1/jasmine-html.js"></script>  <script src="https://cdnjs.cloudflare.com/ajax/libs/jasmine/2.4.1/boot.js"></script>   <script src="https://unpkg.com/reflect-metadata@0.1.8"></script>   <script src="https://unpkg.com/zone.js@0.7.4?main=browser"></script>  <script src="https://unpkg.com/zone.js/dist/long-stack-trace-zone.js?main=browser"></script>  <script src="https://unpkg.com/zone.js/dist/proxy.js?main=browser"></script>  <script src="https://unpkg.com/zone.js/dist/sync-test.js?main=browser"></script>  <script src="https://unpkg.com/zone.js/dist/jasmine-patch.js?main=browser"></script>  <script src="https://unpkg.com/zone.js/dist/async-test.js?main=browser"></script>  <script src="https://unpkg.com/zone.js/dist/fake-async-test.js?main=browser"></script>   **<script>  var \_\_spec\_files\_\_ = [  'app/auth.service.spec',  'app/default.pipe.spec'  ];  </script>  <script src="browser-test-shim.js"></script>** </body>  </html>   <!--  Copyright 2016 Google Inc. All Rights Reserved. Use of this source code is governed by an MIT-style license that can be found in the LICENSE file at http://angular.io/license --> |
|  |
|  |

**Testing with Mocks & Spies**

|  |
| --- |
| Index.html  <!-- Run application specs in a browser --> <!DOCTYPE html> <html> <head>  <script>document.write('<base href="' + document.location + '" />');</script>  <title>Sample App Specs</title>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1">  <link rel="stylesheet" href="styles.css">  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/jasmine/2.4.1/jasmine.css">  </head> <body>  <script src="https://unpkg.com/systemjs@0.19.39/dist/system.src.js"></script>   <script src="https://cdnjs.cloudflare.com/ajax/libs/jasmine/2.4.1/jasmine.js"></script>  <script src="https://cdnjs.cloudflare.com/ajax/libs/jasmine/2.4.1/jasmine-html.js"></script>  <script src="https://cdnjs.cloudflare.com/ajax/libs/jasmine/2.4.1/boot.js"></script>   <script src="https://unpkg.com/reflect-metadata@0.1.8"></script>   <script src="https://unpkg.com/zone.js@0.7.4?main=browser"></script>  <script src="https://unpkg.com/zone.js/dist/long-stack-trace-zone.js?main=browser"></script>  <script src="https://unpkg.com/zone.js/dist/proxy.js?main=browser"></script>  <script src="https://unpkg.com/zone.js/dist/sync-test.js?main=browser"></script>  <script src="https://unpkg.com/zone.js/dist/jasmine-patch.js?main=browser"></script>  <script src="https://unpkg.com/zone.js/dist/async-test.js?main=browser"></script>  <script src="https://unpkg.com/zone.js/dist/fake-async-test.js?main=browser"></script>   <script>  **var** \_\_spec\_files\_\_ = [  'app/auth.service.spec',  'app/login.component.spec',  ];  </script>  <script src="browser-test-shim.js"></script> </body>  </html>   <!--  Copyright 2016 Google Inc. All Rights Reserved. Use of this source code is governed by an MIT-style license that can be found in the LICENSE file at http://angular.io/license --> |
| Auth.service.ts  **export class** AuthService {  isAuthenticated(): **boolean** {  **return** !!localStorage.getItem('token');  } } |
| Login.component.ts  **import** {Component} **from** '@angular/core'; **import** {AuthService} **from** "./auth.service";  @Component({  selector: 'app-login',  template: `<a [hidden]="needsLogin()">Login</a>` }) **export class** LoginComponent {   **constructor**(**private** auth: AuthService) {  }   needsLogin() {  **return** !**this**.auth.isAuthenticated();  } } |
| Login.component.spec.ts  /\* tslint:disable:no-unused-variable \*/ **import** {LoginComponent} **from** './login.component'; **import** {AuthService} **from** "./auth.service";  describe('Component: Login', () => {   **let** component: LoginComponent;  **let** service: AuthService;  **let** spy: **any**;   beforeEach(() => {  service = **new** AuthService();  component = **new** LoginComponent(service);  });   afterEach(() => {  service = **null**;  component = **null**;  });    it('canLogin returns false when the user is not authenticated', () => {  spy = spyOn(service, 'isAuthenticated').and.returnValue(**false**);  expect(component.needsLogin()).toBeTruthy();  expect(service.isAuthenticated).toHaveBeenCalled();   });   it('canLogin returns false when the user is not authenticated', () => {  spy = spyOn(service, 'isAuthenticated').and.returnValue(**true**);  expect(component.needsLogin()).toBeFalsy();  expect(service.isAuthenticated).toHaveBeenCalled();  }); }); |
| Login.component.spec.alternatives.ts 1 |
| **import** {LoginComponent} **from** './login.component'; **import** {AuthService} **from** "./auth.service";  // Mocking by overriding functions  **class** MockAuthService **extends** AuthService {  authenticated = **false**;   isAuthenticated() {  **return this**.authenticated;  } }   - Mocking **with** fake classes  **import** {LoginComponent} **from** './login.component';  **class** MockAuthService {  authenticated = **false**;   isAuthenticated() {  **return this**.authenticated;  } }  describe('Component: Login', () => {   **let** component: LoginComponent;  **let** service: MockAuthService;   beforeEach(() => {  service = **new** MockAuthService();  component = **new** LoginComponent(service);  });   afterEach(() => {  service = **null**;  component = **null**;  });    it('canLogin returns false when the user is not authenticated', () => {  service.authenticated = **false**;  expect(component.needsLogin()).toBeTruthy();  });   it('canLogin returns false when the user is not authenticated', () => {  service.authenticated = **true**;  expect(component.needsLogin()).toBeFalsy();  }); }); |
| **import** {LoginComponent} **from** './login.component'; **import** {AuthService} **from** "./auth.service";  describe('Component: Login', () => {   **let** component: LoginComponent;  **let** service: AuthService;   beforeEach(() => {  service = **new** AuthService();  component = **new** LoginComponent(service);  });   afterEach(() => {  localStorage.removeItem('token');  service = **null**;  component = **null**;  });    it('canLogin returns false when the user is not authenticated', () => {  expect(component.needsLogin()).toBeTruthy();  });   it('canLogin returns false when the user is not authenticated', () => {  localStorage.setItem('token', '12345');  expect(component.needsLogin()).toBeFalsy();  }); }); |
|  |
|  |

**Angular Test Bed**

|  |
| --- |
| Index.html  <!-- Run application specs in a browser --> <!DOCTYPE html> <html> <head>  <script>document.write('<base href="' + document.location + '" />');</script>  <title>Sample App Specs</title>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1">  <link rel="stylesheet" href="styles.css">  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/jasmine/2.4.1/jasmine.css">  </head> <body>  <script src="https://unpkg.com/systemjs@0.19.39/dist/system.src.js"></script>   <script src="https://cdnjs.cloudflare.com/ajax/libs/jasmine/2.4.1/jasmine.js"></script>  <script src="https://cdnjs.cloudflare.com/ajax/libs/jasmine/2.4.1/jasmine-html.js"></script>  <script src="https://cdnjs.cloudflare.com/ajax/libs/jasmine/2.4.1/boot.js"></script>   <script src="https://unpkg.com/reflect-metadata@0.1.8"></script>   <script src="https://unpkg.com/zone.js@0.7.4?main=browser"></script>  <script src="https://unpkg.com/zone.js/dist/long-stack-trace-zone.js?main=browser"></script>  <script src="https://unpkg.com/zone.js/dist/proxy.js?main=browser"></script>  <script src="https://unpkg.com/zone.js/dist/sync-test.js?main=browser"></script>  <script src="https://unpkg.com/zone.js/dist/jasmine-patch.js?main=browser"></script>  <script src="https://unpkg.com/zone.js/dist/async-test.js?main=browser"></script>  <script src="https://unpkg.com/zone.js/dist/fake-async-test.js?main=browser"></script>   <script>  **var** \_\_spec\_files\_\_ = [  'app/auth.service.spec',  'app/login.component.spec',  ];  </script>  <script src="browser-test-shim.js"></script> </body>  </html>   <!--  Copyright 2016 Google Inc. All Rights Reserved. Use of this source code is governed by an MIT-style license that can be found in the LICENSE file at http://angular.io/license --> |
| Auth.service.ts  **export class** AuthService {  isAuthenticated(): **boolean** {  **return** !!localStorage.getItem('token');  } } |
| Auth.service.spec.ts  **import** {AuthService} **from** './auth.service';  describe('Service: Auth', () => {   **let** service: AuthService;   beforeEach(() => {  service = **new** AuthService();  });   afterEach(() => {  service = **null**;  localStorage.removeItem('token');  });   it('should return true from isAuthenticated when there is a token', () => {  localStorage.setItem('token', '1234');  expect(service.isAuthenticated()).toBeTruthy();  });   it('should return false from isAuthenticated when there is no token', () => {  expect(service.isAuthenticated()).toBeFalsy();  });  }); |
| Login.component.ts  **import** {Component} **from** '@angular/core'; **import** {AuthService} **from** "./auth.service";  @Component({  selector: 'app-login',  template: `<a [hidden]="needsLogin()">Login</a>` }) **export class** LoginComponent {   **constructor**(**private** auth: AuthService) {  }   needsLogin() {  **return** !**this**.auth.isAuthenticated();  } } |
| Login.component.spec.ts  /\* tslint:disable:no-unused-variable \*/ **import** {**TestBed**, ComponentFixture} **from** '@angular/core/testing'; **import** {LoginComponent} **from** './login.component'; **import** {AuthService} **from** "./auth.service";  describe('Component: Login', () => {   **let** component: LoginComponent;  **let** fixture: ComponentFixture<LoginComponent>;  **let** authService: AuthService;   beforeEach(() => {   // refine the test module by declaring the test component  TestBed.configureTestingModule({  declarations: [LoginComponent],  providers: [AuthService]  });   // create component and test fixture  fixture = TestBed.createComponent(LoginComponent);   // get test component from the fixture  component = fixture.componentInstance;   // UserService provided to the TestBed  authService = TestBed.get(AuthService);   });   it('canLogin returns false when the user is not authenticated', () => {  spyOn(authService, 'isAuthenticated').and.returnValue(**false**);  expect(component.needsLogin()).toBeTruthy();  expect(authService.isAuthenticated).toHaveBeenCalled();  });   it('canLogin returns false when the user is not authenticated', () => {  spyOn(authService, 'isAuthenticated').and.returnValue(**true**);  expect(component.needsLogin()).toBeFalsy();  expect(authService.isAuthenticated).toHaveBeenCalled();  }); }); |

**Testing Change Detection**

|  |
| --- |
| Index.html  <!-- Run application specs in a browser --> <!DOCTYPE html> <html> <head>  <script>document.write('<base href="' + document.location + '" />');</script>  <title>Sample App Specs</title>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1">  <link rel="stylesheet" href="styles.css">  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/jasmine/2.4.1/jasmine.css">  </head> <body>  <script src="https://unpkg.com/systemjs@0.19.39/dist/system.src.js"></script>   <script src="https://cdnjs.cloudflare.com/ajax/libs/jasmine/2.4.1/jasmine.js"></script>  <script src="https://cdnjs.cloudflare.com/ajax/libs/jasmine/2.4.1/jasmine-html.js"></script>  <script src="https://cdnjs.cloudflare.com/ajax/libs/jasmine/2.4.1/boot.js"></script>   <script src="https://unpkg.com/reflect-metadata@0.1.8"></script>   <script src="https://unpkg.com/zone.js@0.7.4?main=browser"></script>  <script src="https://unpkg.com/zone.js/dist/long-stack-trace-zone.js?main=browser"></script>  <script src="https://unpkg.com/zone.js/dist/proxy.js?main=browser"></script>  <script src="https://unpkg.com/zone.js/dist/sync-test.js?main=browser"></script>  <script src="https://unpkg.com/zone.js/dist/jasmine-patch.js?main=browser"></script>  <script src="https://unpkg.com/zone.js/dist/async-test.js?main=browser"></script>  <script src="https://unpkg.com/zone.js/dist/fake-async-test.js?main=browser"></script>   <script>  **var** \_\_spec\_files\_\_ = [  'app/auth.service.spec',  'app/login.component.spec',  ];  </script>  <script src="browser-test-shim.js"></script> </body>  </html>   <!--  Copyright 2016 Google Inc. All Rights Reserved. Use of this source code is governed by an MIT-style license that can be found in the LICENSE file at http://angular.io/license --> |
| Login.component.ts  **import** {Component} **from** '@angular/core'; **import** {AuthService} **from** "./auth.service";  @Component({  selector: 'app-login',  template: `  <a>   <span \*ngIf="needsLogin()">Login</span>  <span \*ngIf="!needsLogin()">Logout</span>  </a> ` }) **export class** LoginComponent {   **constructor**(**private** auth: AuthService) {  }   needsLogin() {  **return** !**this**.auth.isAuthenticated();  } } |
| Login.component.spec.ts  /\* tslint:disable:no-unused-variable \*/ **import** {TestBed, async, ComponentFixture} **from** '@angular/core/testing'; **import** {LoginComponent} **from** './login.component'; **import** {AuthService} **from** "./auth.service"; import {DebugElement} from "@angular/core"; import {By} from "@angular/platform-browser";  describe('Component: Login', () => {   **let** component: LoginComponent;  **let** fixture: ComponentFixture<LoginComponent>;  **let** authService: AuthService;  **let** el: DebugElement;   beforeEach(() => {   // refine the test module by declaring the test component  TestBed.configureTestingModule({  declarations: [LoginComponent],  providers: [AuthService]  });   // create component and test fixture  fixture = TestBed.createComponent(LoginComponent);   // get test component from the fixture  component = fixture.componentInstance;   // UserService provided to the TestBed  authService = TestBed.get(AuthService);   // get the "a" element by CSS selector (e.g., by class name)  **el = fixture.debugElement.query(By.css('a'));**  });   it('login button hidden when the user is authenticated', () => {  // To being with Angular has not done any change detection so the content is blank.  **expect(el.nativeElement.textContent.trim()).toBe('');**   // Trigger change detection and this lets the template update to the initial value which is Login since by  // default we are not authenticated  fixture.detectChanges();  **expect(el.nativeElement.textContent.trim()).toBe('Login');**   // Change the authetication state to true  spyOn(authService, 'isAuthenticated').and.returnValue(**true**);   // The label is still Login! We need changeDetection to run and for angular to update the template.   **expect(el.nativeElement.textContent.trim()).toBe('Login');**  // Which we can trigger via fixture.detectChange()  fixture.detectChanges();   // Now the label is Logout  **expect(el.nativeElement.textContent.trim()).toBe('Logout');**  }); }); |
|  |

**Testing Asynchronous Code**

|  |
| --- |
| Login.component.ts  **import** {Component} **from** '@angular/core'; **import** {AuthService} **from** "./auth.service";  @Component({  selector: 'app-login',  template: `  <a>   <span \*ngIf="needsLogin">Login</span>  <span \*ngIf="!needsLogin">Logout</span>  </a> ` }) **export class** LoginComponent **implements** OnInit {   needsLogin: **boolean** = **true**;   constructor(**private** auth: AuthService) {  }   ngOnInit() {  **this**.auth.isAuthenticated().then((authenticated) => {  **this**.needsLogin = !authenticated;  })  } } |
| Login.component.spec.ts  /\* tslint:disable:no-unused-variable \*/ **import** {TestBed, async, whenStable, fakeAsync, tick, ComponentFixture} **from** '@angular/core/testing'; **import** {LoginComponent} **from** './login.component'; **import** {AuthService} **from** "./auth.service"; **import** {DebugElement} **from** "@angular/core"; **import** {By} **from** "@angular/platform-browser";  describe('Component: Login', () => {   **let** component: LoginComponent;  **let** fixture: ComponentFixture<LoginComponent>;  **let** authService: AuthService;  **let** el: DebugElement;   beforeEach(() => {   // refine the test module by declaring the test component  TestBed.configureTestingModule({  declarations: [LoginComponent],  providers: [AuthService]  });   // create component and test fixture  fixture = TestBed.createComponent(LoginComponent);   // get test component from the fixture  component = fixture.componentInstance;   // UserService provided to the TestBed  authService = TestBed.get(AuthService);   // get the "a" element by CSS selector (e.g., by class name)  el = fixture.debugElement.query(By.css('a'));  });   it('Button label via fakeAsync() and tick()', fakeAsync(() => {  expect(el.nativeElement.textContent.trim()).toBe('');  fixture.detectChanges();  expect(el.nativeElement.textContent.trim()).toBe('Login');   spyOn(authService, 'isAuthenticated').and.returnValue(Promise.resolve(**true**));   component.ngOnInit();  // Simulates the passage of time until all pending asynchronous activities complete  tick();  fixture.detectChanges();  expect(el.nativeElement.textContent.trim()).toBe('Logout');  }));   it('Button label via async() and whenStable()', async(() => {  // async() knows about all the pending promises defined in it's function body.  fixture.detectChanges();  expect(el.nativeElement.textContent.trim()).toBe('Login');  **spyOn(authService, 'isAuthenticated').and.returnValue(Promise.resolve(true));**   fixture.whenStable().then(() => {  // This is called when ALL pending promises have been resolved  fixture.detectChanges();  expect(el.nativeElement.textContent.trim()).toBe('Logout');  });   component.ngOnInit();   }));   it('Button label via jasmine.done', (done) => {  fixture.detectChanges();  expect(el.nativeElement.textContent.trim()).toBe('Login');   // Make the authService return a promise that resolves to true   **let spy = spyOn(authService, 'isAuthenticated').and.returnValue(Promise.resolve(true));**  // We trigger the component to check the authService again  component.ngOnInit();   // We now want to call a function when the Promise returned from authService.isAuthenticated() is resolved  spy.calls.mostRecent().returnValue.then(() => {  // The needsChanged boolean has been updated on the Component so to update the template we trigger change detection  fixture.detectChanges();  // Now the label is Logout  expect(el.nativeElement.textContent.trim()).toBe('Logout');  // We tell jasmine we are done with this test spec  done();  });  }); }); |

**Testing Dependency Injection**

|  |
| --- |
| Login.component.spec.ts  /\* tslint:disable:no-unused-variable \*/ **import** {TestBed, ComponentFixture, inject} **from** '@angular/core/testing'; **import** {LoginComponent} **from** './login.component'; **import** {AuthService} **from** "./auth.service";  **class** MockAuthService **extends** AuthService {  isAuthenticated() {  **return** 'Mocked';  } }   describe('Component: Login', () => {   **let** component: LoginComponent;  **let** fixture: ComponentFixture<LoginComponent>;  **let** testBedService: AuthService;  **let** componentService: AuthService;   beforeEach(() => {   // refine the test module by declaring the test component  TestBed.configureTestingModule({  declarations: [LoginComponent],  providers: [AuthService]  });   // Configure the component with another set of Providers  TestBed.overrideComponent(  LoginComponent,  {set: {providers: [{provide: AuthService, useClass: MockAuthService}]}}  );   // create component and test fixture  fixture = TestBed.createComponent(LoginComponent);   // get test component from the fixture  component = fixture.componentInstance;   // AuthService provided to the TestBed  testBedService = TestBed.get(AuthService);   // AuthService provided by Component, (should return MockAuthService)  componentService = fixture.debugElement.injector.get(AuthService);  });   it('Service injected via inject(...) and TestBed.get(...) should be the same instance',  **inject([AuthService], (injectService: AuthService) => {  expect(injectService).toBe(testBedService);  })**  );   it('Service injected via component should be and instance of MockAuthService', () => {  expect(componentService **instanceof** MockAuthService).toBeTruthy();  }); }); |
| Login.component.ts  **import** {Component} **from** '@angular/core'; **import** {AuthService} **from** "./auth.service";  @Component({  selector: 'app-login',  template: `  <a>   <span \*ngIf="needsLogin">Login</span>  <span \*ngIf="!needsLogin">Logout</span>  </a> ` }) **export class** LoginComponent **implements** OnInit {   needsLogin: **boolean** = **true**;   **constructor**(**private** auth: AuthService) {  }   ngOnInit() {  **this**.auth.isAuthenticated().then((authenticated) => {  **this**.needsLogin = !authenticated;  })  } } |

**Testing Components**

|  |
| --- |
| Login.component.spec.ts  /\* tslint:disable:no-unused-variable \*/ **import** {TestBed, ComponentFixture, inject, async} **from** '@angular/core/testing'; **import** {LoginComponent, User} **from** './login.component'; **import** {Component, DebugElement} **from** "@angular/core"; **import** {By} **from** "@angular/platform-browser";   describe('Component: Login', () => {   **let** component: LoginComponent;  **let** fixture: ComponentFixture<LoginComponent>;  **let** submitEl: DebugElement;  **let** loginEl: DebugElement;  **let** passwordEl: DebugElement;   beforeEach(() => {   // refine the test module by declaring the test component  TestBed.configureTestingModule({  declarations: [LoginComponent]  });    // create component and test fixture  fixture = TestBed.createComponent(LoginComponent);   // get test component from the fixture  component = fixture.componentInstance;   submitEl = fixture.debugElement.query(By.css('button'));  loginEl = fixture.debugElement.query(By.css('input[type=email]'));  passwordEl = fixture.debugElement.query(By.css('input[type=password]'));  });   it('Setting enabled to false disabled the submit button', () => {  **component.enabled = false;  fixture.detectChanges();  expect(submitEl.nativeElement.disabled).toBeTruthy();**  });   it('Setting enabled to true enables the submit button', () => {  **component.enabled = true;  fixture.detectChanges();  expect(submitEl.nativeElement.disabled).toBeFalsy();**  });   it('Entering email and password emits loggedIn event', () => {  **let** user: User;  loginEl.nativeElement.value = "test@example.com";  passwordEl.nativeElement.value = "123456";   // Subscribe to the Observable and store the user in a local variable.  component.loggedIn.subscribe((value) => user = value);   // This sync emits the event and the subscribe callback gets executed above  submitEl.triggerEventHandler('click', **null**);   // Now we can check to make sure the emitted value is correct  expect(user.email).toBe("test@example.com");  expect(user.password).toBe("123456");  }); }) ; |
| Login.component.ts  **import** {Component, EventEmitter, Input, Output} **from** '@angular/core';  **export class** User {  **constructor**(**public** email: **string**, **public** password: **string**) {  } }  @Component({  selector: 'app-login',  template: ` <form>  <label>Email</label>  <input type="email"  #email>  <label>Password</label>  <input type="password"  #password>  <button type="button"  (click)="login(email.value, password.value)"  [disabled]="!enabled">Login  </button> </form> ` }) **export class** LoginComponent {  @Output() loggedIn = **new** EventEmitter<User>();  @Input() enabled = **true**;   login(email, password) {  console.log(`Login ${email} ${password}`);  **if** (email && password) {  console.log(`Emitting`);  **this**.loggedIn.emit(**new** User(email, password));  }  } } |

**Testing Directives**

|  |
| --- |
| Ohverfocus.directive.spec.ts  /\* tslint:disable:no-unused-variable \*/ **import** {TestBed, ComponentFixture} **from** '@angular/core/testing'; **import** {Component, DebugElement} **from** "@angular/core"; **import** {By} **from** "@angular/platform-browser"; **import** {TestBed} **from** '@angular/core/testing'; **import** {HoverFocusDirective} **from** './hoverfocus.directive';  @Component({  template: `<input type="text" hoverfocus>` }) **class** TestHoverFocusComponent { }   describe('Directive: HoverFocus', () => {   **let** component: TestHoverFocusComponent;  **let** fixture: ComponentFixture<TestHoverFocusComponent>;  **let** inputEl: DebugElement;   beforeEach(() => {  TestBed.configureTestingModule({  declarations: [TestHoverFocusComponent, HoverFocusDirective]  });  fixture = TestBed.createComponent(TestHoverFocusComponent);  component = fixture.componentInstance;  inputEl = fixture.debugElement.query(By.css('input'));  });   **it('hovering over input', () => {  inputEl.triggerEventHandler('mouseover', null);  fixture.detectChanges();  expect(inputEl.nativeElement.style.backgroundColor).toBe('blue');   inputEl.triggerEventHandler('mouseout', null);  fixture.detectChanges();  expect(inputEl.nativeElement.style.backgroundColor).toBe('inherit');  }); });** |
| Hoverfocus.directive.ts  **import** {  Directive,  HostListener,  HostBinding } **from** '@angular/core';  @Directive({  selector: '[hoverfocus]' }) **export class** HoverFocusDirective {   @HostBinding("style.background-color") backgroundColor: **string**;   @HostListener('mouseover') onHover() {  **this**.backgroundColor = 'blue';  }   @HostListener('mouseout') onLeave() {  **this**.backgroundColor = 'inherit';  } } |

**Testing Model Driven Forms**

|  |
| --- |
| Login.component.spec.ts  /\* tslint:disable:no-unused-variable \*/  **import** {TestBed, ComponentFixture} **from** '@angular/core/testing'; **import** {ReactiveFormsModule, FormsModule} **from** "@angular/forms"; **import** {LoginComponent, User} **from** "./login.component";   describe('Component: Login', () => {   **let** component: LoginComponent;  **let** fixture: ComponentFixture<LoginComponent>;   beforeEach(() => {   // refine the test module by declaring the test component  TestBed.configureTestingModule({  imports: [ReactiveFormsModule, FormsModule],  declarations: [LoginComponent]  });   // create component and test fixture  fixture = TestBed.createComponent(LoginComponent);   // get test component from the fixture  component = fixture.componentInstance;  component.ngOnInit();  });   it('form invalid when empty', () => {  expect(component.form.valid).toBeFalsy();  });   it('email field validity', () => {  **let** errors = {};  **let** email = component.form.controls['email'];  expect(email.valid).toBeFalsy();   // Email field is required  errors = email.errors || {};  expect(errors['required']).toBeTruthy();   // Set email to something  email.setValue("test");  errors = email.errors || {};  expect(errors['required']).toBeFalsy();  expect(errors['pattern']).toBeTruthy();   // Set email to something correct  email.setValue("test@example.com");  errors = email.errors || {};  expect(errors['required']).toBeFalsy();  expect(errors['pattern']).toBeFalsy();  });   it('password field validity', () => {  **let** errors = {};  **let** password = component.form.controls['password'];   // Email field is required  errors = password.errors || {};  expect(errors['required']).toBeTruthy();   // Set email to something  password.setValue("123456");  errors = password.errors || {};  expect(errors['required']).toBeFalsy();  expect(errors['minlength']).toBeTruthy();   // Set email to something correct  password.setValue("123456789");  errors = password.errors || {};  expect(errors['required']).toBeFalsy();  expect(errors['minlength']).toBeFalsy();  });   it('submitting a form emits a user', () => {  **expect(component.form.valid).toBeFalsy();  component.form.controls['email'].setValue("test@test.com");  component.form.controls['password'].setValue("123456789");  expect(component.form.valid).toBeTruthy();   let user: User;  // Subscribe to the Observable and store the user in a local variable.  component.loggedIn.subscribe((value) => user = value);   // Trigger the login function  component.login();   // Now we can check to make sure the emitted value is correct  expect(user.email).toBe("test@test.com");  expect(user.password).toBe("123456789");**  }); }) ; |
| Login.component.ts  **import** {  Component,  EventEmitter,  Output } **from** '@angular/core'; **import** {  FormGroup,  Validators,  FormBuilder } **from** "@angular/forms";  **export class** User {  **constructor**(**public** email: **string**,  **public** password: **string**) {  } }  @Component({  selector: 'app-login',  template: ` <form (ngSubmit)="login()"  [formGroup]="form">  <label>Email</label>  <input type="email"  formControlName="email">  <label>Password</label>  <input type="password"  formControlName="password">  <button type="submit">Login</button> </form> ` }) **export class** LoginComponent {  @Output() loggedIn = **new** EventEmitter<User>();  form: FormGroup;   **constructor**(**private** fb: FormBuilder) {  }   ngOnInit() {  **this**.form = **this**.fb.group({  email: ['', [  Validators.required,  Validators.pattern("[^ @]\*@[^ @]\*")]],  password: ['', [  Validators.required,  Validators.minLength(8)]],  });  }   login() {  console.log(`Login ${**this**.form.value}`);  **if** (**this**.form.valid) {  **this**.loggedIn.emit(  **new** User(  **this**.form.value.email,  **this**.form.value.password  )  );  }  } } |

**Testing Http**

|  |
| --- |
| Search.service.spec.ts  /\* tslint:disable:no-unused-variable \*/ **import** {  JsonpModule,  Jsonp,  BaseRequestOptions,  Response,  ResponseOptions,  Http } **from** "@angular/http"; **import** {TestBed, fakeAsync, tick} **from** '@angular/core/testing'; **import** {MockBackend} **from** "@angular/http/testing"; **import** {SearchService} **from** './search.service';  describe('Service: Search', () => {   **let** service: SearchService;  **let** backend: MockBackend;   beforeEach(() => {  TestBed.configureTestingModule({  imports: [JsonpModule],  providers: [  SearchService,  MockBackend,  BaseRequestOptions,  {  provide: Jsonp,  useFactory: (backend, options) => **new** Jsonp(backend, options),  deps: [MockBackend, BaseRequestOptions]  }  ]  });   // Get the MockBackend  backend = TestBed.get(MockBackend);   // Returns a service with the MockBackend so we can test with dummy responses  service = TestBed.get(SearchService);   });   it('search should return SearchItems', fakeAsync(() => {  **let** response = {  "resultCount": 1,  "results": [  {  "artistId": 78500,  "artistName": "U2",  "trackName": "Beautiful Day",  "artworkUrl60": "image.jpg",  }]  };   // When the request subscribes for results on a connection, return a fake response  backend.connections.subscribe(connection => {  connection.mockRespond(**new** Response(<ResponseOptions>{  body: JSON.stringify(response)  }));  });   // Perform a request and make sure we get the response we expect  service.search("U2");  tick();   expect(service.results.length).toBe(1);  expect(service.results[0].artist).toBe("U2");  expect(service.results[0].name).toBe("Beautiful Day");  expect(service.results[0].thumbnail).toBe("image.jpg");  expect(service.results[0].artistId).toBe(78500);  })); }); |
| Search.service.ts  **import** {Injectable} **from** '@angular/core'; **import** {Jsonp} **from** '@angular/http'; **import** 'rxjs/add/operator/toPromise';  **class** SearchItem {  **constructor**(**public** name: **string**,  **public** artist: **string**,  **public** thumbnail: **string**,  **public** artistId: **string**) {  } }  @Injectable() **export class** SearchService {  apiRoot: **string** = 'https://itunes.apple.com/search';  results: SearchItem[];   **constructor**(**private** jsonp: Jsonp) {  **this**.results = [];  }   search(term: **string**) {  **return new** Promise((resolve, reject) => {  **this**.results = [];  **let** apiURL = `${**this**.apiRoot}?term=${term}&media=music&limit=20&callback=JSONP\_CALLBACK`;  **this**.jsonp.request(apiURL)  .toPromise()  .then(  res => { // Success  **this**.results = res.json().results.map(item => {  console.log(item);  **return new** SearchItem(  item.trackName,  item.artistName,  item.artworkUrl60,  item.artistId  );  });  resolve(**this**.results);  },  msg => { // Error  reject(msg);  }  );  });  } } |

**Testing Routing**

|  |
| --- |
| Router.spect.ts  /\* tslint:disable:no-unused-variable \*/ **import** {Location} **from** "@angular/common"; **import** {TestBed, fakeAsync, tick} **from** '@angular/core/testing'; **import** {RouterTestingModule} **from** "@angular/router/testing"; **import** {Router} **from** "@angular/router";  **import** {  HomeComponent,  SearchComponent,  AppComponent,  routes } **from** "./router"  describe('Router: App', () => {   **let** location: Location;  **let** router: Router;  **let** fixture;   beforeEach(() => {  TestBed.configureTestingModule({  imports: [ RouterTestingModule.withRoutes(routes)],  declarations: [  HomeComponent,  SearchComponent,  AppComponent  ]  });   router = TestBed.get(Router);  location = TestBed.get(Location);   fixture = TestBed.createComponent(AppComponent);  router.initialNavigation();  });   it('fakeAsync works', fakeAsync(() => {  **let** promise = **new** Promise((resolve) => {  setTimeout(resolve, 10)  });  **let** done = **false**;  promise.then(() => done = **true**);  tick(50);  expect(done).toBeTruthy();  }));   it('navigate to "" redirects you to /home', fakeAsync(() => {  router.navigate(['']);  tick(50);  expect(location.path()).toBe('/home');  }));   it('navigate to "search" takes you to /search', fakeAsync(() => {  router.navigate(['/search']);  tick(50);  expect(location.path()).toBe('/search');  })); }); |
| Router.ts  **import** {Component} **from** "@angular/core"; **import** {Routes} **from** "@angular/router";  @Component({  template: `Search` }) **export class** SearchComponent { }  @Component({  template: `Home` }) **export class** HomeComponent { }  @Component({  template: `<router-outlet></router-outlet>` }) **export class** AppComponent { }  **export const** routes: Routes = [  // {path: '', redirectTo: 'home', pathMatch: 'full'},  {path: 'home', component: HomeComponent},  {path: 'search', component: SearchComponent} ]; |

**Advanced Topics**

**Custom Form Validators**

|  |
| --- |
| **model-driven script.ts**  **import** {  NgModule,  Component,  Pipe,  OnInit } **from** '@angular/core'; **import** {  ReactiveFormsModule,  FormsModule,  FormGroup,  FormControl,  Validators,  FormBuilder } **from** '@angular/forms'; **import** {BrowserModule} **from** '@angular/platform-browser'; **import** {platformBrowserDynamic} **from** '@angular/platform-browser-dynamic';  **function** emailDomainValidator(control: FormControl) {  **let** email = control.value;  **if** (email && email.indexOf("@") != -1) {  **let** [\_, domain] = email.split("@");  **if** (domain !== "codecraft.tv") {  **return** {  emailDomain: {  parsedDomain: domain  }  }  }  }  **return null**; }  @Component({  selector: 'model-form',  template: `<form novalidate  [formGroup]="myform">   <fieldset formGroupName="name">  <div class="form-group"  [ngClass]="{  'has-danger': firstName.invalid && (firstName.dirty || firstName.touched),  'has-success': firstName.valid && (firstName.dirty || firstName.touched)  }">  <label>First Name</label>  <input type="text"  class="form-control"  formControlName="firstName"  required>  <div class="form-control-feedback"  \*ngIf="firstName.errors && (firstName.dirty || firstName.touched)">  <p \*ngIf="firstName.errors.required">First Name is required</p>  </div>   </div>   <div class="form-group"  [ngClass]="{  'has-danger': lastName.invalid && (lastName.dirty || lastName.touched),  'has-success': lastName.valid && (lastName.dirty || lastName.touched)  }">  <label>Last Name</label>  <input type="text"  class="form-control"  formControlName="lastName"  required>  <div class="form-control-feedback"  \*ngIf="lastName.errors && (lastName.dirty || lastName.touched)">  <p \*ngIf="lastName.errors.required">Last Name is required</p>  </div>  </div>  </fieldset>    <div class="form-group"  [ngClass]="{  'has-danger': email.invalid && (email.dirty || email.touched),  'has-success': email.valid && (email.dirty || email.touched)  }">  <label>Email</label>  <input type="email"  class="form-control"  formControlName="email"  required>  <div class="form-control-feedback"  \*ngIf="email.errors && (email.dirty || email.touched)">  <p \*ngIf="email.errors.required">Email is required</p>  <p \*ngIf="password.errors.pattern">The email address must contain at least the @ character</p>  <p \*ngIf="email.errors.emailDomain">Email must be on the codecraft.tv domain</p>  </div>   </div>   <div class="form-group"  [ngClass]="{  'has-danger': password.invalid && (password.dirty || password.touched),  'has-success': password.valid && (password.dirty || password.touched)  }">  <label>Password</label>  <input type="password"  class="form-control"  formControlName="password"  required>  <div class="form-control-feedback"  \*ngIf="password.errors && (password.dirty || password.touched)">  <p \*ngIf="password.errors.required">Password is required</p>  <p \*ngIf="password.errors.minlength">Password must be 8 characters long, we need another {{password.errors.minlength.requiredLength - password.errors.minlength.actualLength}} characters </p>  </div>  </div>   <div class="form-group"  [ngClass]="{  'has-danger': language.invalid && (language.dirty || language.touched),  'has-success': language.valid && (language.dirty || language.touched)  }">  <label>Language</label>  <select class="form-control"  formControlName="language">  <option value="">Please select a language</option>  <option \*ngFor="let lang of langs"  [value]="lang">{{lang}}  </option>  </select>  </div>   <pre>{{myform.value | json}}</pre> </form>` }) **class** ModelFormComponent **implements** OnInit {  langs: **string**[] = [  'English',  'French',  'German',  ];  myform: FormGroup;  firstName: FormControl;  lastName: FormControl;  email: FormControl;  password: FormControl;  language: FormControl;    ngOnInit() {  **this**.createFormControls();  **this**.createForm();  }   createFormControls() {  **this**.firstName = **new** FormControl('', Validators.required);  **this**.lastName = **new** FormControl('', Validators.required);  **this**.email = **new** FormControl('', [  Validators.required,  Validators.pattern("[^ @]\*@[^ @]\*"),  emailDomainValidator  ]);  **this**.password = **new** FormControl('', [  Validators.required,  Validators.minLength(8)  ]);  **this**.language = **new** FormControl('');  }   createForm() {  **this**.myform = **new** FormGroup({  name: **new** FormGroup({  firstName: **this**.firstName,  lastName: **this**.lastName,  }),  email: **this**.email,  password: **this**.password,  language: **this**.language  });  } }   @Component({  selector: 'app',  template: `<model-form></model-form>` }) **class** AppComponent { }   @NgModule({  imports: [  BrowserModule,  FormsModule,  ReactiveFormsModule],  declarations: [  AppComponent,  ModelFormComponent  ],  bootstrap: [  AppComponent  ] }) **class** AppModule { }  platformBrowserDynamic().bootstrapModule(AppModule); |
| **Template-driven script.ts**  **import** {  NgModule,  Component,  OnInit,  ViewChild,  Directive,  Inject,  Input, } **from** '@angular/core'; **import** {  NG\_VALIDATORS,  FormsModule,  FormGroup,  FormControl,  ValidatorFn,  Validators } **from** '@angular/forms'; **import** {BrowserModule} **from** '@angular/platform-browser'; **import** {platformBrowserDynamic} **from** '@angular/platform-browser-dynamic';  **class** Signup {  **constructor**(**public** firstName: **string** = '',  **public** lastName: **string** = '',  **public** email: **string** = '',  **public** password: **string** = '',  **public** language: **string** = '') {  } }  **function** emailDomainValidator(control: FormControl) {  **let** email = control.value;  **if** (email && email.indexOf("@") != -1) {  **let** [\_, domain] = email.split("@");  **if** (domain !== "codecraft.tv") {  **return** {  emailDomain: {  parsedDomain: domain  }  }  }  }  **return null**; }  @Directive({  selector: '[emailDomain][ngModel]',  providers: [  {  provide: NG\_VALIDATORS,  useValue: emailDomainValidator,  multi: **true** }  ] }) **class** EmailDomainValidator { }   @Component({  selector: 'template-form',  template: `<!--suppress ALL --> <form novalidate  (ngSubmit)="onSubmit()"  #f="ngForm">   <fieldset ngModelGroup="name">  <div class="form-group"  [ngClass]="{  'has-danger': firstName.invalid && (firstName.dirty || firstName.touched),  'has-success': firstName.valid && (firstName.dirty || firstName.touched)  }">  <label>First Name</label>  <input type="text"  class="form-control"  name="firstName"  [(ngModel)]="model.firstName"  required  #firstName="ngModel">  <div class="form-control-feedback"  \*ngIf="firstName.errors && (firstName.dirty || firstName.touched)">  <p \*ngIf="firstName.errors.required">First name is required</p>  </div>  </div>   <div class="form-group"  [ngClass]="{  'has-danger': lastName.invalid && (lastName.dirty || lastName.touched),  'has-success': lastName.valid && (lastName.dirty || lastName.touched)  }">  <label>Last Name</label>  <input type="text"  class="form-control"  name="lastName"  [(ngModel)]="model.lastName"  required  #lastName="ngModel">  <div class="form-control-feedback"  \*ngIf="lastName.errors && (lastName.dirty || lastName.touched)">  <p \*ngIf="lastName.errors.required">Last name is required</p>  </div>  </div>  </fieldset>    <div class="form-group"  [ngClass]="{  'has-danger': email.invalid && (email.dirty || email.touched),  'has-success': email.valid && (email.dirty || email.touched)  }">  <label>Email</label>  <input type="email"  class="form-control"  name="email"  [(ngModel)]="model.email"  required  pattern="[^ @]\*@[^ @]\*"  emailDomain  #email="ngModel">  <div class="form-control-feedback"  \*ngIf="email.errors && (email.dirty || email.touched)">  <p \*ngIf="email.errors.required">Email is required</p>  <p \*ngIf="email.errors.pattern">Email must contain at least the @ character</p>  <!--<p \*ngIf="email.errors.emailDomain">Email must be on the codecraft.tv domain</p>-->  <p \*ngIf="email.errors.emailDomain">Email must be on the {{ email.errors.emailDomain.requiredDomain }} domain</p>  </div>  </div>    <div class="form-group"  [ngClass]="{  'has-danger': password.invalid && (password.dirty || password.touched),  'has-success': password.valid && (password.dirty || password.touched)  }">  <label>Password</label>  <input type="password"  class="form-control"  name="password"  [(ngModel)]="model.password"  required  minlength="8"  #password="ngModel">  <div class="form-control-feedback"  \*ngIf="password.errors && (password.dirty || password.touched)">  <p \*ngIf="password.errors.required">Password is required</p>  <p \*ngIf="password.errors.minlength">Password must be at least 8 characters long</p>  </div>  </div>   <div class="form-group">  <label>Language</label>  <select class="form-control"  name="language"  [(ngModel)]="model.language">  <option value="">Please select a language</option>  <option \*ngFor="let lang of langs"  [value]="lang">{{lang}}  </option>  </select>  </div>   <button type="submit"  class="btn btn-primary"  [disabled]="f.invalid">Submit  </button>   <pre>{{f.value | json}}</pre> </form>  ` }) **class** TemplateFormComponent {   model: Signup = **new** Signup();  @ViewChild('f') form: **any**;   langs: **string**[] = [  'English',  'French',  'German',  ];   onSubmit() {  **if** (**this**.form.valid) {  console.log("Form Submitted!");  **this**.form.reset();  }  } }  @Component({  selector: 'app',  template: `<template-form></template-form>` }) **class** AppComponent { }   @NgModule({  imports: [  BrowserModule,  FormsModule  ],  declarations: [  AppComponent,  TemplateFormComponent,  EmailDomainValidator  ],  bootstrap: [  AppComponent  ] }) **class** AppModule { }  platformBrowserDynamic().bootstrapModule(AppModule); |

**Configurable Custom Form Validators**

|  |
| --- |
| **Model-driven script.ts**  **import** {  NgModule,  Component,  Pipe,  OnInit } **from** '@angular/core'; **import** {  ReactiveFormsModule,  FormsModule,  FormGroup,  FormControl,  Validators,  FormBuilder } **from** '@angular/forms'; **import** {BrowserModule} **from** '@angular/platform-browser'; **import** {platformBrowserDynamic} **from** '@angular/platform-browser-dynamic';   // Basic hardcoded validator function // **function** emailDomainValidator(control: FormControl) {  **let** email = control.value;  **if** (email && email.indexOf("@") != -1) {  **let** [\_, domain] = email.split("@");  **if** (domain !== "codecraft.tv") {  **return** {  emailDomain: {  parsedDomain: domain  }  }  }  }  **return null**; }  // Configurable validator function // **class** CodeCraftValidators {  **static** *emailDomain*(requiredDomain) {  **return function** (control: FormControl) {  **let** email = control.value;  **if** (email && email.indexOf("@") != -1) {  **let** [\_, domain] = email.split("@");  **if** (domain !== requiredDomain) {  **return** {  emailDomain: {  parsedDomain: domain,  requiredDomain: requiredDomain  }  }  }  }  **return null**;  }  } }  @Component({  selector: 'model-form',  template: `<form novalidate  [formGroup]="myform">   <fieldset formGroupName="name">  <div class="form-group"  [ngClass]="{  'has-danger': firstName.invalid && (firstName.dirty || firstName.touched),  'has-success': firstName.valid && (firstName.dirty || firstName.touched)  }">  <label>First Name</label>  <input type="text"  class="form-control"  formControlName="firstName"  required>  <div class="form-control-feedback"  \*ngIf="firstName.errors && (firstName.dirty || firstName.touched)">  <p \*ngIf="firstName.errors.required">Last Name is required</p>  </div>   </div>   <div class="form-group"  [ngClass]="{  'has-danger': lastName.invalid && (lastName.dirty || lastName.touched),  'has-success': lastName.valid && (lastName.dirty || lastName.touched)  }">  <label>Last Name</label>  <input type="text"  class="form-control"  formControlName="lastName"  required>  <div class="form-control-feedback"  \*ngIf="lastName.errors && (lastName.dirty || lastName.touched)">  <p \*ngIf="lastName.errors.required">Last Name is required</p>  </div>  </div>  </fieldset>    <div class="form-group"  [ngClass]="{  'has-danger': email.invalid && (email.dirty || email.touched),  'has-success': email.valid && (email.dirty || email.touched)  }">  <label>Email</label>  <input type="email"  class="form-control"  formControlName="email"  required>  <div class="form-control-feedback"  \*ngIf="email.errors && (email.dirty || email.touched)">  <p \*ngIf="email.errors.required">Email is required</p>  <p \*ngIf="password.errors.pattern">The email address must contain at least the @ character</p>  <!--<p \*ngIf="email.errors.emailDomain">Email must be on the codecraft.tv domain</p>-->  <p \*ngIf="email.errors.emailDomain">Email must be on the {{ email.errors.emailDomain.requiredDomain }} domain</p>  </div>   </div>   <div class="form-group"  [ngClass]="{  'has-danger': password.invalid && (password.dirty || password.touched),  'has-success': password.valid && (password.dirty || password.touched)  }">  <label>Password</label>  <input type="password"  class="form-control"  formControlName="password"  required>  <div class="form-control-feedback"  \*ngIf="password.errors && (password.dirty || password.touched)">  <p \*ngIf="password.errors.required">Password is required</p>  <p \*ngIf="password.errors.minlength">Password must be 8 characters long, we need another {{password.errors.minlength.requiredLength - password.errors.minlength.actualLength}} characters </p>  </div>  </div>   <div class="form-group"  [ngClass]="{  'has-danger': language.invalid && (language.dirty || language.touched),  'has-success': language.valid && (language.dirty || language.touched)  }">  <label>Language</label>  <select class="form-control"  formControlName="language">  <option value="">Please select a language</option>  <option \*ngFor="let lang of langs"  [value]="lang">{{lang}}  </option>  </select>  </div>   <pre>{{myform.value | json}}</pre> </form>` }) **class** ModelFormComponent **implements** OnInit {  langs: **string**[] = [  'English',  'French',  'German',  ];  myform: FormGroup;  firstName: FormControl;  lastName: FormControl;  email: FormControl;  password: FormControl;  language: FormControl;    ngOnInit() {  **this**.createFormControls();  **this**.createForm();  }   createFormControls() {  **this**.firstName = **new** FormControl('', Validators.required);  **this**.lastName = **new** FormControl('', Validators.required);  **this**.email = **new** FormControl('', [  Validators.required,  Validators.pattern("[^ @]\*@[^ @]\*"),  CodeCraftValidators.*emailDomain*('codecraft.tv')  ]);  **this**.password = **new** FormControl('', [  Validators.required,  Validators.minLength(8)  ]);  **this**.language = **new** FormControl('');  }   createForm() {  **this**.myform = **new** FormGroup({  name: **new** FormGroup({  firstName: **this**.firstName,  lastName: **this**.lastName,  }),  email: **this**.email,  password: **this**.password,  language: **this**.language  });  } }   @Component({  selector: 'app',  template: `<model-form></model-form>` }) **class** AppComponent { }   @NgModule({  imports: [  BrowserModule,  FormsModule,  ReactiveFormsModule],  declarations: [  AppComponent,  ModelFormComponent  ],  bootstrap: [  AppComponent  ] }) **class** AppModule { }  platformBrowserDynamic().bootstrapModule(AppModule); |
| **Template-driven script.ts**  **import** {  NgModule,  Component,  OnInit,  ViewChild,  Directive,  Inject,  Input, } **from** '@angular/core'; **import** {  NG\_VALIDATORS,  FormsModule,  FormGroup,  FormControl,  ValidatorFn,  Validators } **from** '@angular/forms'; **import** {BrowserModule} **from** '@angular/platform-browser'; **import** {platformBrowserDynamic} **from** '@angular/platform-browser-dynamic';  **class** Signup {  **constructor**(**public** firstName: **string** = '',  **public** lastName: **string** = '',  **public** email: **string** = '',  **public** password: **string** = '',  **public** language: **string** = '') {  } }  // Basic hardcoded validator function // **function** emailDomainValidator(control: FormControl) {  **let** email = control.value;  **if** (email && email.indexOf("@") != -1) {  **let** [\_, domain] = email.split("@");  **if** (domain !== "codecraft.tv") {  **return** {  emailDomain: {  parsedDomain: domain  }  }  }  }  **return null**; }  // Configurable validator function // **class** CodeCraftValidators {  **static** *emailDomain*(requiredDomain) {  **return function** (control: FormControl) {  **let** email = control.value;  **if** (email && email.indexOf("@") != -1) {  **let** [\_, domain] = email.split("@");  **if** (domain !== requiredDomain) {  **return** {  emailDomain: {  parsedDomain: domain,  requiredDomain: requiredDomain  }  }  }  }  **return null**;  }  } }  // Basic hardcoded directive // // @Directive({ // selector: '[emailDomain][ngModel]', // providers: [ // { // provide: NG\_VALIDATORS, // useValue: emailDomainValidator, // multi: true // } // ] // }) // class EmailDomainValidator { // }   // Directive configured via DI // // @Directive({ // selector: '[emailDomain][ngModel]', // providers: [ // { // provide: NG\_VALIDATORS, // useClass: EmailDomainValidator, // multi: true // } // ] // }) // class EmailDomainValidator { // // private valFn = ValidatorFn; // // constructor(@Inject('RequiredDomain') requiredDomain: string) { // this.valFn = CodeCraftValidators.emailDomain(requiredDomain) // } // // validate(control: FormControl) { // return this.valFn(control); // } // }  // Directive configured via input property binding // @Directive({  selector: '[emailDomain][ngModel]',  providers: [  {  provide: NG\_VALIDATORS,  useExisting: EmailDomainValidator,  multi: **true** }  ] }) **class** EmailDomainValidator {  @Input('emailDomain') emailDomain: **string**;  **private** valFn = Validators.nullValidator;   ngOnChanges(): **void** {  **if** (**this**.emailDomain) {  **this**.valFn = CodeCraftValidators.*emailDomain*(**this**.emailDomain)  } **else** {  **this**.valFn = Validators.nullValidator;  }  }   validate(control: FormControl) {  **return this**.valFn(control);  } }   @Component({  selector: 'template-form',  template: `<!--suppress ALL --> <form novalidate  (ngSubmit)="onSubmit()"  #f="ngForm">   <fieldset ngModelGroup="name">  <div class="form-group"  [ngClass]="{  'has-danger': firstName.invalid && (firstName.dirty || firstName.touched),  'has-success': firstName.valid && (firstName.dirty || firstName.touched)  }">  <label>First Name</label>  <input type="text"  class="form-control"  name="firstName"  [(ngModel)]="model.firstName"  required  #firstName="ngModel">  <div class="form-control-feedback"  \*ngIf="firstName.errors && (firstName.dirty || firstName.touched)">  <p \*ngIf="firstName.errors.required">First name is required</p>  </div>  </div>   <div class="form-group"  [ngClass]="{  'has-danger': lastName.invalid && (lastName.dirty || lastName.touched),  'has-success': lastName.valid && (lastName.dirty || lastName.touched)  }">  <label>Last Name</label>  <input type="text"  class="form-control"  name="lastName"  [(ngModel)]="model.lastName"  required  #lastName="ngModel">  <div class="form-control-feedback"  \*ngIf="lastName.errors && (lastName.dirty || lastName.touched)">  <p \*ngIf="lastName.errors.required">Last name is required</p>  </div>  </div>  </fieldset>    <div class="form-group"  [ngClass]="{  'has-danger': email.invalid && (email.dirty || email.touched),  'has-success': email.valid && (email.dirty || email.touched)  }">  <label>Email</label>  <input type="email"  class="form-control"  name="email"  [(ngModel)]="model.email"  required  pattern="[^ @]\*@[^ @]\*"  [emailDomain]="'codecraft.tv'"  #email="ngModel">  <div class="form-control-feedback"  \*ngIf="email.errors && (email.dirty || email.touched)">  <p \*ngIf="email.errors.required">Email is required</p>  <p \*ngIf="email.errors.pattern">Email must contain at least the @ character</p>  <!--<p \*ngIf="email.errors.emailDomain">Email must be on the codecraft.tv domain</p>-->  <p \*ngIf="email.errors.emailDomain">Email must be on the {{ email.errors.emailDomain.requiredDomain }} domain</p>  </div>  </div>    <div class="form-group"  [ngClass]="{  'has-danger': password.invalid && (password.dirty || password.touched),  'has-success': password.valid && (password.dirty || password.touched)  }">  <label>Password</label>  <input type="password"  class="form-control"  name="password"  [(ngModel)]="model.password"  required  minlength="8"  #password="ngModel">  <div class="form-control-feedback"  \*ngIf="password.errors && (password.dirty || password.touched)">  <p \*ngIf="password.errors.required">Password is required</p>  <p \*ngIf="password.errors.minlength">Password must be at least 8 characters long</p>  </div>  </div>   <div class="form-group">  <label>Language</label>  <select class="form-control"  name="language"  [(ngModel)]="model.language">  <option value="">Please select a language</option>  <option \*ngFor="let lang of langs"  [value]="lang">{{lang}}  </option>  </select>  </div>   <button type="submit"  class="btn btn-primary"  [disabled]="f.invalid">Submit  </button>   <pre>{{f.value | json}}</pre> </form>  ` }) **class** TemplateFormComponent {   model: Signup = **new** Signup();  @ViewChild('f') form: **any**;   langs: **string**[] = [  'English',  'French',  'German',  ];   onSubmit() {  **if** (**this**.form.valid) {  console.log("Form Submitted!");  **this**.form.reset();  }  } }  @Component({  selector: 'app',  template: `<template-form></template-form>` }) **class** AppComponent { }   @NgModule({  imports: [  BrowserModule,  FormsModule  ],  declarations: [  AppComponent,  TemplateFormComponent,  EmailDomainValidator  ],  bootstrap: [  AppComponent  ],  providers: [  {provide: 'RequiredDomain', useValue: 'example.com'}  ] }) **class** AppModule { }  platformBrowserDynamic().bootstrapModule(AppModule); |