



R a i s i n g t h e b a r

Angular Component & Template

Mục tiêu

- Tạo được các component tùy biến.
- Truyền được dữ liệu vào component.
- Giao tiếp được giữa các component.
- Sử dụng được các directive trong template.
- Sử dụng được style cho template.

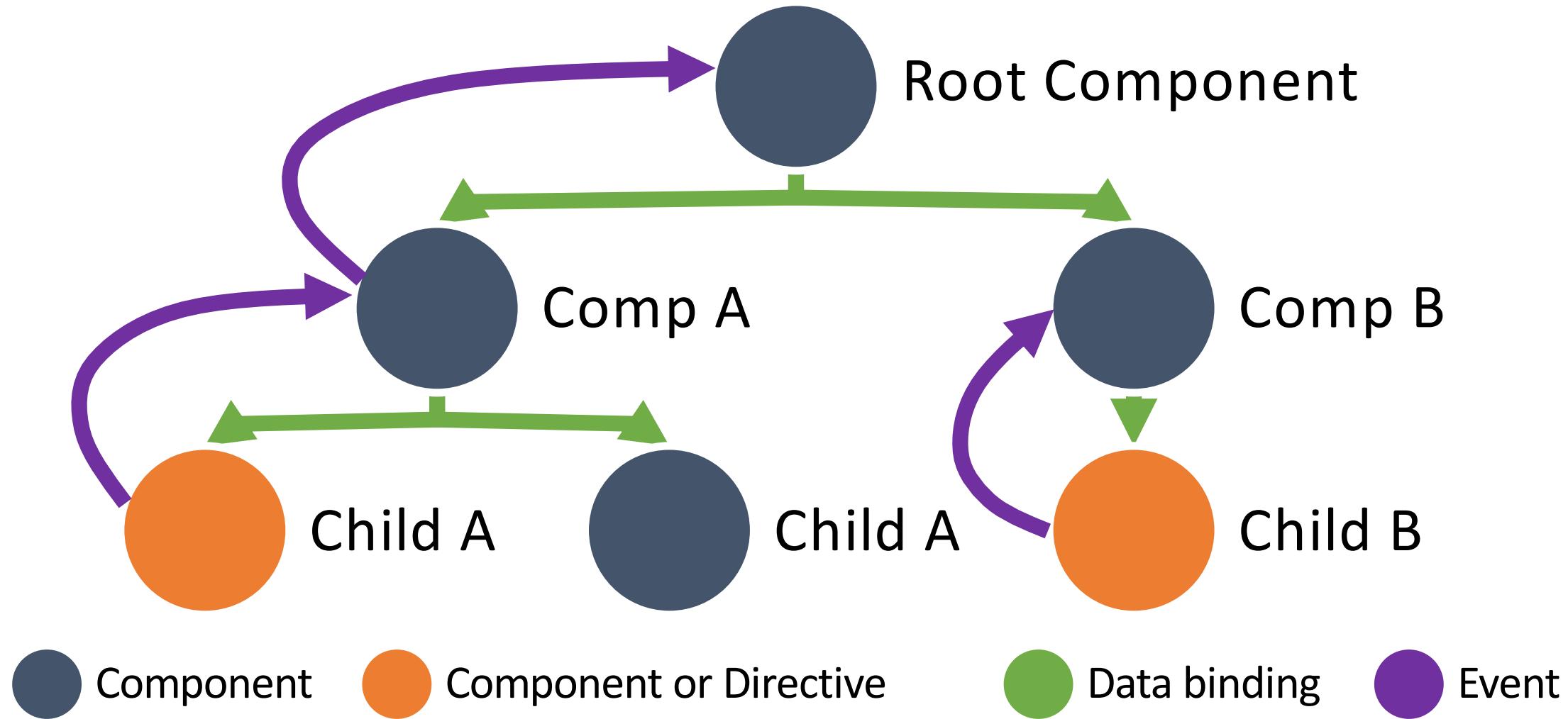
Project mẫu

- Project mẫu tại folder: angular-component-template
- Hoặc bạn có thể tạo mới project với câu lệnh:

```
ng new angular-component-template --style scss
```

Tương tác giữa các component

Component interactions

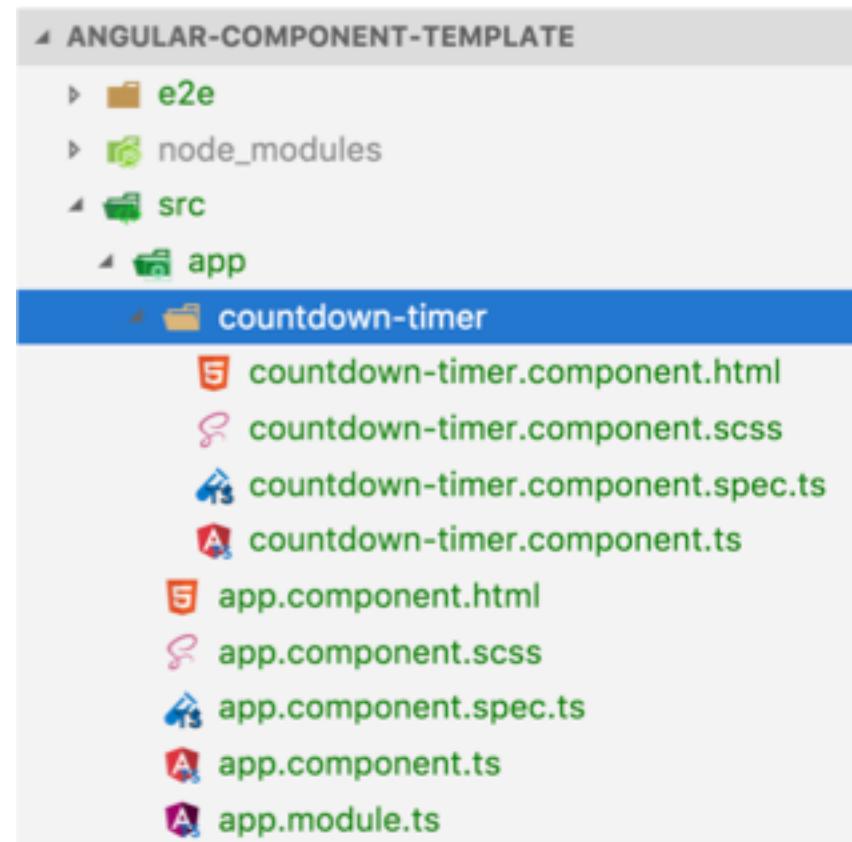


Truyền dữ liệu với @Input

- Component cha có thể truyền dữ liệu vào cho component con bằng cách sử dụng @Input.
- Chuẩn hóa dữ liệu đầu vào với get/set, ngOnChanges.
- Tạo alias cho @Input property.

Truyền dữ liệu với @Input

ng generate component countdown-timer



Truyền dữ liệu với @Input

```
import { Component, Input, OnDestroy, OnInit } from '@angular/core';

@Component({
  selector: 'app-countdown-timer',
  templateUrl: './countdown-timer.component.html',
  styleUrls: ['./countdown-timer.component.scss']
})
export class CountdownTimerComponent implements OnInit, OnDestroy {

  @Input()
  seconds = 11;
}
```

Decorator @Input để đánh dấu property **seconds** sẽ là input mà component này cần nhận dữ liệu.
Property **seconds** có thể được thiết lập giá trị mặc định, nếu người dùng không truyền vào giá trị nào.

Truyền dữ liệu với @Input

- Giả sử chúng ta tạo component để hiển thị Countdown
- Component này cho phép người dùng truyền vào thời gian cần đếm ngược.
- Nếu người dùng không truyền vào thời gian cần đếm, chúng ta có thể thiết lập một giá trị mặc định.

Code thực thi của component

```
export class CountdownTimerComponent implements OnInit,  
OnDestroy {  
    private intervalId = 0;  
    message = '';  
    remainingTime: number;  
  
    @Input()  
    seconds = 11;  
    // ...  
}
```

Code thực thi của component

```
export class CountdownTimerComponent implements OnInit,  
OnDestroy {  
    // ...  
    ngOnInit() {}  
    ngOnDestroy() {}  
    clearTimer() {}  
    start() {}  
    stop() {}  
    reset() {}  
    private countDown() {}  
}
```



Angular component lifecycle, các hàm này sẽ được Angular tự động gọi khi khởi tạo(ngOnInit) và khi hủy(ngOnDestroy) component.

Template của component

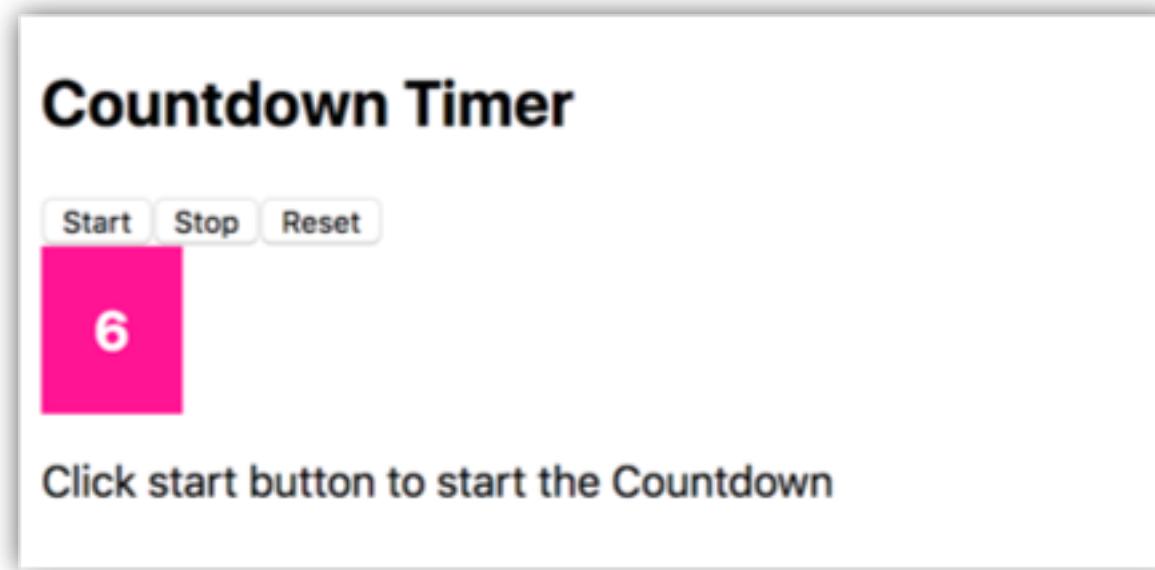
```
<h2>Countdown Timer</h2>
<div class="group-actions">
  <button (click)="start()">Start</button>
  <button (click)="stop()">Stop</button>
  <button (click)="reset()">Reset</button>
</div>
<div class="timer">
  <span class="timer-text">{{ remainingTime }}</span>
</div>
<p>{{ message }}</p>
```

Template của App component (parent)

```
<app-countdown-timer [seconds]="6">  
</app-countdown-timer>
```

Truyền dữ liệu cho component con

Kết quả



Chuẩn hóa dữ liệu đầu vào

- Sẽ thế nào nếu người dùng truyền vào **seconds** là một kiểu dữ liệu khác **number**?

The screenshot shows a web application titled "Countdown Timer". At the top, there are three buttons: "Start", "Stop", and "Reset". Below them, a large pink rectangular area displays the text "NaN". Underneath this area, the text "T-NaN seconds and counting" is visible. At the bottom of the screen, a browser's developer tools interface is shown, specifically the "Elements" tab. It displays the file structure: "src > app > app.component.html > ...". Below this, the HTML code for the component is shown:

```
1 <app-countdown-timer seconds="a string">
2 </app-countdown-timer>
```

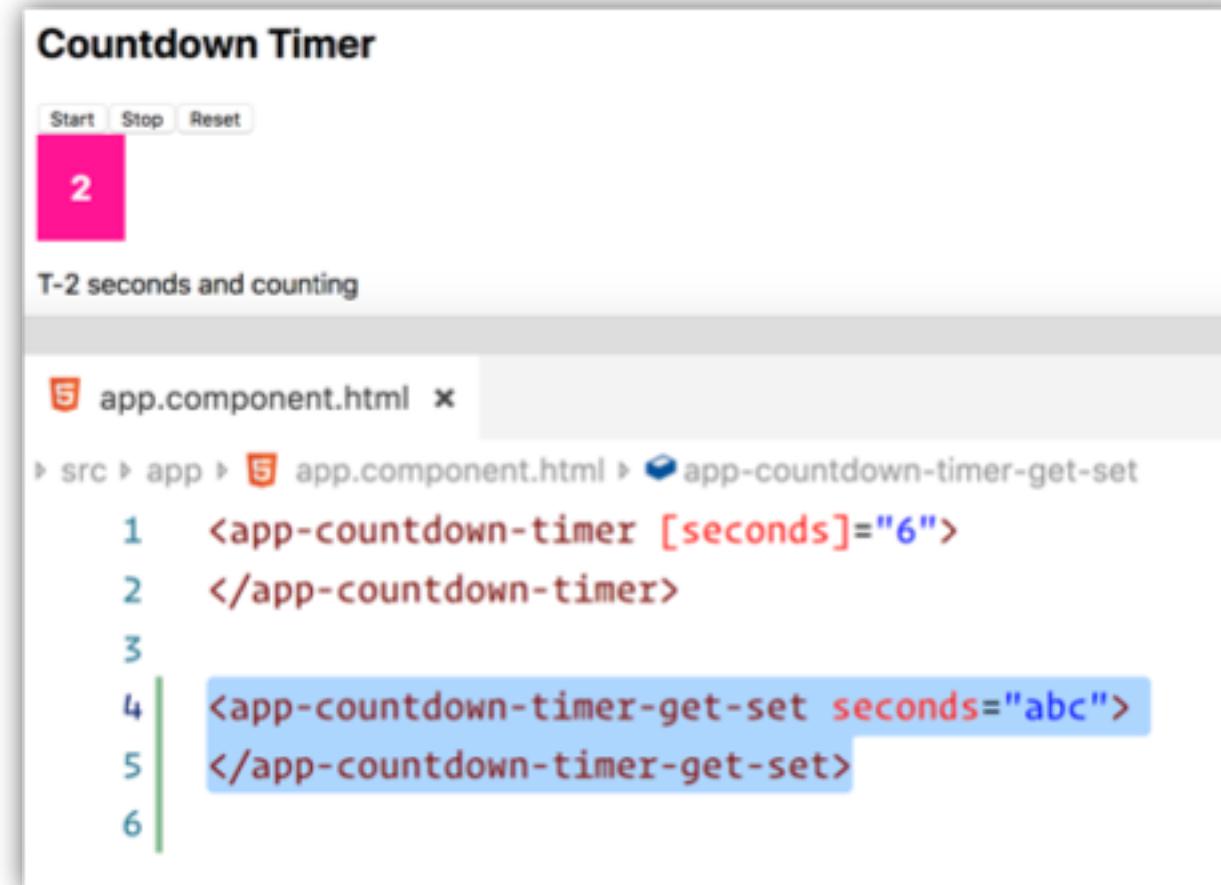
Chuẩn hóa dữ liệu đầu vào

- Generate component mới:
 - ng generate component countdown-timer-get-set
- Copy các phần code của countdown-timer component sang component vừa tạo.
- Update logic để sử dụng get/set cho @Input.

Chuẩn hóa dữ liệu đầu vào

```
export class CountdownTimerGetSetComponent {  
    private _seconds = 11;  
  
    @Input()  
    get seconds(): number {  
        return this._seconds;  
    }  
  
    set seconds(v) {  
        v = typeof v === 'undefined' ? 11 : v;  
        const vFixed = Number(v);  
        this._seconds = Number.isNaN(vFixed) ? 11 : vFixed;  
    }  
}
```

Chuẩn hóa dữ liệu đầu vào



The screenshot shows a web application titled "Countdown Timer". At the top, there are three buttons: "Start", "Stop", and "Reset". Below them, a large pink square displays the number "2". Underneath the square, the text "T-2 seconds and counting" is visible. At the bottom, a code editor window displays the file "app.component.html". The code is as follows:

```
<app-countdown-timer [seconds]="6">
</app-countdown-timer>
<app-countdown-timer-get-set seconds="abc">
</app-countdown-timer-get-set>
```

The line "seconds='abc'" is highlighted with a blue background.

Chuẩn hóa dữ liệu đầu vào

- Generate component mới:
 - ng generate component countdown-timer-onchanges
- Copy các phần code của countdown-timer component sang component vừa tạo.
- Update logic để sử dụng lifecycle hook (ngOnChanges) cho @Input.

Chuẩn hóa dữ liệu đầu vào

```
export class CountdownTimerOnchangesComponent implements OnChanges {  
  @Input()  
  seconds = 11;  
  ngOnChanges(changes: SimpleChanges) {  
    if ('seconds' in changes) {  
      let v = changes.seconds.currentValue;  
      v = typeof v === 'undefined' ? 11 : v;  
      const vFixed = Number(v);  
      this.seconds = Number.isNaN(vFixed) ? 11 : vFixed;  
    }  
  }  
}
```

Angular component lifecycle, hàm này sẽ được Angular tự động gọi mỗi khi có một @Input property bị thay đổi.

Chuẩn hóa dữ liệu đầu vào

Countdown Timer

Start Stop Reset

2

T-2 seconds and counting

app.component.html

```
src > app > app.component.html > app-countdown-timer-onchanges
7   <app-countdown-timer-onchanges seconds="str">
8   </app-countdown-timer-onchanges>
```

Tạo alias cho @Input

- Generate component mới:
 - ng generate component countdown-timer-alias
- Copy các phần code của countdown-timer-onchanges component sang component vừa tạo.
- Tạo alias cho @Input.

Tạo alias cho @Input

```
export class CountdownTimerAliasComponent {  
  @Input('remaining-time')  
  seconds = 11;  
}
```

Tên gọi của property **seconds** mà component cha sẽ
sử dụng để truyền dữ liệu cho component này

```
<app-countdown-timer-alias remaining-time="60">  
</app-countdown-timer-alias>
```

Component Event với @Output

- Component con gửi dữ liệu cho component cha thông qua Event với @Output.
- Sử dụng alias cho @Output property.

Component Event với @Output

- Generate component mới:
 - ng generate component countdown-timer-event
- Copy các phần code của countdown-timer-onchanges component sang component vừa tạo.
- Tạo Event cho component vừa tạo để thông báo khi nào quá trình đếm ngược kết thúc.
- Ở component cha thực hiện hiển thị thông báo cho người dùng biết quá trình đếm ngược đã kết thúc.

Component Event với @Output

```
export class CountdownTimerEventComponent {  
  @Output()  
  finish = new EventEmitter<boolean>();  
  private countDown() {  
    // other code  
    if (this.remainingTime === 0) {  
      this.finish.emit(true);  
    }  
  }  
}
```

Component Event với @Output

app.component.html

```
<app-countdown-timer-event  
    seconds="10"  
    (finish)="finishCountdown()">  
</app-countdown-timer-event>  
<p>{{ countdownMsg }}</p>
```

app.component.ts

```
countdownMsg = '';  
finishCountdown() {  
    this.countdownMsg = 'Finished!';  
}
```

Component Event với @Output

- Generate component mới:
 - ng generate component countdown-timer-event-alias
- Copy các phần code của countdown-timer-event component sang component vừa tạo.
- Tạo Event alias.

Component Event với @Output

```
export class CountdownTimerEventAliasComponent {  
  @Output('timerEnd')  
  finish = new EventEmitter<boolean>();  
}
```

Component Event với @Output

app.component.html

```
<app-countdown-timer-event-alias  
    seconds="10"  
    (timerEnd)="endCountdown()">  
</app-countdown-timer-event-alias>  
<p>{{ countdownAliasMsg }}</p>
```

app.component.ts

```
countdownAliasMsg = '';  
endCountdown() {  
    this.countdownAliasMsg = 'Ended!';  
}
```

Angular built-in directives

Chuẩn bị

- Generate component mới:
 - generate component rating-bar

Structure directives - ngIf

```
<div *ngIf="condition">...</div>
<ng-template [ngIf]="condition">
  <div>...</div>
</ng-template>
```

Structure directives - `ngIf`

```
<div *ngIf="condition; then thenBlock else elseBlock"></div>
<ng-template #thenBlock>...</ng-template>
<ng-template #elseBlock>...</ng-template>
```

```
<div *ngIf="condition as value; else elseBlock">
  {{value}}
</div>
<ng-template #elseBlock>...</ng-template>
```

Structure directives - ngForOf

```
<div *ngFor="let item of items; index as i; trackBy:  
trackByFn">...</div>
```

```
<ng-template ngFor let-item [ngForOf]="items" let-i="index"  
[ngForTrackBy]="trackByFn">  
  <div>...</div>  
</ng-template>
```

Rating Bar component

```
interface IRatingUnit {  
    value: number;  
    active: boolean;  
}
```

Rating Bar component

```
export class RatingBarComponent implements OnInit,  
OnChanges {  
    @Input() max = 5;  
    @Input() ratingValue = 5;  
    @Input() showRatingValue = true;  
  
    ratingUnits: Array<IRatingUnit> = [];  
}
```

Rating Bar component

```
export class RatingBarComponent implements OnInit, OnChanges {
  ngOnChanges(changes: SimpleChanges) {
    if ('max' in changes) {
      let max = changes.max.currentValue;
      max = typeof max === 'undefined' ? 5 : max;
      this.max = max;
      this.calculate(max, this.ratingValue);
    }
  }
}
```

Rating Bar component

```
export class RatingBarComponent implements OnInit, OnChanges {
  calculate(max, ratingValue) {
    this.ratingUnits = Array.from({length: max},
      (_, index) => ({
        value: index + 1,
        active: index < ratingValue
      }));
  }
  ngOnInit() {
    this.calculate(this.max, this.ratingValue);
  }
}
```

Rating Bar component

```
<h2>Rating Bar</h2>
<div *ngIf="showRatingValue">
  {{ ratingValue }}
</div>

<div class="rating-bar">
  <div class="rating-unit" *ngFor="let item of ratingUnits">
    {{ item.value }}
  </div>
</div>
```

Rating Bar component



Attribute directives – class/ngClass

```
<some-element [class.class-name]="expression"></some-element>
```

```
<some-element [ngClass]="'first second'">...</some-element>
```

```
<some-element [ngClass]=["'first', 'second']">...</some-element>
```

```
<some-element [ngClass]="{{'first': true, 'second': true, 'third':  
false}}">...</some-element>
```

```
<some-element [ngClass]="stringExp|arrayExp|objExp">...</some-element>
```

```
<some-element [ngClass]="{{'class1 class2 class3' : true}}>
```

```
...
```

```
</some-element>
```

Attribute directives – style/ngStyle

```
<some-element [style.style-property]="expression"></some-element>
<some-element [style.styleProperty]="expression"></some-element>
<some-element [style.style-property.unit]="expression"></some-element>
<some-element [style.styleProperty.unit]="expression"></some-element>

<some-element [ngStyle]="{{'font-style': styleExp}}">...</some-element>
<some-element [ngStyle]="{{'max-width.px': widthExp}}">...</some-element>
<some-element [ngStyle]="objExp">...</some-element>
```

Rating Bar component – updated

```
export class RatingBarComponent implements OnInit,  
OnChanges {  
  @Output()  
  rateChange = new EventEmitter<number>();  
  
}
```

Rating Bar component – updated

```
export class RatingBarComponent implements OnInit, OnChanges {
  select(index) {
    this.ratingValue = index + 1;
    this.ratingUnits.forEach((item, idx) =>
      item.active = idx < this.ratingValue);
    this.rateChange.emit(this.ratingValue);
  }
}
```

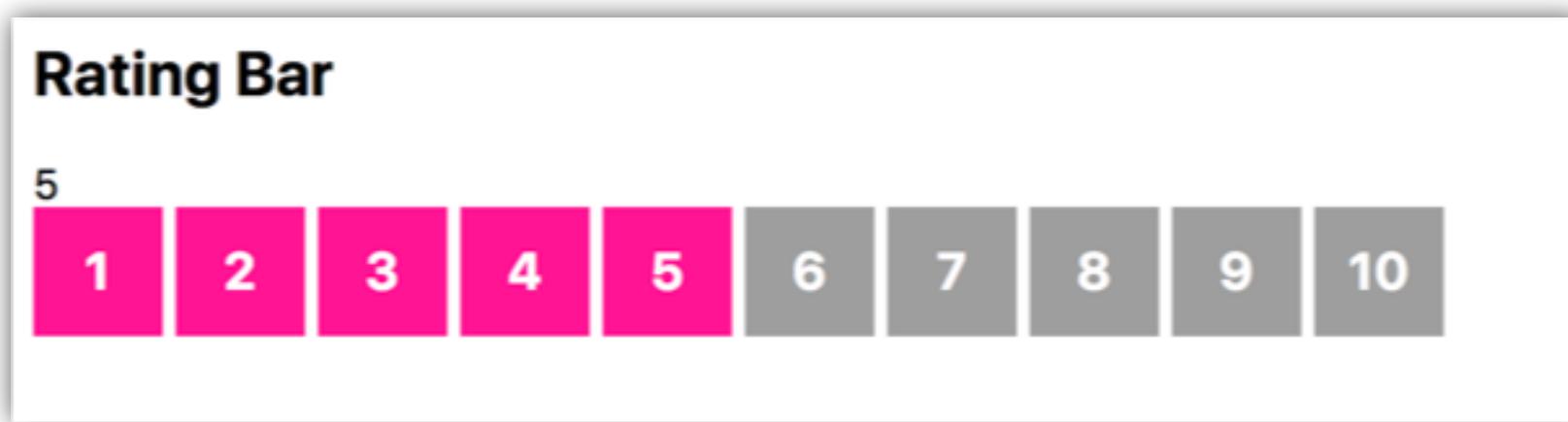
Rating Bar component – updated

```
export class RatingBarComponent implements OnInit, OnChanges {
  enter(index) {
    this.ratingUnits.forEach((item, idx) =>
      item.active = idx <= index);
  }
  reset() {
    this.ratingUnits.forEach((item, idx) =>
      item.active = idx < this.ratingValue);
  }
}
```

Rating Bar component – updated

```
<div class="rating-bar" (mouseleave)="reset()">
  <div class="rating-unit"
    [class.active]="item.active"
    *ngFor="let item of ratingUnits; index as i"
    (click)="select(i)" (mouseenter)="enter(i)">
    {{ item.value }}
  </div>
</div>
```

Rating Bar component – updated



Angular Pipes

Chuẩn bị

- Generate component mới:
 - ng generate component pipe-demo

Built-in Pipes

- AsyncPipe
- CurrencyPipe
- DatePipe
- DecimalPipe
- JsonPipe
- LowerCasePipe
- PercentPipe
- SlicePipe
- TitleCasePipe
- UpperCasePipe
- ...

Sử dụng Pipe

pipe-demo.component.html

```
<h2>Pipes</h2>
<p>
Today: {{ today | date:'fullDate' }}
</p>
<p>
Username: {{user.name | titlecase}}
</p>
```

pipe-demo.component.ts

```
today = new Date();
user = {
  name: 'bob mike'
};
```

Pipes

Today: Tuesday, September 4, 2018
Username: Bob Mike

Angular component style

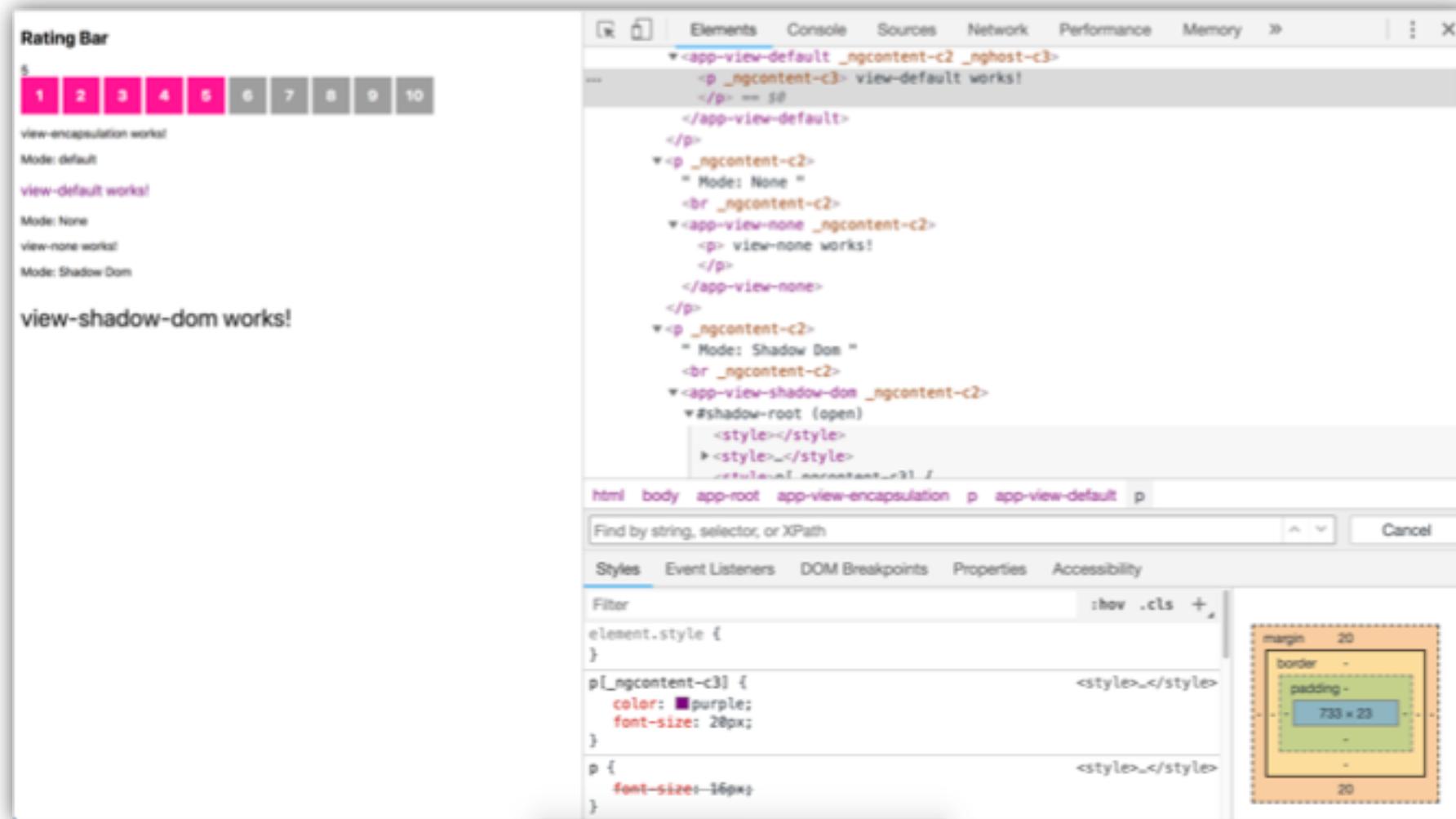
Chuẩn bị

- Generate component mới:
 - ng generate component view-encapsulation
 - ng generate component view-default
 - ng generate component view-shadow-dom
 - ng generate component view-none

Scope style

- Global style (styles.scss/styles.css/etc): ảnh hưởng chung đến toàn bộ app.
- Component style – View Encapsulation:
 - Emulated – default: đóng gói thành scope cho component đó, không bị leak style dẫn đến ảnh hưởng tới component khác.
 - None: không đóng gói gì cả, style trong component sẽ trở thành global style và ảnh hưởng chung đến toàn bộ app.
 - ShadowDom (Angular v6.1+)/Native (deprecated từ Angular v6.1+): sử dụng cơ chế Shadow Dom để đóng gói, không bị leak style sang component khác.

Kết quả khi apply scope style



Link tham khảo

- <https://angular.io/guide/component-interaction>
- <https://angular.io/guide/lifecycle-hooks>
- <https://angular.io/guide/component-styles>
- <https://www.tiepphan.com/thu-nghiem-voi-angular-2-truyen-du-lieu-cho-component-voi-input/>
- <https://www.tiepphan.com/thu-nghiem-voi-angular-2-component-event-voi-eventemitter-output/>
- <https://www.tiepphan.com/thu-nghiem-voi-angular-pipe-trong-angular/>

CODEGYM

CODEGYM

R a i s i n g t h e b a r