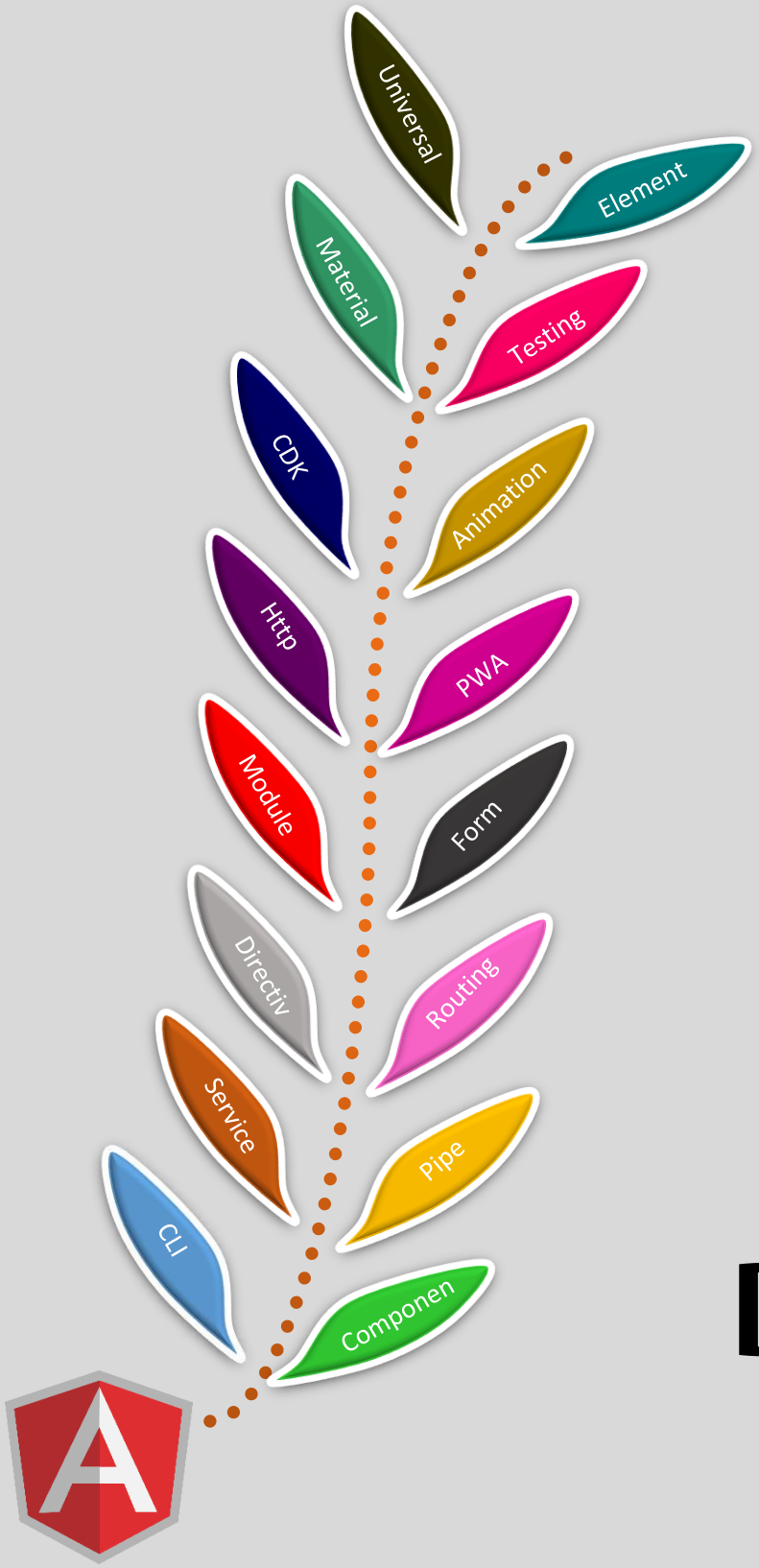
سلسلة تعلم Angular بالعربي



3

Angular Pipes And Directives

فيصل الفهد

Angular Pipes and Directives

الكتاب الثالث من

سلسلة تعلم Angular بالعربي

تم انتاج هذا العمل في عام

٢٠٢٠

المؤلف

فيصل الفهد

<u>إهداء</u>

إلى أطهر قلبين في حياتي... والديَّ العزيزين.

إلى من شاركني السرَّاء والضراء، ولم أرها عابسة يومًا..... زوجتي المخلصة.

إلى من أتشوَّق لأن أرى مستقبلهما المشرق بإذن الله..... ابنائي وبناتي.

إلى جميع متلهف للعلم ويتوق للمعرفة

أهديكم هذا الكتاب المتواضع، وأدعو الله أن يحوز إعجابكم.

المؤلف

مقدمة السلسلة

اللهم علمنا ما ينفعنا وانفعنا بما علمتنا إنك انت العليم الحكيم..

أضع بين إياديكم أحبتي وأخوتي مطوري الويب وبالتحديد مطوري Front End أول سلسلة عربية تتكلم عن إطار عمل Angular، حيث تقوم فكرة هذه السلسلة على حصر جميع الميزات التي يُقدمها Angular ومن ثم في كل كتاب يتم أخذ ميزة او ميزتين والإبحار فيها بشكل مستقل محاولاً بذلك تغطية أغلب وأهم جوانبها مع الشرح المفصل والأمثلة المتعددة.

ومن هذا المنطبق يجب التنويه أن هذه السلسلة ليست للمبتدئين في البرمجة، وانما لابد ان تمتلك عزيزي المتعلم على الأقل أساسيات أركان تطوير الويب الثلاثة HTML-CSS-JavaScript ومن ثم لابد ان تمتلك اساسيات Typescript لأن إطار عمل Angular يعتمد على هذه التقنية، ولا يخفى على كل مطور واجهات أمامية أهمية Typescript والتي من وجهة نظري أرى انه يجب على كل مطور ويب تعلمها واتقانها، ونكتفي ان نقول ان Deno.js وهي اللغة الجديدة من Node.js أصبحت تدعم هذه التقنية بشكل كامل، لذلك انصح كل مطور عربي بتعلم هذه التقنية ومحاولة الإلمام بجوانبها وكيفية الاستفادة منها في إطار عمل Angular بشكل خاص ولتطوير الويب بشكل عام.

ولا يخفى عليك عزيزي المتعلم أن الكمال لله ولا يخلوا عمل من الأخطاء فالخطأ وارد والتقصير موجود، لذلك في حال وجدت أي تقصير او أي خطأ في هذه السلسلة او أردت تقديم أي اقتراح لتطوير هذه السلسلة فلا تتردد بمراسلتي على البريد الإلكتروني الذي سوف اضعه بأسفل هذه الصفحة.

وأخيراً أسأل الله العلي العظيم أن ينفع به وان يتقبله عملاً خالصاً لوجهه سبحانه،

ولا تنسوني أخوتي من صالح دعائكم.

المؤلف

فيصل الفهد

Faisal.alfahd.ksa@gmail.com

جدول المحتويات

مقدمة الكتاب

بسم الله وصلى اللهم وسلم على نبينا محمد وعلى اله وصحبه أجمعين ...

أضع بين أيديكم أحبتي الكتاب الثالث من هذه السلسلة التي أسال المولى جل في علاه أن يجعلها في ميزان حسناتي ويكتب لي فيها الأجر لي ولوالدي ولجميع أحبابي.

اما محتوى الكتاب فهو مقسم إلى فصلين، الفصل الأول افردت الكلام فيه عن Pipes بداية في التعريف بها ومن ثم ذكر أنواعها وامثلة عليها.

اما الفصل الثاني فتم تخصيصه للحديث عن Directives بنوعيها Structure Directives وAttribute Directives مع التدعيم بالعديد من الأمثلة.

المؤلف

فيصل الفهد

Faisal.alfahd.ksa@gmail.com

الفصل الأول

Pipes

1.1. المقدمة:

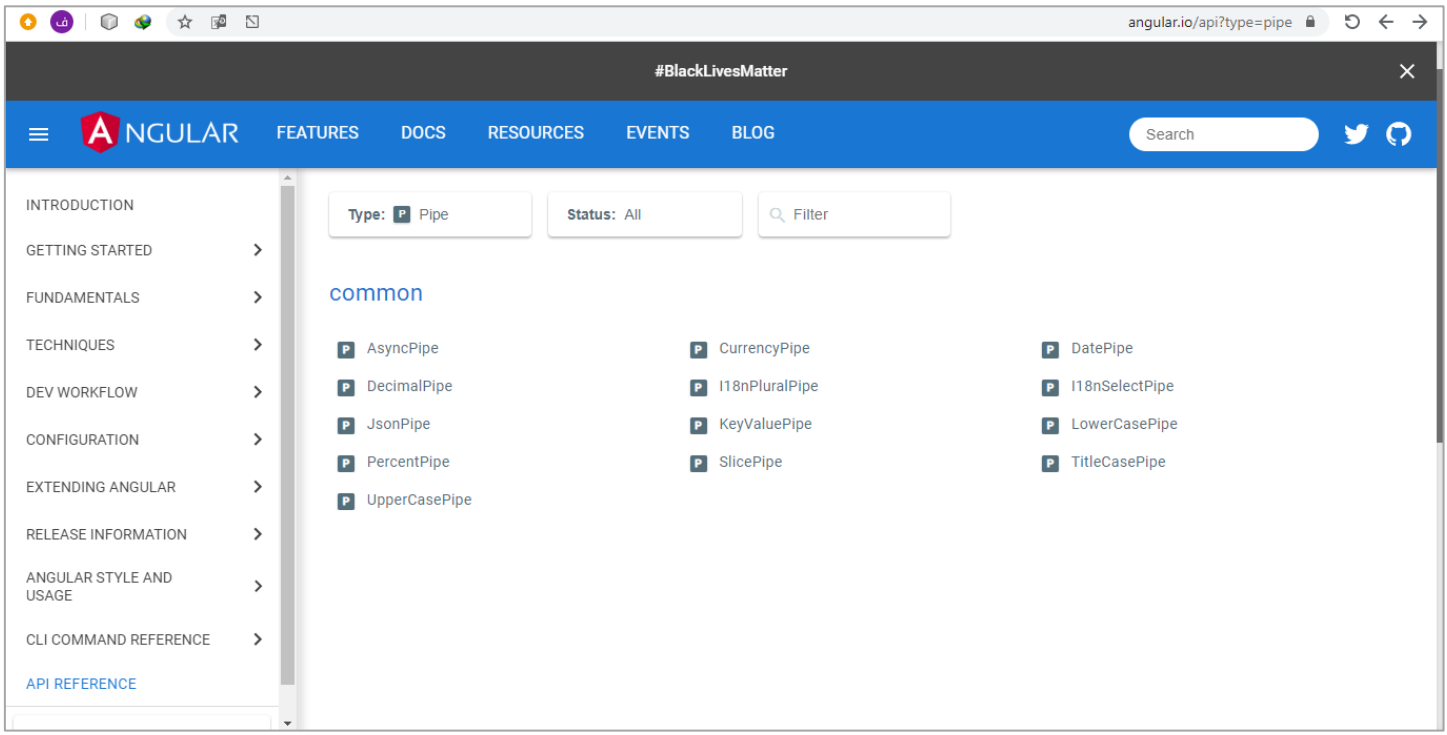
وهي عبارة عن دوال Functions يتم كتابتها في Template او بصيغة أخرى ملف HTML الخاص بالComponent لتنفيذ مهمة معينة (لمعرفة اكثر عن Components وطرق التعامل معها الرجاء مراجعة الكتاب الأول من هذه السلسلة Angular Components and Services)، وممكن أن تأخذ بارامترات واحياناً لا تأخذ أي بارامتر، كما ان Angular قدم لنا مجموعة من pipes الجاهزة مثل التعامل مع تنسيقات التاريخ والوقت والعملات والحروف الكبيرة والصغير او حتى لتعامل مع البيانات Async ...الخ، وبنفس الوقت يتيح لنا Angular أن نبني Pipes الخاصة بنا في حال احتجنا إلى ذلك.

مع العلم ان جميع هذه الPipes موجودة من ضمن Common Module، الذي يُضاف بشكل تلقائي لأي مشروع Angular، بمعنى آخر انها متوفرة بشكل تلقائي عند انشاء أي مشروع angular جديد ولا تحتاج إلى أي أضافات، ولمعرفة أكثر عن Modules الرجاء مراجعة الكتاب الرابع من هذه السلسلة Angular Routing and Modules.

2.1. Built-in Pipes:

وهي عبارة عن مجموعة من Pipes الجاهزة التي تقدمها لنا Angular، لتسهيل القيام ببعض المهام، فعلى سبيل المثال هنالك DatePipe لتعامل مع تنسيقات التاريخ والوقت المختلفة، وهنالك CurrencyPipe لتعامل مع تنسيقات العملة المختلفة، ...الخ.

ولاستعراض جميع هذه الأنواع عن طريق الرابط التالي: https://angular.io/api?type=pipe



وسوف استعرض أهمها وأكثرها استخداماً، مع العلم ان جميعها تعمل بنفس الآلية.

ولكن قبل استعراضها لابد ان ننشئ مشروع Angular جديد، ولمعرفة طريقة انشاء مشروع Angular جديد الرجاء مراجعة الكتاب الأول من هذه السلسلة Angular Environment Setup.

1.2.1. TitleCasePipe:

والصيغة العامة لها هي: **{{value_expression | titlecase}}**

حيث value_expression هو عبارة عن القطعة النصية المراد تطبيق هذا pipe عليها، وtitlecase هو عبارة عن pipe نفسه، ومهمته تحويل اول حرف من كل كلمة إلى حرف كبير.

ملف app.component.html

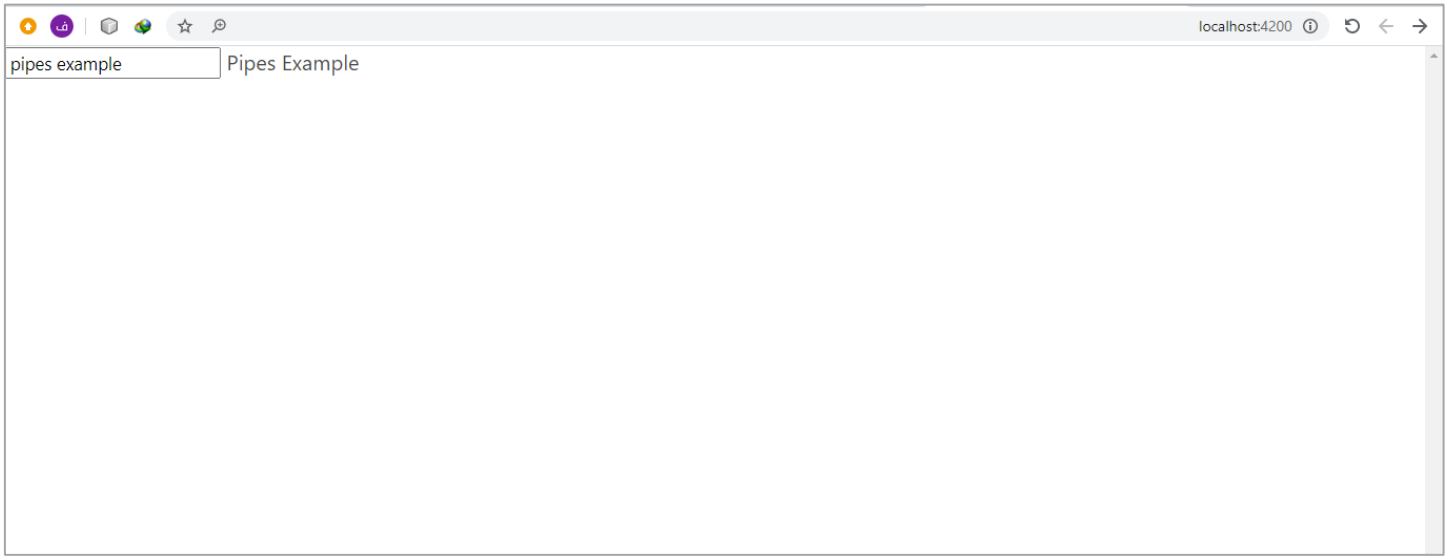
```
 <input type="text" (input)="onType($event.target.value)" />

 {{value | titlecase}}
```

وفي ملف class نجري التعديل التالي:

ملف app.component.ts

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

@Component({
   selector: 'app-root',
   templateUrl: './app.component.html',
   styleUrls: ['./app.component.css']
})

export class AppComponent {
  value = '';

  onType(value: string) {
    this.value = value;
  }

}
```

والنتيجة في المتصفح، كالتالي، (لكي نشاهد النتيجة لابد ان نعمل serve للمشروع لكي يعمل في المتصفح وذلك عن طريقة كتابة الأمر ng s -o ولمعرفة أكثر عن أوامر Angular CLI وطريقة التعامل معها الرجاء مراجعة الكتاب الأول من هذه السلسلة Angular Environment Setup).



1.2.2. DatePipe:

كما ذكرت سابقاً في المقدمة أن Pipes هي عبارة عن Function تستقبل مدخل وهي القيمة value وتقوم بإخراج مخرج بحسب نوع ومهمة هذا Pipe، وبنفس الوقت أشرنا ان بعض Pipes تستقبل بارامترات وبعضها لا تستقبل، فمثلاً في titlecase pipe تستقبل مدخل (قيمة) وهي النص وتخرج مُخرج وهو النص بعد تعديل اول حرف من كل كلمة ليصبح حرف كبير ولكنها لا تستقبل أي بارامتر، بعكس DatePipe التي تسمح بإضافة مجموعة من البارامترات الاختيارية بالإضافة إلى القيمة value.

أما الصيغة العامة لها فهي: **{{ value_expression | date [: format [: timezone [: locale]]] }}**

حيث ان date يمثل القيمة value.

وformat تمثل البارامترات الاختيارية التي ممكن ان نمررها إلى هذا Pipe، والتي عن طريقها نستطيع ان نتحكم بشكل وهيئة عرض التاريخ والوقت، مع العلم أن default هو mediumDate في حال عدم اختيار أي format، اما الآن سوف استعرض جميع هذه format، كما في الشكل التالي:

Examples are given in en-US locale.

- 'short': equivalent to 'M/d/yy, h:mm a' (6/15/15, 9:03 AM).
- 'medium': equivalent to 'MMM d, y, h:mm:ss a' (Jun 15, 2015, 9:03:01 AM).
- 'long': equivalent to 'MMMM d, y, h:mm:ss a z' (June 15, 2015 at 9:03:01 AM GMT+1).
- 'full': equivalent to 'EEEE, MMMM d, y, h:mm:ss a zzzz' (Monday, June 15, 2015 at 9:03:01 AM GMT+01:00).
- 'shortDate': equivalent to 'M/d/yy' (6/15/15).
- 'mediumDate': equivalent to 'MMM d, y' (Jun 15, 2015).
- 'longDate': equivalent to 'MMMM d, y' (June 15, 2015).
- 'fullDate': equivalent to 'EEEE, MMMM d, y' (Monday, June 15, 2015).
- 'shortTime': equivalent to 'h:mm a' (9:03 AM).
- 'mediumTime': equivalent to 'h:mm:ss a' (9:03:01 AM).
- 'longTime': equivalent to 'h:mm:ss a z' (9:03:01 AM GMT+1).
- 'fullTime': equivalent to 'h:mm:ss a zzzz' (9:03:01 AM GMT+01:00).

أما في حال أردنا ان نقوم ببناء هذه Optional Format بأنفسنا فنستطيع ذلك بمساعدة هذا الجدول، كالتالي:

Field type	Format	Description	Example Value
Era	G, GG & GGG	Abbreviated	AD
	GGGG	Wide	Anno Domini
	GGGGG	Narrow	A
Year	y	Numeric: minimum digits	2, 20, 201, 2017, 20173
	yy	Numeric: 2 digits + zero padded	02, 20, 01, 17, 73
	yyy	Numeric: 3 digits + zero padded	002, 020, 201, 2017, 20173
	yyyy	Numeric: 4 digits or more + zero padded	0002, 0020, 0201, 2017, 20173
Month	M	Numeric: 1 digit	9, 12
	MM	Numeric: 2 digits + zero padded	09, 12

Field type	Format	Description	Example Value
	MMM	Abbreviated	Sep
	MMMM	Wide	September
	MMMMM	Narrow	S
Month standalone	L	Numeric: 1 digit	9, 12
	LL	Numeric: 2 digits + zero padded	09, 12
	LLL	Abbreviated	Sep
	LLLL	Wide	September
	LLLLL	Narrow	S
Week of year	w	Numeric: minimum digits	1... 53
	ww	Numeric: 2 digits + zero padded	01... 53
Week of month	W	Numeric: 1 digit	1... 5
Day of month	d	Numeric: minimum digits	1
	dd	Numeric: 2 digits + zero padded	01
Week day	E, EE & EEE	Abbreviated	Tue
	EEEE	Wide	Tuesday
	EEEEE	Narrow	T
	EEEEEE	Short	Tu

Field type	Format	Description	Example Value
Period	a, aa & aaa	Abbreviated	am/pm or AM/PM
	aaaa	Wide (fallback to a when missing)	ante meridiem/post meridiem
	aaaaa	Narrow	a/p
Period*	B, BB & BBB	Abbreviated	mid.
	BBBB	Wide	am, pm, midnight, noon, morning, afternoon, evening, night
	BBBBB	Narrow	md
Period standalone*	b, bb & bbb	Abbreviated	mid.
	bbbb	Wide	am, pm, midnight, noon, morning, afternoon, evening, night
	bbbbb	Narrow	md
Hour 1-12	h	Numeric: minimum digits	1, 12
	hh	Numeric: 2 digits + zero padded	01, 12
Hour 0-23	H	Numeric: minimum digits	0, 23
	HH	Numeric: 2 digits + zero padded	00, 23
Minute	m	Numeric: minimum digits	8, 59
	mm	Numeric: 2 digits + zero padded	08, 59
Second	s	Numeric: minimum digits	0... 59
	ss	Numeric: 2 digits + zero padded	00... 59

Field type	Format	Description	Example Value
Fractional seconds	S	Numeric: 1 digit	0... 9
	SS	Numeric: 2 digits + zero padded	00... 99
	SSS	Numeric: 3 digits + zero padded (= milliseconds)	000... 999
Zone	z, zz & zzz	Short specific non location format (fallback to O)	GMT-8
	zzzz	Long specific non location format (fallback to OOOO)	GMT-08:00
	Z, ZZ & ZZZ	ISO8601 basic format	-0800
	ZZZZ	Long localized GMT format	GMT-8:00
	ZZZZZ	ISO8601 extended format + Z indicator for offset 0 (= XXXXX)	-08:00
	O, OO & OOO	Short localized GMT format	GMT-8
	OOOO	Long localized GMT format	GMT-08:00

ومن المهم ملاحظة أن local الافتراضي هو en-US، وفي حال اردنا تغيير إلى منطقة معينة لابد من الذهاب إلى هذا الموقع https://github.com/angular/angular/tree/master/packages/common/locales والبحث عن المنطقة التي تريدها وتحميل الملف الخاص بها.

الآن لنقوم بتطبيق ما تعلمناه على مثالنا حيث سوف نستخدم Custom Format Options، كالتالي:

app.component.html ملف

```
<div class="container">
    <h3>Payment Form</h3>
    <div class="row">
        <div class="form-group col">
            <label>Payee Name</label>
            <input
                type="text"
                class="form-control"
```

```
                (input)="onType($event.target.value)"
            />
        </div>
        <div class="col">
            <label><p>Name</p></label>
            <div>
                {{value | titlecase}}
            </div>
        </div>
    </div>
    <div class="row">
        <div class="form-group col">
            <label>Payment Date</label>
            <input
                type="date"
                class="form-control"
                (input)="onDateChange($event.target.value)"
            />
        </div>
        <div class="col">
            <label><p>Payment Date</p></label>
            <div>
                {{date| date:"MMMM d, y - h:m:s" : "en-SA"}}
            </div>
        </div>
    </div>
</div>
```

ملف app.component.ts

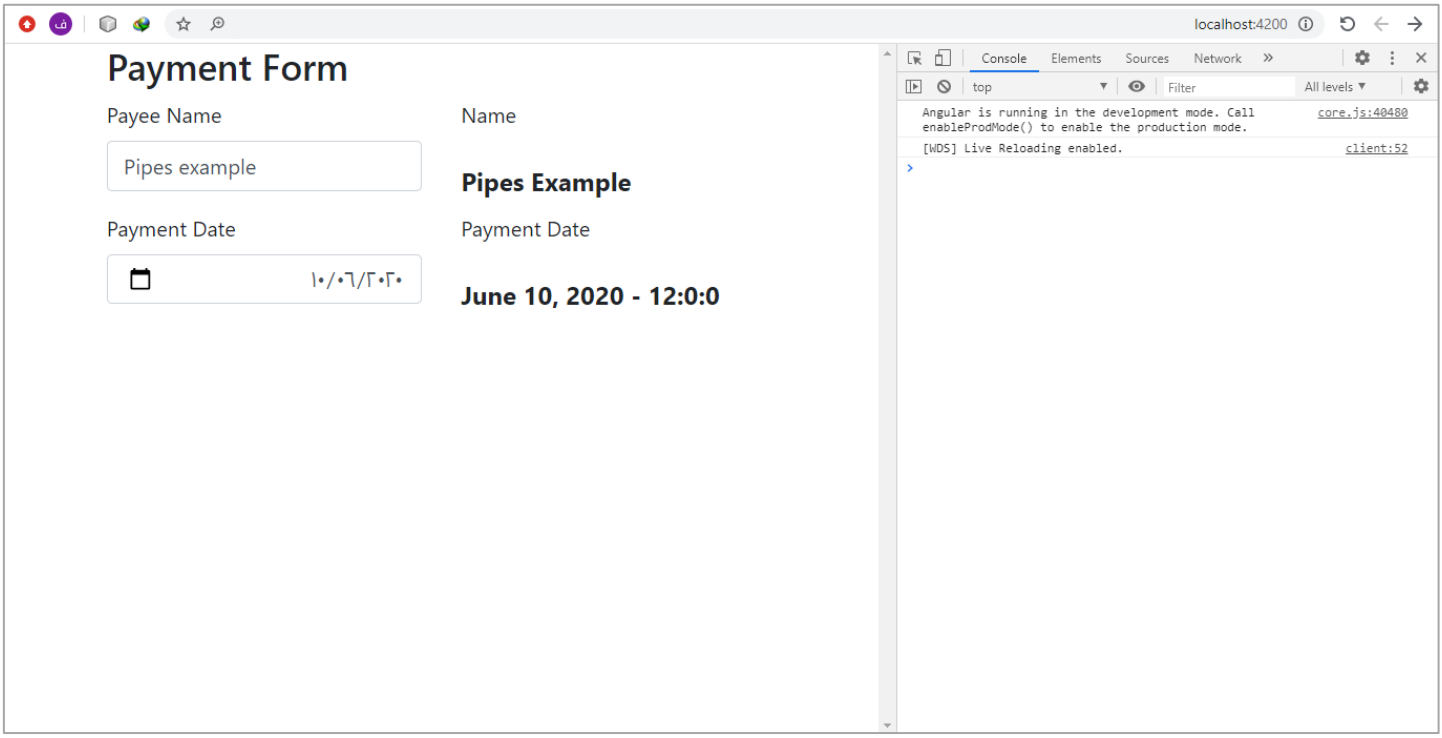
```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})

export class AppComponent {
  value = '';
  date = '';

  onType(value: string) {
    this.value = value;
  }
  onDateChange(value: string) {
    this.date = value;
  }

}
```

ملف styles.css

```
@import "bootstrap/dist/css/bootstrap.css";
```

ملف app.component.css

```
.result {
    font-weight: bolder;
    font-size: larger;
}
```



3.2.1. CurrencyPipe:

وهذا الPipe خاص بتنسيق الرقم على شكل عملة، وهو ايضاً نفس السابق يستقبل بارامتر.

والصيغة العامة له: **{{ value_expression | currency [: currencyCode [: display [: digitsInfo [: locale]]]] }}**

حيث currencyCode هو كود العملة الخاص بكل دولة ويمكن استعراض جميع هذه الأكواد على هذا الرابط:

https://en.wikipedia.org/wiki/ISO_4217

أما display فهو عبارة عن طريقة عرض هذا الكود (الرمز – الاسم – ...الخ) حيث تأخذ مجموعة من القيم كالتالي:

- code: ومعناها نعرض الكود الخاص بالعملة مثل SAR او USD...الخ.
- symbol : وهي القيمة الافتراضية، ومعناها اعرض الرمز وليس الكود (في حال كان له رمز) مثل $ او €.
- symbol-narrow: بعض العملات تأخذ بالإضافة الى الرمز والكود ايضاً لها رموز خاص، مثل الدولار الكندي CA$.
- String: حيث نستطيع عن طريقها كتابة نص معين من عندنا ليكون بديلاً لرمز او الكود، وفي حال كان فارغ سوف يلغي الرمز والكود، أما في حال تركه فارغ سوف يقوم بإلغاء أي رمز او كود مع إبقاء صيغة رقم العملة كما هو، مثال/ لو كان لدينا الرقم التالي 25000 ونريد تحويله إلى رقم عملة، فلو كتبنا "ريال سعودي" فسوف يكون الرقم ريال سعودي ٢٥,٠٠٠,٠٠، اما لو تركناه فارغ فسوف يكون الرقم ٢٥,٠٠٠,٠٠.

أما digitsInfo فهي عدد الخانات الصحيحة والعشرية للرقم، فمثلاً لو كان الرقم 35.12345، فلو جعلنا قيمة digitsInfo هي 4.3 فأنه سيجعل للرقم أربع خانات صحيحة وثلاث عشرية ليصبح الرقم بهذه الطريقة 0035.123.

أما locale فقد أشرنا لها سابقاً وتشير إلى المنطقة الموجود فيها والقيمة الافتراضية لها هنا هي undefined بعكس Date Pipe.

وفي المثال التالي نستعرض مجموعة من الحالات التي نستخدم فيها هذا pipe، كالتالي:

```
@Component({
  selector: 'currency-pipe',
  template: `<div>
    <!--output '$0.26'-->
    <p>A: {{a | currency}}</p>

    <!--output 'CA$0.26'-->
    <p>A: {{a | currency:'CAD'}}</p>

    <!--output 'CAD0.26'-->
    <p>A: {{a | currency:'CAD':'code'}}</p>

    <!--output 'CA$0,001.35'-->
    <p>B: {{b | currency:'CAD':'symbol':'4.2-2'}}</p>

    <!--output '$0,001.35'-->
    <p>B: {{b | currency:'CAD':'symbol-narrow':'4.2-2'}}</p>

    <!--output '0 001,35 CA$'-->
    <p>B: {{b | currency:'CAD':'symbol':'4.2-2':'fr'}}</p>

    <!--output 'CLP1' because CLP has no cents-->
    <p>B: {{b | currency:'CLP'}}</p>
  </div>`
})
export class CurrencyPipeComponent {
  a: number = 0.259;
  b: number = 1.3495;
}
```

اما الان لنطبق ما تعلمناه على المثال الرئيسي، كالتالي:

ملف app.component.html

```
<div class="container">
    <h3>Payment Form</h3>
    <div class="row">
        <div class="form-group col">
            <label>Payee Name</label>
            <input
               type="text"
               class="form-control"
               (input)="onType($event.target.value)"
            />
        </div>
        <div class="col">
            <label><p>Name</p></label>
            <div class="result"> {{value | titlecase}} </div>
        </div>
    </div>
    <div class="row">
        <div class="form-group col">
            <label>Payment Date</label>
            <input
               type="date"
```

```
                class="form-control"
                (input)="onDateChange($event.target.value)"
            />
        </div>
        <div class="col">
            <label><p>Payment Date</p></label>
            <div class="result">
                {{date| date:"MMMM d, y - h:m:s" : "en-SA"}}
            </div>
        </div>
    </div>
    <div class="row">
        <div class="form-group col">
            <label>Payment Amount</label>
            <input
                type="text"
                class="form-control"
                (input)="onAmountChange($event.target.value)"
            />
        </div>
        <div class="col">
            <label><p>Payment Amount</p></label>
            <div class="result">
                {{ amount|currency: "SAR" : "symbol" : "3.3" }}
            </div>
        </div>
    </div>
</div>
```

ملف app.component.ts

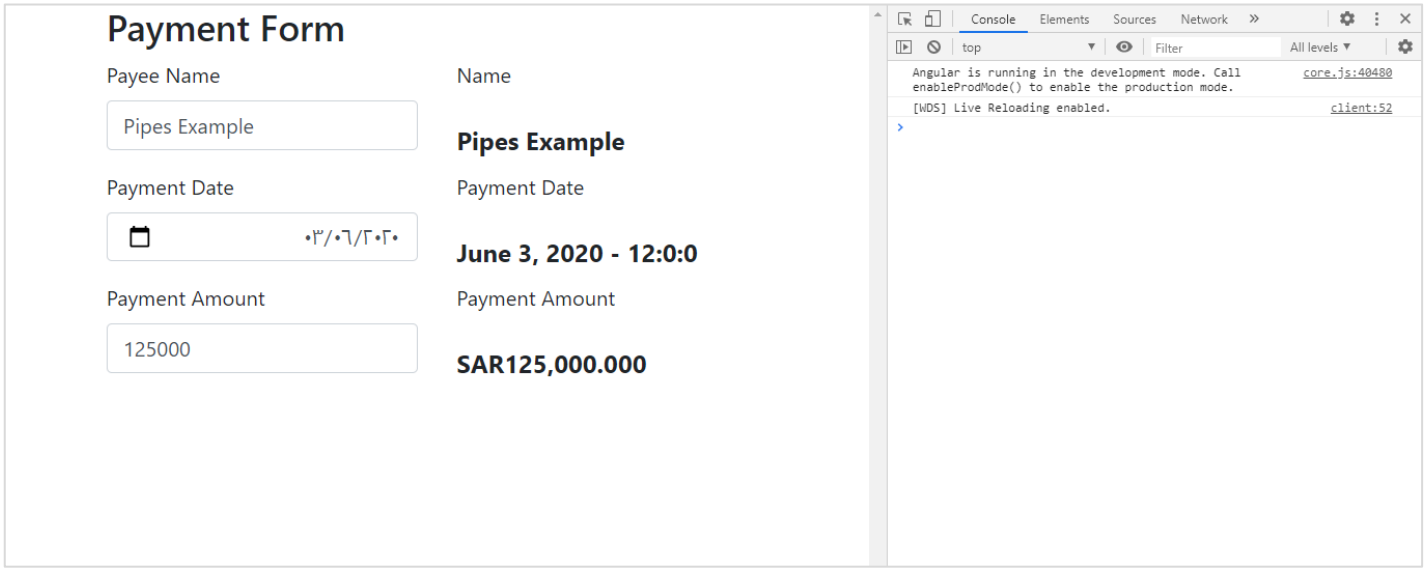
```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})

export class AppComponent {

  value: string;
  date: string;
  amount: number;


  onType(value: string) {
    this.value = value;
  }

  onDateChange(value: string) {
    this.date = value;
  }
```

```
  onAmountChange(value: string) {
    this.amount = +value;
  }

}
```

ملف app.component.css

```
.result {
    font-weight: bolder;
    font-size: larger;
}
```

اما النتيجة في المتصفح، فتكون كالتالي:



4.2.1. DecimalPipe:

وهذا الPipe خاص بالتعامل مع الأرقام من حيث تحديد عدد خانات الاعداد الصحيحة او العشرية.

والصيغة العامة له: **{{ value_expression | number [: digitsInfo [: locale]] }}**

ويمكن التحكم بعدد الخانات الصحيحة والعشرية من خلال كتابة الأمر التالي:

عدد خانات الاعداد الصحيحة	.	أقل عدد للخانات العشرية	-	اقصى عدد للخانات العشرية

فمثلاً لو كان لدينا العدد 24.10248399 واردنا تقريبه بحيث يكون خمس خانات عشرية كحد اقصى وخانتين كحد أدنى مع خانتين اعداد صحيحة، فنكتب الامر التالي 2.2-5 وتكون النتيجة 24.10249، وفي حال أردنا الاستغناء عن اقل عدد خانات والاكتفاء بأقصى عدد خانات عشرية فنكتب الامر 2.5 فقط.

وفي الشكل التالي مجموعة أخرى من الأمثلة، كالتالي:

Example

```
@Component({
  selector: 'number-pipe',
  template: `<div>
    <!--output '2.718'-->
    <p>e (no formatting): {{e | number}}</p>

    <!--output '002.71828'-->
    <p>e (3.1-5): {{e | number:'3.1-5'}}</p>

    <!--output '0,002.71828'-->
    <p>e (4.5-5): {{e | number:'4.5-5'}}</p>

    <!--output '0 002,71828'-->
    <p>e (french): {{e | number:'4.5-5':'fr'}}</p>

    <!--output '3.14'-->
    <p>pi (no formatting): {{pi | number}}</p>

    <!--output '003.14'-->
    <p>pi (3.1-5): {{pi | number:'3.1-5'}}</p>

    <!--output '003.14000'-->
    <p>pi (3.5-5): {{pi | number:'3.5-5'}}</p>

    <!--output '-3' / unlike '-2' by Math.round()-->
    <p>-2.5 (1.0-0): {{-2.5 | number:'1.0-0'}}</p>
  </div>`
})
export class NumberPipeComponent {
  pi: number = 3.14;
  e: number = 2.718281828459045;
}
```

الآن لنقوم بتطبيق ما تعلمناه على مثالنا الرئيسي كالتالي:

ملف app.component.html

```
<div class="container">
    <h3>Payment Form</h3>
    <div class="row">
        <div class="form-group col">
            <label>Payee Name</label>
            <input type="text" class="form-control" (input)="onType($event.target.value)" />
        </div>
        <div class="col">
            <label><p>Name</p></label>
            <div class="result">
                {{value | titlecase}}
            </div>
        </div>
    </div>
    <div class="row">
        <div class="form-group col">
            <label>Payment Date</label>
            <input
               type="date"
               class="form-control"
               (input)="onDateChange($event.target.value)"
            />
        </div>
```

```
        <div class="col">
            <label><p>Payment Date</p></label>
            <div class="result">
                {{date| date:"MMMM d, y - h:m:s" : "en-SA"}}
            </div>
        </div>
    </div>
    <div class="row">
        <div class="form-group col">
            <label>Payment Amount</label>
            <input
               type="text"
               class="form-control"
               (input)="onAmountChange($event.target.value)"
            />
        </div>
        <div class="col">
            <label><p>Payment Amount</p></label>
            <div class="result">
                {{ amount|currency: "SAR" : "symbol" : "3.3" }}
            </div>
        </div>
    </div>
    <div class="row">
        <div class="form-group col">
            <label>Height</label>
            <input
               type="text"
               class="form-control"
               (input)="onheightChange($event.target.value)"
            />
        </div>
        <div class="col">
            <label><p>Height</p></label>
            <div class="result">
                {{ height | number: "2.2-5" }}
            </div>
        </div>
    </div>
</div>
```

ملف app.component.ts

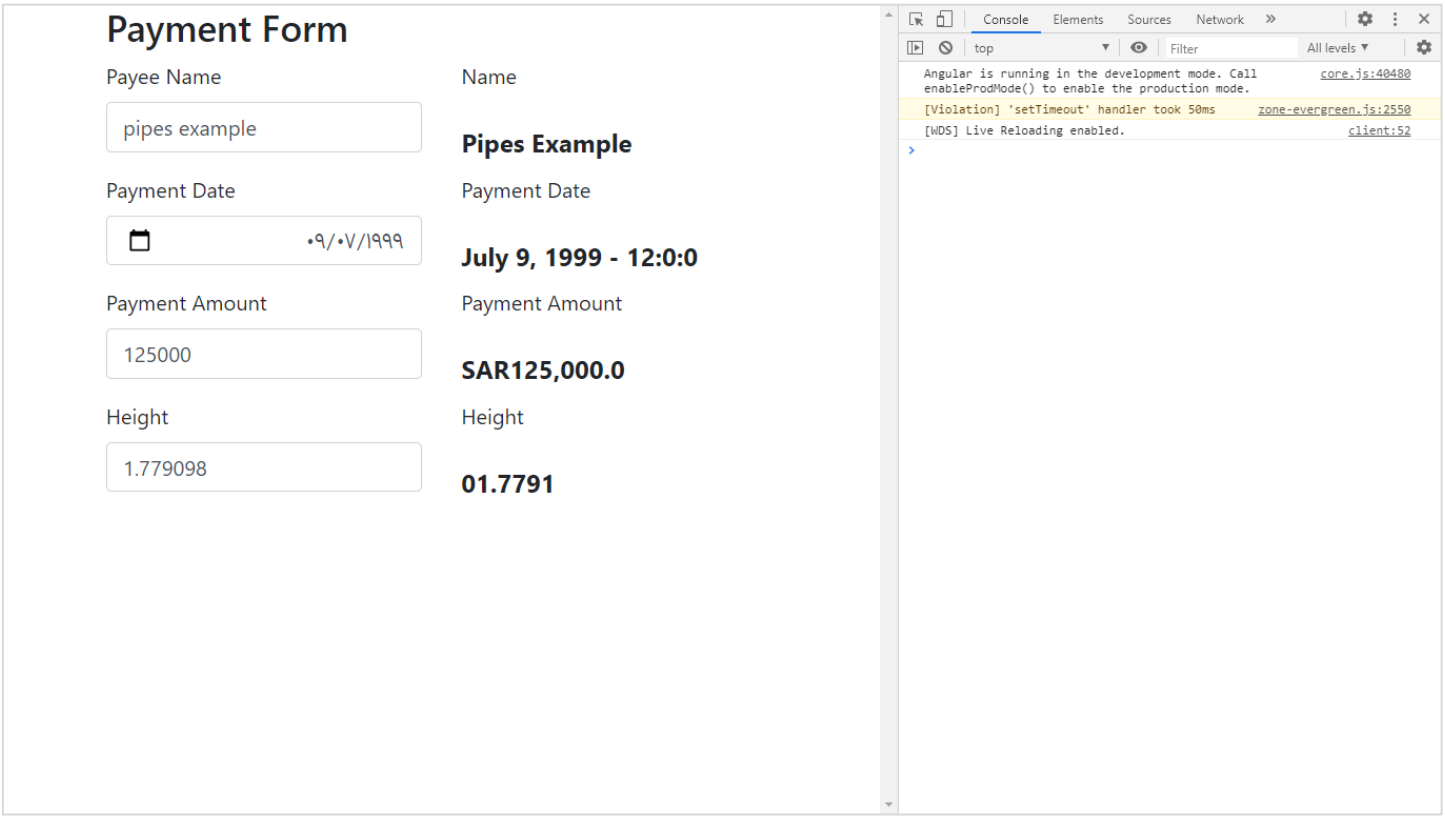
```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {

  value: string;
  date: string;
```

```
  amount: number;
  height: number;

  onType(value: string) {
    this.value = value;
  }
  onDateChange(value: string) {
    this.date = value;
  }
  onAmountChange(value: string) {
    this.amount = +value;
  }
  onheightChange(value: string) {
    this.height = +value;
  }

}
```

ملف app.component.css

```
.result {
    font-weight: bolder;
    font-size: larger;
}
```

اما النتيجة في المتصفح، فتكون كالتالي:



5.2.1. JsonPipe:

وهذا النوع يستفيد منه المطور لعملية تصحيح الأخطاء debugging بحيث يستطيع التعامل مع البيانات القادمة ومعرفة ما فيها عن طريق تحويلها إلى صيغة json.

وبذلك نكون استعرضنا أهم أنواع Pipes، وفي حال أردت عزيزي المتعلم الاطلاع على باقي Pipes تستطيع الحصول عليها من الموقع الرئيسي مع معرفة طرق التعامل معها في مشاريع Angular.

والآن لننتقل الى النوع الثاني من Pipes وهو Custom Pipes.

3.1. Custom Pipes:

أتاح لنا Angular ان نقوم ببناء Pipes الخاص بنا بكل سهولة وذلك من خلال كتابة الأمر التالي عن طريق Angular CLI

ng g pipe (Name of Pipe)

ولمعرفة أكثر عن كيفية التعامل مع الأمر generate بشكل خاص وAngular CLI بشكل عام الرجاء مراجعة الكتاب الأول من هذه السلسلة Angular Environment Setup.

أما الآن لنقوم ببناء Pipe يقوم بالتحويل من الميل إلى كلم، وفي البداية يجب كتابة الامر السابق في terminal الخاص بمحرر الأكواد VS Code كما تعلمنا في الكتاب الأول مع التأكد اننا بنفس مجلد المشروع، مع العلم أنني سوف اختار الاسم convert لهذا Pipe (لك حرية اختيار الاسم الذي تُريده). وعند الانتهاء من كتابة الامر السابق والضغط على زر Enter من لوحة المفاتيح سوف ينشئ لنا ملف باسم convert.pipe.ts، وعند استعراض محتويات هذا الملف، نجد التالي:

convert.pipe.ts ملف

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'convert'   1
})

export class ConvertPipe implements PipeTransform {

  transform(value: unknown, ...args: unknown[]): unknown {
    return null;                                          2
  }

}
```

حيث انه في الرقم 1 يوجد اسم الPipe وهو الذي سوف نستخدمه في ملف HTML كما كنا نستخدم سابقاً date او number وغيرها.

اما في الرقم 2 فيوجد الدالة ذات الاسم transform حيث محتوى هذه الدالة هو الذي يتم فيه كتابة Logic البرمجي الخاص بهذا Pipe.

ونلاحظ ان هذه الدالة تستقبل مدخلين، كالتالي:

- value: وهذا المدخل هو عبارة عن القيمة التي تُريد ان يقوم هذا pipe بتنفيذ مهمة ما عليها، كما كنا نفعل سابقاً عندما نستقبل قيمة التاريخ ونطبق عليها date pipe او القيمة النصية ونطبق عليها titlecase pipe.
- args: المدخل الثاني هو عبارة عن البارامترات التي يستقبلها هذا البارامتر لتحديد أي مهمة نريد ان يقوم بها pipe على القيمة، وهي عبارة عن مصفوفة لأنه ممكن ان يكون لدينا أكثر من بارامتر، كما شاهدنا سابقاً عندما تعاملنا مع CurrencyPipe او بارامتر واحد كما تعاملنا مع DecimalPipe.

الآن لنقوم بكتابة Logic البرمجي الذي يقوم باستقبال القيمة بالميل وتحويلها إلى كلم او متر او سم بناءً على البارامتر الذي تم تمريره له، كالتالي:

convert.pipe.ts ملف

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'convert'
})

export class ConvertPipe implements PipeTransform {

  transform(value: number, targetUnits: string): unknown {

    if (!value) {
      return '';
    }

    switch (targetUnits) {
      case 'km':
        return value * 1.60934;
      case 'm':
        return value * 1.60934 * 1000;
      case 'cm':
        return value * 1.60934 * 1000 * 1000;
      default:
        throw new Error('Target Unit not Supported');
    }
  }
}
```

نلاحظ اننا قمنا بتغيير الأنواع للمدخلات بحيث value جعلناها رقم (لأننا هنا نتعامل مع ارقام) اما المدخل الثاني فوضعناه نص (لأننا نتعامل مع بارامتر نصي واحد)، بمعنى ان Angular يُعطي للمطورين مرونة أكثر من حيث تمكينهم من تحديد الأنواع المناسب للمهمة التي سوف ينفذها هذا pipe ولا يلزمهم بالأنواع المحدد افتراضية.

اما logic فهو بسيط حيث في البداية نتأكد من ان قيمة للمدخل value وفي حال عدم وجود أي قيمة نعيد قيمة فارغة، ومن ثم عن طريق switch نفحص قيمة المدخل الثاني فإذا km نجري عملية رياضية للتحويل إلى كلم، وهكذا بقية الأنواع، وفي حال كانت القيمة غير هذه الأنواع الثلاثة فنقوم بإظهار رسالة خطأ.

الآن لنقوم بتطبيق هذا الpipe في ملف template للcomponent، كالتالي:

ملف app.component.html

```
<div class="container">
    <h3>Payment Form</h3>
    <div class="row">
        <div class="form-group col">
            <label>Payee Name</label>
            <input type="text" class="form-control" (input)="onType($event.target.value)" />
        </div>
        <div class="col">
            <label><p>Name</p></label>
            <div class="result">
                {{value | titlecase}}
            </div>
        </div>
    </div>
    <div class="row">
        <div class="form-group col">
            <label>Payment Date</label>
            <input
               type="date"
               class="form-control"
               (input)="onDateChange($event.target.value)"
            />
        </div>
        <div class="col">
            <label><p>Payment Date</p></label>
            <div class="result">
                {{date| date:"MMMM d, y - h:m:s" : "en-SA"}}
            </div>
        </div>
    </div>
    <div class="row">
        <div class="form-group col">
            <label>Payment Amount</label>
            <input
               type="text"
               class="form-control"
               (input)="onAmountChange($event.target.value)"
            />
        </div>
        <div class="col">
            <label><p>Payment Amount</p></label>
            <div class="result">
                {{ amount|currency: "SAR" : "symbol" : "3.1-2" }}
            </div>
```

```
        </div>
    </div>
    <div class="row">
        <div class="form-group col">
            <label>Height</label>
            <input
               type="text"
               class="form-control"
               (input)="onHeightChange($event.target.value)"
            />
        </div>
        <div class="col">
            <label><p>Height</p></label>
            <div class="result">
                {{ height | number: "2.2-5" }}
            </div>
        </div>
    </div>
    <div class="row">
        <div class="form-group col">
            <label>Miles</label>
            <input
               type="text"
               class="form-control"
               (input)="onMilesChange($event.target.value)"
            />
        </div>
        <div class="col">
            <label><p>Kilometers</p></label>
            <div class="result">
                {{ miles | convert: 'km' }}
            </div>
        </div>
    </div>
</div>
```

app.component.ts ملف

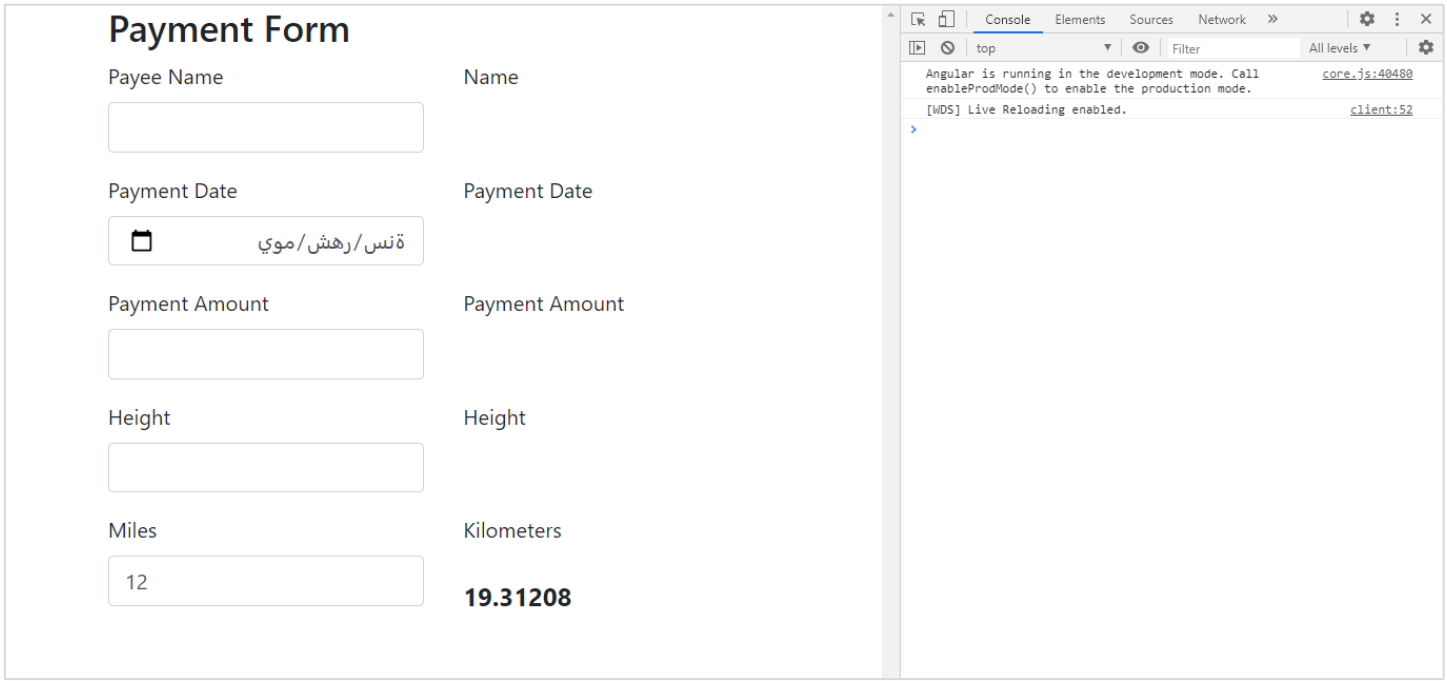
```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})

export class AppComponent {

  value: string;
  date: string;
  amount: number;
  height: number;
  miles: number;
```

```
  onType(value: string) {
    this.value = value;
  }

  onDateChange(value: string) {
    this.date = value;
  }

  onAmountChange(value: string) {
    this.amount = +value;
  }

  onHeightChange(value: string) {
    this.height = +value;
  }

  onMilesChange(value: string) {
    this.miles = +value;
  }

}
```

ملف app.component.css

```
.result {
    font-weight: bolder;
    font-size: larger;
}
```

اما النتيجة في المتصفح فتكون كالتالي:



3.1. ملاحظات مهمة:

اولاً/ يجب التنويه انه يمكن كتابة الPipes سواء كانت Built-In او Custom في أي مكان نكتب فيه الLogic البرمجي في ملف HTML او المسمى ايضاً ملف Template، مثل Interpolation {{ }} – Property Binding [ ] – Event Binding ( ) .

أو Built-In Directives مثل *ngIf.

مثلاً لو أردنا أن نخفي نتيجة التحويل إلى كلم بحيث لا تظهر النتيجة إلى إذا كانت النتيجة أكبر من 10 كلم، فنستطيع ذلك، كالتالي:

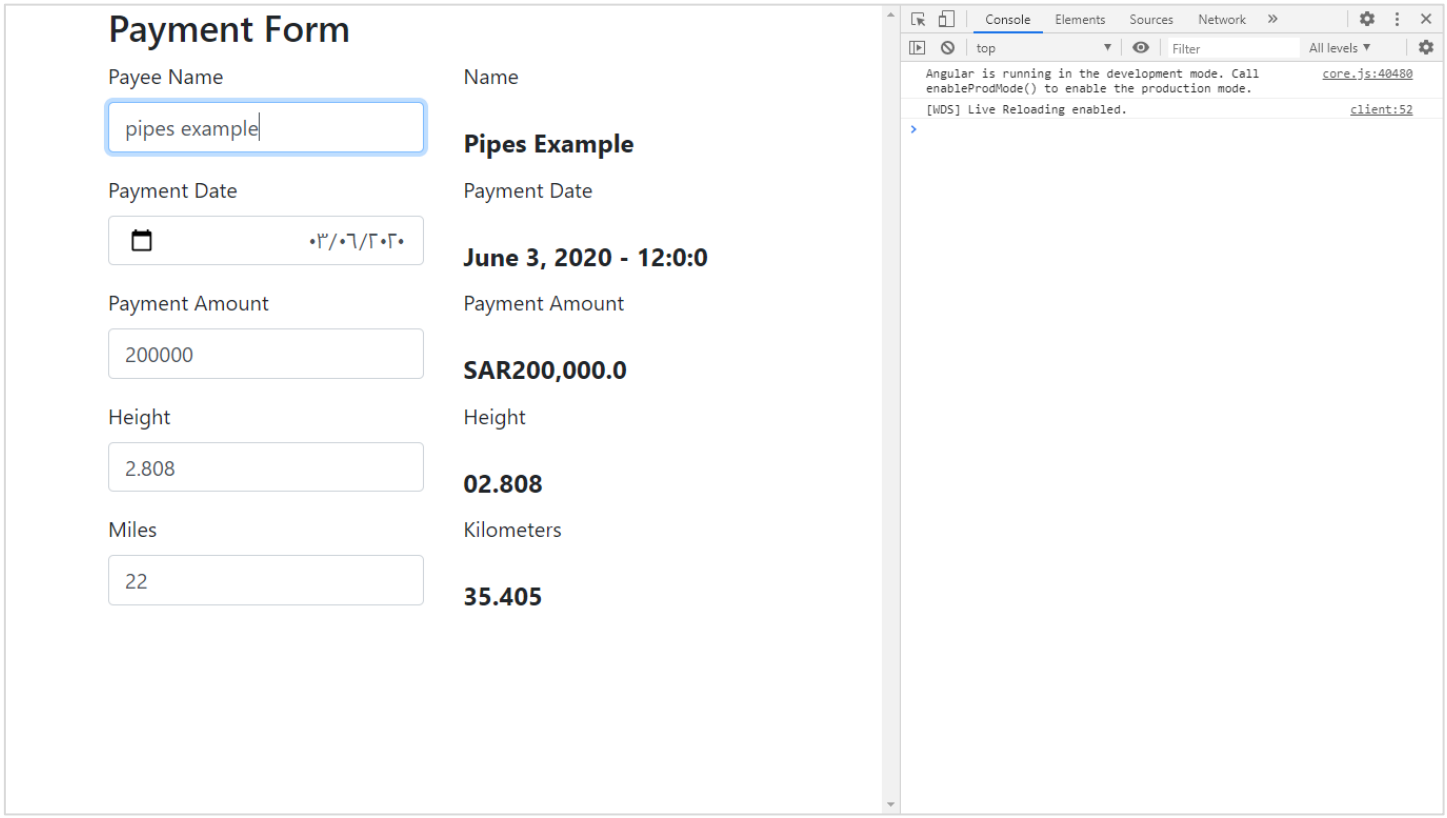
جزء من ملف app.component.html

```
<div class="row">
    <div class="form-group col">
        <label>Miles</label>
        <input
           type="text"
           class="form-control"
           (input)="onMilesChange($event.target.value)"
        />
    </div>
    <div class="col">
        <label><p>Kilometers</p></label>
        <div class="result" *ngIf="(miles | convert: 'km') > 10">
            {{ miles | convert: 'km' }}
        </div>
    </div>
</div>
```

ثانياً/ يمكن الدمج بين أكثر من pipe فمثلاً لو أردنا ان نقوم بعملية تقريب للرقم بعد تحويله من ميل إلى cm، نستطيع القيام بذلك كالتالي:

جزء من ملف app.component.html

```
<div class="row">
    <div class="form-group col">
        <label>Miles</label>
        <input
           type="text"
           class="form-control"
           (input)="onMilesChange($event.target.value)"
        />
    </div>
    <div class="col">
        <label><p>Kilometers</p></label>
        <div class="result" *ngIf="(miles | convert: 'km') > 10">
            {{ miles | convert: "km" | number: "2.2-3" }}
        </div>
    </div>
</div>
```



ثالثاً/ عند بناء Custom Pipe فأننا لا نحتاج أن نقوم باستدعائه، فعلى سبيل المثال لو أردنا ان نتعامل مع Forms لابد من ان نستدعي FormsModule، وأما Pipe نقوم بكتابته مباشرة في ملف template، كما كنا نفعل بالعادة.

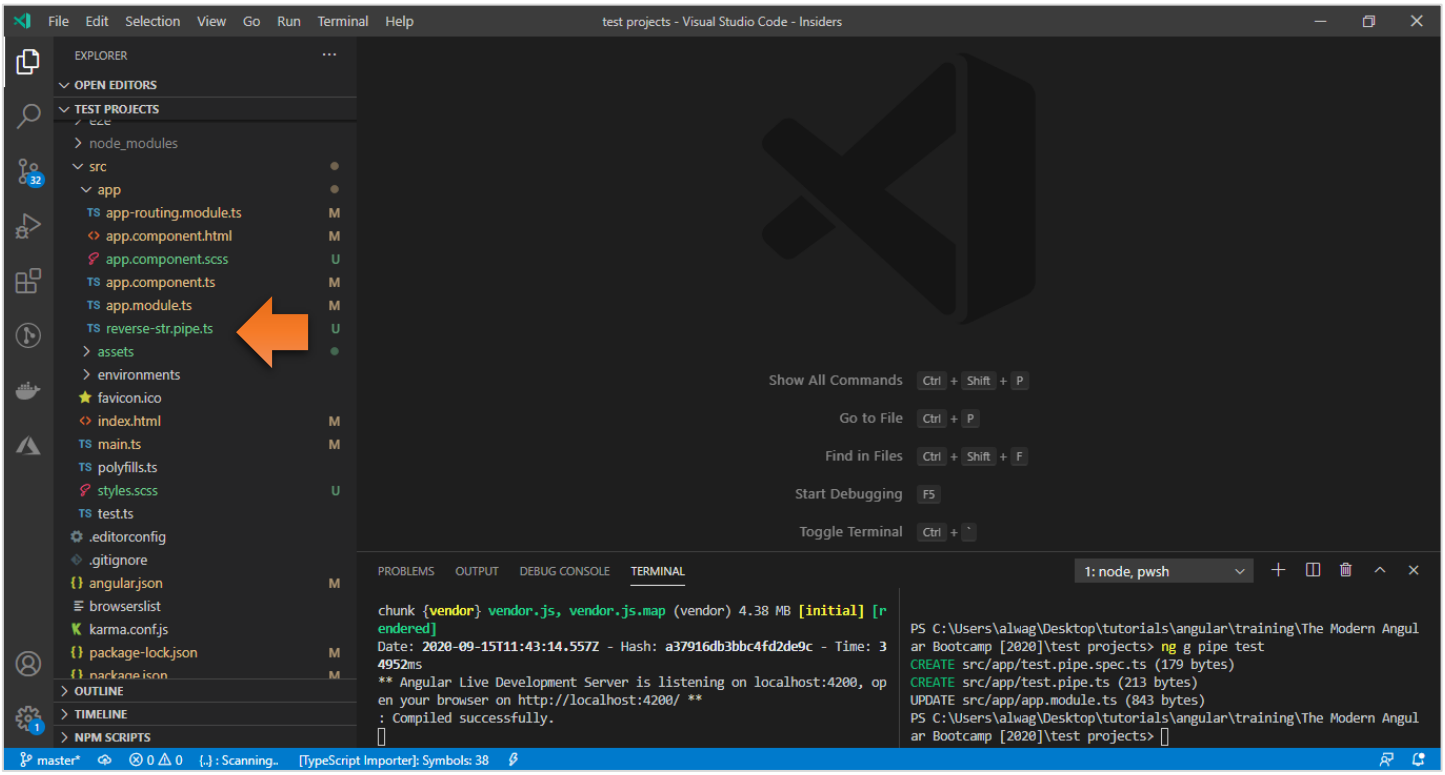
4.1. Pure/Impure Pipes:

في حالة بناء custom pipe فإننا امام خيارين الأول هو ان نجعل هذا pipe يكون pure وهو الوضع الافتراضي، او نغيره ليكون impure، والسؤال هنا ما هو الفرق بين الاثنين او بعبارة اصح ما هو pure؟ وما هو impure؟

وللإجابة على هذه التساؤلات لنسترجع ما قلناه سابقاً من ناحية ان Pipes هي في حقيقتها عبارة عن دوال Functions وهذه الدوال لها مدخل وهو القيمة، وهذه القيمة بطبيعة الحال لها نوع بيانات سواء نصي string او رقمي number او منطقي boolean او مصفوفة او كائن، ...الخ.

ومما قيل سابقاً نستطيع ان نقول ان Pure Pipe تقوم بمراقبة هذا المدخل (القيمة) في حال تم تغيير قيمتها سيتم تطبيق هذا Pipe على هذه القيمة بشرط ان يكون النوع الخاص بهذا المدخل (القيمة) هو اما يكون من الأنواع Primitive Type مثل (string, number, boolean) او تم تغيير References الخاص بالكائن مثل تغيير References لي (Array, Objects).

بمعنى لو افترضنا انه لدينا Custom Pipe يقوم بعكس كلمة نصية وهذه الكلمة تمثل القيمة value التي سوف يتم تطبيق هذا pipe عليها، وبما ان هذه القيمة هي قطعة نصية فان نوعها سوف يكون string، ولذلك في حال وجود زر على سبيل المثال يقوم بتغيير هذه القيمة من قيمة إلى قيمة أخرى فإن Pure Pipe سوف يلتقط هذه التغيير ويطبق ايضاً pipe عليه وهو عكس الكلمة او بمعنى آخر عكس القيمة value، ولتوضيح لنقوم بإعطاء مثال يقوم باستقبال قيمة وعكس النص، وللقيام بذلك بدايةً ننشأ Pipe جديد عن طريق Angular CLI وليكن اسمه reverse-str، كالتالي:



ومن ثم نقوم بكتابة Logic الخاص بعكس القيمة في الملف reverse-str، كالتالي:

ملف reverse-str.pipe.ts

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'ReverseStr'
})

export class ReverseStrPipe implements PipeTransform {

  transform(value: string, ...args: unknown[]): unknown {
    if (!value) {
      return null;
    }
    let newStr = '';
    for (let i = value.length - 1; i >= 0; i--) {
      newStr += value.charAt(i);
    }
    return newStr;
  }

}
```

والآن لنذهب إلى Root Component وبالتحديد في ملف class ونضيف خاصية ونسند لها أي قيمة وليكن "12345"، كالتالي:

ملف app.component.ts

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

@Component({
   selector: 'app-root',
```

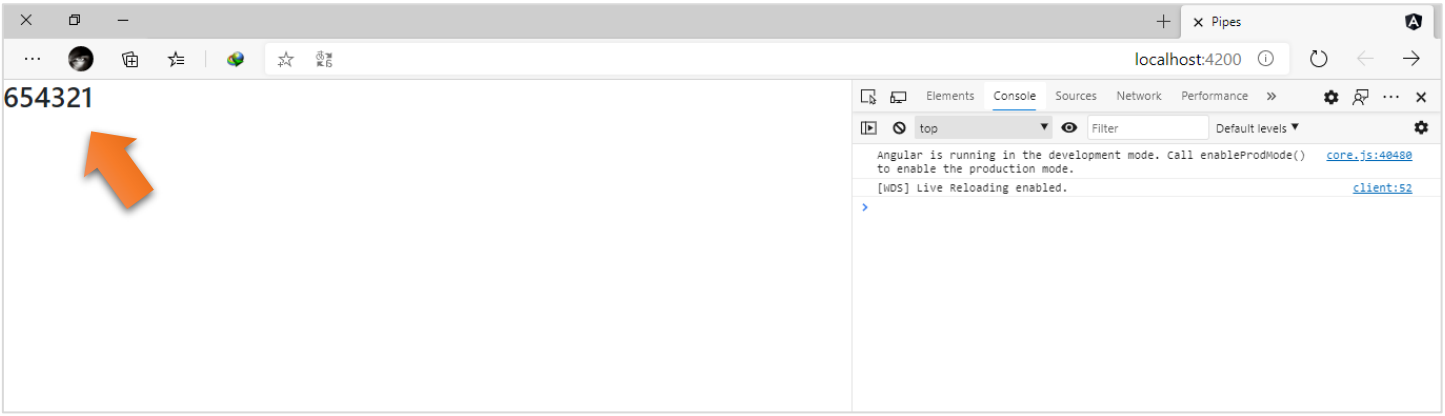
```
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss'],
})

export class AppComponent implements OnInit {

  str = '123456';

  constructor() { }

  ngOnInit(): void { }

}
```

ومن ثم لنقوم بعمل bind لهذه الخاصية في ملف template مع إضافة Pipe المنشأ سابقاً إليه، كالتالي:

ملف app.component.html

```
<h3>{{ str | ReverseStr }}</h3>
```

اما النتيجة في المتصفح فتكون كالتالي:



إلى الآن جميع ما ذكر هو مراجعة لما قيل سابقاً، ولم نقم بتطبيق Pure or Impure Pipes، لذلك سوف نقوم في البداية بمحاولة فهم Pure Pipe.

قلنا سابقاً ان أي pipe نقوم بإنشائه فأن Angular بشكل افتراضي سوف يعامله على انه Pure Pipe ومن خصائص هذا النوع انه يراقب القيمة في حال كان نوعها Primitive Type فإن أي تغيير سوف يحدث على هذه القيمة سوف يطبق Pipe عليها، لذلك لنقوم بإنشاء زر وهذا الزر يقوم بتغيير قيمة الخاصية str المنشأة سابقاً، كالتالي:

ملف app.component.html

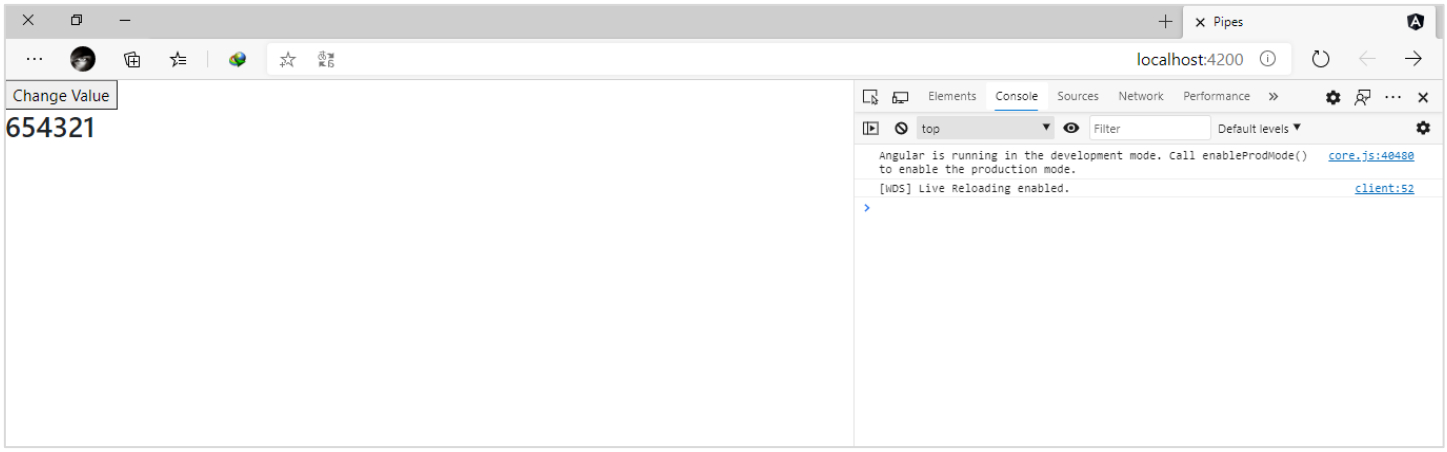
```
<button (click)="changeValue()">Change Value</button>

<h3>{{ str | ReverseStr }}</h3>
```

وفي ملف class لهذا component نكتب محتوى هذه الدالة، كالتالي:

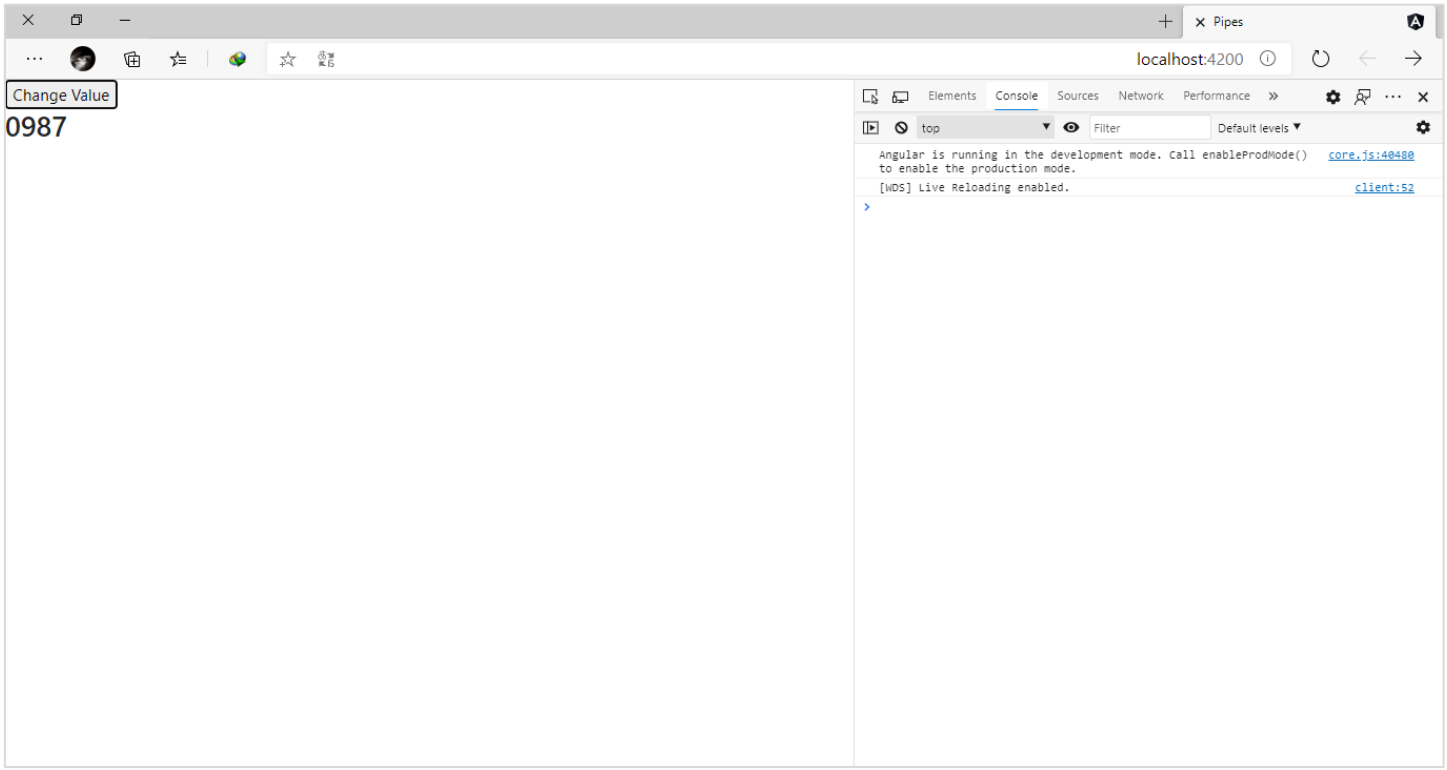
ملف app.component.ts

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

@Component({
   selector: 'app-root',
   templateUrl: './app.component.html',
   styleUrls: ['./app.component.scss'],
})

export class AppComponent implements OnInit {
   str = '123456';
   constructor() { }

   ngOnInit(): void { }

   changeValue() {
      this.str = '7890';
   }
}
```

اما النتيجة فسوف تكون، كالتالي:



والآن لنقوم بالضغط على الزر لكي نغير القيمة، والنتيجة سوف تكون كالتالي:



نلاحظ مع تغير القيمة تم تطبيق Pipe وهذا الذي نقصده عندما قلنا انه في حال كان Pipe هو Pure ونوع البيانات للقيمة هي اما string, number, boolean فإن أي تغيير في القيمة سوف يطبق pipe على القيمة الجديدة والتي هي في حالتنا هذه عكس النص المُدخل.

اما الحالة الثانية التي يستطيع Pure Pipe مراقبتها وعند تغيير قيمتها يقوم بتطبيق هذا pipe عليها هي في حال تغيير references الخاص بالكائن (Array, Object, Date) في الذاكرة، لذلك لنقوم بتغيير تعريف الخاصية لتصبح مثلاً مصفوفة، كالتالي:

ملف app.component.ts

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

@Component({
   selector: 'app-root',
   templateUrl: './app.component.html',
   styleUrls: ['./app.component.scss'],
})

export class AppComponent implements OnInit {

   str = ['123456', '876', '6243'];

   constructor() { }

   ngOnInit(): void { }

   changeValue() {
   }
}
```

ومن ثم نحتاج إلى أن نقوم بتغيير Logic الخاص في ملف Pipe لكي يتعامل مع مصفوفة، كالتالي:

ملف reverse-str.pipe.ts

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'ReverseStr'
})

export class ReverseStrPipe implements PipeTransform {

  transform(value: any[], ...args: unknown[]): unknown {
    if (!value) {
      return null;
    }
    let newStr = '';
    for (let i = value[0].length - 1; i >= 0; i--) {
      newStr += value[0].charAt(i);
    }
    return newStr;
  }

}
```

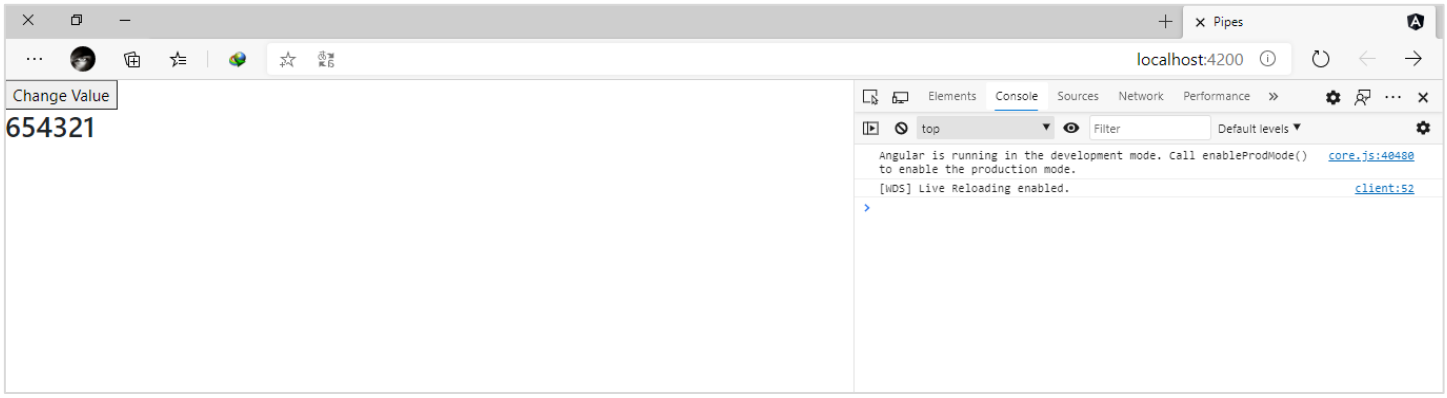
حيث بعد هذا التعديل سوف يأخذ هذا Pipe اول عنصر من المصفوفة ويقوم بتغيير ترتيبها بشكل عكسي.

اما في ملف template لي Root Component فلن نقوم بإجراء أي تعديل، وملف class نضيف الاسطر البرمجية التالية التي تقوم بتغيير reference لي المصفوفة المنشأة سابقاً ذات الاسم str، كالتالي:

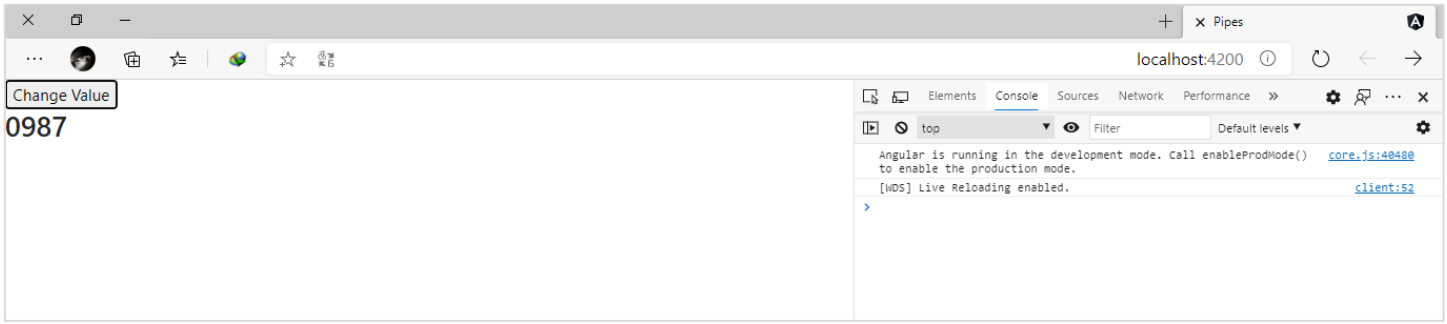
ملف app.component.ts

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

@Component({
   selector: 'app-root',
   templateUrl: './app.component.html',
   styleUrls: ['./app.component.scss'],
})

export class AppComponent implements OnInit {
   str = ['123456', '876', '6243'];

   constructor() { }
   ngOnInit(): void { }

   changeValue() {
      // this.str = '7890';
      const newStr = Object.assign([], this.str);
      newStr[0] = '7890';
      this.str = newStr;
   }
}
```

نلاحظ قمنا بعمل تعطيل لسطر القديم واضفنا Logic الجديد والخاص بتغيير المرجع Reference لي المصفوفة، اما النتيجة فستكون كالتالي:



الآن لنقوم بالضغط على الزر ونرى النتيجة، كالتالي:



نلاحظ ان Pipe اكتشف التعديل في القيمة وطبق عليها Logic الخاص بتغيير ترتيب القيمة بشكل عكسي، وهذا الذي قصدناه عندما أشرنا في السابق بإن الحالة الثانية في Pure Pipe في حالة تم تغيير Reference للكائن فإن Pure Pipe سوف يكتشف هذا التعديل.

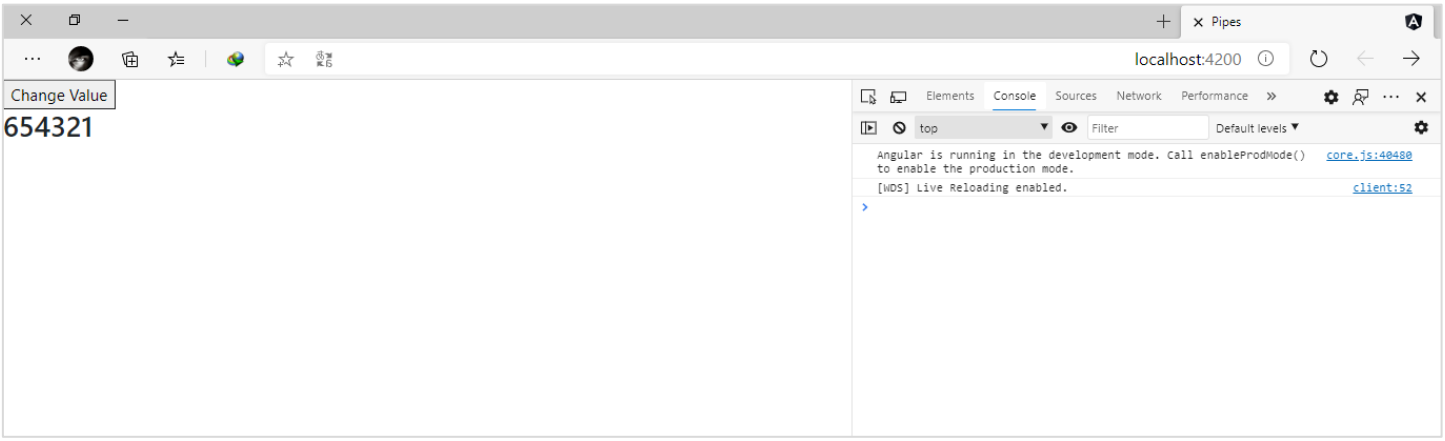
ولكن ماذا لو لم نقم بتغيير Reference للكائن سواء كانت مصفوفة او غيرها، بحيث قمنا بكتابة الأمر التالي:

ملف app.component.ts

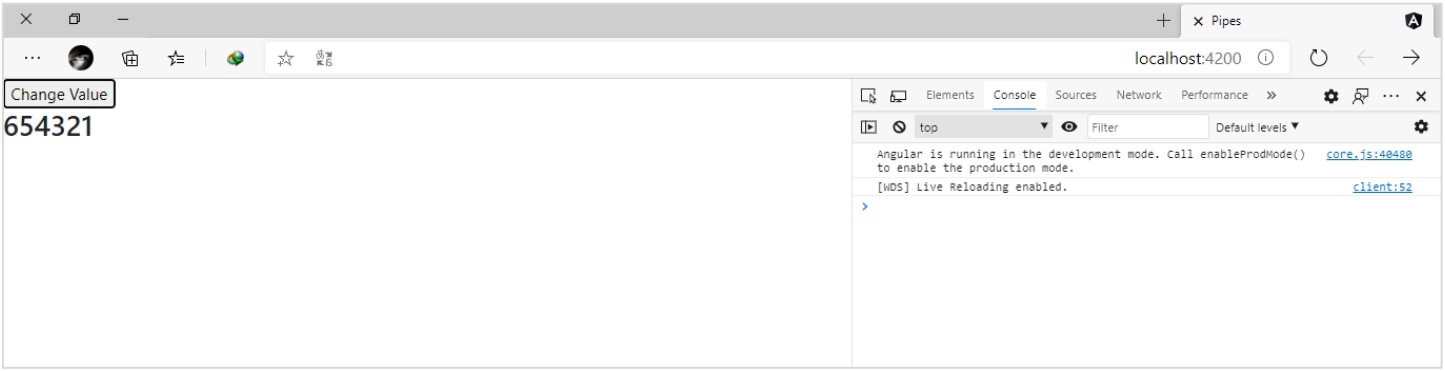
```
import { Component, OnInit } from '@angular/core';
@Component({
   selector: 'app-root',
   templateUrl: './app.component.html',
   styleUrls: ['./app.component.scss'],
})
export class AppComponent implements OnInit {
   str = ['123456', '876', '6243'];
   constructor() { }
   ngOnInit(): void { }

   changeValue() {
      // this.str = '7890';

      // const newStr = Object.assign([], this.str);
      // newStr[0] = '7890';
      // this.str = newStr;

      this.str[0] = '7890';
   }
```

```
}
```

هنا اختلف الأمر حيث قمنا بالوصول إلى عنصر داخل المصفوفة ومن ثم أجرينا تغيير على قيمته، ولم نقم بتغيير Reference الخاص بالمصفوفة نفسها في الذاكرة RAM، لذلك لنرى النتيجة في المتصفح ومن ثم نعلق عليها، كالتالي:



الشاشة السابقة في حالة بداية التشغيل، لنقوم بالضغط على الزر ولنرى النتيجة:



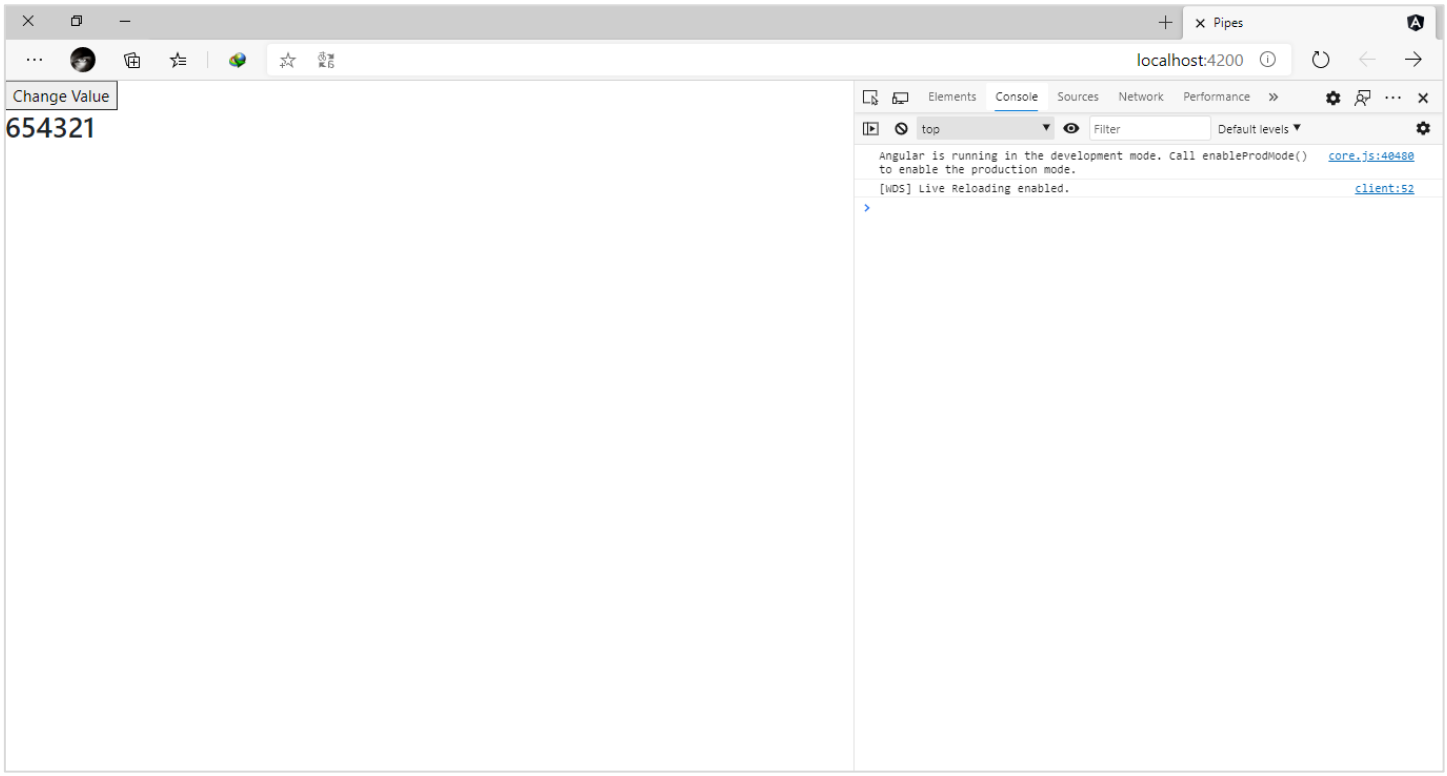
نلاحظ لم يتم تحديث القيمة في ملف template مع العلم ان القيمة في المصفوفة تم تغييرها ولكن لم يتم تحديثها في template لذلك مستخدم التطبيق لم يلاحظ وجود أي تغيير، ومن الحلول المطروحة لمثل هذا الأمر هو استخدام Impure Pipe، وطريقة استخدامه سهلة جداً، حيث نذهب إلى ملف pipe ونضيف السطر البرمجي التالي:

reverse-str.pipe.ts ملف

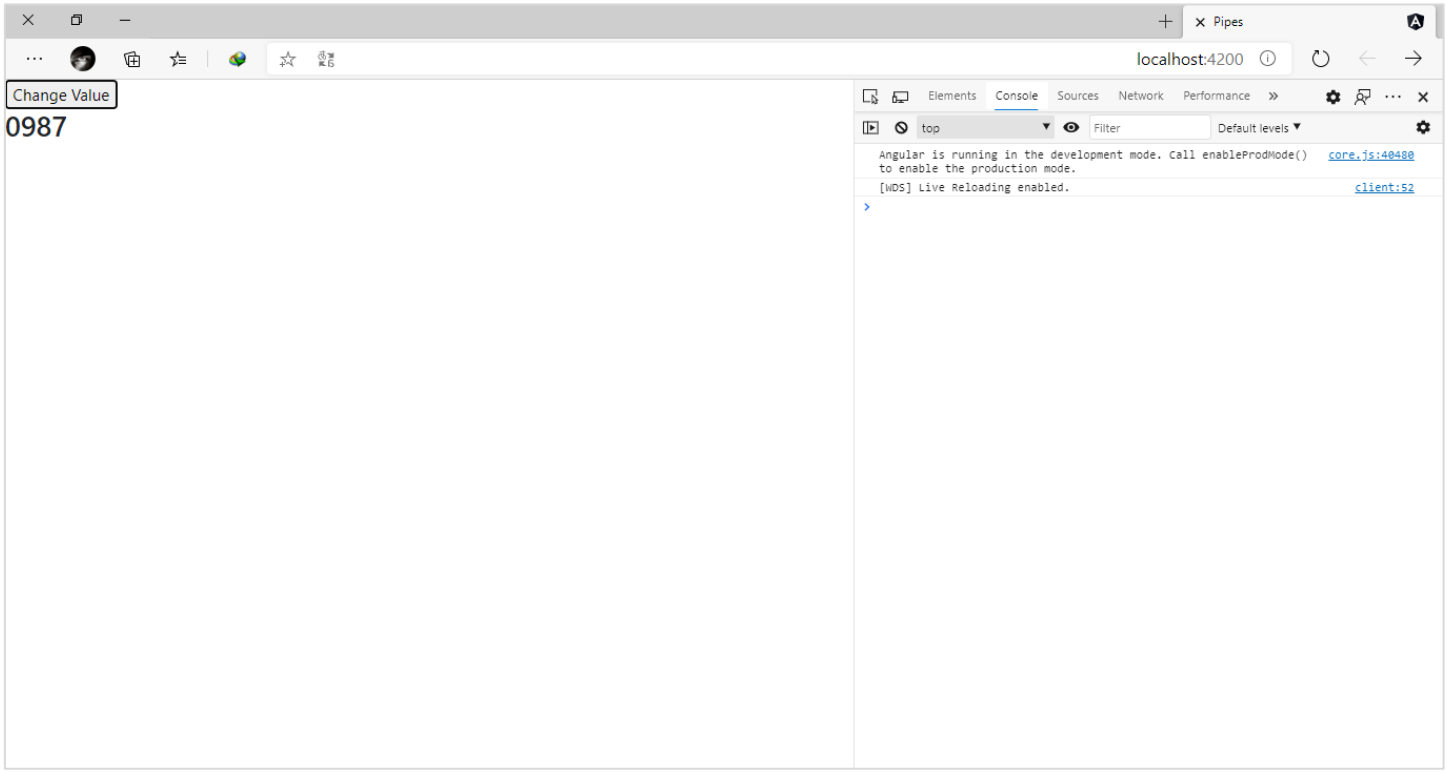
```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'ReverseStr',
  pure: false
})

export class ReverseStrPipe implements PipeTransform {
  transform(value: any[], ...args: unknown[]): unknown {
    if (!value) {
      return null;
    }
    let newStr = '';
    for (let i = value[0].length - 1; i >= 0; i--) {
      newStr += value[0].charAt(i);
    }
    return newStr;
  }
```

```
}
```

والآن لنذهب إلى المتصفح ولنرى النتيجة:



ولنضغط على الزر ولنرى النتيجة:



نلاحظ اكتشف التعديل وقام بتطبيقه، ولكن يجب الانتباه وان تكون حذراً عزيزي المتعلم عند استخدام Impure Pipe، لأن أي تعديل يتم في هذا component الذي تم فيه استدعاء هذا pipe فإن Angular سوف يقوم بتنفيذ هذا pipe وإن كان التغيير في قيمة جزء آخر من هذا component، طبعاً في حالة التطبيقات البسيطة مثل مثالنا هذا فإن هذا الأمر لن يؤثر، ولكن في

حال كان التطبيق متوسط إلى كبير فعندئذً يجب التفكير جيداً قبل استخدام impure pipe لأنها سوف تستهلك موارد أجهزة مستخدمي التطبيق الخاص بك بشكل كبير، ولتوضيح لننفرض اننا احتجنا لأن نراقب حركة مؤشر الفأرة خلال مرورها فوق عنصر محدد وليكن div في هذا component الذي يستدعي هذا pipe، كالتالي:

ملف app.component.html

```
<button (click)="changeValue()">Change Value</button>
<h3>{{ str | ReverseStr }}</h3>
<div
   style="
      width: 500px;
      height: 500px;
      background: silver;
      border: 1px solid black;
      margin: 0 auto;
   "
   (mousemove)="onMouseMove()"
></div>
```

قمت بإضافة بعض التنسيقات لكي يكون div واضح وظاهر وبنفس الوقت عند مرور مؤشر الفأرة نقوم بتنفيذ دالة معينة اسميتها onMouseMove، اما محتوى هذه الدالة فلا شيء لأننا فقط نريد ان نبين تأثير أي حدث يتم على تنفيذ Impure Pipe:

ملف app.component.ts

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

@Component({
   selector: 'app-root',
   templateUrl: './app.component.html',
   styleUrls: ['./app.component.scss'],
})

export class AppComponent implements OnInit {
   str = ['123456', '876', '6243'];

   constructor() { }

   ngOnInit(): void { }

   changeValue() {
      // this.str = '7890';

      // const newStr = Object.assign([], this.str);
      // newStr[0] = '7890';
      // this.str = newStr;

      this.str[0] = '7890';
   }

   onMouseMove() {

   }
```

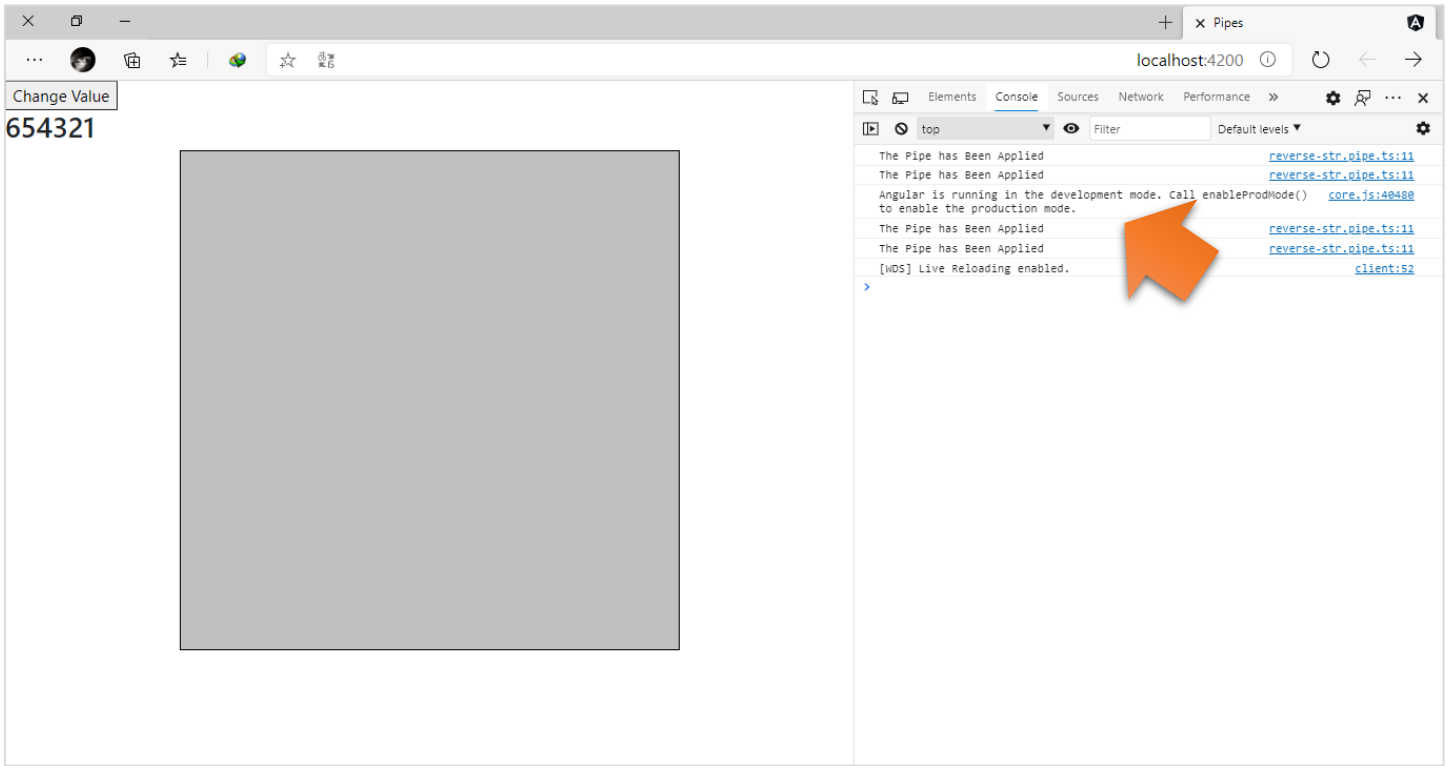
```
}
```

وفي نفس Pipe لنقوم بطباعة رسالة في console بحيث نتأكد ان Angular يستدعي وينفذ هذا Pipe ولو كان لم يتم على القيمة الخاصة بهذا pipe، كالتالي:

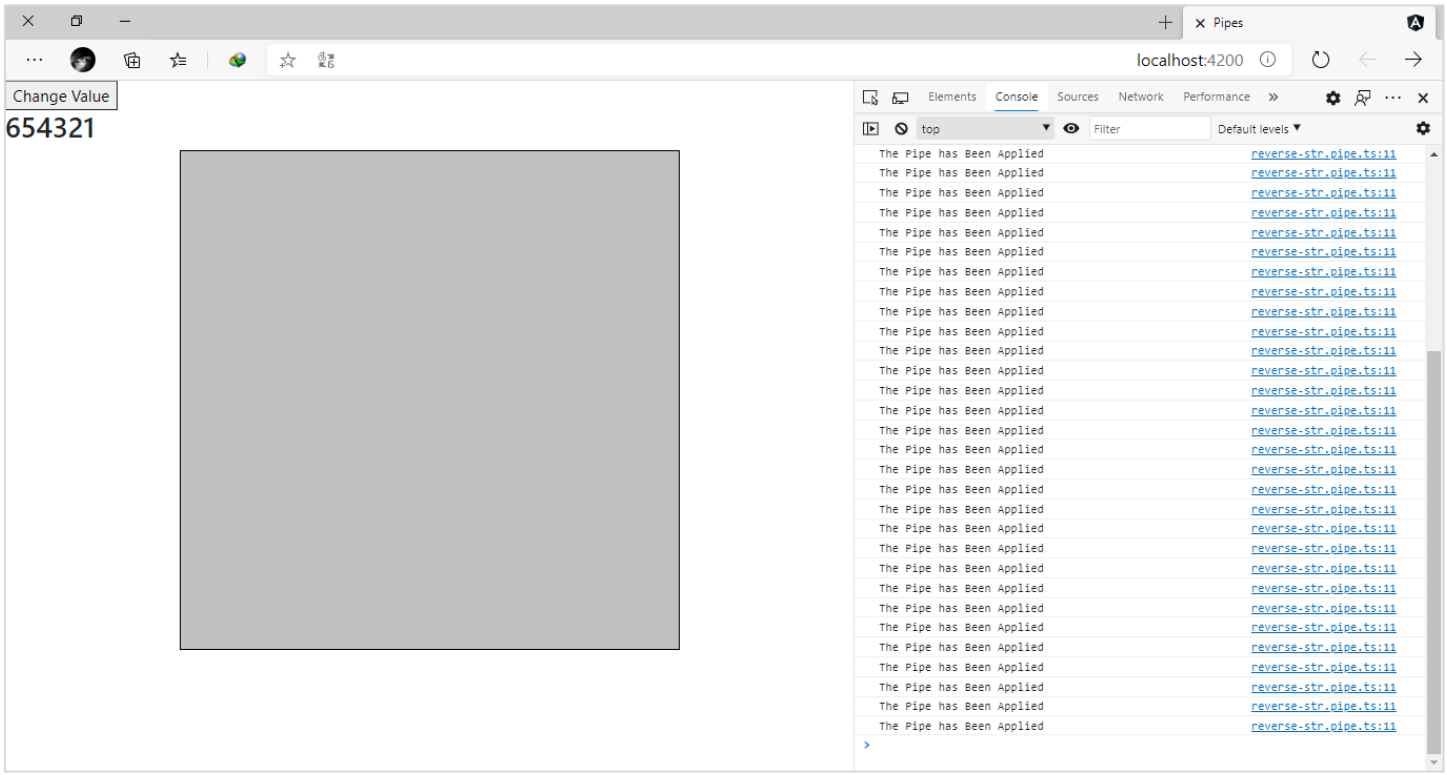
ملف reverse-str.pipe.ts

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'ReverseStr',
  pure: false
})

export class ReverseStrPipe implements PipeTransform {

  transform(value: any[], ...args: unknown[]): unknown {

    console.log('The Pipe has Been Applied');

    if (!value) {
      return null;
    }
    let newStr = '';
    for (let i = value[0].length - 1; i >= 0; i--) {
      newStr += value[0].charAt(i);
    }
    return newStr;
  }

}
```

اما النتيجة فستكون كالتالي:



نلاحظ انه ومع اننا لم نقم بتمرير المؤشر إلى الآن، فقط في بداية تشغيل التطبيق تم استدعاء وتنفيذ هذا Pipe لأن Angular يقوم بعمل بعض Checks عند بداية تشغيل أي تطبيق (لمعرفة اكثر عن هذه checks يُرجى مراجعة الكتاب الثاني من هذه السلسلة Angular Components and Services وبالتحديد القسم الخاص lifecycle hooks وايضاً change detection)، ولنقوم الآن بتحريك مؤشر الفأرة بداخل هذا div، ولنرى النتيجة:



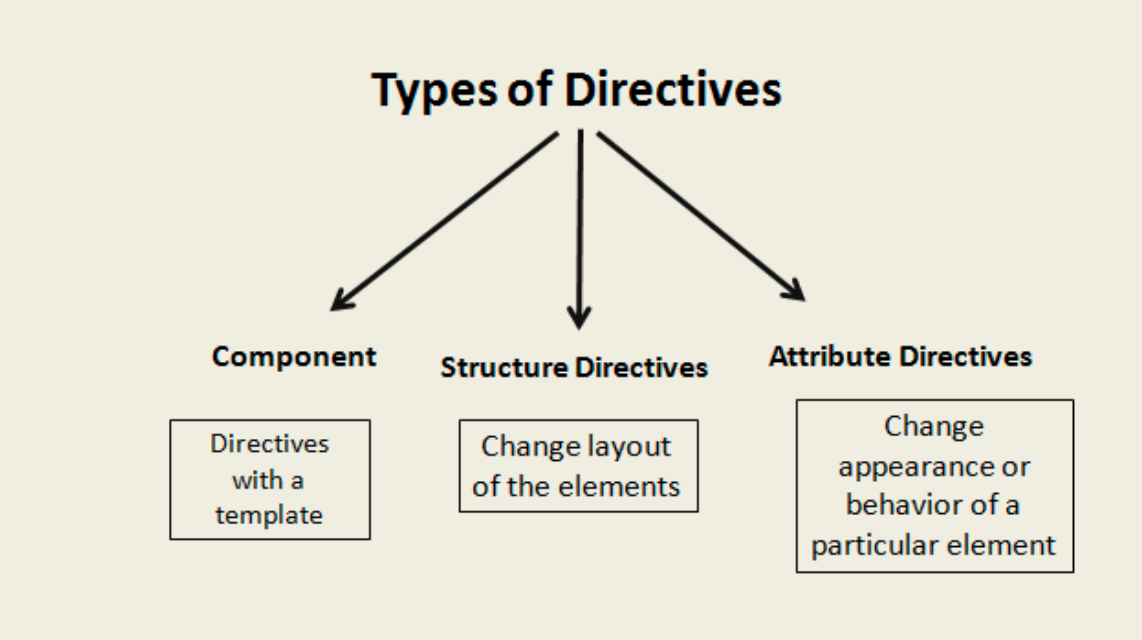
نلاحظ كمية الاستدعاءات لهذا pipe، مع العلم ان الحدث وقع على div وليس له علاقة بالقيمة الخاصة بهذا pipe، فلك عزيزي المتعلم ان تتخيل لو كان لديك Pipe يقوم بإجراء Logic معقد معتمداً على بيانات قادمة من Server على شبكة الانترنت فكم من الموارد سوف يتم استهلاكها في حال اردنا استخدام Impure Pipe.

الفصل الثاني

Directives

1.2. مقدمة:

هي عبارة عن function ممكن ان يتم وضعها في عنصر html معين سواء لتغيير سلوك هذا العنصر في DOM او حتى تغيير بنية هذا العنصر في DOM، كما انها ممكن ان تكون عبارة عن عناصر Tags مشابهه لعناصر HTML لتؤدي مهمة معينة، وممكن تقسيم Directives إلى ثلاث أنواع، هي:



Component: وقد تكلمنا عن هذا النوع سابقاً، وتستطيع الرجوع إلى الكتاب الثاني من هذه السلسلة Angular Components and Services

Structure Directives: وهذه الDirectives تقوم بتعديل وتغيير بُنية العنصر الذي يُضاف له هذا الDirective في DOM، مثل ngIf* التي تقوم بحذف عنصر معين او اضافته إلى DOM وفق شروط معينة، او ngFor* التي تقوم بتكرار عنصر معين في DOM، وهنالك مجموعة أخرى سوف نستعرضها جميعاً بإذن الله.

Attribute Directives: اما هذه فتقوم بتحكم في سلوك behavior الخاص بالعنصر المُضاف له هذا Directive كأن يقوم مثلاً بتغيير لون العنصر او الحجم او غيره من الخصائص المتنوعة.

وجميع هذه الأنواع تصنف على انها Built-In Directives، وايضاً يتيح لنا Angular أن نقوم ببناء Directives الخاصة بنا، وفي هذه الحالة تصنف على انها Custom Directives.

2.2. Structure Directives:

ويصنف هذا النوع على انه Built-In Directives أي أنه مبني ضمناً داخل أطار عمل Angular وهو موجود ضمن Module المسمى CommonModule وهو من Modules التي تُضاف تلقائياً عند أنشاء أي مشروع Angular جديد لذلك لاستخدام هذه Directives لا تحتاج إلى إضافة أي Modules أخرى، اما من ناحية وظيفتها فتقوم بتغيير بنية العنصر في DOM، كأن تقوم بحذفه او اضافته او تكراره، ولنقم في البداية بإنشاء مشروع Angular جديد (لمعرفة كيفية إنشاء مشروع جديد الرجاء مراجعة الكتاب الأول من هذه السلسلة Angular Environment Setup)، ومن أمثلتها ما يلي:

1.2.2. ngFor Directive:

ويقوم هذا Directive بتكرار عنصر معين في DOM بناءً على عدد عناصر مصفوفة معينة يتم تمريرها إليه.

والصيغة العامة له:

< العنصر *ngFor="let (اسم المتغير) of (اسم المصفوفة)" > </العنصر>

مثال/ لنفرض أن لدينا مصفوفة عدد عناصرها ست عناصر وتحتوي على مجموعة قيم وليكن اسمها arrayNum، كالتالي:

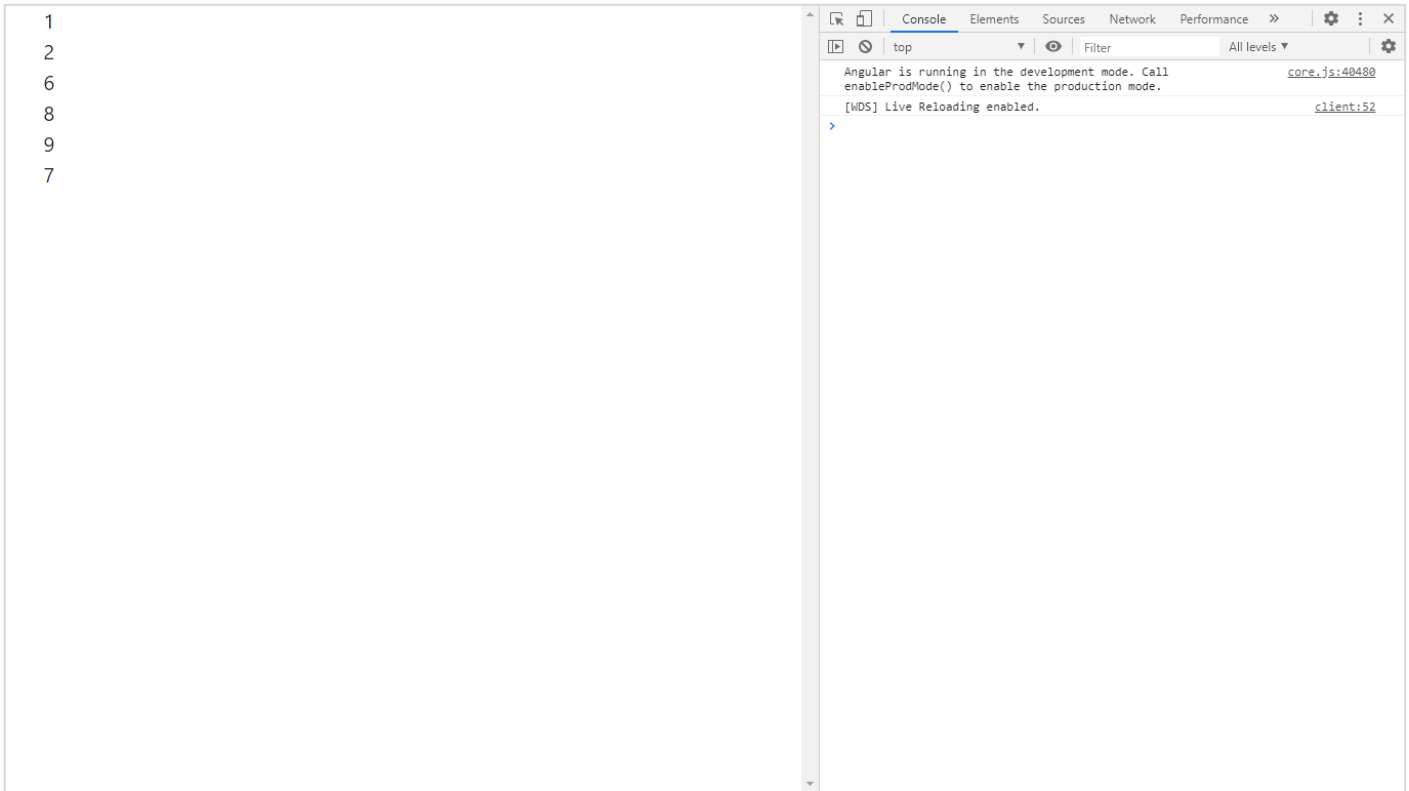
arrayNum = [1, 2, 6, 8, 9, 7]

بحيث ان هذه الخاصية arrayNum يتم تعريفها واسناد القيم لها في ملف class لي Root Component او بصيغة أخرى ملف app.component.ts، وفي ملف HTML الخاص بهذا component نضيف ngFor*، كالتالي:

ملف app.component.html

```
<div *ngFor="let arr of arrayNum">{{ arr }} </div>
```

وسوف تكون النتيجة أن هذا العنصر div سوف يتم تكراره (إعادة بنائه) في DOM ست مرات بعدد المصفوفة التي تم تمريره لها وبنفس الوقت سوف يضيف القيمة الموجودة في المتغير arr في العنصر بحيث اول div سوف تكون قيمته 1 والdiv الثاني سوف تكون قيمته 2 وهكذا بحسب القيم الموجودة في arr، كما في الشكل التالي: (لابد ان نعمل serve للمشروع لكي يعمل في server عن طريق كتابة الأمر ng s -o في terminal، ولمعرفة أكثر عن هذا الأمر وطرق التعامل مع Angular CLI الرجاء مراجعة الكتاب الأول من هذه السلسلة Angular Environment Setup)

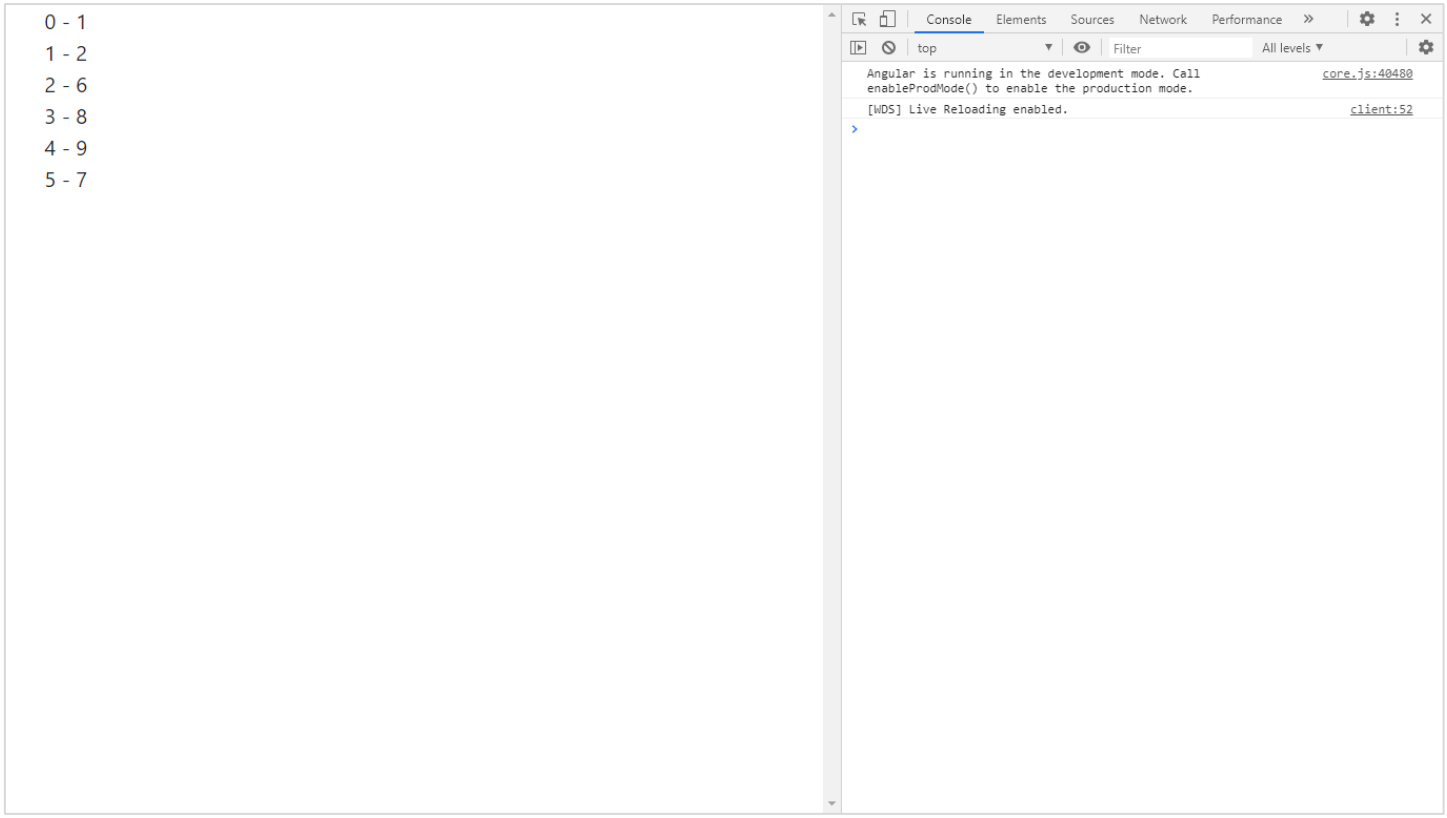


الصورة السابقة لهذا Directive هي الصيغة العامة وهي بشكلها البسيط ولكن هنالك إمكانيات أكثر يقدمها لنا هذا Directive، منها على سبيل المثال الحصول على index لكل عنصر من عناصر هذه المصفوفة، كالتالي:

ملف app.component.html

```
<div *ngFor="let arr of arrayNum; let i=index"> {{ i }} - {{ arr }} </div>
```

وسو تكون النتيجة:



نلاحظ انه اظهر لنا القيمة الموجودة في المتغير arr وبنفس الوقت اظهر لنا الindex لهذه القيمة بحسب ما هو موجود في المصفوفة.

كما يجدر الإشارة انه يمكن تعريف index بطريقة ثانية، كالتالي:

ملف app.component.html

```
<div *ngFor="let arr of arrayNum; index as i"> {{ i }} - {{ arr }} </div>
```

نلاحظ اننا قمنا بتعريفه عن طريق index as i وليس بالطريقة السابقة let i=index، وسواء استخدم الطريقة الأولى او الثانية فكلاهما يقومان بنفس المهمة.

وللاطلاع على قائمة الأوامر الإضافية التي تأتي مع ngFor من خلال الرابط التالي: https://angular.io/api/common/NgForOf

ونستطيع حصر أشهر هذه الأوامر من خلال الجدول التالي:

الأمر	الشرح
index	الحصول على index للعنصر داخل المصفوفة وهي تُعيد رقم
count	الحصول على عدد عناصر المصفوفة وهي تُعيد رقم
first	تُعيد قيمة منطقية true في حال كان العنصر هو أول عنصر في المصفوفة
last	تُعيد قيمة منطقية true في حال كان العنصر هو آخر عنصر في المصفوفة
even	تُعيد قيمة منطقية true في حال كان index للعنصر في المصفوفة زوجي، ونستطيع الاستفادة منها في حال اضفنا مجموعة من الصفوف في جدول بحيث نميز الصفوف الزوجية بلون مختلف، ولها استخدامات أخرى بحسب الحاجة
odd	تُعيد قيمة منطقية true في حال كان index للعنصر في المصفوفة فردي، ونستطيع الاستفادة منها في حال اضفنا مجموعة من الصفوف في جدول بحيث نميز الصفوف الفردية بلون مختلف، ولها استخدامات أخرى بحسب الحاجة
trackBy	ونسند لها دالة، ونستفيد منها في تحسين أداء التطبيق لديك عن استخدام ngFor

والأوامر السابقة بعضها اعطينا عليها أمثلة وأخرى لا تحتاج إلى شرح فتستطيع تطبيقها بنفسك ومعرفة كيفية التعامل معها، ولم يبقى معنا إلا الأمر الأخير وهو trackBy، وهذا الأمر لفهم عمله سوف اضرب مثال، لنفرض أن لدينا زر وهذا الزر يقرأ بيانات من server ومن ثم يستقبلها على شكل مصفوفة ويعرضها للمستخدم عن طريق ngFor، ومع كل ضغطة لهذا الزر سوف يعيد قراءة البيانات في حال انه كان هنالك بيانات جديدة لأضافتها هي الأخرى، افتراضياً يقوم Angular مع بداية الضغط على هذا الزر بقراءة المصفوفة ومن ثم إضافة (بناء) مجموعة من العناصر إلى DOM بناءً على عدد هذه المصفوفة، وعند الضغط مرة أخرى فأنه سوف يقوم بحذف جميع هذه العناصر من DOM وأضافتها مرة أخرى وهكذا، طبعاً في حال البيانات البسيطة فالأمر بسيط ولا يؤثر على اداء التطبيق ولكن في حال كانت البيانات كبيرة فإن هذا الامر يستهلك DOM ويقلل من أداء التطبيق ويسبب البطء مما يؤثر على تجربة المستخدم لتطبيقك، لذلك لحل هذه المشكلة نريده ان يقوم بقراءة هذه البيانات في كل ضغطة ولكن يضيف البيانات الجديدة فقط مع الإبقاء على البيانات القديمة كما هي، ونستطيع عمل ذلك عن طريق trackBy، كالتالي:

لنفرض ان لدينا مصفوفة كائنات بحيث كل كائن يحتوي على رقم الموظف واسم الموظف، وهنالك زر وكل ضغطة على هذا الزر نقوم بقراءة هذه المصفوفة وعرضها للمستخدم عن طريق ngFor، كالتالي:

ملف app.component.ts

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})

export class AppComponent {
  empArray = [];
```

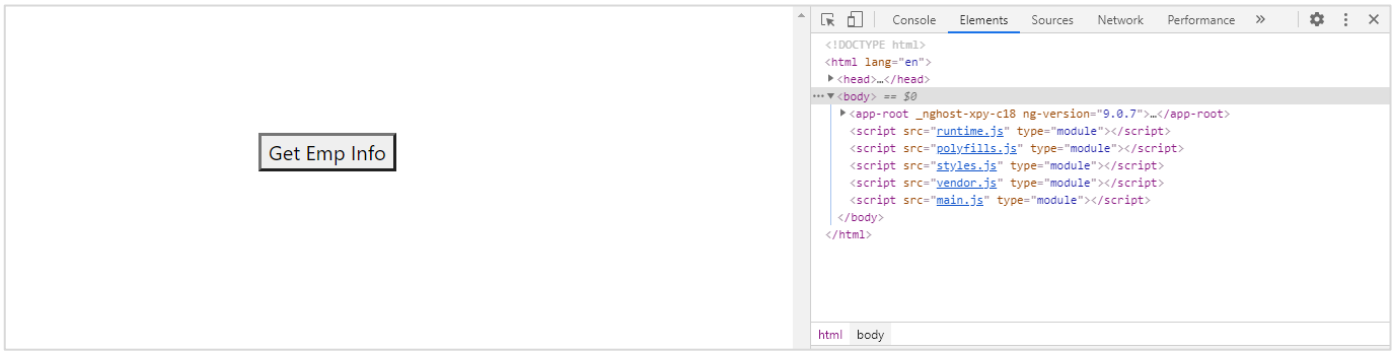
```
  loadEmp() {
   this.empArray = [
      {emp_no: 1, emp_name: 'mohd'},
      {emp_no: 2, emp_name: 'saad'},
      {emp_no: 3, emp_name: 'khalid'},
    ];
  }

}
```

أما في ملف HTML نضيف الكود التالي:

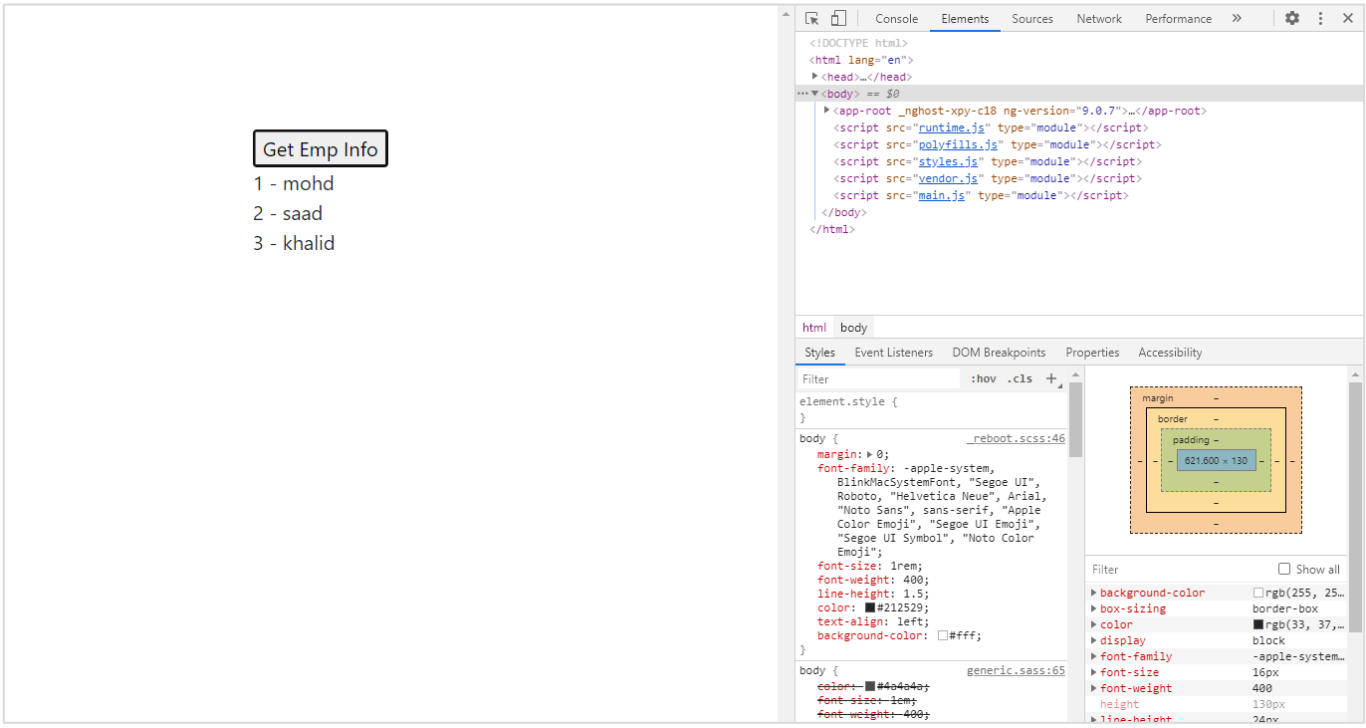
ملف app.component.html

```
<button (click)="loadEmp()">Get Emp Info</button>

<div *ngFor="let emp of empArray">
      {{ emp.emp_no }} - {{ emp.emp_name }}
</div>
```

ولمشاهدة النتيجة:



وعند الضغط على الزر:



نلاحظ عندما قمنا بالضغط على الزر اضفنا عناصر لهذه المصفوفة وتلقائياً قام Angular بإضافة العناصر باستخدام ngFor، وفي كل مرة يقوم المستخدم بالضغط على هذا الزر سوف يقوم Angular بحذف هذه العناصر وإعادة بنائها مرة أخرى في DOM، وهذا السلوك غير محبب كما أشرنا سابقاً خصوصا عند التعامل مع البيانات الكبيرة، لذلك سوف نستخدم trackBy ونسند لها دالة وهذه الدالة تستقبل بارامترين هما index للعنصر والمتغير الخاص بالمصفوفة الذي عرفناه داخل ngFor والذي هو في حالتنا هذه اسميناه emp، ومن ثم تقوم بالتأكد هل هذا العنصر موجود ام لا، وفي حال كان موجود فلا تعُيد أي قيمة ولنجعلها undefined اما إذا كانت هذه القيمة غير موجودة فأضف هذه القيمة، كالتالي:

ملف app.component.html

```
<button (click)="loadEmp()">Get Emp Info</button>

<div *ngFor="let emp of empArray; trackBy: trackEmp">
      {{ emp.emp_no }} - {{ emp.emp_name }}
</div>
```

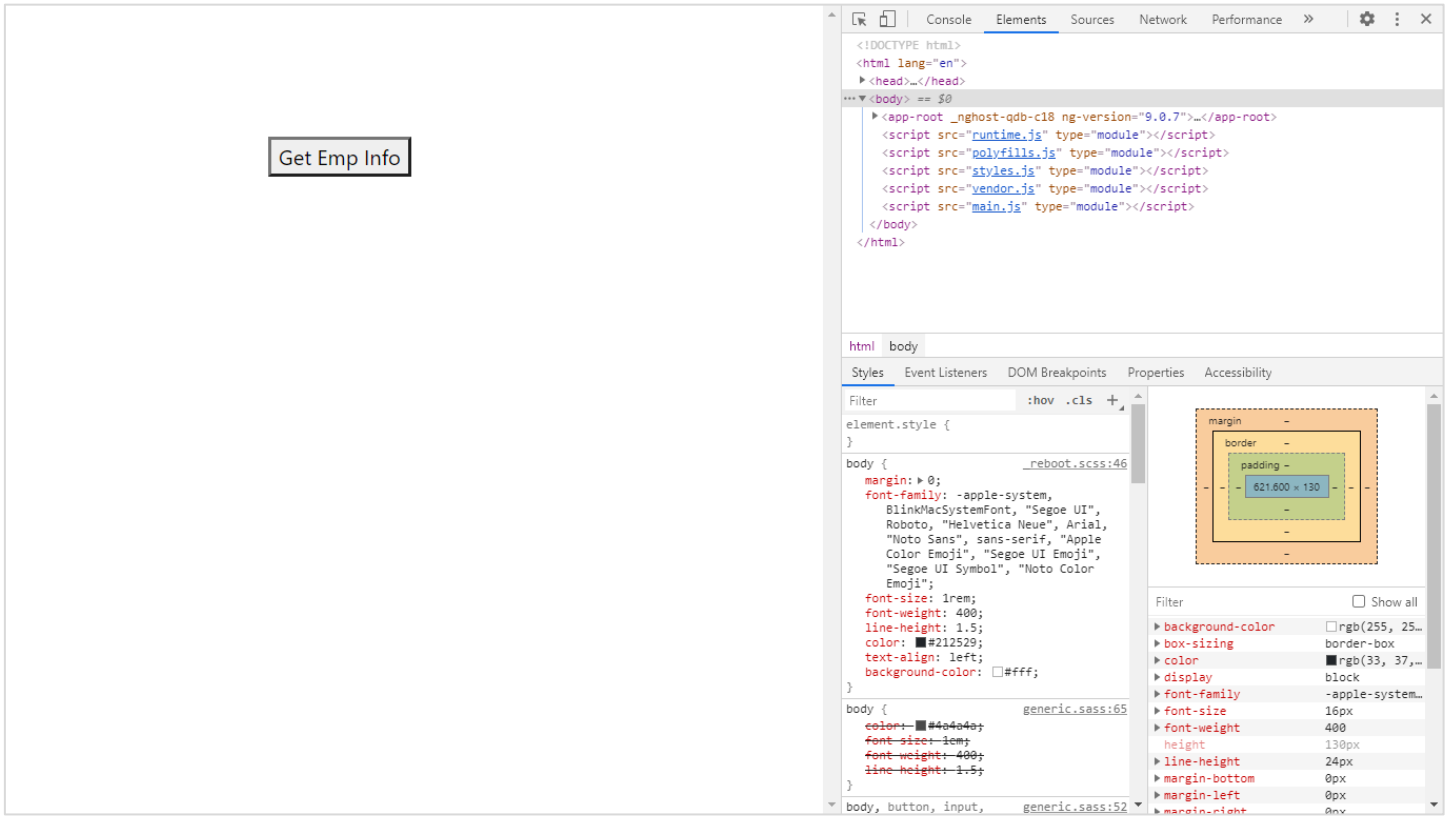
نلاحظ أسندنا له دالة واسميناها trackEmp (لك حرية اختيار الاسم الذي تُريده)، اما في ملف class لهذا component فنكتب محتوى هذه الدالة، كالتالي:

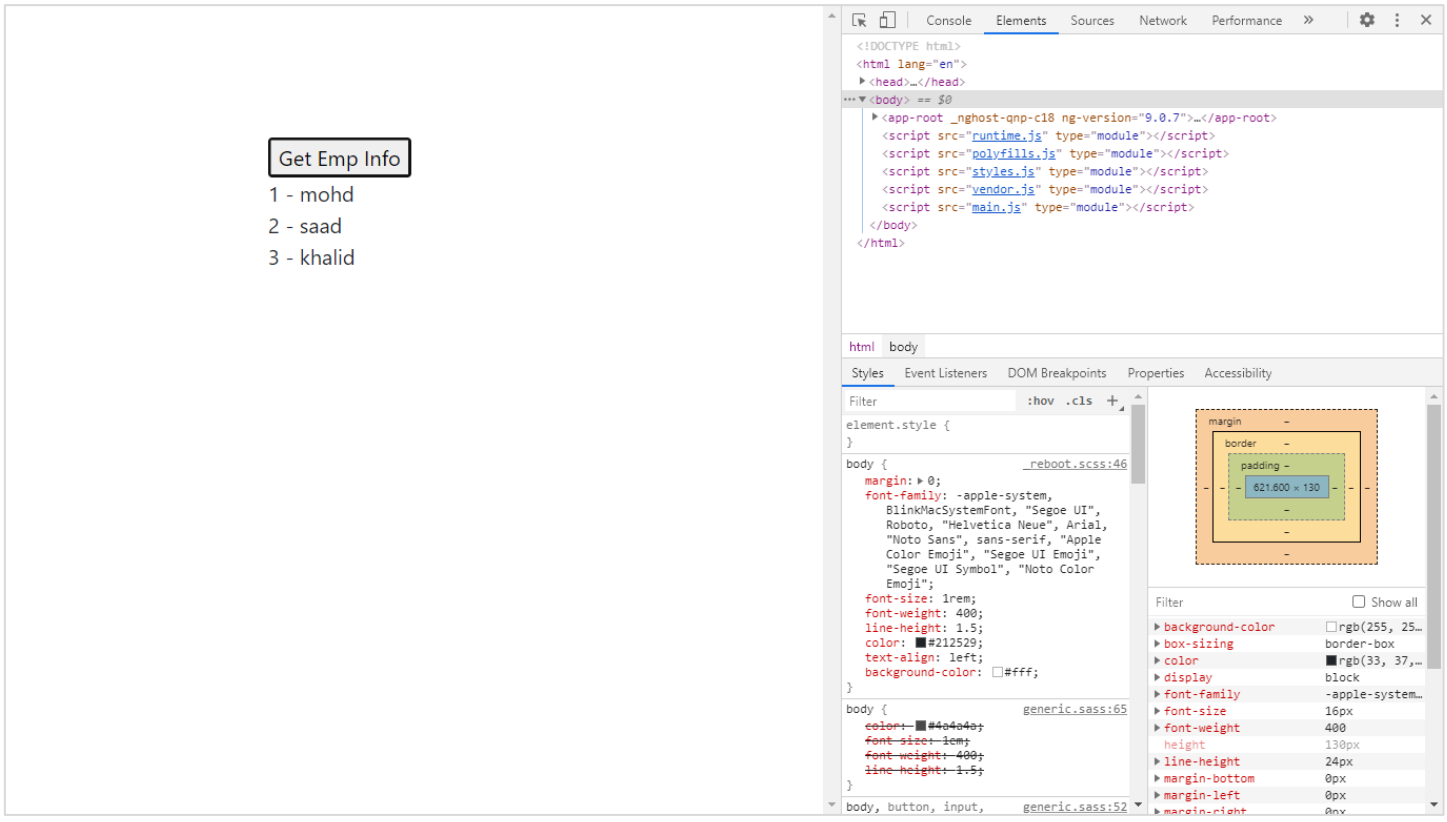
ملف app.component.ts

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})

export class AppComponent {
  empArray = [];

  loadEmp() {
   this.empArray = [
      {emp_no: 1, emp_name: 'mohd'},
      {emp_no: 2, emp_name: 'saad'},
      {emp_no: 3, emp_name: 'khalid'},
    ];
  }

  trackEmp(index, emp) {
    return !emp ? emp.emp_no : undefined;
  }

}
```

وهنا كتبنا محتوى الدالة، حيث وضعنا شرط في حال كان العنصر فقم ببنائه وإن لم يكن فأرجع قيمة undefined.

والآن لنشاهد النتيجة في المتصفح:





عند اول ضغطة سوف تلاحظ انه أضاف العناصر وبنفس الوقت سوف تشاهد وميض في الجزء الأيمن من أدوات المطور الموجودة في متصفح chrome، ولكن عند الضغط مرة أخرى فلن تشاهد أي وميض والسبب انه قام بقراءة المصفوفة ووجدها لم تتغير لذلك أبقى على العناصر كما هي في DOM، وهذا هو السلوك المتوقع والذي نُريده.

وبذلك نكون قمنا بتغطية اغلب الأمور عن هذا Directive، والآن لنقوم بإعطاء مثال عملي، قم بإنشاء مشروع Angular جديد واضف له مكتبة bootstrap (اختياري):

ملف app.component.ts

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})

export class AppComponent {

images = [
  {
    title: 'Beach',
    url: 'https://images.unsplash.com/photo-1507525428034-b723cf961d3e?ixlib=rb-
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=500&q=60'
  },
  {
    title: 'Sunset',
    url: 'https://images.unsplash.com/photo-1494548162494-384bba4ab999?ixlib=rb-
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=500&q=60'
  },
  {
    title: 'Flowers',
    url: 'https://images.unsplash.com/photo-1533907650686-70576141c030?ixlib=rb-
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=500&q=60'
  },
  {
    title: 'Ice',
    url: 'https://images.unsplash.com/photo-1548097160-627fd636ee56?ixlib=rb-
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=500&q=60'
  },
  {
    title: 'Desert',
    url: 'https://images.unsplash.com/photo-1488197047962-b48492212cda?ixlib=rb-
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=500&q=60'
  },
  {
    title: 'Jungle',
    url: 'https://images.unsplash.com/photo-1564460549828-f0219a31bf90?ixlib=rb-
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=500&q=60'
  },
];

}
```

وفي ملف app.component.html، أضف الكود التالي:

ملف app.component.html

```
<nav>
    <ul class="pagination">
        <li class="page-item" *ngFor="let image of images; let i=index">
            <a class="page-link">{{i + 1}}</a>
```

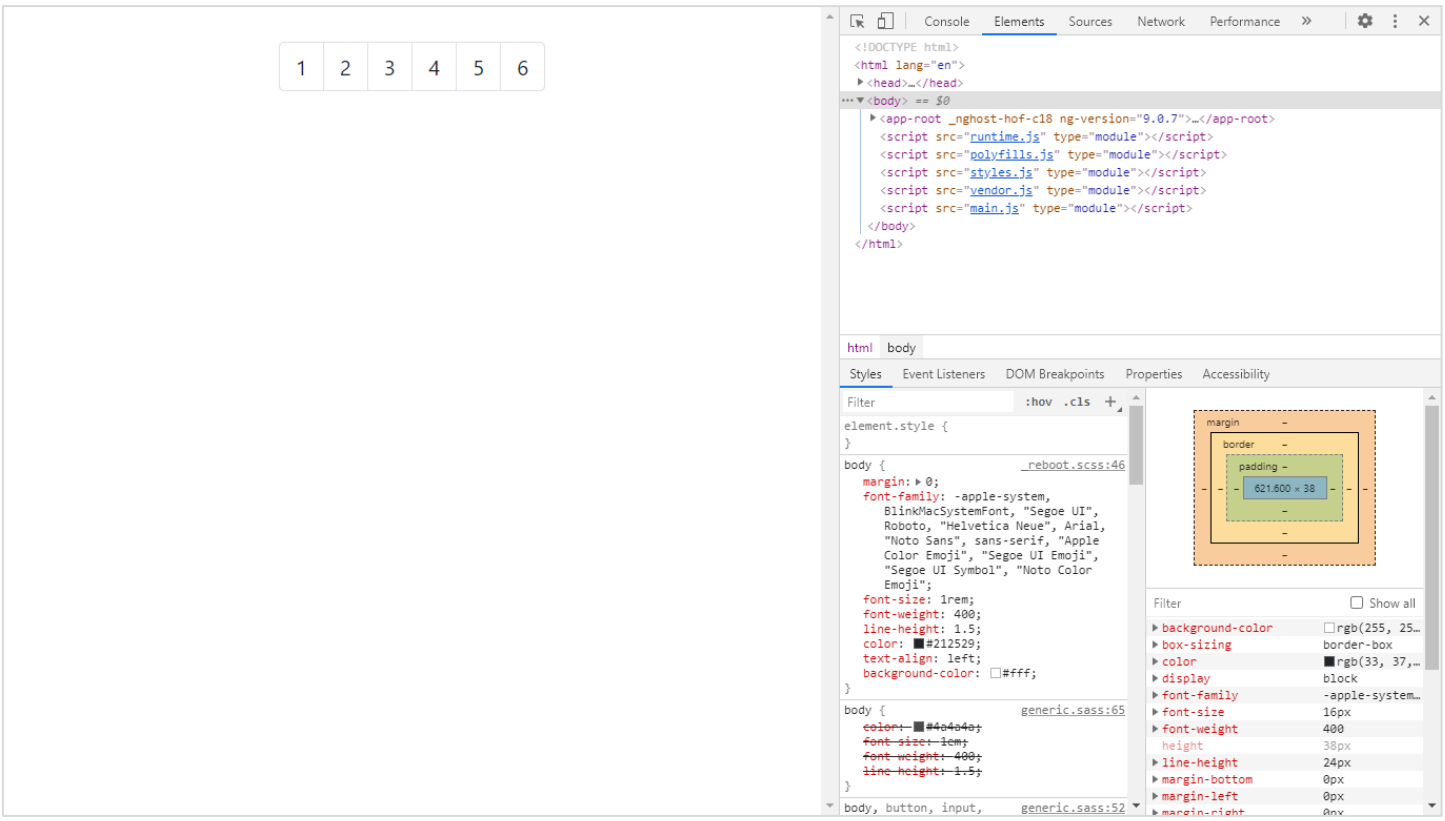
```
        </li>
    </ul>
</nav>
```

وفي ملف app.component.css، فنضيف الكود التالي:

ملف app.component.css

```
nav {
    margin-top: 30px;
}
```

أما النتيجة في المتصفح، فهي كالتالي:



2.2.2. nglf Directive:

وهذا النوع هو ايضاً من Structure Directive حيث يؤثر على بنية العنصر في DOM، ويقوم بحذف أو أضافة هذا العنصر بناءً على شرط معين، ولابد أن يُعيد هذا الشرط قيمة منطقية true او false بحيث إذا كانت true يضيف العنصر إلى DOM وإذا كانت false يقوم بحذف هذا العنصر من DOM.

ويجب التنويه إلى بعض الأمور:

- القيمة !true = false وتعني في حال كان لدينا متغير من النوع Boolean وليكن اسمه isEmpty فأذا وضعنا قبله !isEmpty فنقصد فيها القيمة false وهو مساوي isEmpty === false، ونفس الوضع بالنسبة إذا كانت القيمة true، فلو كتبنا اسم المتغير فقط isEmpty فنقصد بها ان قيمته true وهو مساوي isEmpty === true.

- وفي حال كان المتغير من النوع النصي string واردنا حذف او بناء عنصر معين في DOM بناءً على هذا المتغير، وليكن على سبيل المثال اسم المتغير name فنستطيع كتابة اسم المتغير فقط name في شرط وسوف يقوم Angular ببقية العمل نيابة عنك بحيث يبحث عن هذا المتغير وفي حال كان يحتوي على قيمة فانه يُعيد القيمة true وفي حال كان فارغ لا يحتوي على أي قيمة فانه يُعيد false.
- كما اننا نستطيع كتابة الشرط على شكل دالة وهذه الدالة تُعيد قيمة منطقية true او false.
- كما نستطيع كتابة مقارنة منطقية ونتيجة هذه المقارنة تُعيد true في حال تحقق هذه المقارنة او false في حال عدم تحققها، كأن يكون لدينا متغيرين x و y ونضع الشرط x===y ففي حال كانا متساويين يُعيد true والعكس صحيح.

وأقصر طريق لفهم أي جزئية في البرمجة هي من خلال المثال العملي لذلك لنذهب إلى ملف app.component.html، ونضيف الكود التالي:

ملف app.component.html

```
<nav>
    <ul class="pagination">
        <li class="page-item">
            <a class="page-link">Prev</a>
        </li>
        <li class="page-item" *ngFor="let image of images; let i=index">
            <a class="page-link">{{i + 1}}</a>
        </li>
        <li class="page-item">
            <a class="page-link">Next</a>
        </li>
    </ul>
</nav>
```

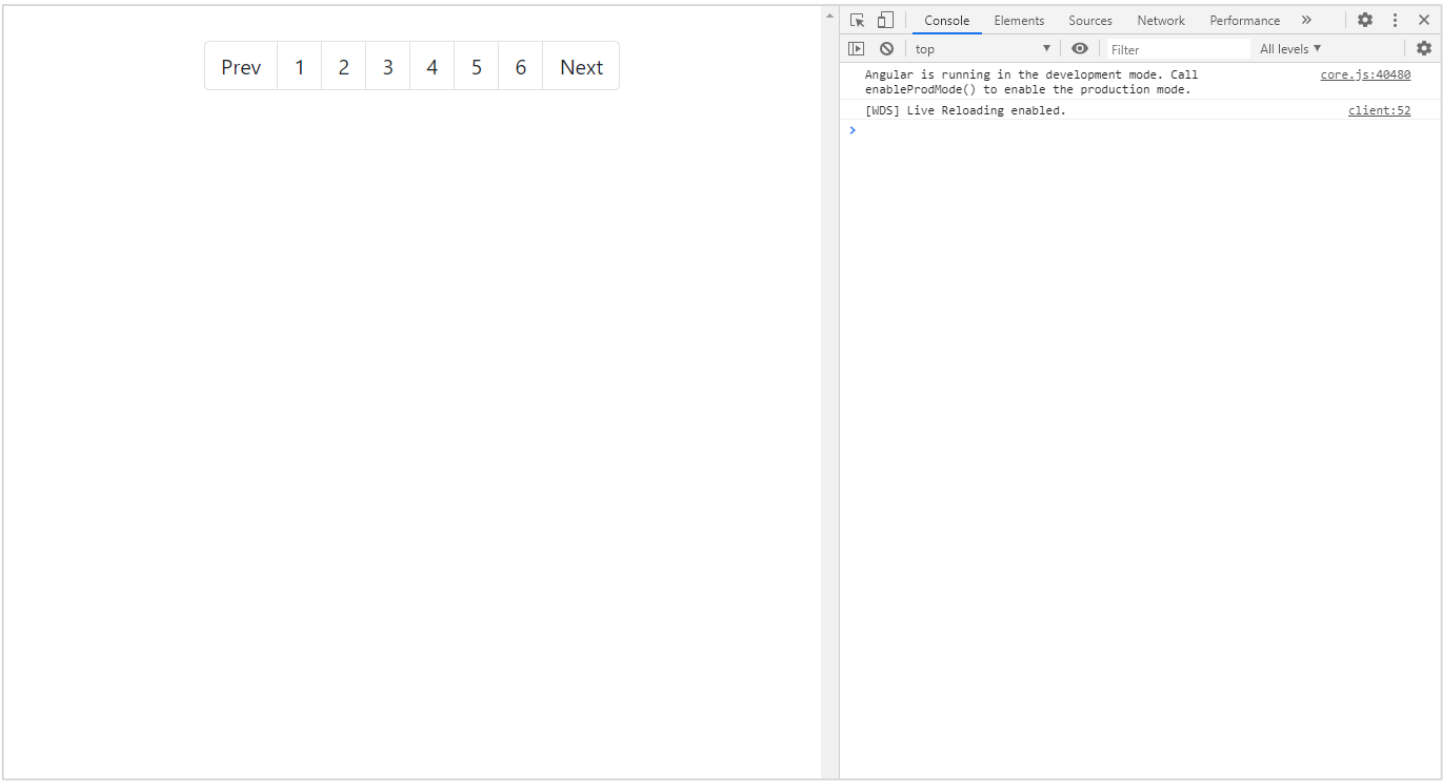
ولنعرف متغير في ملف app.component.ts بحيث يقوم بتخزين index الحالي لكل عنصر من عناصر المصفوفة، لذلك لنقم بتعريفه الآن وفي وقت لاحق نتعامل معه ليقوم بتخزين index، وليكن اسم هذا المتغير currentPage ونسند له القيمة صفر، كالتالي:

ملف app.component.ts

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})

export class AppComponent {
  currentPage = 0;
  images = [
    {
      title: 'Beach',
```

```
      url: 'https://images.unsplash.com/photo-1507525428034-b723cf961d3e?ixlib=rb-
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=500&q=60'
    },
    {
      title: 'Sunset',
      url: 'https://images.unsplash.com/photo-1494548162494-384bba4ab999?ixlib=rb-
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=500&q=60'
    },
    {
      title: 'Flowers',
      url: 'https://images.unsplash.com/photo-1533907650686-70576141c030?ixlib=rb-
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=500&q=60'
    },
    {
      title: 'Ice',
      url: 'https://images.unsplash.com/photo-1548097160-627fd636ee56?ixlib=rb-
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=500&q=60'
    },
    {
      title: 'Desert',
      url: 'https://images.unsplash.com/photo-1488197047962-b48492212cda?ixlib=rb-
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=500&q=60'
    },
    {
      title: 'Jungle',
      url: 'https://images.unsplash.com/photo-1564460549828-f0219a31bf90?ixlib=rb-
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=500&q=60'
    },
  ];
}
```

الآن لنرى النتيجة إلى الآن في المتصفح من ثم نكمل الباقي:



الان نريد ان نخزن قيمة index في المتغير currentPage بحيث إذا ضغط المستخدم على رقم 4 نخزن الindex لهذا الرقم والذي هو 3 في المتغير currentPage، وبنفس الوقت نريد أن نضيف الكلاس active (وهو من كلاسات bootstrap) عن طريق class binding التي شرحناها في الأقسام السابقة والفائدة منها لكي يعطي انطباع للمستخدم انه تم الضغط على هذا العنصر عن طريق تغيير لون الخلفية له، وأخيراً عند الضغط على زر Next نزيد قيمة currentPage بواحد وعند الضغط على زر Prev ننقص قيمة المتغير بواحد، كالتالي:

ملف app.component.html

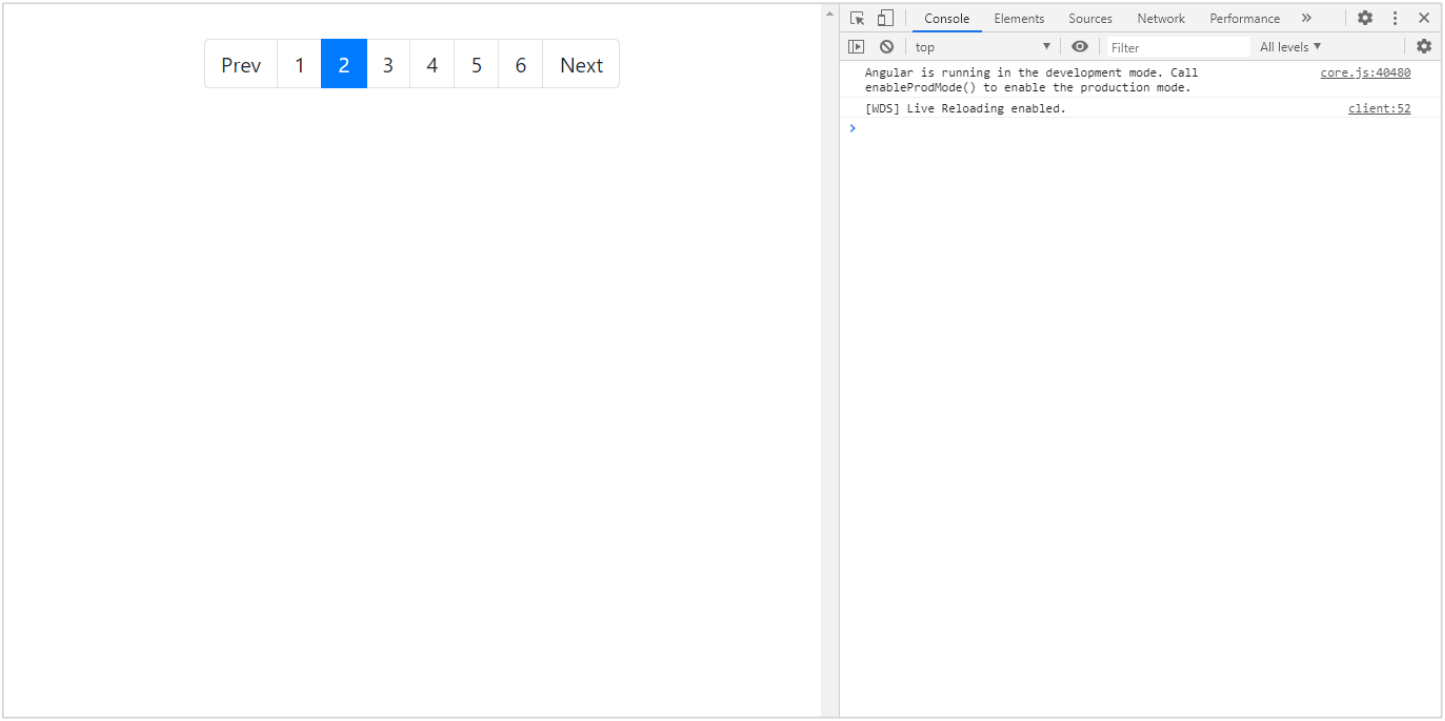
```
<nav>
    <ul class="pagination">
        <li class="page-item">
            <a class="page-link" (click)="currentPage = currentPage - 1">
                Prev
            </a>
        </li>
        <li
          class="page-item"
          *ngFor="let image of images; let i=index"
          [class.active]="i===currentPage"
        >
            <a (click)="currentPage = i" class="page-link">{{i + 1}}</a>
        </li>
        <li class="page-item">
            <a class="page-link" (click)="currentPage = currentPage + 1">
                Next
            </a>
        </li>
    </ul>
</nav>
```

نضيف الكلاس active عن طريق class binding في حال تحقق الشرط

هنا نأخذ قيمة index لكل عنصر مع كل ضغطة على هذا العنصر ونخزنها في المتغير

هنا نضيف أن ننقص قيمة المتغير

ولنرى النتيجة في المتصفح، كالتالي:



جميع ما قمنا به هو تهيئة لكي نتعامل مع nglf، حيث نريد في حال وصل المستخدم إلى آخر زر وهو في حالتنا هذه الزر الذي يحمل القيمة 6 أن يقوم بحذف الزر Next وإذا كان الزر المختار اقل من 6 يعيد إضافة الزر Next ونفس الوضع مع الزر Prev ولكن مع الزر الذي يحمل القيمة 1.

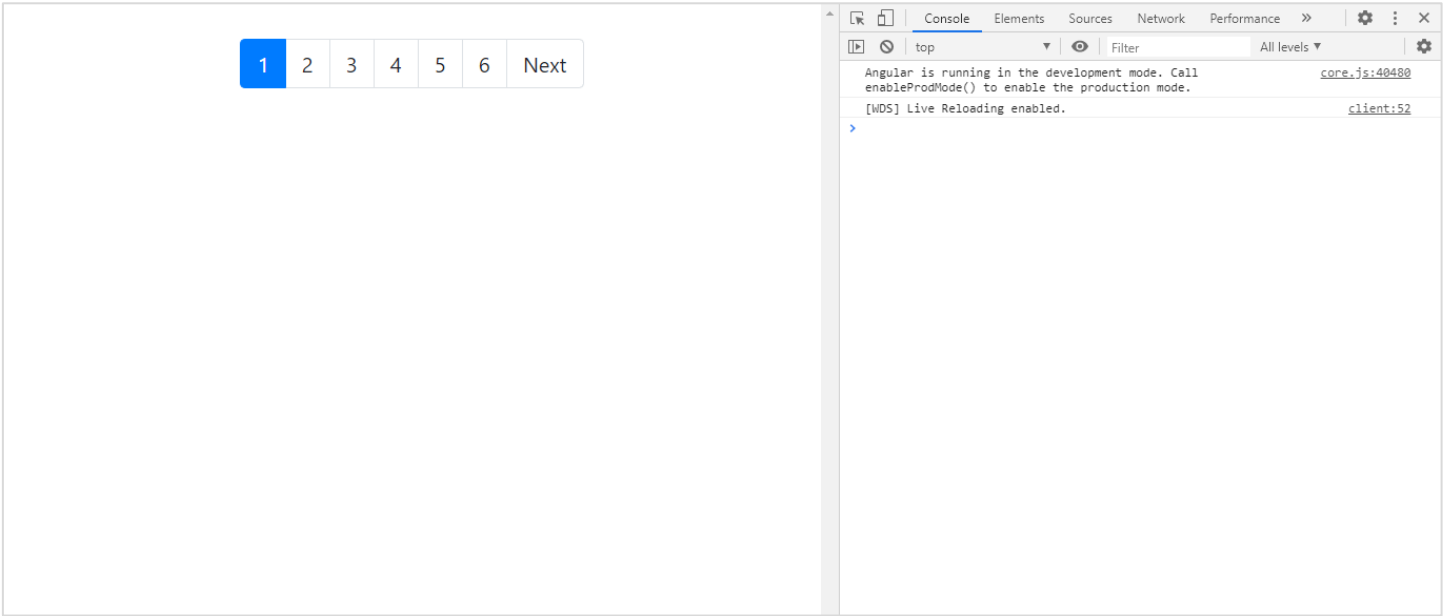
ونستطيع ان نعمل هذا الأمر بكل بساطة عن طريق nglf، كالتالي:

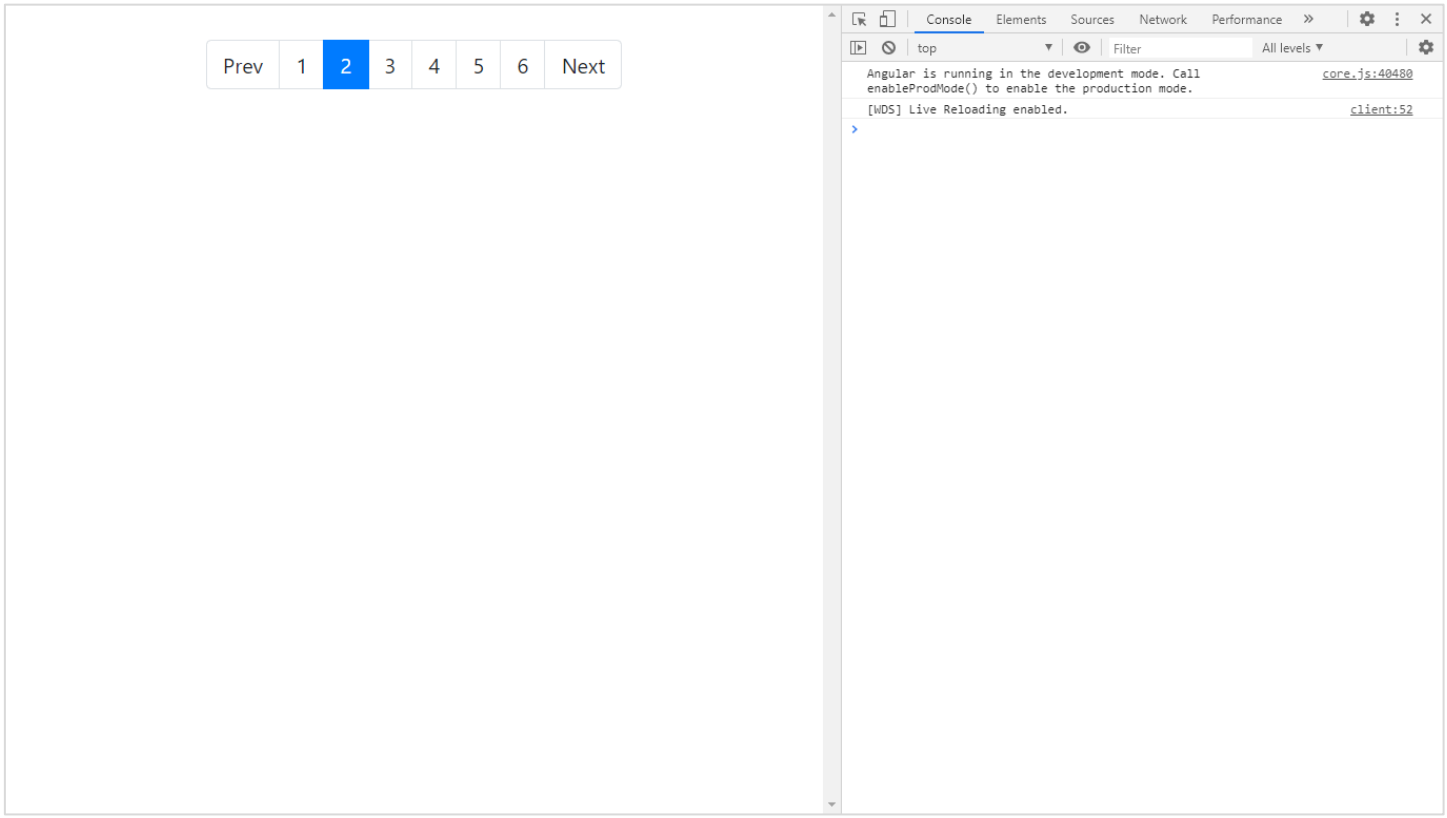
ملف app.component.html

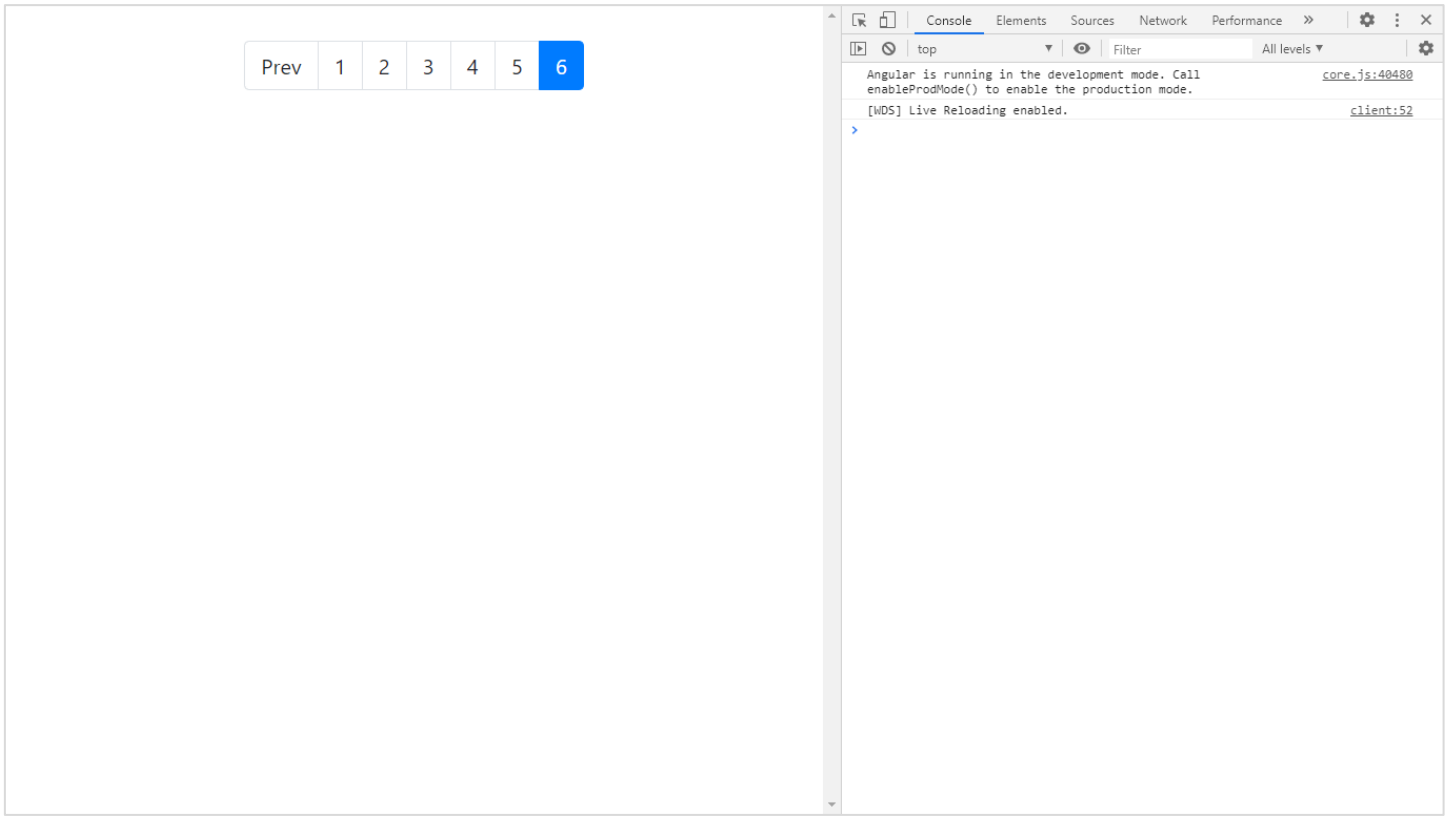
```
<nav>
    <ul class="pagination">
        <li class="page-item" *ngIf="currentPage > 0">
            <a class="page-link" (click)="currentPage = currentPage - 1">
                Prev
            </a>
        </li>
        <li
          class="page-item"
          [class.active]="i===currentPage"
          *ngFor="let image of images; let i=index"
         >
            <a (click)="currentPage = i" class="page-link">{{i + 1}}</a>
        </li>
        <li class="page-item" *ngIf="currentPage < images.length - 1">
            <a class="page-link" (click)="currentPage = currentPage + 1">
                Next
            </a>
        </li>
    </ul>
</nav>
```

هنا وضعنا شرط في حال تحقق الشرط اضف العنصر li وهو ان قيمة المتغير اكبر من الصفر أي انه موجود على العنصر رقم 2 لأن هذا العنصر يحمل index ذو القيمة 1، والعكس صحيح مع الزر Next.

الآن لنرى النتيجة على المتصفح:







وهنالك صياغة أخرى لجملة nglf بحيث يتم استخدامها بدلاً من استخدام اثنين nglf نستطيع استخدام nglf واحدة عن طريق الاستفادة من selector ذو الاسم <ng-template> وهو عبارة عن selector خاص يقدمه لنا angular وفي الغالب يتم استخدامه من Structure Directives حيث من مهامه أي أكواد html يتم كتابتها داخله لن تظهر للمستخدم إلا في حال تحقق الشرط عن طريق nglf او ngSwitch، كالتالي:

لنفرض أن لدينا مصفوفة تحتوي على مجموعة من بيانات الموظفين وليكن اسمها empNames=['mohd', 'saad', 'bader'] ونريد أن نظهر هذه البيانات في حال كان هنالك بيانات عن طريق nglf وngFor، وفي الطريقة التقليدية نستخدم الكود التالي:

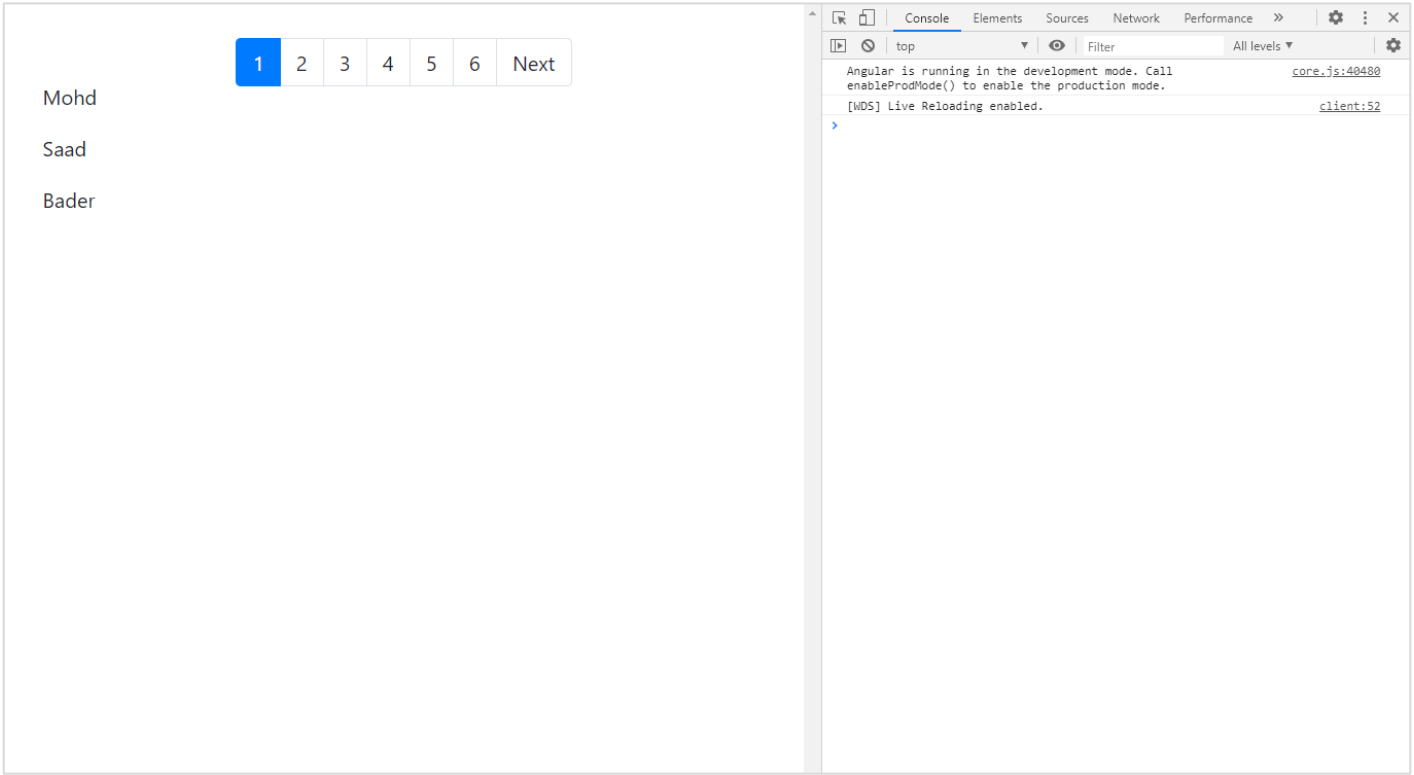
ملف app.component.ts

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {

  empNames = ['Mohd', 'Saad', 'Bader'];
  currentPage = 0;

  images = [
    {
      title: 'Beach',
      url: 'https://images.unsplash.com/photo-1507525428034-b723cf961d3e?ixlib=rb-
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=500&q=60'
    },
    {
      title: 'Sunset',
      url: 'https://images.unsplash.com/photo-1494548162494-384bba4ab999?ixlib=rb-
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=500&q=60'
    },
    {
      title: 'Flowers',
      url: 'https://images.unsplash.com/photo-1533907650686-70576141c030?ixlib=rb-
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=500&q=60'
    },
    {
      title: 'Ice',
      url: 'https://images.unsplash.com/photo-1548097160-627fd636ee56?ixlib=rb-
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=500&q=60'
    },
    {
      title: 'Desert',
      url: 'https://images.unsplash.com/photo-1488197047962-b48492212cda?ixlib=rb-
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=500&q=60'
    },
    {
      title: 'Jungle',
      url: 'https://images.unsplash.com/photo-1564460549828-f0219a31bf90?ixlib=rb-
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=500&q=60'
    },
  ];
}
```

app.component.html ملف

```
<nav>
    <ul class="pagination">
        <li class="page-item" *ngIf="currentPage > 0">
            <a class="page-link" (click)="currentPage = currentPage - 1">
                Prev
            </a>
        </li>
        <li
            class="page-item"
            [class.active]="i===currentPage"
            *ngFor="let image of images; let i=index"
        >
            <a (click)="currentPage = i" class="page-link">{{i + 1}}</a>
        </li>
        <li class="page-item" *ngIf="currentPage < images.length - 1">
            <a class="page-link" (click)="currentPage = currentPage + 1">
                Next
            </a>
        </li>
    </ul>
</nav>


<div *ngIf="empNames">
    <p *ngFor="let emp of empNames">{{ emp }}</p>
</div>
<div *ngIf="!empNames">
    <p>No Employees Names</p>
</div>
```

والنتيجة:



هذا بالنسبة للطريقة الاعتيادية ولكن نستطيع اختصار الكود ليصبح بهذه الطريقة:

جزء من ملف app.component.html

```
<div *ngIf="empNames else NoNames">
    <p *ngFor="let emp of empNames">{{ emp }}</p>
</div>

<ng-template #NoNames>
    <p>No Employees Names</p>
</ng-template>
```

نلاحظ أننا اضفنا <ng-template> واضفنا له template variable اسميناه #NoNames واستغنينا عن ngIf الثانية، اما في ngIf الأولى استخدمنا else ومررنا template variable لي <ng-template>، ولو حاولنا قراءة الشرط بطريقة مبسطة، قم بإظهار هذا div إذا كان هنالك بيانات في empNames وإذ لم يكن أظهر template الذي يحمل المتغير NoNames.

وايضاً نستطيع اختصار هذا الكود بشكل أفضل للقراء كالتالي:

جزء من ملف app.component.html

```
<div *ngIf="empNames then Names else NoNames"></div>

<ng-template #Names>
    <p *ngFor="let emp of empNames">{{ emp }}</p>
</ng-template>

<ng-template #NoNames>
    <p>No Employees Names</p>
</ng-template>
```

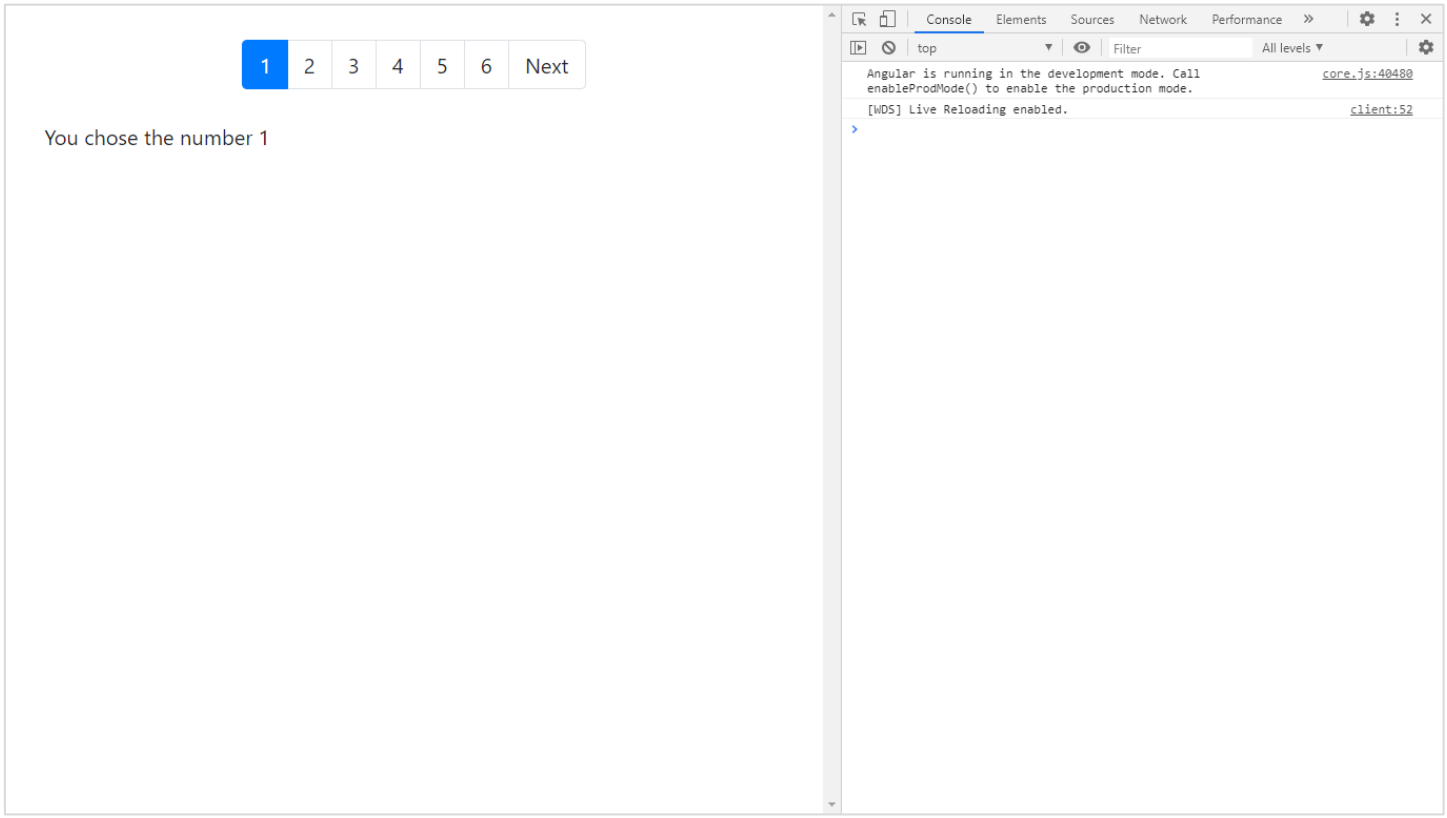
3.2.2. ngSwitch Directive:

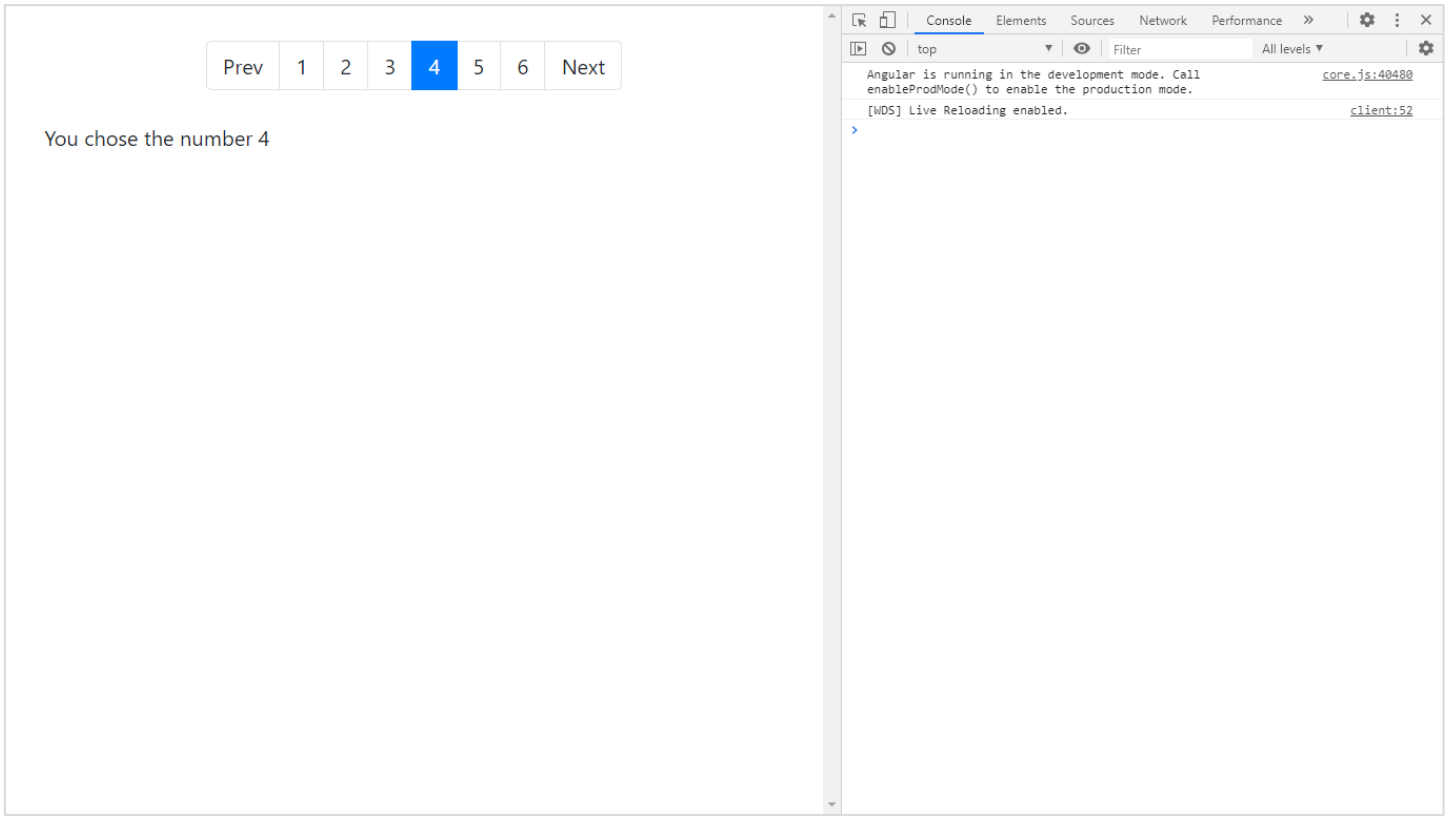
وهي مشابهه لي switch case الموجودة في لغلب لغات البرمجة، ولكن هنا تم تطويرها لاستخدامها في Template بحيث يتم إضافة او حذف عنصر بناءً على شرط معين، وتستخدم في حال كان لدينا مجموعة من الاحتمالات وليس احتمالين كما في ngIf.

مثال/

app.component.html ملف

```
<nav>
    <ul class="pagination">
        <li class="page-item" *ngIf="currentPage > 0">
            <a class="page-link" (click)="currentPage = currentPage - 1">
                Prev
            </a>
        </li>
        <li class="page-item"
            [class.active]="i===currentPage"
            *ngFor="let image of images; let i=index"
        >
            <a (click)="currentPage = i" class="page-link">{{i + 1}}</a>
        </li>
        <li class="page-item" *ngIf="currentPage < images.length - 1">
            <a class="page-link" (click)="currentPage = currentPage + 1">
                Next
            </a>
        </li>
    </ul>
    <div [ngSwitch]="currentPage">
        <div *ngSwitchCase="0">
            <p>You chose the number {{currentPage + 1 }}</p>
        </div>
        <div *ngSwitchCase="1">
            <p>You chose the number {{currentPage + 1 }}</p>
        </div>
        <div *ngSwitchCase="2">
            <p>You chose the number {{currentPage + 1 }}</p>
        </div>
        <div *ngSwitchCase="3">
            <p>You chose the number {{currentPage + 1 }}</p>
        </div>
        <div *ngSwitchCase="4">
            <p>You chose the number {{currentPage + 1 }}</p>
        </div>
        <div *ngSwitchCase="5">
            <p>You chose the number {{currentPage + 1 }}</p>
        </div>
        <div *ngSwitchCase="6">
            <p>You chose the number {{currentPage + 1 }}</p>
        </div>
        <div *ngSwitchDefault>
            <p>Unknown current page !</p>
        </div>
    </div>
</nav>
```





وبذلك نكون انتهينا من شرح اهم ثلاث أنواع في Structure Directives وهي ngFor وngIf وngSwitch.

3.2. Attribute Directives:

وهذا النوع يقوم بتغيير وتعديل الخصائص الخاصة بالعنصر في DOM، ومن أشهرها نوعين ngStyle وngClass.

3.2.1. ngClass:

وتقوم بإضافة مجموعة من الكلاسات لعنصر معين على شكل كائن Object { } وفق شروط معينة، ونستطيع إضافة اكثر من شرط لكل كلاس، وتكون الخاصية لهذا object هو اسم الكلاس والقيمة هي الشرط.

الصيغة العامة:

```
[ngClass]="{class1: condition1 OR (some conditions), class2, condition2 OR (some conditions) ...etc.}"
```

مثال:

لنقم بتغيير Class Binding الذي عملناه سابقاً بngClass، مع العلم ان Class Binding (لفهم Class Binding الرجاء مراجعة الكتاب الثاني من هذه السلسلة Angular Components and Services) مشابه لي ngClass والفرق الوحيد ان ngClass نستطيع ان نمرر لها اكثر من كلاس وليس كلاس واحد فقط، الآن لنقم بتطبيق المثال، كالتالي:

ملف app.component.html

```
<nav>
    <ul class="pagination">
        <li class="page-item" *ngIf="currentPage > 0">
            <a class="page-link" (click)="currentPage = currentPage - 1">
                Prev
            </a>
        </li>
        <li
            class="page-item"
            [ngClass]="{active: i===currentPage}"
            *ngFor="let image of images; let i=index"
        >
            <a (click)="currentPage = i" class="page-link">{{i + 1}}</a>
        </li>
        <li class="page-item" *ngIf="currentPage < images.length - 1">
            <a class="page-link" (click)="currentPage = currentPage + 1">
                Next
            </a>
        </li>
    </ul>
</nav>
```

ولو ذهبنا إلى المتصفح لوجدنا نفس النتيجة.

ngStyle .2.3.2:

وهي مشابهه لي ngClass ولكن الفرق هنا نقوم بتمرير خواص css بشكل مباشر وتنفذ وفق شرط محدد بدلاً من إضافة الكلاسات، كما انها تشبه Style Binding ولكن الفرق الوحيد ngStyle نستطيع تمرير أكثر من خاصية من خواص css بدلاً من خاصية واحدة فقط.

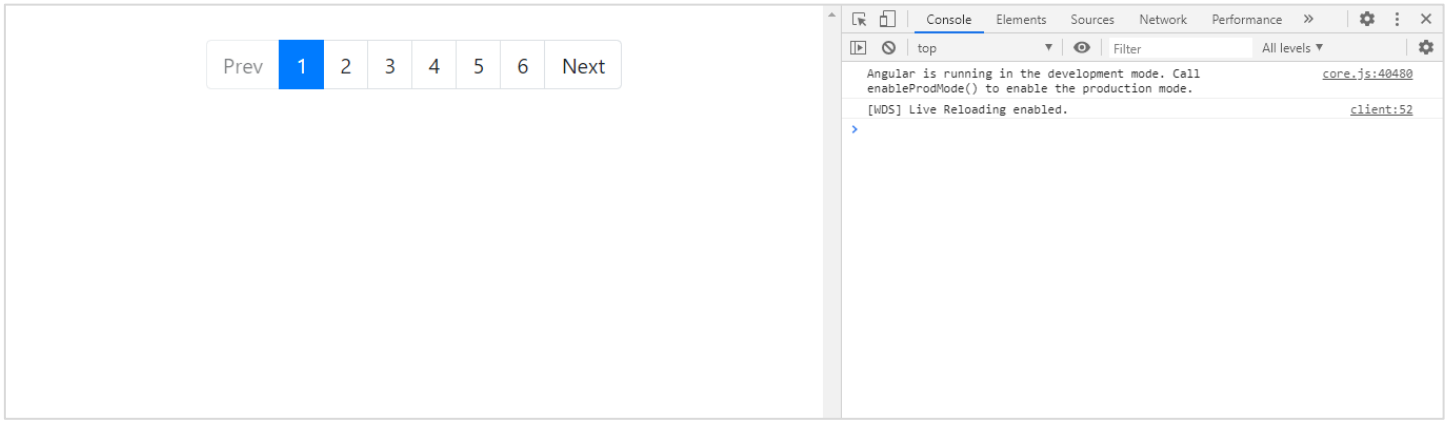
مثال/ لنقوم بتعطيل الزر Next وPrev عن طريق ngStyle بدلاً من اخفائهم عن طريق ngIf، كالتالي:

ملف app.component.html

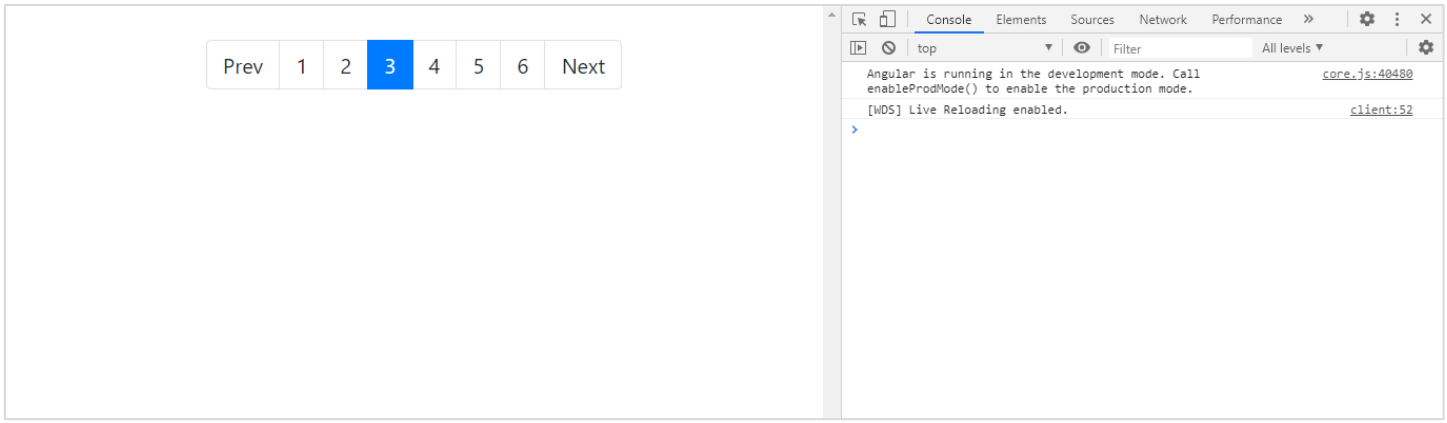
```
<nav>
    <ul class="pagination">
        <li class="page-item" [ngStyle]="{
            'opacity': currentPage > 0 ? 1 : 0.6,
            'pointer-events':  currentPage > 0 ? 'auto' : 'none'
            }"
        >
            <a class="page-link" (click)="currentPage = currentPage - 1">
                Prev
            </a>
        </li>
        <li
            class="page-item"
            [ngClass]="{active: i===currentPage}"
            *ngFor="let image of images; let i=index"
        >
            <a (click)="currentPage = i" class="page-link">{{i + 1}}</a>
        </li>
        <li class="page-item" [ngStyle]="{
            'opacity': currentPage < images.length - 1 ? 1 : 0.6,
            'pointer-events':  currentPage < images.length - 1 ? 'auto' : 'none'
            }"
        >
            <a class="page-link" (click)="currentPage = currentPage + 1">
                Next
            </a>
        </li>
    </ul>
</nav>
```

يجب ملاحظة انه ان الخاصية pointer-events هي من خواص css يمكن كتابتها كما هي او نستطيع كتابتها pointerEvents وهذا ينطبق على أي خاصية تتكون من كلمتين.

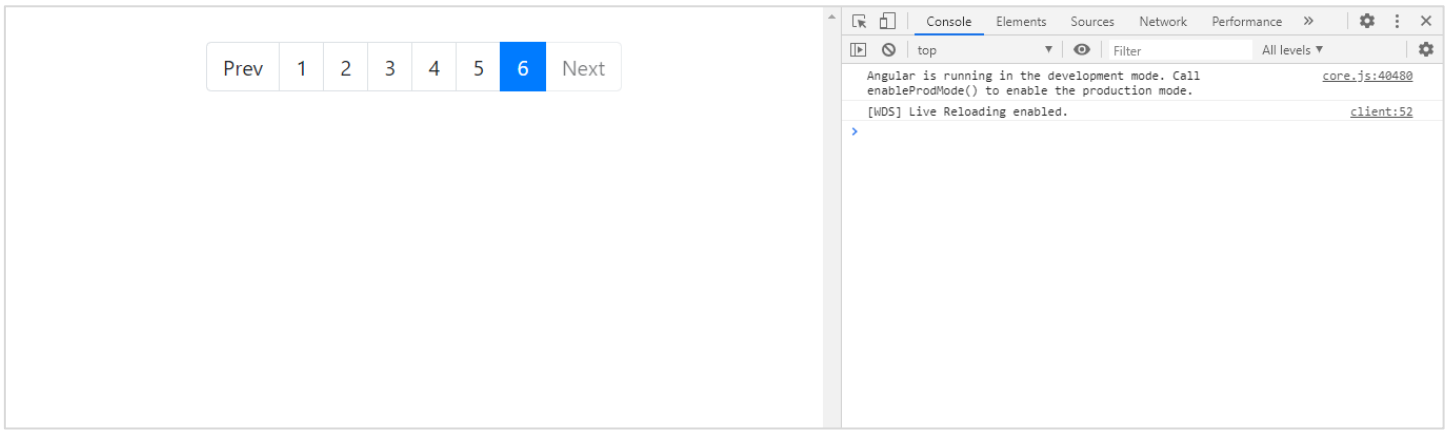
الآن لنشاهد النتيجة، على المتصفح:



```
Angular is running in the development mode. Call      core.js:40480
enableProdMode() to enable the production mode.
[WDS] Live Reloading enabled.                          client:52
```



```
Angular is running in the development mode. Call      core.js:40480
enableProdMode() to enable the production mode.
[WDS] Live Reloading enabled.                          client:52
```



```
Angular is running in the development mode. Call      core.js:40480
enableProdMode() to enable the production mode.
[WDS] Live Reloading enabled.                          client:52
```

4.2. Multiple Structure Directives with Ng-Container:

قبل البدء بشرح كيف يمكننا ان نضيف أكثر من Structure Directive لنفس العنصر، لنقم بأجراء بعض التعديلات على الكود، كالتالي:

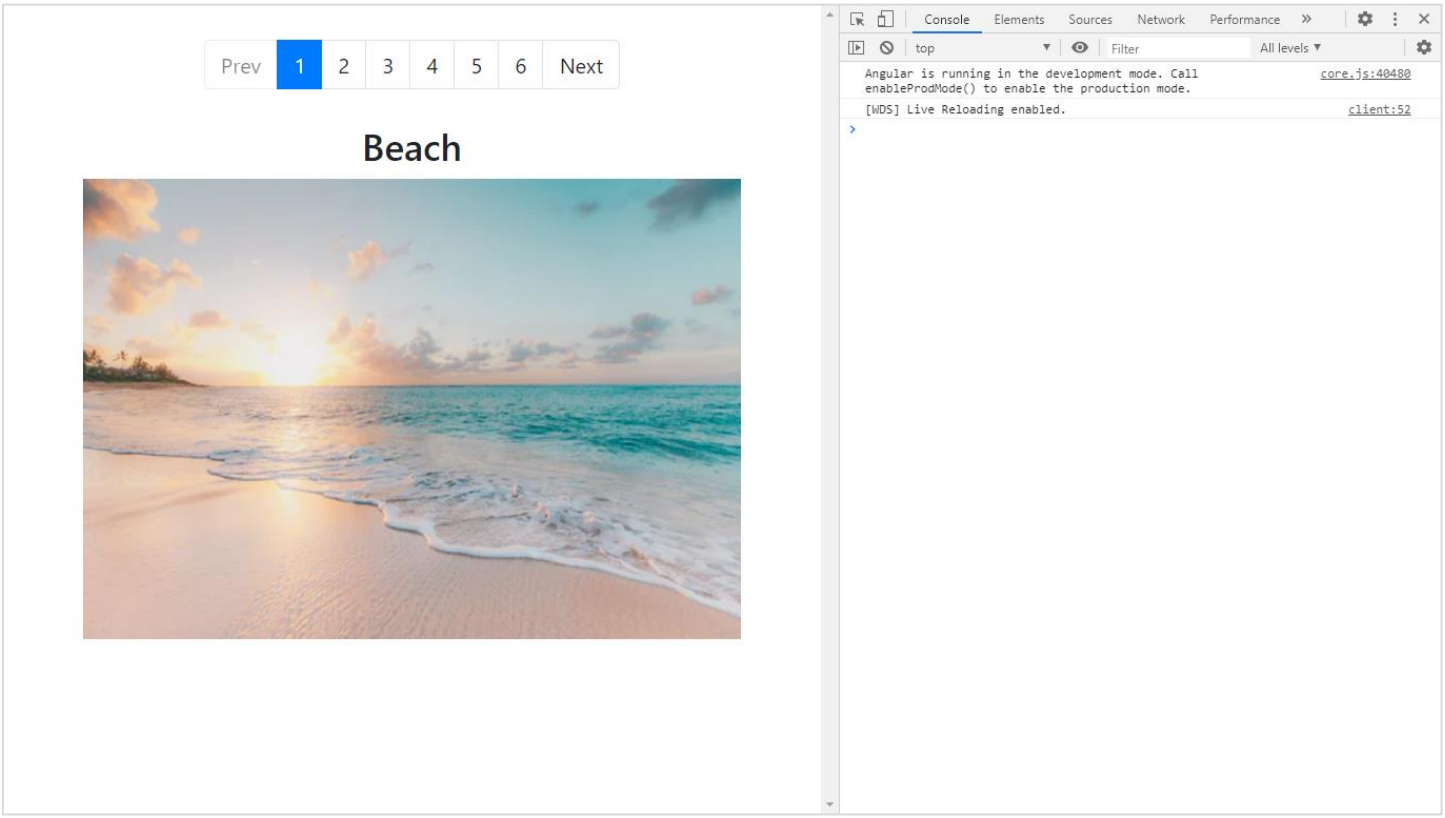
ملف app.component.html

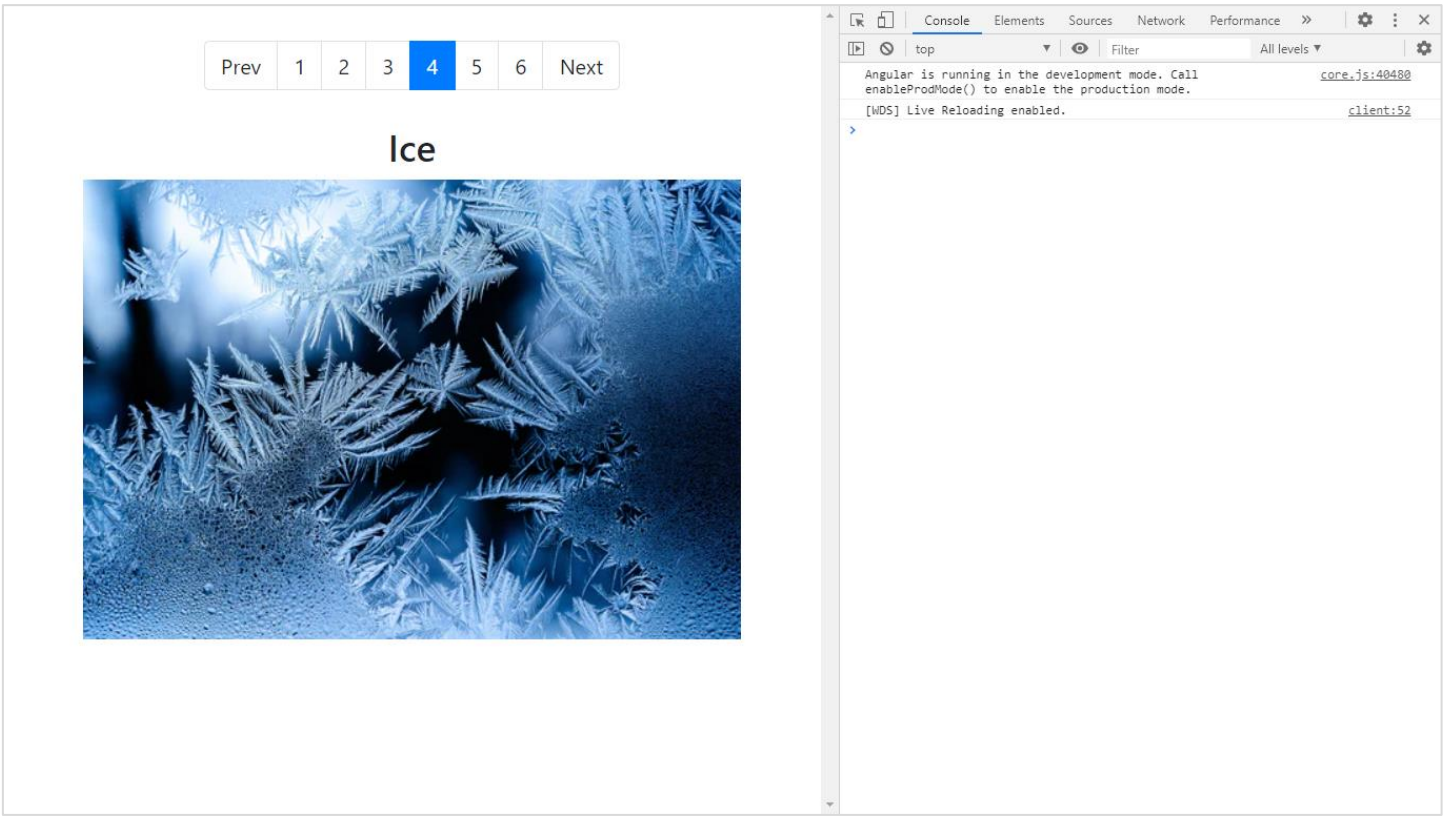
```
<nav>
    <ul class="pagination">
        <li class="page-item" [ngStyle]="{
            'opacity': currentPage > 0 ? 1 : 0.6,
            'pointerEvents':  currentPage > 0 ? 'auto' : 'none'
            }">
            <a class="page-link" (click)="currentPage = currentPage - 1">
                Prev
            </a>
        </li>
        <li class="page-item"
```

```
            [ngClass]="{active: i===currentPage}"
            *ngFor="let image of images; let i=index"
        >
            <a (click)="currentPage = i" class="page-link">{{i + 1}}</a>
        </li>
        <li class="page-item" [ngStyle]="{
            'opacity': currentPage < images.length - 1 ? 1 : 0.6,
            'pointer-events':  currentPage < images.length - 1 ? 'auto' : 'none'
            }"
        >
            <a class="page-link" (click)="currentPage = currentPage + 1">
                Next
            </a>
        </li>
    </ul>
</nav>

<div class="img-box">
    <h3>{{ images[currentPage].title }}</h3>
    <img [src]="images[currentPage].url" />
</div>
```

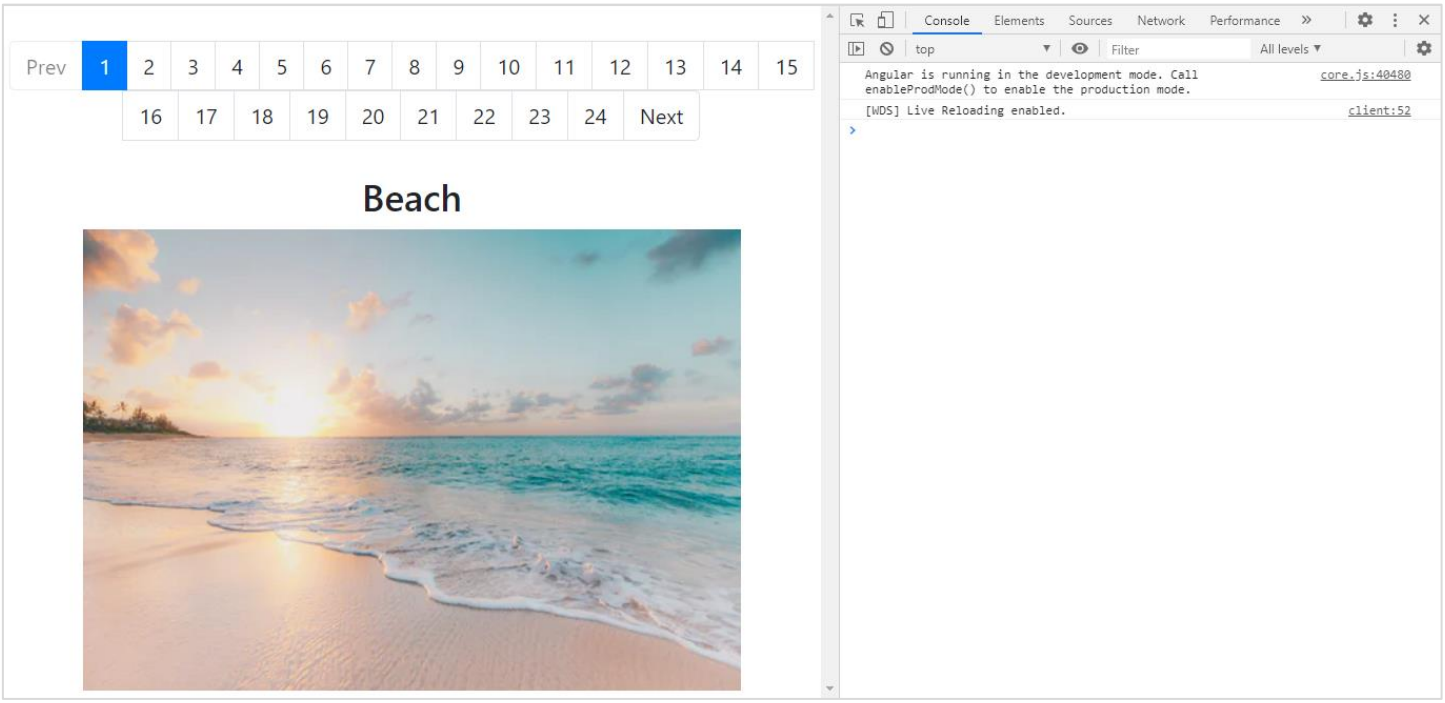
app.component.css ملف

```
nav {
    margin-top: 30px;
}

.img-box {
    text-align: center;
    padding-top: 30px;
}

.img-box img {
    width: 500px;
    height: 350px;
}
```





نلاحظ انه لدينا ست عناصر فقط ولكن ماذا لو ضاعفنا هذه الأعداد في المصفوفة images، فسوف يصبح التطبيق، كالتالي:



نلاحظ أن عدد الصفحات أصبح كثير، ومن الحلول والتي تخدمنا لتوضيح ما نريد توضيحه هنا هو استخدام ngIf بنفس العنصر الذي يوجد به ngFor ولكن هنا تظهر لنا الإشكالية وهي انه لا يمكن استخدام أكثر من Structure Directive لنفس العنصر، لذلك لحل هذه المشكلة هي عن طريق ng-container وهي بمثابة حاوية افتراضية ولا يتم بنائها في DOM ولا تحويلها إلى أي عنصر من عناصر HTML في DOM ايضاً، بمعنى آخر أن Angular سوف يقوم ببناء جميع ما تحتويه من عناصر فقط ولكن لن يتم بناءها هي بنفسها، والفائدة منها لحل بعض الإشكاليات منها هذه الإشكالية وهي استخدام أكثر من Structure Directive لنفس العنصر، كالتالي:

ملف app.component.html

```
<nav>
    <ul class="pagination">
        <li class="page-item" [ngStyle]="{
            'opacity': currentPage > 0 ? 1 : 0.6,
            'pointerEvents':  currentPage > 0 ? 'auto' : 'none'
            }">
            <a class="page-link" (click)="currentPage = currentPage - 1">
                Prev
            </a>
        </li>
        <ng-container *ngFor="let image of images; let i=index">
            <li
                class="page-item"
                [ngClass]="{active: i===currentPage}"
                *ngIf="showFivePages(i)"
            >
                <a (click)="currentPage = i" class="page-link">{{i + 1}}</a>
            </li>
        </ng-container>
        <li class="page-item" [ngStyle]="{
            'opacity': currentPage < images.length - 1 ? 1 : 0.6,
            'pointer-events':  currentPage < images.length - 1 ? 'auto' : 'none'
```

```
            }">
            <a class="page-link" (click)="currentPage = currentPage + 1">
                Next
            </a>
        </li>
    </ul>
</nav>

<div class="img-box">
    <h3>{{ images[currentPage].title }}</h3>
    <img [src]="images[currentPage].url" />
</div>
```

مع اننا وضعنا ngFor في <ng-container> ولكن Angular سوف يقوم ببناء وتكرار العنصر li في DOM، وهذا هو الذي نريده فبذلك نستطيع ان نضع الشرط عن طريق ngIf في العنصر li وبنفس الوقت نقوم ببناء وتكرار عنصر li في DOM.

اما في ملف app.component.ts فنقوم بكتابة محتوى الدالة showFivePages، كالتالي:

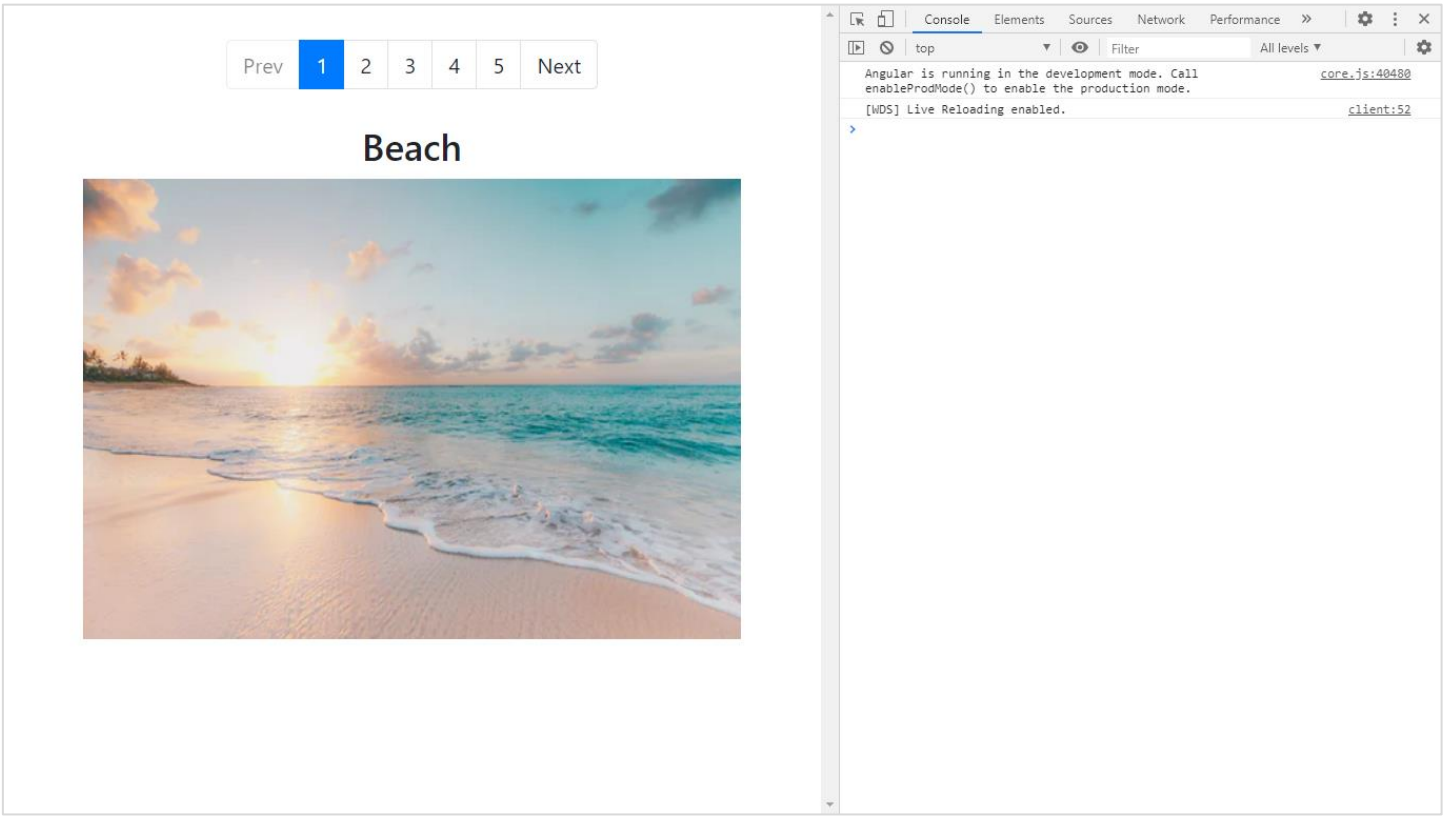
ملف app.component.css

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  currentPage = 0;

  images = [
    {
      title: 'Beach',
      url: 'https://images.unsplash.com/photo-1507525428034-b723cf961d3e?ixlib=rb-
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=500&q=60'
    },
    {
      title: 'Sunset',
      url: 'https://images.unsplash.com/photo-1494548162494-384bba4ab999?ixlib=rb-
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=500&q=60'
    },
    {
      title: 'Flowers',
      url: 'https://images.unsplash.com/photo-1533907650686-70576141c030?ixlib=rb-
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=500&q=60'
    },
    {
      title: 'Ice',
      url: 'https://images.unsplash.com/photo-1548097160-627fd636ee56?ixlib=rb-
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=500&q=60'
    },
    {
      title: 'Desert',
```

```
      url: 'https://images.unsplash.com/photo-1488197047962-b48492212cda?ixlib=rb-
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=500&q=60'
    },
    {
      title: 'Jungle',
      url: 'https://images.unsplash.com/photo-1564460549828-f0219a31bf90?ixlib=rb-
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=500&q=60'
    },    {
      title: 'Beach',
      url: 'https://images.unsplash.com/photo-1507525428034-b723cf961d3e?ixlib=rb-
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=500&q=60'
    },
    {
      title: 'Sunset',
      url: 'https://images.unsplash.com/photo-1494548162494-384bba4ab999?ixlib=rb-
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=500&q=60'
    },
    {
      title: 'Flowers',
      url: 'https://images.unsplash.com/photo-1533907650686-70576141c030?ixlib=rb-
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=500&q=60'
    },
    {
      title: 'Ice',
      url: 'https://images.unsplash.com/photo-1548097160-627fd636ee56?ixlib=rb-
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=500&q=60'
    },
    {
      title: 'Desert',
      url: 'https://images.unsplash.com/photo-1488197047962-b48492212cda?ixlib=rb-
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=500&q=60'
    },
    {
      title: 'Jungle',
      url: 'https://images.unsplash.com/photo-1564460549828-f0219a31bf90?ixlib=rb-
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=500&q=60'
    },    {
      title: 'Beach',
      url: 'https://images.unsplash.com/photo-1507525428034-b723cf961d3e?ixlib=rb-
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=500&q=60'
    },
    {
      title: 'Sunset',
      url: 'https://images.unsplash.com/photo-1494548162494-384bba4ab999?ixlib=rb-
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=500&q=60'
    },
    {
      title: 'Flowers',
      url: 'https://images.unsplash.com/photo-1533907650686-70576141c030?ixlib=rb-
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=500&q=60'
    },
    {
      title: 'Ice',
```

```
      url: 'https://images.unsplash.com/photo-1548097160-627fd636ee56?ixlib=rb-
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=500&q=60'
    },
    {
      title: 'Desert',
      url: 'https://images.unsplash.com/photo-1488197047962-b48492212cda?ixlib=rb-
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=500&q=60'
    },
    {
      title: 'Jungle',
      url: 'https://images.unsplash.com/photo-1564460549828-f0219a31bf90?ixlib=rb-
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=500&q=60'
    },    {
      title: 'Beach',
      url: 'https://images.unsplash.com/photo-1507525428034-b723cf961d3e?ixlib=rb-
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=500&q=60'
    },
    {
      title: 'Sunset',
      url: 'https://images.unsplash.com/photo-1494548162494-384bba4ab999?ixlib=rb-
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=500&q=60'
    },
    {
      title: 'Flowers',
      url: 'https://images.unsplash.com/photo-1533907650686-70576141c030?ixlib=rb-
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=500&q=60'
    },
    {
      title: 'Ice',
      url: 'https://images.unsplash.com/photo-1548097160-627fd636ee56?ixlib=rb-
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=500&q=60'
    },
    {
      title: 'Desert',
      url: 'https://images.unsplash.com/photo-1488197047962-b48492212cda?ixlib=rb-
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=500&q=60'
    },
    {
      title: 'Jungle',
      url: 'https://images.unsplash.com/photo-1564460549828-f0219a31bf90?ixlib=rb-
1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=500&q=60'
    },
  ];

  showFivePages(index: number) {
    return (this.currentPage - index) < 5;
  }
}
```

والنتيجة في المتصفح، كالتالي:



ويجب التنويه ان هذا الحل ليس أفضل الحلول ولكن تم اختياره لتوضيح مفهوم استخدام أكثر من Structure Directive لنفس العنصر.

5.2. Custom Directives:

يتيح لنا Angular ان نقوم ببناء Directive الخاص بنا والذي يلبي احتياجاتنا، ولكن قبل البدء في بناء Directive لابد اننا سوف نمر اثناء بناء هذه Directives بمجموعة من Directors وServices التي تسهل لنا بناء هذه Directives، وهي:

١. ElementRef
٢. Renderer2
٣. @Input
٤. ViewContainerRef
٥. TemplateRef<generic type>
٦. @HostListener
٧. @HostBinding

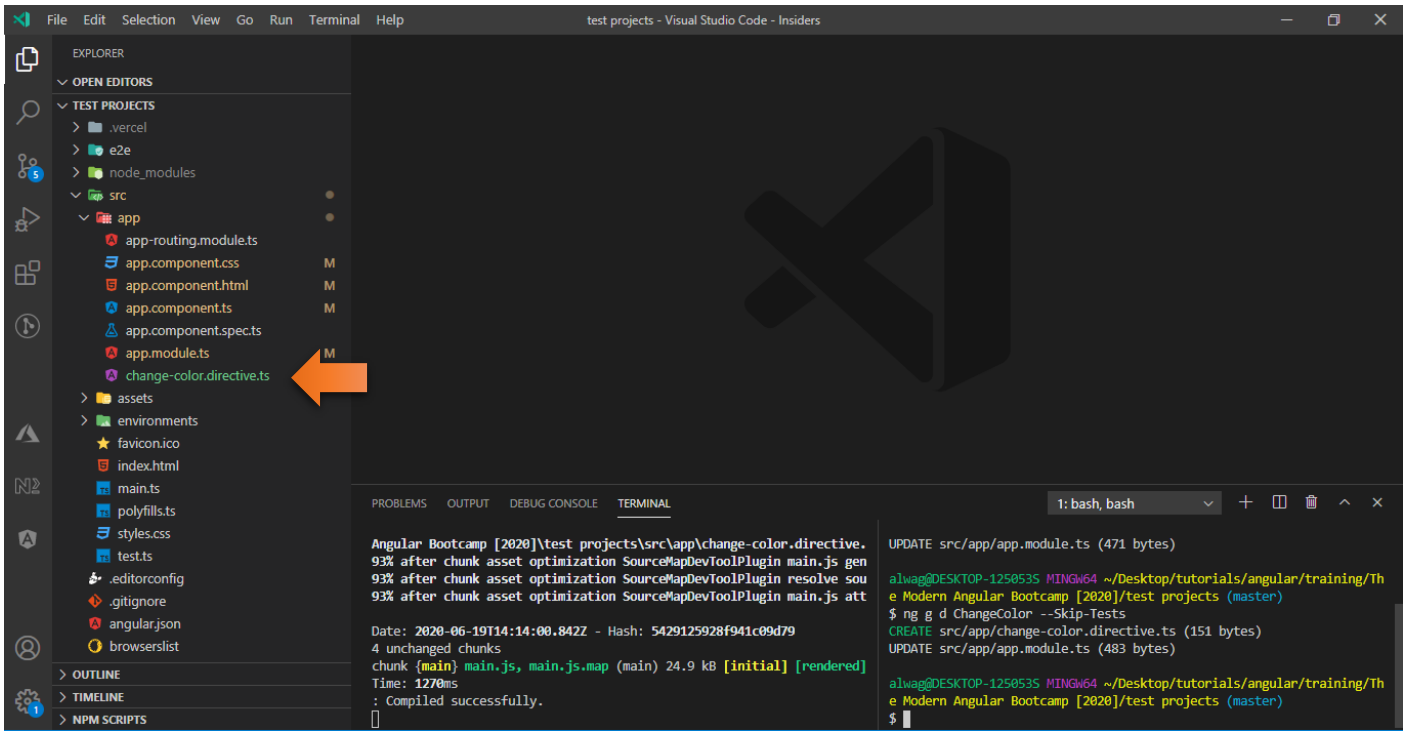
مع العلم ان Decorator وServices من الرقم واحد إلى الرقم خمسة تم شرحها بشكل مفصل في الكتاب الثاني من هذه السلسلة Angular Components and Services، لذلك لن نعيد شرحها وانما فقط سوف نذكر كيف نستفيد منها في بناء Custom Directives، أما من ناحية HostListener وHostBinding فسوف نتطرق لها هنا من ناحية شرحها وتوضيحها.

وأفضل طريقة لفهم من خلال التطبيق العملي لذلك سوف نقوم ببناء مثال والذي نحتاجه من هذه Directors أو Services سوف نقوم بشرح وظيفتها والفائدة منها.

1.5.2. المثال الأول: بناء Directive بسيط يقوم بتغيير لون النص والخلفية لعنصر HTML:-

في هذا المثال البسيط سوف نقوم بتوضيح بعض المفاهيم وبالتدريج سوف ندخل في أمثلة أكثر صعوبة إلى أن نلم بجميع المفاهيم، وتستطيع عزيزي المتعلم بناء Directive الخاص بك.

في البداية نقوم بإنشاء Directive جديد عن طريق Angular CLI وذلك بكتابة الأمر التالي ng g d ChangeColor، حيث أن ChangeColor هو اسم الملف – لك حرية اختيار الاسم الذي تُريده – وسوف ينتج لدينا الملف التالي:



change-color.directive.ts ملف

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

@Directive({
  selector: '[appChangeColor]'
})

export class ChangeColorDirective {

  constructor() { }

}
```

نلاحظ ان الملف مشابه إلى حد ما لملف class الخاص بي Component ولديه selector وهو نفس اسم Directive مضاف له app بين الاقواس المربعة، لكي نضيفه للعنصر المستهدف على شكل خاصية مع العلم اننا نستطيع حذف هذه الاقواس واضافة نقطة قبل الاسم appChangeColor. ولكن في هذه الحالة سوف نضيفه إلى عنصر HTML على شكل كلاس css، والمتعارف عليه هو ابقائه على حالة هذه واضافته على شكل خاصية للعنصر.

ولكي نستطيع تغيير خصائص عنصر HTML عن طريق هذا Directive نحتاج إلى طريقة معينة للوصول إلى هذا العنصر وتغيير خصائصه، ونستطيع عمل ذلك عن طريق Service اسمها ElementRef حيث نقوم بحقنه (نعمل له inject) في constructor، كالتالي:

change-color.directive.ts ملف

```
import { Directive, ElementRef } from '@angular/core';

@Directive({
  selector: '[appChangeColor]'
})
export class ChangeColorDirective {

  constructor(private elementRef: ElementRef) {
    this.elementRef.nativeElement.style.backgroundColor = 'gray';
   }

}
```

نلاحظ اننا عملنا inject لهذا Service في متغير اسميناه elementRef وبذلك اصبح هذا المتغير يشير إلى العنصر الذي سوف نضيف له Directive بحيث لو قمنا على سبيل المثال بإضافته إلى عنصر <p appChangeColor ></p> فكأننا نقوم هنا p.nativeElement.style.backgroundColor = 'gray' وهكذا بقية العناصر، وايضاً عن طريق هذا المتغير استطعنا الوصول إلى خاصية backgrounColor والخاصة بتغيير لون الخلفية للعنصر.

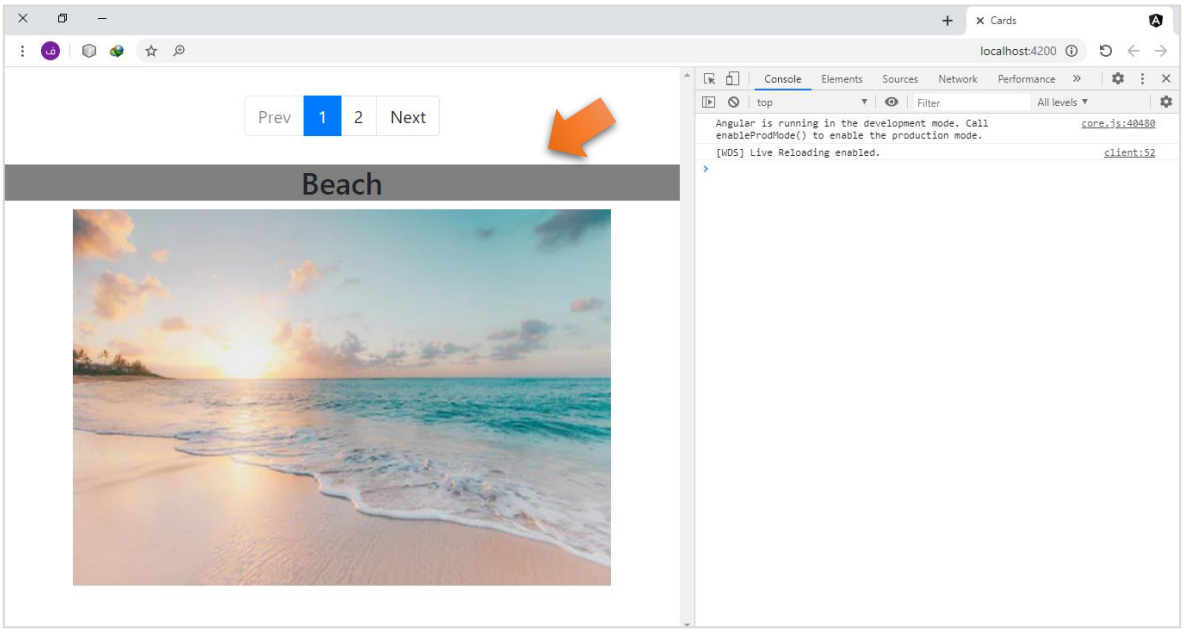
الآن لنقوم بإضافة هذا Directive إلى العنصر <h3></h3> والذي يحتوي على اسم الصورة في ملف app.component.html، كالتالي:

app.component.html ملف

```
<nav>
    <ul class="pagination">
        <li class="page-item" [ngStyle]="{
            'opacity': currentPage > 0 ? 1 : 0.6,
            'pointerEvents':  currentPage > 0 ? 'auto' : 'none'
            }">
            <a class="page-link" (click)="currentPage = currentPage - 1">
                Prev
            </a>
        </li>
        <ng-container *ngFor="let image of images; let i=index">
            <li class="page-item"
                [ngClass]="{active: i===currentPage}"
                *ngIf="showFivePages(i)"
            >
                <a (click)="currentPage = i" class="page-link">{{i + 1}}</a>
            </li>
        </ng-container>
        <li class="page-item" [ngStyle]="{
            'opacity': currentPage < images.length - 1 ? 1 : 0.6,
            'pointer-events':  currentPage < images.length - 1 ? 'auto' : 'none'
```

```
            }">
            <a class="page-link" (click)="currentPage = currentPage + 1">
                Next
            </a>
        </li>
    </ul>
</nav>

<div class="img-box">
    <h3 appChangeColor>{{ images[currentPage].title }}</h3>
    <img [src]="images[currentPage].url" />
</div>
```

والنتيجة:



مع اننا استطعنا الوصول إلى العنصر والتلاعب بخصائصه مباشرة في DOM عن طريق ElementRef إلى ان Angular لا يفضل هذه الطريقة لعدة أسباب منها الأمان ومنها أن اكواد Angular لا تعمل فقط في DOM فنستطيع كتابة اكواد Angular لبناء تطبيقات موبايل او تطبيقات سطح المكتب، لذلك قدمت لنا Angular طريقة أخرى وهي المفضلة وذلك عن طريق Service آخر اسمه Renderer2، حيث نصل إلى العنصر عن طريق ElmentRef ولكن نتلاعب بخصائص هذا العنصر عن طريق Renderer2، كالتالي:

change-color.directive.ts ملف

```
import { Directive, ElementRef, Renderer2 } from '@angular/core';

@Directive({
  selector: '[appChangeColor]'
})
export class ChangeColorDirective {
  constructor(private elementRef: ElementRef, private renderer2: Renderer2) {
    this.renderer2.setStyle(this.elementRef.nativeElement, 'backgroundColor', 'gray');
   }

}
```

2.5.2. المثال الثاني: بناء Directive بسيط يقوم بتغيير لون النص والخلفية لعنصر مع HTMLمع تمرير قيم ديناميكية: -

المثال السابق يقوم بتغيير لون الخلفية ولكن القيم ثابتة بمعنى القيمة التي قمنا بتمريرها إلى الخاصية backgroundColor ثابته وهي gray، ولكن الفائدة الأساسية من Directive هو الديناميكية بمعنى اننا ننشأ Directive ونضيفه لأي عنصر ونقوم بتمرير القيمة من خلال هذا العنصر ويستقبلها Directive ويقوم بإسنادها إلى هذه الخاصية، ونستطيع القيام بذلك عن طريق @Input() وميزة Property Binding، لأن هذا Directive عند إضافته لأي عنصر يعتبر بمثابة أبن لهذا العنصر وبذلك نستطيع تمرير البيانات له عن طريق Decorator ذو الاسم @Input() كما كنا نفعل مع components عند أرسال البيانات من الاب إلى الأبن، او عن طريق @Output() لتمرير البيانات من الأبن إلى الأب، ونستطيع ان نطلق على هذا العنصر ايضاً مسمى آخر هو مُضيف Host لهذا Directive.

ونستطيع القيام بهذا الامر، كالتالي:

change-color.directive.ts ملف

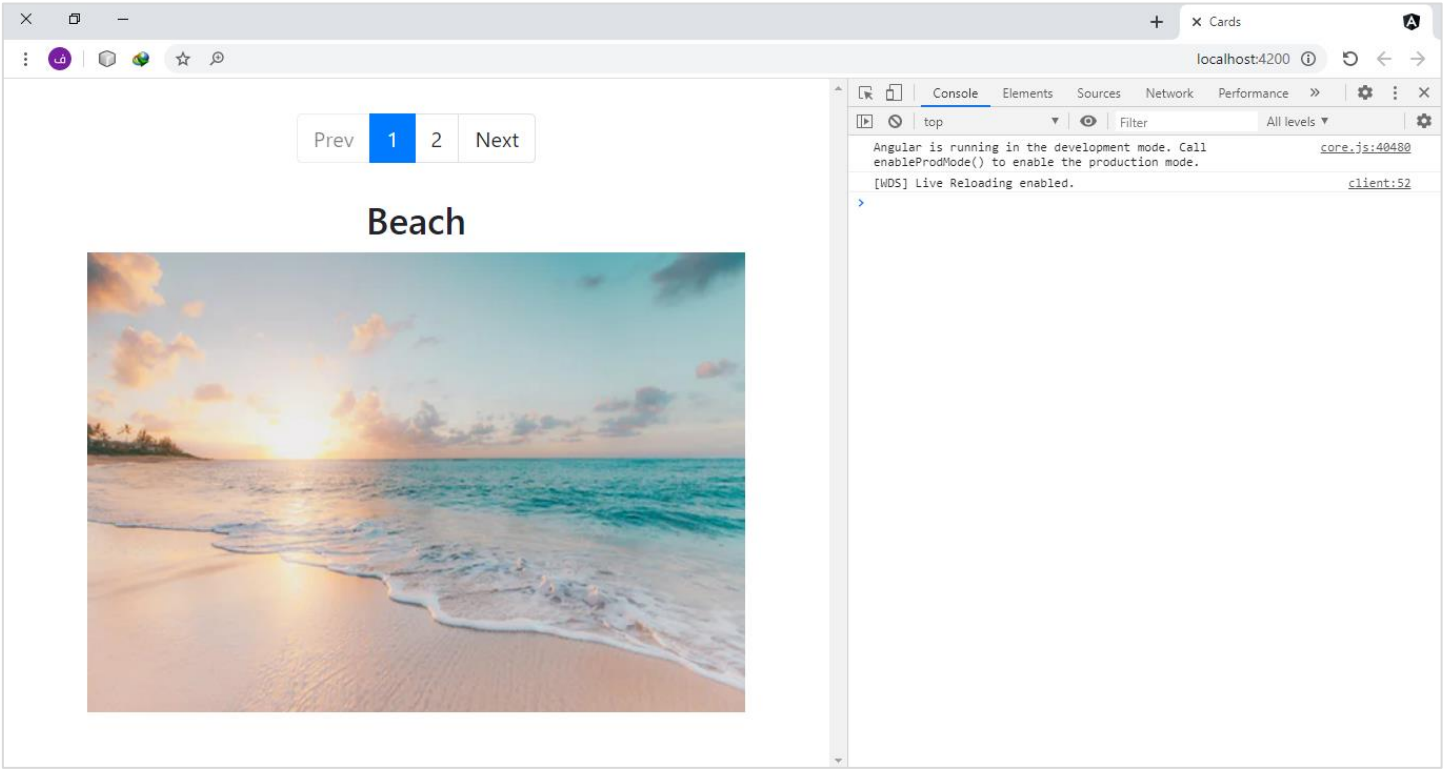
```
import { Directive, ElementRef, Renderer2, Input } from '@angular/core';

@Directive({
  selector: '[appChangeColor]'
})

export class ChangeColorDirective {
  @Input() color: string;

  constructor(private elementRef: ElementRef, private renderer2: Renderer2) {
    this.renderer2.setStyle(this.elementRef.nativeElement, 'backgroundColor', this.color);
   }

}
```

app.component.html ملف

```
<nav>
    <ul class="pagination">
        <li class="page-item" [ngStyle]="{
            'opacity': currentPage > 0 ? 1 : 0.6,
            'pointerEvents':  currentPage > 0 ? 'auto' : 'none'
            }">
            <a class="page-link" (click)="currentPage = currentPage - 1">
                Prev
            </a>
        </li>
        <ng-container *ngFor="let image of images; let i=index">
            <li class="page-item"
                [ngClass]="{active: i===currentPage}"
                *ngIf="showFivePages(i)"
            >
                <a (click)="currentPage = i" class="page-link">{{i + 1}}</a>
            </li>
```

```
        </ng-container>
        <li class="page-item" [ngStyle]="{
            'opacity': currentPage < images.length - 1 ? 1 : 0.6,
            'pointer-events':  currentPage < images.length - 1 ? 'auto' : 'none'
            }">
            <a class="page-link" (click)="currentPage = currentPage + 1">
                Next
            </a>
        </li>
    </ul>
</nav>

<div class="img-box">
    <h3 appChangeColor [color]=" 'lightblue' ">{{ images[currentPage].title }}</h3>
    <img [src]="images[currentPage].url" />
</div>
```

نلاحظ اننا مررنا القيمة lightblue على شكل نص عن طريق وضعه بين علامتي تنصيص مفردة، باستخدام ميزة Property Binding، كما نستطيع تمرير متغير كأن يستقبل اللون عن طريق أداة الألوان ونقوم بتخزينها في متغير ومن ثم نمرر هذا المتغير إلى [color] وفي هذه الحالة لابد ان نلغي علامتي التنصيص المفردة (' ') الآن لنقم بالذهاب إلى المتصفح ونرى النتيجة:



نلاحظ لم يظهر أي شيء، وحقيقة هذا هو المتوقع لأن Angular عندما يقوم ببناء العنصر <h3></h3> سوف يجد appChangeColor وسوف يقوم بتنفيذه وتنفيذ دالة constructor وسوف يجد السطر البرمجي الذي قمنا بكتابته ولكن هو إلى الآن لم يتعرف على الخاصية color التي قمنا بعمل لها Property Binding لذلك سوف تكون قيمتها undefined، وبعد الانتهاء من تنفيذه هذا Directive سوف يكمل بنائه لهذا العنصر وسوف يجد خاصية [color] وعندها سوف يذهب إلى هذا Directive ويُعرف القيمة التي تم تمريرها لهذا Directive ولكن لن يُعيد قراءة constructor لأنه يقرأها مرة واحدة فقط وبذلك لن يتم تنفيذ السطر البرمجي ولن يقوم بتغيير الخلفية لهذا العنصر، وهنالك حلين لهذا الأمر الأول وهو وضع السطر البرمجي

في الدالة ngOnInit لأن هذه الدالة يقوم بتنفيذها Angular بعد بناء Directive، ولفهم اكثر حو هذا الامر الرجاء مراجعة كتابي Angular Components and Directives وبالتحديد الجزء الخاص Lifecycle Hooks، مع العلم ان دوال Lifecycle hooks هي بالأساس خاص بي Component ولكن نستطيع تطبيقها على Directive، أما الثاني عن طريق استخدام دالة Setter، كالتالي:

change-color.directive.ts جزء من ملف

```
@Input() set color(colorValue: string) {
   this.renderer2.setStyle(this.elementRef.nativeElement, 'backgroundColor', colorValue);
}
```

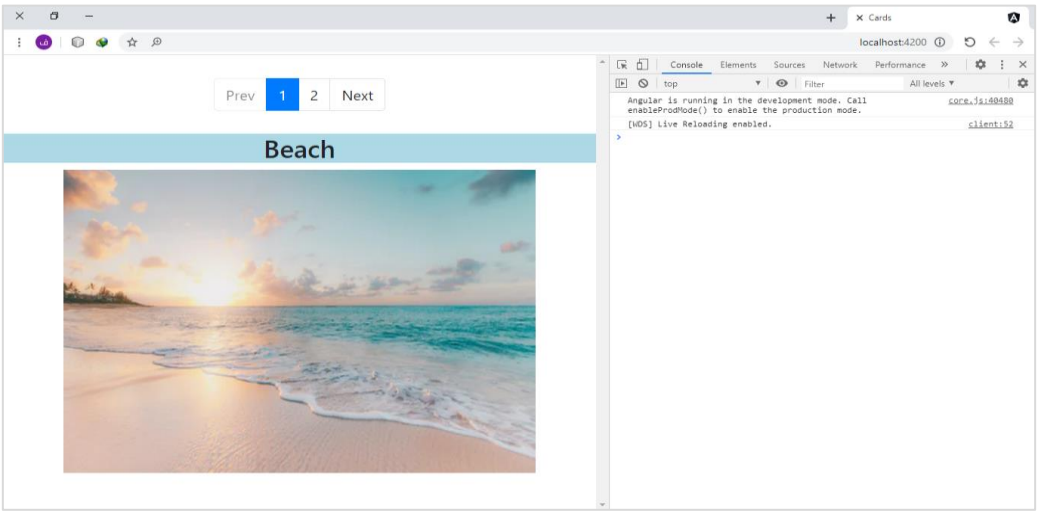
app.component.html جزء من ملف

```
<div class="img-box">
    <h3 appChangeColor [color]="'lightblue'">{{ images[currentPage].title }}</h3>
    <img [src]="images[currentPage].url" />
</div>
```

أما الطريقة الثانية فنقوم بتطبيقها عن طريق استدعاء الدالة ngOnInit ونقل السطر البرمجي لها، كالتالي:

change-color.directive.ts ملف

```
import { Directive, ElementRef, Renderer2, Input, OnInit } from '@angular/core';
@Directive({
  selector: '[appChangeColor]'
})

export class ChangeColorDirective implements OnInit {
  @Input() color: string;
  constructor(private elementRef: ElementRef, private renderer2: Renderer2) {

  }

  ngOnInit(): void {
    this.renderer2.setStyle(this.elementRef.nativeElement, 'backgroundColor', this.color);
  }

}
```



ولكن بما أنه لدينا متغير واحد فقط قمنا بتمريره فنستطيع إعادة صياغته بهذه الطريقة:

change-color.directive.ts ملف

```
import { Directive, ElementRef, Renderer2, Input, OnInit } from '@angular/core';

@Directive({
  selector: '[appChangeColor]'
})

export class ChangeColorDirective implements OnInit {
  @Input('appChangeColor') color: string;

  constructor(private elementRef: ElementRef, private renderer2: Renderer2) {

  }

  ngOnInit(): void {
    this.renderer2.setStyle(this.elementRef.nativeElement, 'backgroundColor', this.color);
  }

}
```

نلاحظ أننا قمنا بتمرير أسم Input لهذا Alias على انه Directive

اما في ملف app.component.html، فنمرر القيمة مباشرة إلى هذا Directive كالتالي:

app.component.html ملف

```
<nav>
    <ul class="pagination">
        <li class="page-item" [ngStyle]="{
            'opacity': currentPage > 0 ? 1 : 0.6,
            'pointerEvents':  currentPage > 0 ? 'auto' : 'none'
            }">
            <a class="page-link" (click)="currentPage = currentPage - 1">
                Prev
            </a>
        </li>
        <ng-container *ngFor="let image of images; let i=index">
            <li class="page-item"
                [ngClass]="{active: i===currentPage}"
                *ngIf="showFivePages(i)"
            >
                <a (click)="currentPage = i" class="page-link">{{i + 1}}</a>
            </li>
        </ng-container>
        <li class="page-item" [ngStyle]="{
            'opacity': currentPage < images.length - 1 ? 1 : 0.6,
            'pointer-events':  currentPage < images.length - 1 ? 'auto' : 'none'
            }">
            <a class="page-link" (click)="currentPage = currentPage + 1">
                Next
            </a>
        </li>
    </ul>
```

```
</nav>

<div class="img-box">
    <h3 [appChangeColor]=" 'lightblue' ">{{ images[currentPage].title }}</h3>
    <img [src]="images[currentPage].url" />
</div>
```

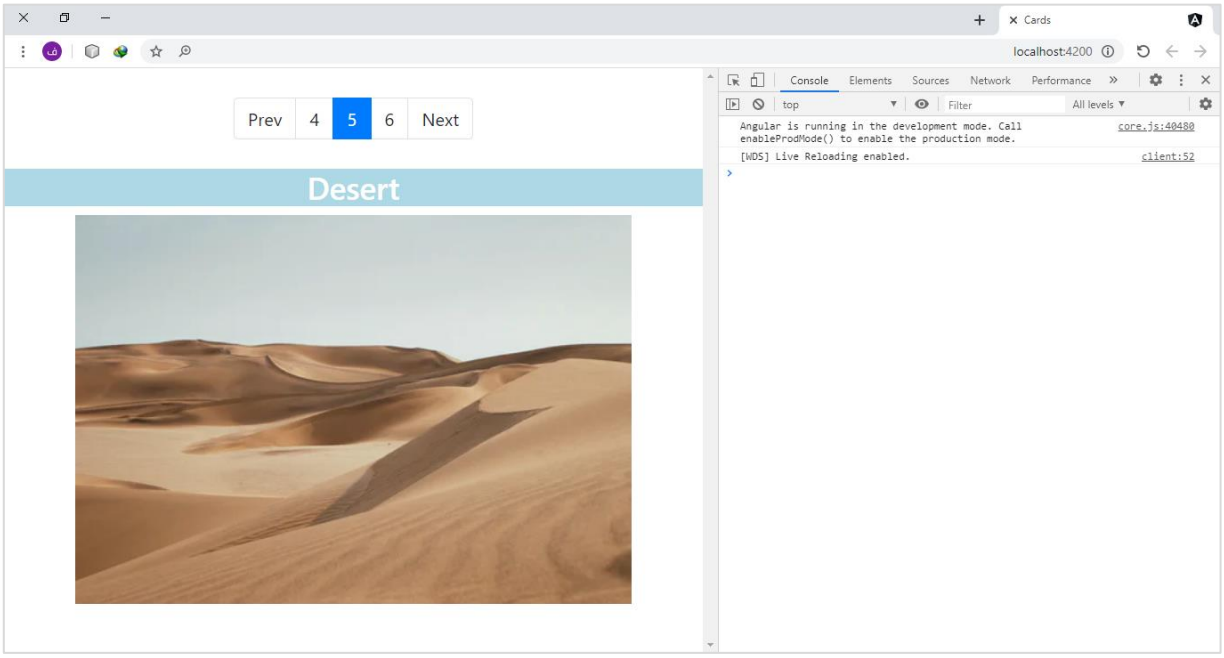
أما في حالة انه كان لدينا خصائص كثيرة نريد ان نمررها فنستخدم الطريقة الأولى بحيث كل خاصية نستقبلها بي Input خاص بها ونعمل لها Property Binding في العنصر.

change-color.directive.ts ملف

```
import { Directive, ElementRef, Renderer2, Input, OnInit } from '@angular/core';
@Directive({
  selector: '[appChangeColor]'
})

export class ChangeColorDirective implements OnInit {
  @Input('appChangeColor') color: string;
  @Input() ForeColor: string;

  constructor(private elementRef: ElementRef, private renderer2: Renderer2) {

  }
  ngOnInit(): void {
    this.renderer2.setStyle(this.elementRef.nativeElement, 'backgroundColor', this.color);
    this.renderer2.setStyle(this.elementRef.nativeElement, 'color', this.ForeColor);
  }

}
```

app.component.html جزء من ملف

```
<div class="img-box">
    <h3
       [appChangeColor]=" 'lightblue' "
       [ForeColor]="'white'"
    >
       {{ images[currentPage].title }}</h3>

    <img [src]="images[currentPage].url" />
</div>
```



نلاحظ تم تطبيق كلا اللونين (الخلفية والخط).

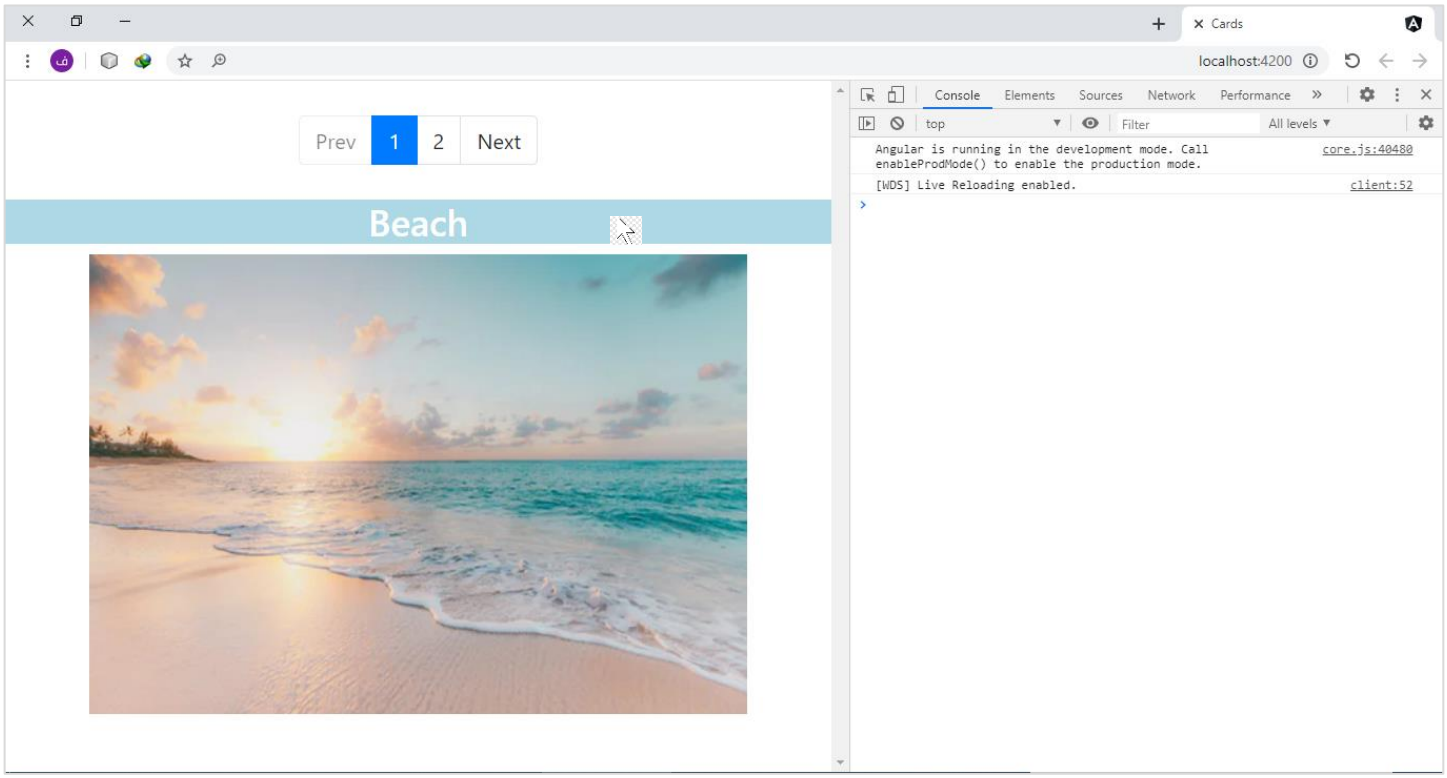
3.5.2. المثال الثالث: بناء Directive بسيط يقوم بتغيير لون النص والخلفية لعنصر HTML مع تمرير قيم ديناميكية، ويتم تنفيذ هذا Directive فقط في حال وقوع حدث Event معين على العنصر:-

في المثال السابق قمنا بتمرير قيم ديناميكية لي Directive من العنصر المضيف Host (العنصر الأب) لهذا Directive وبناء على هذه القيم نقوم بتغيير لو الخلفية والنص، وهنا سوف نتقدم خطوة إلى الأمام ونريد أن نراقب الأحداث التي تتم على العنصر المُضيف Host لهذا Directive (مثل حدث النقر على زر الفأرة على العنصر او حدث مرور مؤشر الفأرة على العنصر او حدث خروج مؤشر الفأرة عن العنصر او...الخ) عن طريق Decorator ذو الاسم @HostListener حيث نمرر له الحدث الذي نريد منه ان يراقبه وعند وقوع هذا الحدث ينفذ دالة معينة، كالتالي:

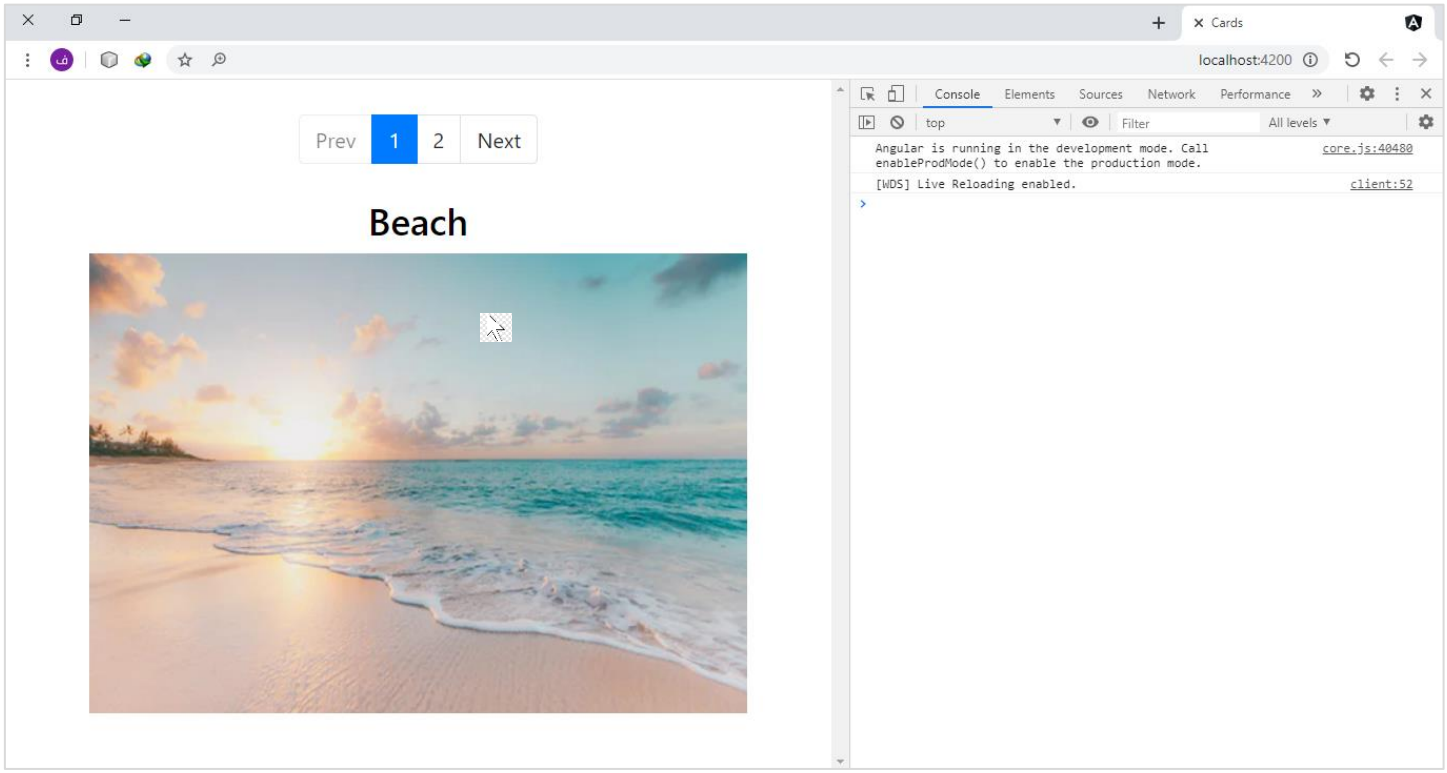
change-color.directive.ts ملف

```
import {
      Directive,
      ElementRef,
      Renderer2,
      Input,
      OnInit,
      HostListener
} from '@angular/core';

@Directive({
  selector: '[appChangeColor]'
})

export class ChangeColorDirective implements OnInit {
  @Input('appChangeColor') color: string;
  @Input() ForeColor: string;

  constructor(private elementRef: ElementRef, private renderer2: Renderer2) {}
```

```
  ngOnInit(): void {}

  @HostListener('mouseover') onMouseMove() {
    this.renderer2.setStyle(this.elementRef.nativeElement, 'backgroundColor', this.color);
    this.renderer2.setStyle(this.elementRef.nativeElement, 'color', this.ForeColor);
  }

  @HostListener('mouseout') onMouseOut() {
    this.renderer2.setStyle(this.elementRef.nativeElement, 'backgroundColor', 'transparent');
    this.renderer2.setStyle(this.elementRef.nativeElement, 'color', 'black');
  }
}
```



هذا في حال مرور مؤشر الفأرة على العنصر <h3></h3> (الحدث mouse over)، اما في حال الخروج خروج مؤشر الفأرة من العنصر (الحدث mouse out) فيصبح العنصر، كالتالي:



هذه مجرد امثلة والهدف منها فهم كيف يتم الاستماع إلى حدث معين وقع على العنصر الموجود فيه هذا Directive ومن ثم تنفيذ مهمة معينة، وتستطيع عزيزي المتعلم ان تطلق العنان لمخيلتك ومحاولة التجريب بنفسك واعلم ان الطريقة واحدة.

4.5.2. المثال الرابع: بناء Directive بسيط يقوم بإضافة border للعنصر عن طريق HostBinding:-

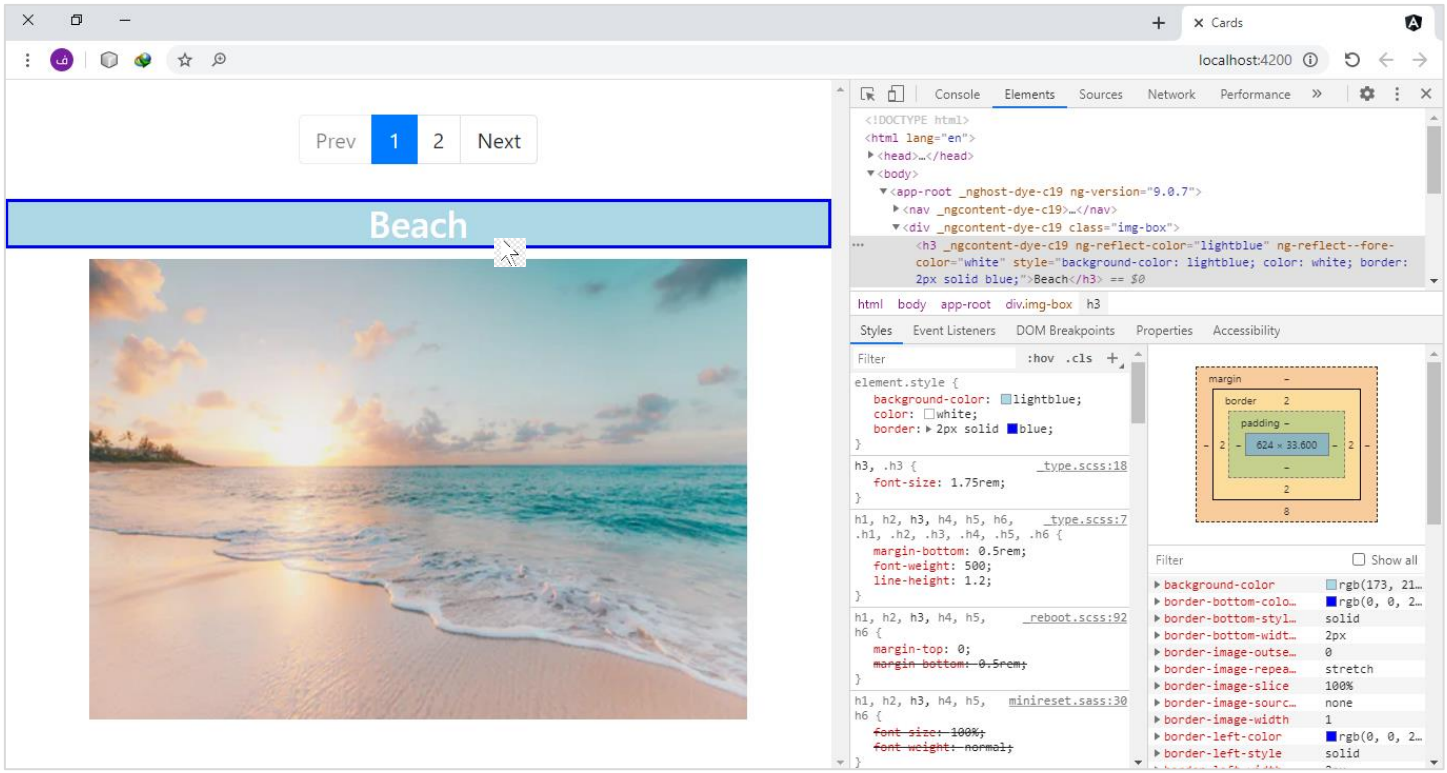
نستطيع ايضاً تمرير قيم والتحكم بخصائص الstyle للعنصر المُضيف (height, width, color, margin, border, etc.) بطريقة أخرى غير طريقة Input وهي باستخدام Decorator ذو الاسم HostBinding حيث نمرر له الخاصية التي نريد ان نُغير قيمتها مسبوقاً بكلمة style، كالتالي:

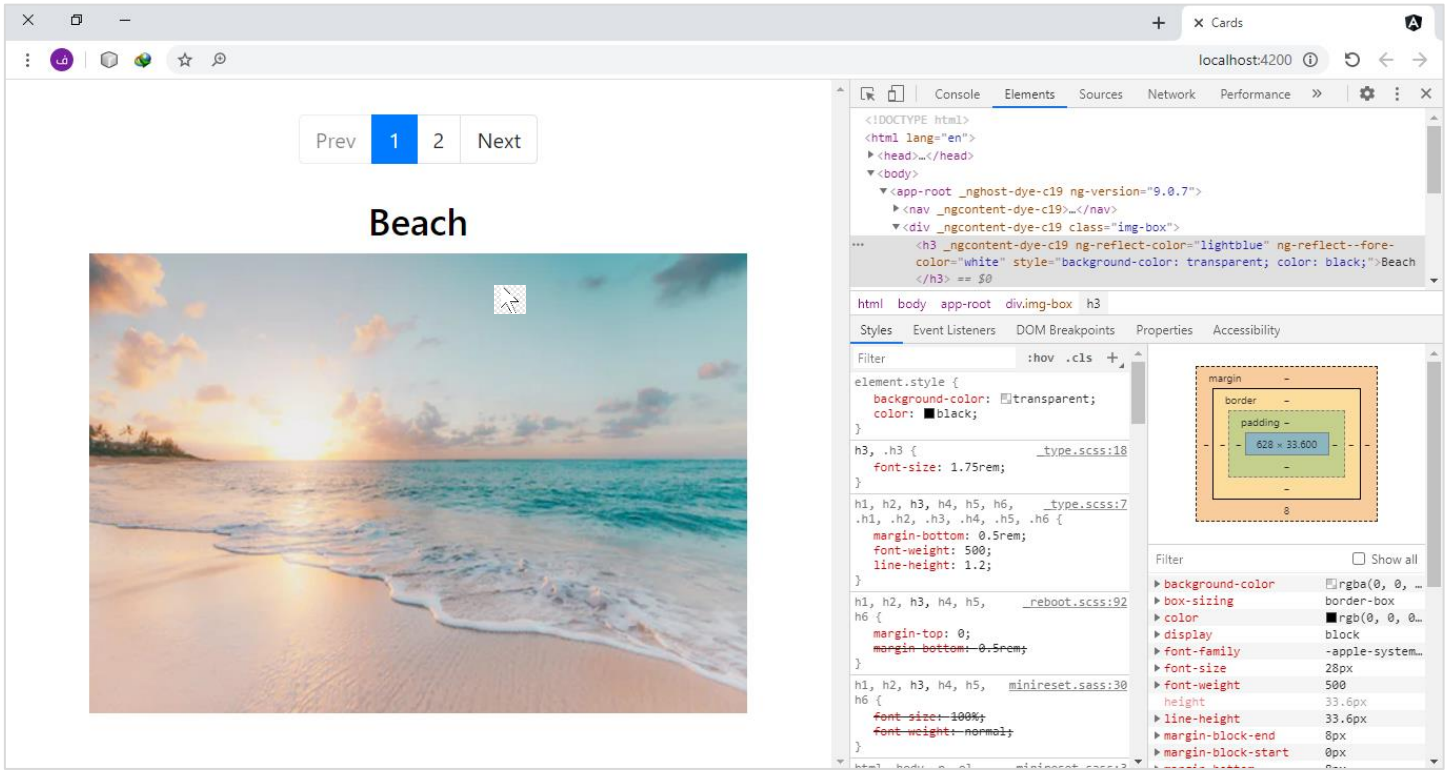
change-color.directive.ts ملف

```
import {
      Directive,
      ElementRef,
      Renderer2,
      Input,
      OnInit,
      HostListener,
      HostBinding
} from '@angular/core';

@Directive({
  selector: '[appChangeColor]'
})

export class ChangeColorDirective implements OnInit {
  @Input('appChangeColor') color: string;
  @Input() ForeColor: string;
  @HostBinding('style.border') border: string;
```

```
  constructor(private elementRef: ElementRef, private renderer2: Renderer2) {}

  ngOnInit(): void {}

  @HostListener('mouseover') onMouseMove() {
    this.renderer2.setStyle(this.elementRef.nativeElement, 'backgroundColor', this.color);
    this.renderer2.setStyle(this.elementRef.nativeElement, 'color', this.ForeColor);
    this.border = '2px solid blue';
  }

  @HostListener('mouseout') onMouseOut() {
    this.renderer2.setStyle(this.elementRef.nativeElement, 'backgroundColor', 'transparent');
    this.renderer2.setStyle(this.elementRef.nativeElement, 'color', 'black');
    this.border = '';
  }

}
```

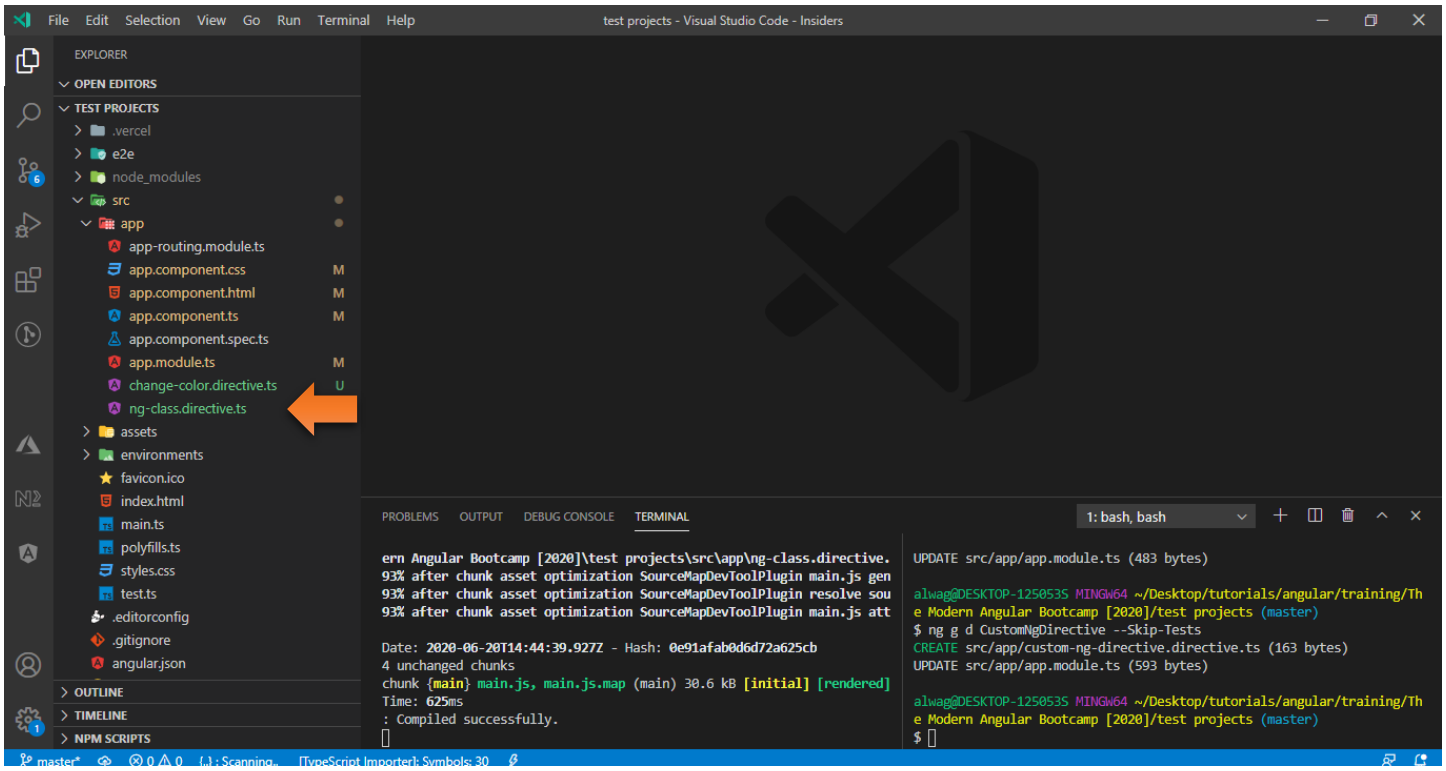
نلاحظ ان هذه الطريقة هي أسهل من الطريقة الأولى، ولك عزيزي المتعلم حرية الاختيار مع أي طريقة تراها مناسبة لك.





5.5.2. المثال الخامس: بناء Directive نحاكي فيه ngClass: -

في هذا المثال سوف نقوم ببناء ngClass مشابهه لما هو موجود في Angular لفهم أعمق لهذا Directive، وسوف نستخدم التقنيات التي شرحناها سابقاً لبناء هذا Directive، وأول خطوة هو انشاء Directive جديد وليكن اسمه NgClass، كالتالي:



ولو رجعنا إلى ngClass لوجدنا انها تستقبل كائن object وهذا الكائن ممكن ان يحتوي على أكثر من مفتاح key وقيمة value حيث المفتاح يمثل الكلاس والقيمة تمثل الشرط الذي بناءً على قيمته نضيف هذا الكلاس (القيمة true) او نحذف هذا الكلاس

(القيمة false)، لذلك لابد ان نستخدم حلقة تكرار لقراءة جميع المفاتيح، وبنفس الوقت نقوم بوضع شرط لمعرفة القيمة لكل مفتاح هل هذ true او false، كالتالي:

ملف ng-class.directive.ts

```
import { Directive, ElementRef, Renderer2, Input } from '@angular/core';

@Directive({
  selector: '[appNgClass]'
})
export class NgClassDirective {

  constructor(private elementRef: ElementRef, private renderer: Renderer2) { }

  @Input('appNgClass') set NgClass(objClass: any) {
    for (const key in objClass) {
      if (objClass[key]) {
        this.renderer.addClass(this.elementRef.nativeElement, key);
      } else {
        this.renderer.removeClass(this.elementRef.nativeElement, key);
      }
    }
  }

}
```

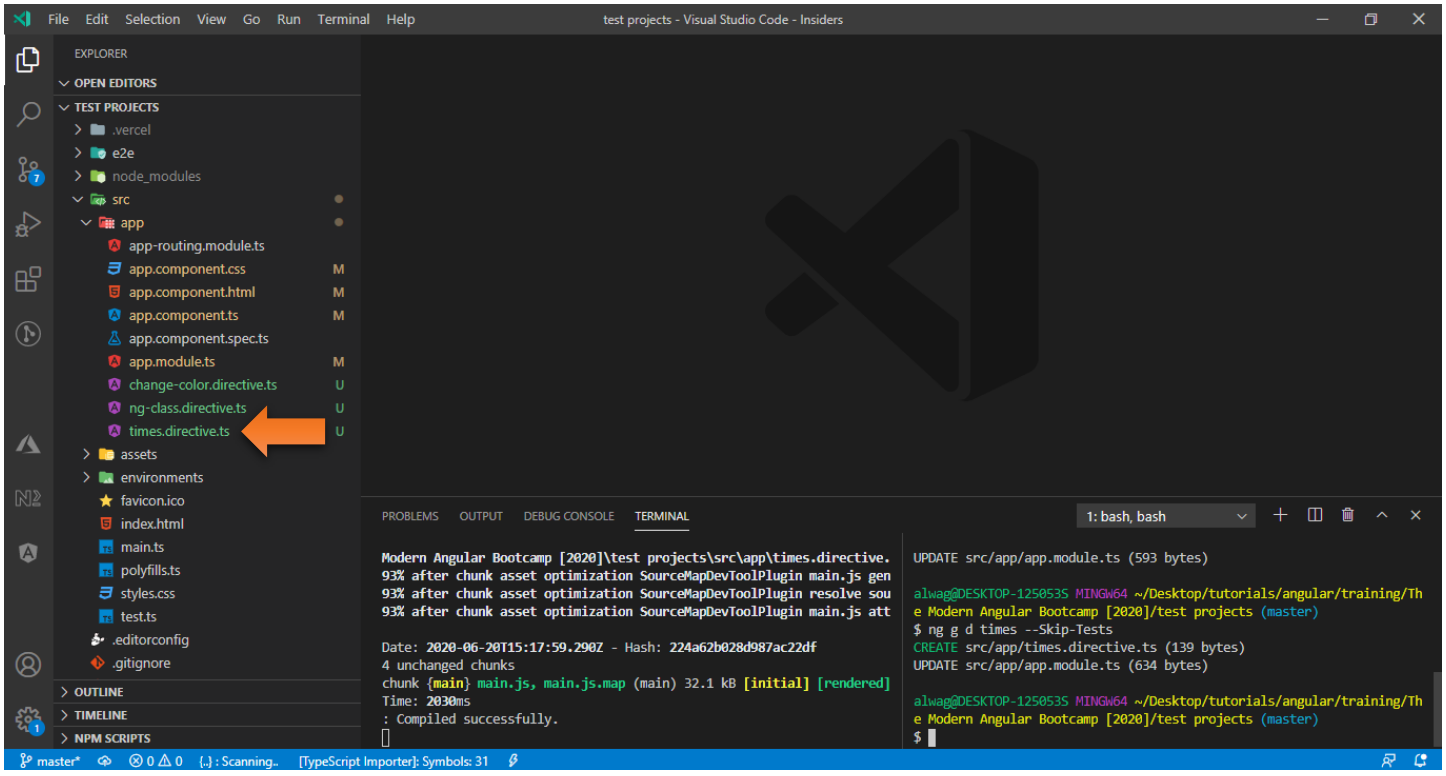
ملف app.component.html

```
<nav>
    <ul class="pagination">
        <li class="page-item" [ngStyle]="{
            'opacity': currentPage > 0 ? 1 : 0.6,
            'pointerEvents':  currentPage > 0 ? 'auto' : 'none'
            }">
            <a class="page-link" (click)="currentPage = currentPage - 1">
                Prev
            </a>
        </li>
        <ng-container *ngFor="let image of images; let i=index">
            <li class="page-item"
                [appNgClass]="{active: i===currentPage}"
                *ngIf="showFivePages(i)"
        >
                <a (click)="currentPage = i" class="page-link">{{i + 1}}</a>
            </li>
        </ng-container>
        <li class="page-item" [ngStyle]="{
            'opacity': currentPage < images.length - 1 ? 1 : 0.6,
            'pointer-events':  currentPage < images.length - 1 ? 'auto' : 'none'
            }">
            <a class="page-link" (click)="currentPage = currentPage + 1">
                Next
            </a>
        </li>
```

```
    </ul>
</nav>

<div class="img-box">
    <h3 [appChangeColor]=" 'lightblue' "
        [ForeColor]="'white'">
        {{ images[currentPage].title }}
    </h3>
    <img [src]="images[currentPage].url" />
</div>
```

6.5.2. المثال السادس: بناء Directive نحاكي فيه ngFor*: -

في هذا المثال سوف نحاول بناء Structure Directive، وسوف نُطبق المفاهيم viewContainer وtempalteRef، حيث ان الأول يمثل الحاوية التي تحتوي الكود الذي نريد تكراره وهو مشابه ng-container اما الثاني فيمثل template أو اكواد او تاغات HTML التي نريد تكرارها، وأول خطوة نقوم فيها هي انشاء Directive جديد وليكن اسمه times، كالتالي:



وتكمن الفكرة في قراءة المدخل والذي هو عبارة عن رقم وبناءً على هذا الرقم نقوم ببناء templateRef في viewContainer ونمرر index لكل template، كالتالي:

times.directive.ts ملف

```
import { Directive, ViewContainerRef, TemplateRef, Input } from '@angular/core';

@Directive({
  selector: '[appTimes]'
})
export class TimesDirective {

  constructor(
      private viewContainerRef: ViewContainerRef,
```

```
      private templateRef: TemplateRef<any>
) { }

  @Input('appTimes') set render(times: number) {
    for (let i = 0; i < times; i++) {
      this.viewContainerRef.createEmbeddedView(this.templateRef, {
        index: i
      });
    }
  }

}
```

حيث البارمتر times الذي مررناه لدالة render يمثل الرقم الذي نريد تكرار TemplateRef بناءً عليه. وindex مررناه هنا وجعلنا قيمته تساوي i لكي نستدعيه في ملف app.component.html كما كنا نفعل مع *ngFor (let i=index)، والآن لنقوم باستبدال *ngFor بهذا Directive، كالتالي:

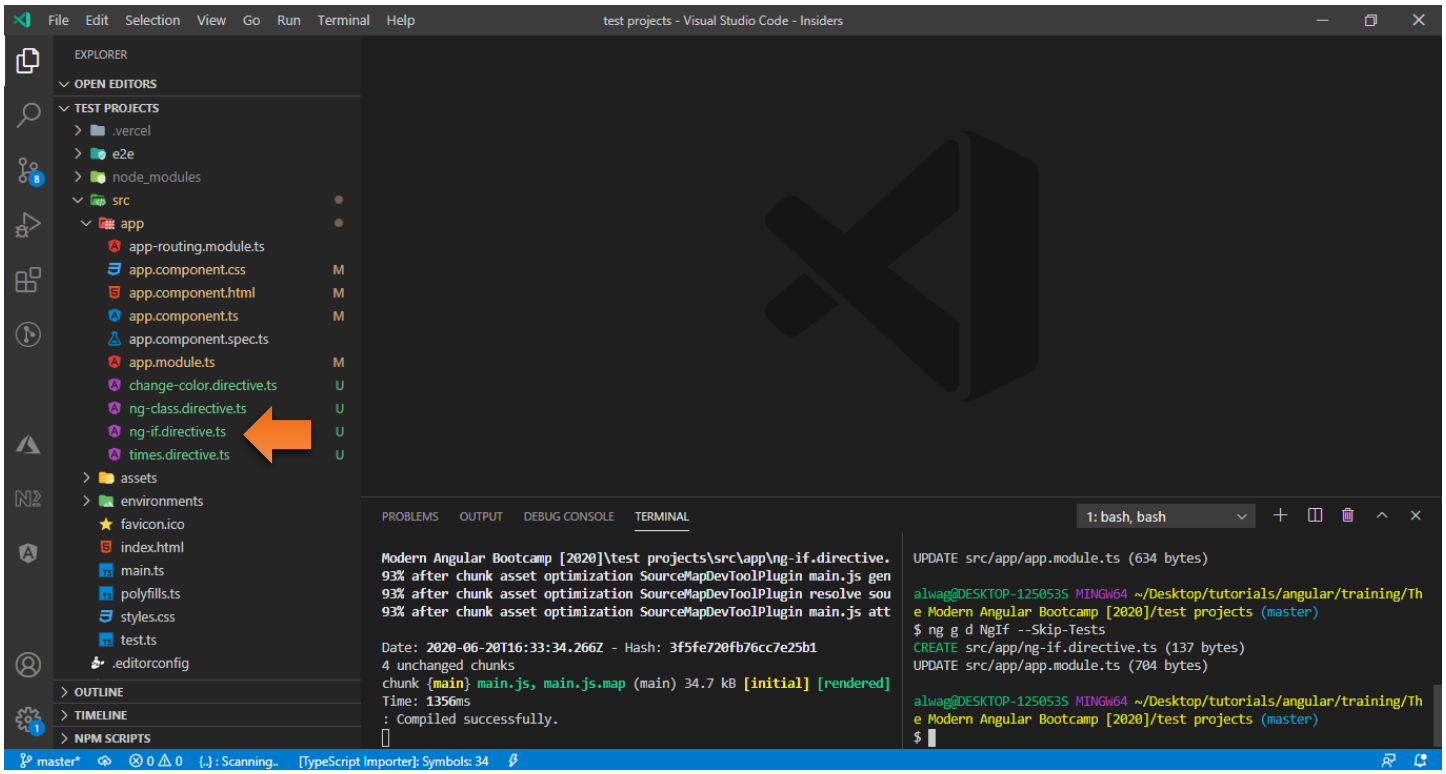
ملف app.component.html

```
<nav>
    <ul class="pagination">
        <li class="page-item" [ngStyle]="{
            'opacity': currentPage > 0 ? 1 : 0.6,
            'pointerEvents':  currentPage > 0 ? 'auto' : 'none'
            }">
            <a class="page-link" (click)="currentPage = currentPage - 1">
                Prev
            </a>
        </li>
        <ng-container *appTimes="images.length; let i=index">
            <li
                class="page-item"
                [appNgClass]="{active: i===currentPage}"
                *ngIf="showFivePages(i)"
            >
                <a (click)="currentPage = i" class="page-link">{{i + 1}}</a>
            </li>
        </ng-container>
        <li class="page-item" [ngStyle]="{
            'opacity': currentPage < images.length - 1 ? 1 : 0.6,
            'pointer-events':  currentPage < images.length - 1 ? 'auto' : 'none'
            }">
            <a class="page-link" (click)="currentPage = currentPage + 1">
                Next
            </a>
        </li>
    </ul>
</nav>

<div class="img-box">
    <h3
        [appChangeColor]=" 'lightblue' "
        [ForeColor]="'white'">
```

```
        {{ images[currentPage].title }}
    </h3>
    <img [src]="images[currentPage].url" />
</div>
```

7.5.2. المثال السابع: بناء Directive نحاكي فيه ngIf*:-

وهو مشابهه للمثال السابق من حيث المفاهيم، ولكن يختلف من حيث Logic البرمجي، كالتالي:



ملف ng-if.directive.ts

```
import { Directive, Input, ViewContainerRef, TemplateRef } from '@angular/core';

@Directive({
  selector: '[appNgIf]'
})
export class NgIfDirective {

  constructor(
    private viewContainerRef: ViewContainerRef,
    private templateRef: TemplateRef<any>
    ) { }

  @Input('appNgIf') set checkCondition(condition: boolean) {
    if (condition) {
      this.viewContainerRef.createEmbeddedView(this.templateRef);
    } else {
      this.viewContainerRef.clear();
    }
  }

}
```

app.component.html ملف

```
<nav>
    <ul class="pagination">
        <li class="page-item" [ngStyle]="{
            'opacity': currentPage > 0 ? 1 : 0.6,
            'pointerEvents':  currentPage > 0 ? 'auto' : 'none'
            }">
            <a class="page-link" (click)="currentPage = currentPage - 1">
                Prev
            </a>
        </li>
        <ng-container *appTimes="images.length; let i=index">
            <li
               class="page-item"
               [appNgClass]="{active: i===currentPage}"
               *appNgIf="showThreePages(i)"
            >
                <a (click)="currentPage = i" class="page-link">{{i + 1}}</a>
            </li>
        </ng-container>
        <li class="page-item" [ngStyle]="{
            'opacity': currentPage < images.length - 1 ? 1 : 0.6,
            'pointer-events':  currentPage < images.length - 1 ? 'auto' : 'none'
            }">
            <a class="page-link" (click)="currentPage = currentPage + 1">
                Next
            </a>
        </li>
    </ul>
</nav>

<div class="img-box">
    <h3 [appChangeColor]=" 'lightblue' " [ForeColor]="'white'">
        {{ images[currentPage].title }}
    </h3>
    <img [src]="images[currentPage].url" />
</div>
```

المراجع

References

References:

1. Freeman, **Adam, Pro Angular 9**, Apress, 2020.

2. Agarwal, Uttam, **Hands-On Full Stack Development with Angular 5 and Firebase**, Packt, 2018.

3. Delaney, Jeff, **The Angular Firebase Survival Guide**, leanpub, 2018.

4. Saha, Depasis, **Angular 7.0 for Beginners**, 2018.

5. Vijay, Santhosh, **Material for Angular Developer Course**, Leanpub, 2019.

6. Hussain, Asim, **Angular From Theory to Practice**, 2017.

7. D. Booth, Joseph, **Angular Succinctly**, Syncfusion, 2019.

8. Squad, Ninja, **Become a ninja with Angular**, 2019.

9. Steyer, Manfred, **Enterprise Angular**, Leanpub, 2020.

10. Clow, Mark, **Angular 5 Projects**, Apress 2018.

11. Nate Murray, Felipe Coury, Ari Lerner, and Carlos Taborda, **ng-book The Complete Guide to Angular**, Fullstack.io, 2018.

12. Bouchefra, Ahmed, **Practical Angular: Build your first web apps with Angular 8**, Leanpub, 2019.

13. Hajian, Majid, **Progressive Web Apps with Angular**, Apress, 2019.

14. Savkin, Victor, **Angular Router**, Leanpub, 2018.