

# Angular Demos

## Core Dynamic Features

`{{ expr }}, *ngFor, *ngIf  
@Component ... @Input @Output  
Chrome Debugger`

[arnaud.nauwynck@gmail.com](mailto:arnaud.nauwynck@gmail.com)

Course Esilv 2023

This document:

<https://github.com/Arnaud-Nauwynck/presentations/web/angular-demos-2-ngFor-ngIf-component.pdf>

# Outline

- templateHtml {{ expr }}
- \*ngFor
- \*ngIf
- Chrome Debugger
- @Component
- [] .. @Input field
- () .. @Output changed
- [()] .. Bidirectional Binding
- router

# Interpolating {{ expr }} in template Html

In component.ts

```
@Component({  
  selector: 'app-demo2-interpolation',  
  templateUrl: './demo2-interpolation.component.html',  
})  
export class Demo2InterpolationComponent {  
  
  value1Text = "text1";  
}
```

In (template) component.html

```
value1Text: {{ value1Text }}
```

Result DOM

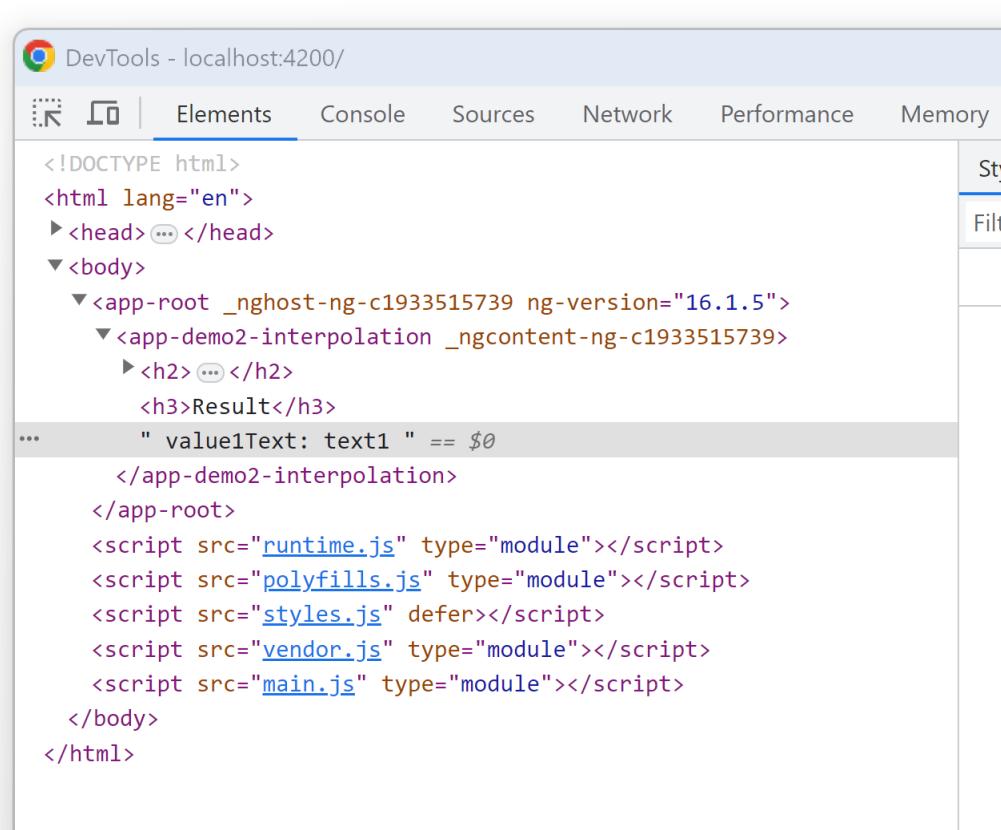
value1Text: text1

# Checking with Chrome debugger (F12)

Demo {{ interpolationExpression }}

Result

value1Text: text1

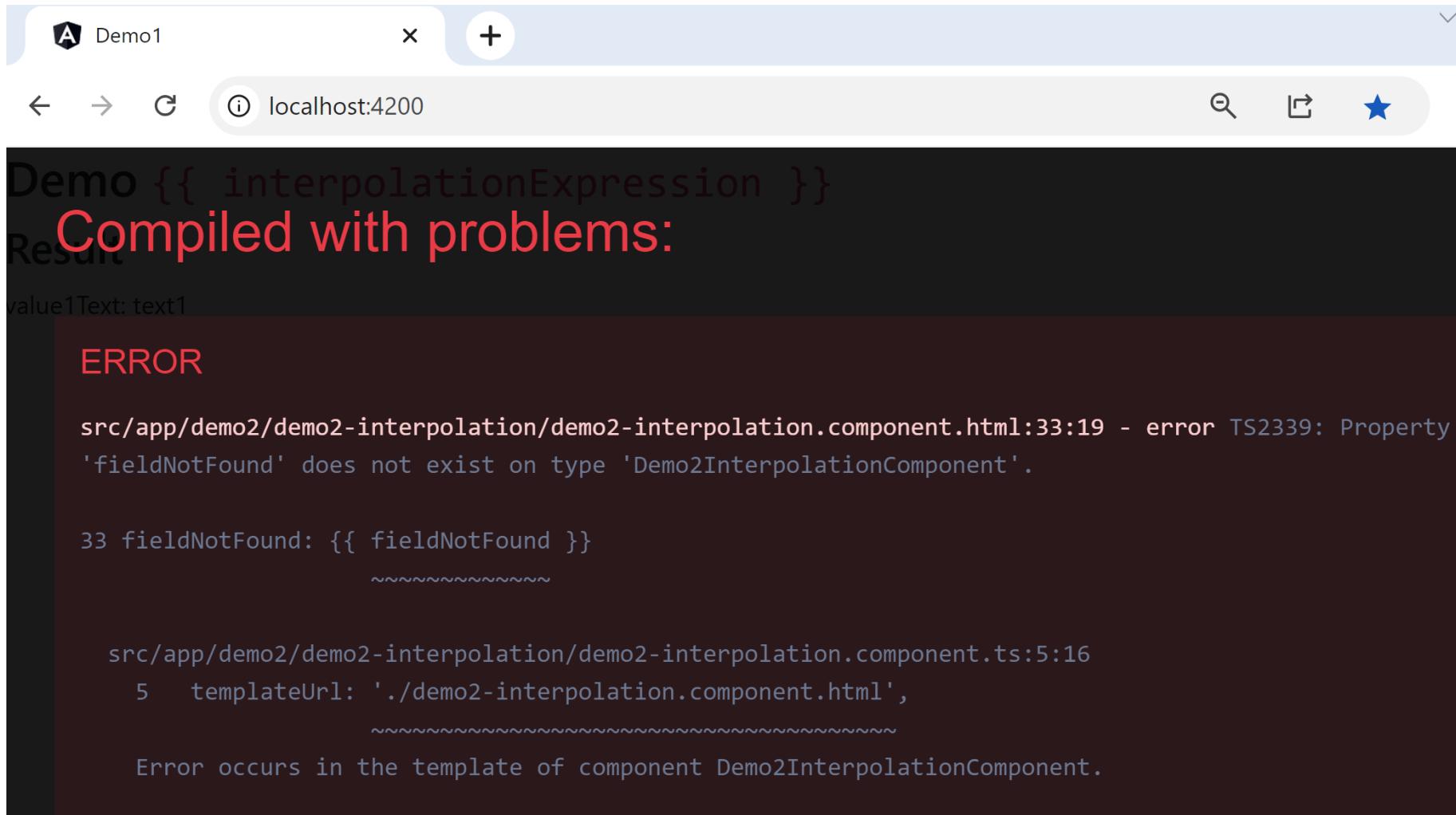


The screenshot shows the Chrome DevTools Elements tab with the following DOM structure:

```
<!DOCTYPE html>
<html lang="en">
  <head> ...
  </head>
  <body>
    <app-root _ngc="c1933515739" ng-version="16.1.5">
      <app-demo2-interpolation _ngcontent-c1933515739>
        <h2> ...
        <h3>Result</h3>
        ...
        " value1Text: text1 " == $0
      </app-demo2-interpolation>
    </app-root>
    <script src="runtime.js" type="module"></script>
    <script src="polyfills.js" type="module"></script>
    <script src="styles.js" defer></script>
    <script src="vendor.js" type="module"></script>
    <script src="main.js" type="module"></script>
  </body>
</html>
```

The highlighted line in the DOM tree is the expression " value1Text: text1 " == \$0, which corresponds to the interpolation expression shown in the browser's result panel.

# What if {{ fieldNotFound }} ?



# Demo {{ complexExpr }}

```
value1Text = "text1";
value2Text = "text2";

value1Number = 12;
value2Number = 3.14159;

value1Bool = true;
value2Bool = false;

value10bj = { text: "objectText1", fieldInt: 1 };

get value1Getter(): string { return this.value1Text; }

function1(): string { return this.value1Text; }

{{ value1Text }}
{{ value1Number }}
{{ value1Text + value2Text }}
{{ value1Number + value2Number }}
{{ value1Text + ' concatenate ' + value1Number }}
{{ (value1Number > 10)? '>10' : '<10' }}
{{ value1Obj.text }}
{{ value1Getter }}
{{ function1() }}
change field values
value1Text:
```

value1Text	text1
value1Number	12
value1Text + value2Text	text1text2
value1Number + value2Number	15.14159
value1Text + ' concatenate ' + value1Number	text1 concatenate 12
(value1Number > 10)? '>10' : '<10'	>10
value1Obj.text	objectText1
value1Getter	text1
function1()	text1
change field values	
value1Text:	text1

# Outline

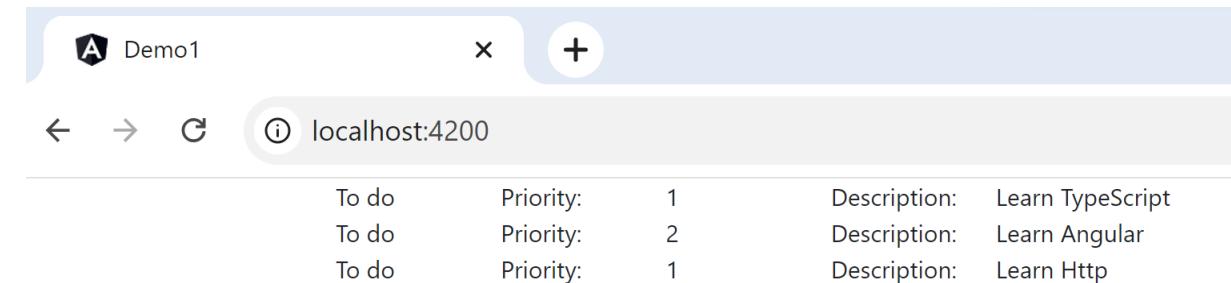


- templateHtml {{ expr }}
- \*ngFor
- \*ngIf
- Chrome Debugger
- @Component
- [] .. @Input field
- () .. @Output changed
- [()] .. Bidirectional Binding
- router

# Demo \*ngFor

Expected html ...  
but without copy&paste

```
app.component.html x
1
2 <div class="container">
3   <div class="row">
4     <label class="col-md-1">To do</label>
5     <label class="col-md-1">Priority: </label>
6     <span class="col-md-1">1</span>
7     <label class="col-md-1">Description: </label>
8     <span class="col-md-8">Learn TypeScript</span>
9   </div>
10  <div class="row">
11    <label class="col-md-1">To do</label>
12    <label class="col-md-1">Priority: </label>
13    <span class="col-md-1">2</span>
14    <label class="col-md-1">Description: </label>
15    <span class="col-md-8">Learn Angular</span>
16  </div>
17  <div class="row">
18    <label class="col-md-1">To do</label>
19    <label class="col-md-1">Priority: </label>
20    <span class="col-md-1">1</span>
21    <label class="col-md-1">Description: </label>
22    <span class="col-md-8">Learn Http</span>
23  </div>
24 </div>
```



A screenshot of a web browser window titled "Demo1" showing a table of tasks. The table has four columns: "To do", "Priority:", "Description:", and another "Description:" column. There are three rows of data.

To do	Priority:	Description:	Description:
To do	Priority:	1	Learn TypeScript
To do	Priority:	2	Learn Angular
To do	Priority:	1	Learn Http

# \*ngFor: Applying MVC Design Pattern : splitting Model (json / javascript) – View (Html)

The image shows a code editor with two tabs: `app.component.ts` and `app.component.html`.

**app.component.ts:**

```
1 import { Component } from '@angular/core';
2
3 interface TodoItem {
4     priority: number;
5     description: string;
6 }
7
8 @Component({
9     selector: 'app-root',
10    templateUrl: './app.component.html',
11    styleUrls: ['./app.component.css']
12 })
13 export class AppComponent {
14     title = 'demo1';
15
16     items: TodoItem[] = [
17         { priority: 1, description: 'learn TypeScript' },
18         { priority: 2, description: 'learn Angular' },
19         { priority: 1, description: 'learn Http' },
20     ];
}
```

**app.component.html:**

```
1
2
3
4
5 <div class="container">
6     <div class="row" *ngFor="let item of items">
7         <label class="col-md-1">To do</label>
8         <label class="col-md-1">Priority: </label>
9         <span class="col-md-1">{{item.priority}}</span>
10        <label class="col-md-1">Description: </label>
11        <span class="col-md-8">{{item.description}}</span>
12     </div>
13 </div>
14
15
16
17
18
19
20
```

The `*ngFor` directive in the `app.component.html` file is highlighted in yellow.

# \*ngFor

← → ⌂ angular.io/api/common/NgFor



DOCS

COMMUNITY

BLOG ↗

Se

Introduction

Getting started >

Understanding Angular >

Developer guides >

Best practices >

Angular tools >

## Description

The `ngForOf` directive is generally used in the **shorthand form** `*ngFor`. In this form, the template to be rendered for each iteration is the content of an anchor element containing the directive.

The following example shows the shorthand syntax with some options, contained in an `<li>` element.

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



Local variable aliases:

Index, count, first / last, even / odd

trackBy:

For incremental optims and transitions

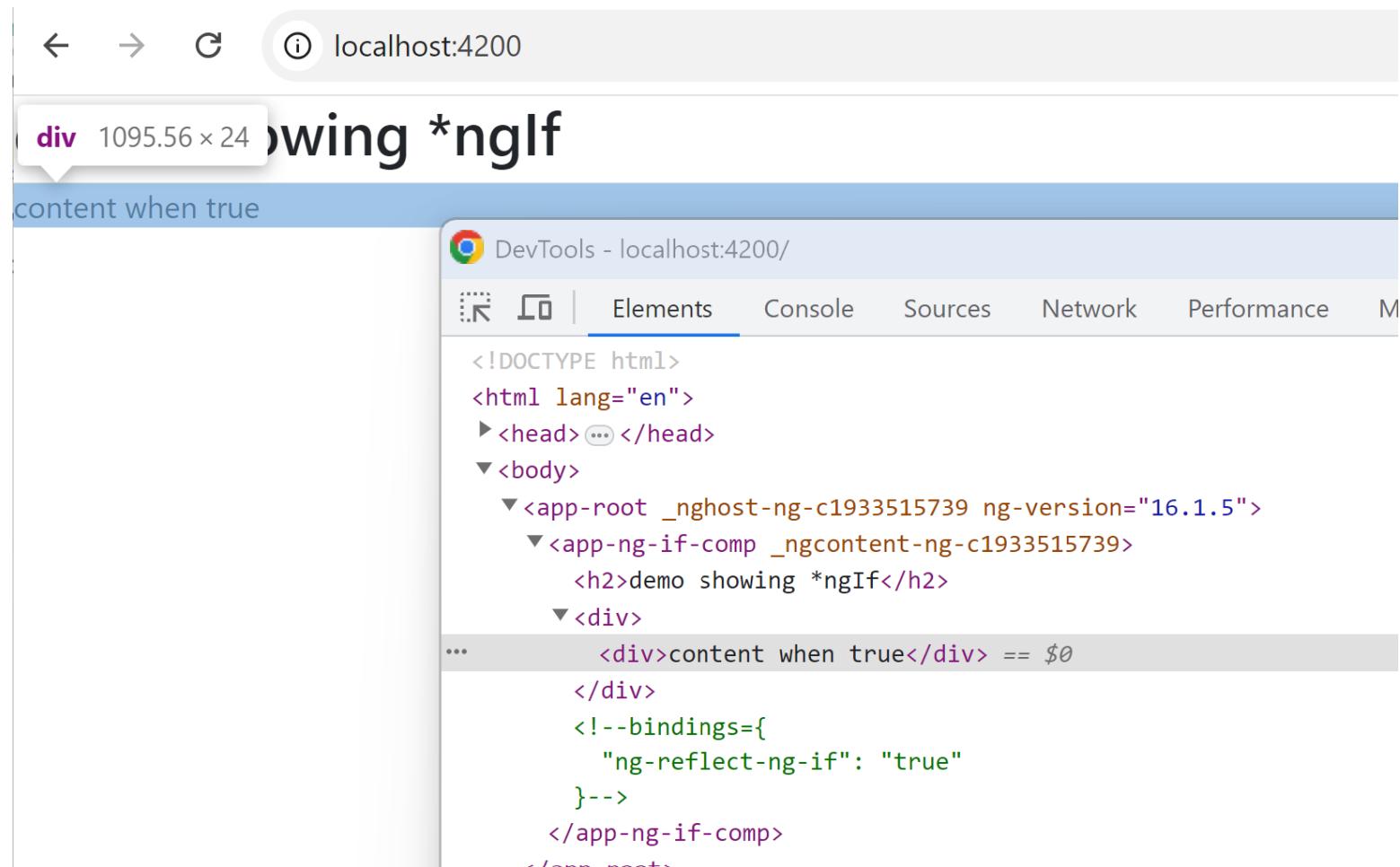
# Outline

templateHtml {{ expr }}  
\*ngFor  
\*ngIf  
 Chrome Debugger  
@Component  
[] .. @Input field  
() .. @Output changed  
[()] .. Bidirectional Binding  
router

# \*ngIf



# Result \*ngIf => attribute removed, comment added



# \*ngIf when false => NO content

The screenshot shows a browser window at localhost:4200 displaying two sections of content:

- demo showing \*ngIf**: This section contains the text "content when true".
- demo showing \*ngIf using !condition**: This section also contains the text "content when true".

Below the browser window, the DevTools Elements tab is open, showing the DOM structure:

```
<!DOCTYPE html>
<html lang="en">
  <head> ...
  <body>
    <app-root _nghost-ng-c1933515739 ng-version="16.1.5">
      <app-ng-if-comp _ngcontent-ng-c1933515739>
        <h2>demo showing *ngIf</h2>
        <div>
          <div>content when true</div>
        </div>
        <!--bindings={<br/>
          "ng-reflect-ng-if": "true"<br/>
        }-->
        <h2>demo showing *ngIf using !condition</h2>
        ...
        <!--bindings={<br/>
          "ng-reflect-ng-if": "false"<br/>
        }--> == $0
      </app-ng-if-comp>
    </app-root>
```

\*ngIf ... ; else contentTemplate  
=> ng-template

Solution 1 : use both \*ngIf="« condition »" and \*ngIf="« ! condition »"

Solution 2 : use ng-template:

```
<div *ngIf="condition; else templateWhenFalse">
  <div>content when true</div>
</div>
<ng-template #templateWhenFalse>
  <div>content when false</div>
</ng-template>
```

Solution 3 : use \*ngSwitch

# How to avoid extra <div> on \*ngIf with multiple elements content ?

```
<span>
  <div *ngIf="condition">
    <span>element 1</span>
    <span>element 2</span>
  </div>
</span>
```

Actual content:



```
<span>
  <div>
    <span>element 1</span>
    <span>element 2</span>
  </div>
</span>
```

Possible problems:  
CSS, divs overhead

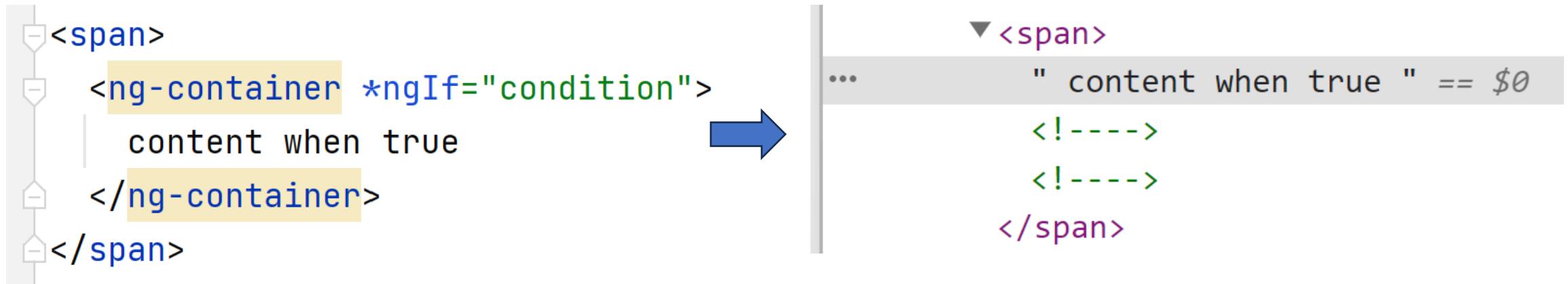
How to ???

INSTEAD... Expected content:



```
<span>
  <span>element 1</span>
  <span>element 2</span>
</span>
```

# Using <ng-container \*ngIf="...">



# Demo \*ngIf

A screenshot of a web browser window displaying a demo application at localhost:4200. The page title is "demo showing \*ngIf". The content area contains several examples of the \*ngIf directive:

- content when true**: A bolded section containing three examples.
- demo showing \*ngIf using !condition**: Shows the directive with a negated condition.
- demo showing \*ngIf with else ng-template #templateId**: Shows the directive with an else template.
- content when true**: Another bolded section.
- Button to toggle model condition**: A heading followed by a button labeled "Toggle condition". Below the button, the text "condition = true" is displayed.
- demo showing ng-container: no extra div or span inserted**: A bolded section.
- content when true**: A final bolded section.

# Remove <!-- bindings ..comment..--> ? « development » vs « production » conf

**Default conf « development » :**

```
$ ng serve  
(implicit: ng serve -c development )  
=> Has Chrome debugging comment
```

```
...      <!--bindings={  
    "ng-reflect-ng-if": "false"  
}--> == $0
```

**conf « production » :**

```
$ ng serve -c production  
=> Still empty placeholder comment  
but NO debugging info in it
```

```
...      <!----> == $0
```

# ng serve --help ; ng serve -c production

```
$ ng serve --help  
ng serve [project]
```

Builds and serves your application, rebuilding on file changes.

#### Arguments:

project	The name of the project to build. Can be an application or a library.	[string] [choices: "demo1"]
---------	---	-----------------------------

#### Options:

-h, --help	Shows a help message for this command in the console.	[boolean]
-c, --configuration	One or more named builder configurations as a comma-separated list as specified in the "configurations" section in angular.json. The builder uses the named configurations to run the given target. For more information, see <a href="https://angular.io/guide/workspace-config#alternate-build-configurations">https://angular.io/guide/workspace-config#alternate-build-configurations</a> .	[string] [choices: "development", "production"]
--allowed-hosts	List of hosts that are allowed to access the dev server.	[array]

# angular.json « development » vs « production » conf

```
angular.json
61      "defaultConfiguration": "production"
62    },
63    "serve": {
64      "builder": "@angular-devkit/build-angular:dev-server",
65      "configurations": {
66        "production": {
67          "browserTarget": "demo1:build:production"
68        },
69        "development": {
70          "browserTarget": "demo1:build:development"
71        }
72      },
73      "defaultConfiguration": "development"
74    },
--
```

# Outline

templateHtml {{ expr }}

\*ngFor

\*ngIf

Chrome Debugger



@Component

[] .. @Input field

() .. @Output changed

[()] .. Bidirectional Binding

router

# @Component

Goal : Avoid copy & paste,  
Be able to insert re-usable html fragment  
aka « component »

« component declaration » → « component re-use »

( Like « method declaration » → method call )

\$ ng generate component

(short) \$ ng g c

```
$ ng g c demo2/mycomp
CREATE src/app/demo2/mycomp/mycomp.component.html (21 bytes)
CREATE src/app/demo2/mycomp/mycomp.component.spec.ts (559 bytes)
CREATE src/app/demo2/mycomp/mycomp.component.ts (202 bytes)
CREATE src/app/demo2/mycomp/mycomp.component.css (0 bytes)
UPDATE src/app/app.module.ts (793 bytes)
```

# Generated @Component code

1 dir + 4 new files added

mycomp

- mycomp.component.css
- mycomp.component.html
- mycomp.component.spec.ts
- mycomp.component.ts →

+ 2 lines inserted in module  
(cf next !)

app.module.ts

```
mycomp.component.html
<p>mycomp works!</p>
```

```
mycomp.component.ts
import { Component } from '@angular/core';
@Component({
  selector: 'app-mycomp',
  templateUrl: './mycomp.component.html',
  styleUrls: ['./mycomp.component.css']
})
export class MycompComponent { }
```

# Using @Component

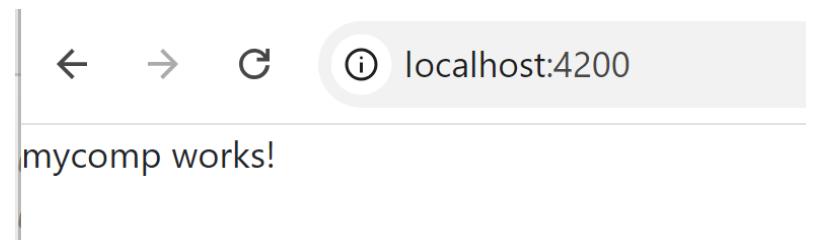
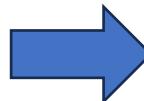
... similar to html « Custom Element »

**Declare component =**  
**assign selector (element name) to class**

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

**Use component =**  
**use selector as element type in html**

```
<app-mycomp></app-mycomp>
```



# Short @Component declaration

1/ **styleUrls** is optional (why global CSS overwrite?)



```
styleUrls: ['./mycomp.component.css']
```

2/ **templateUrl** can be inlined (why using a file for 1..2 lines?)



```
templateUrl: './mycomp.component.html',
```



```
template: `<p>mycomp works!</p>`
```

3/ **\*.spec.ts generated unit tests may be removed or completed**



( do not have assertions!)

# Short @Component declaration



The image shows a code editor interface with a file tree on the left and a code editor window on the right.

**File Tree:**

- mycomp (selected)
- mycomp.component.ts

**Code Editor (mycomp.component.ts):**

```
1 import { Component } from '@angular/core';
2
3 @Component({
4   selector: 'app-mycomp',
5   template: `<p>mycomp works!</p>`
6 })
7 export class MycompComponent {
8
9 }
```

# @Component... ADD in @NgModule declarations: [...] !!!

1 dir + 4 new files added

```
▼ └── mycomp
    ├── mycomp.component.css
    ├── mycomp.component.html
    ├── mycomp.component.spec.ts
    └── mycomp.component.ts
```

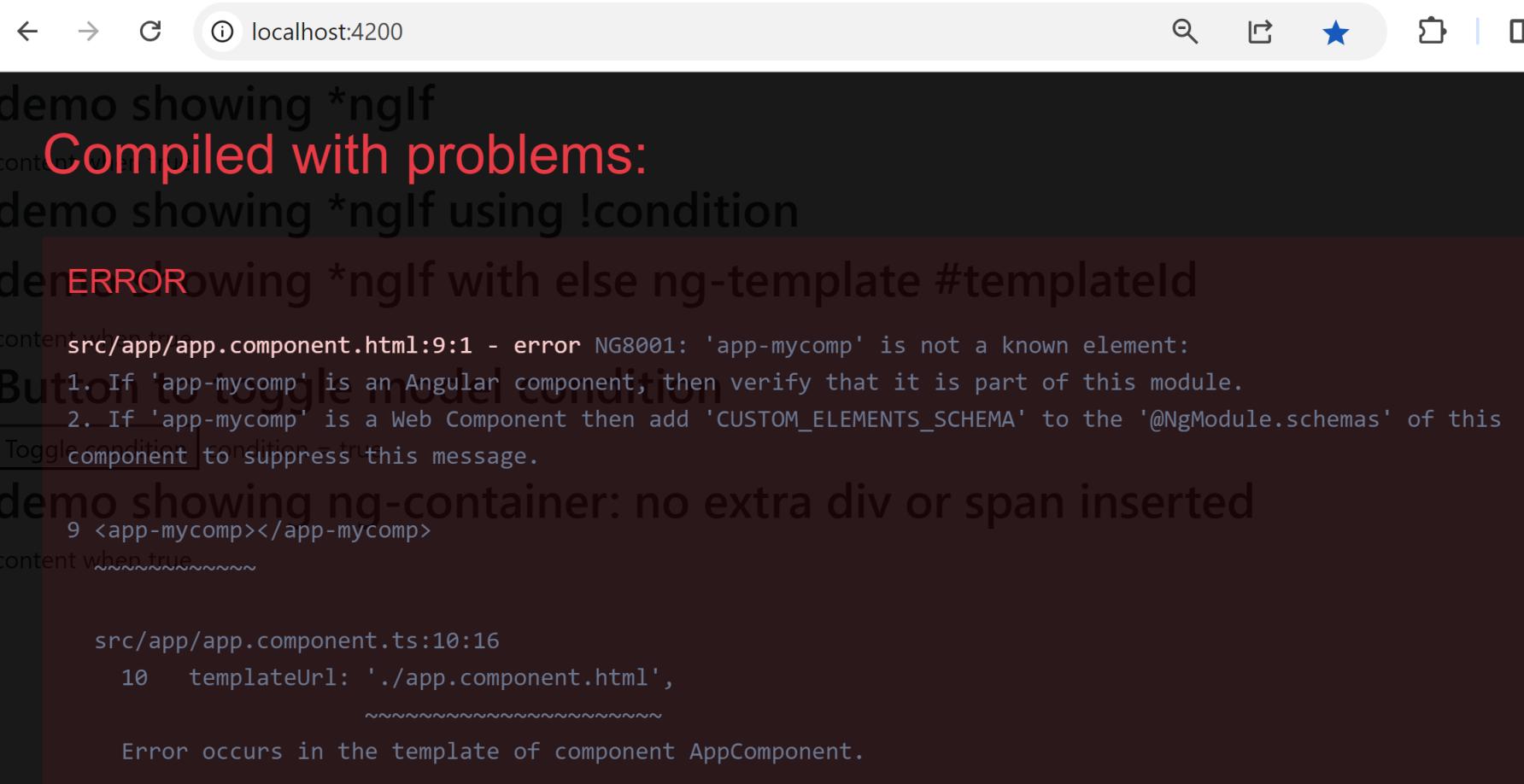
+ 2 lines inserted  
in existing module !

app.module.ts

```
import { MycompComponent } from './demo2/mycomp/mycomp.component';

@NgModule({
  declarations: [
    AppComponent,
    MycompComponent,
  ],
  imports: [
```

# Using component, but forgetting to add in module



A screenshot of a web browser window displaying an Angular application at localhost:4200. The page title is "demo showing \*ngIf". The main content area shows several examples of using the \*ngIf directive:

- "Compiled with problems:"
- "demo showing \*ngIf using !condition"
- "den~~ERRO~~ showing \*ngIf with else ng-template #templateId"
- "src/app/app.component.html:9:1 - error NG8001: 'app-mycomp' is not a known element:
  - 1. If 'app-mycomp' is an Angular component, then verify that it is part of this module.
  - 2. If 'app-mycomp' is a Web Component then add 'CUSTOM\_ELEMENTS\_SCHEMA' to the '@NgModule.schemas' of this component to suppress this message.
- "Toggle condition: condition: true"
- "demo showing ng-container: no extra div or span inserted"

The browser interface includes standard navigation buttons (back, forward, search), a star icon for bookmarks, and a copy/paste/share toolbar.

Edit, copy&paste  
... file module.ts becomes unmaintainable

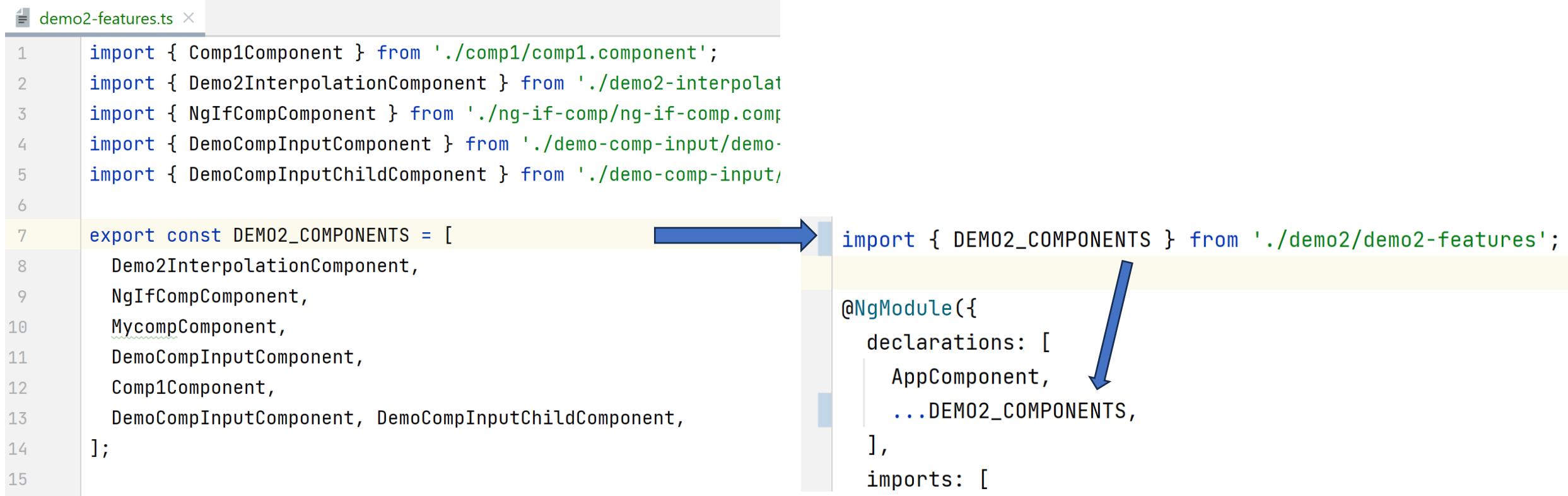
⇒ Avoid modifying too many git commits / conflicts in same file

Common practices :

Solution 1/ split module in sub-modules

Solution 2/ split array declarations into constants sub-arrays

# Split declarations with array (...spreading) const

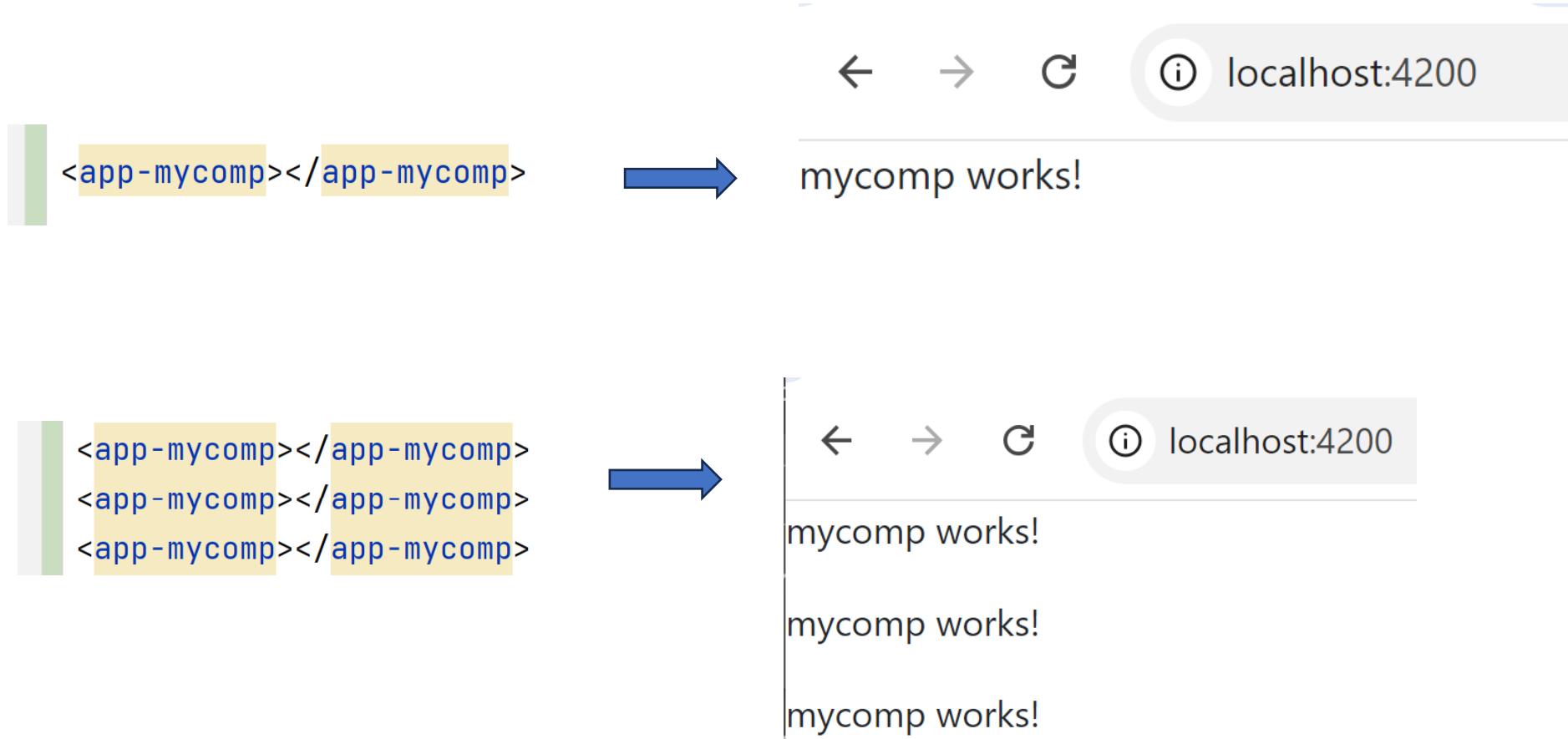


The diagram illustrates the flow of component declarations from a source file to a destination file. A blue arrow points from the declaration in `demo2-features.ts` to its import statement in `app.module.ts`. Another blue arrow points from the import statement in `app.module.ts` back to the declaration in `demo2-features.ts`, indicating a bidirectional relationship or a self-referencing import.

```
demo2-features.ts
1 import { Comp1Component } from './comp1/comp1.component';
2 import { Demo2InterpolationComponent } from './demo2-interpolat
3 import { NgIfCompComponent } from './ng-if-comp/ng-if-comp.comp
4 import { DemoCompInputComponent } from './demo-comp-input/demo-
5 import { DemoCompInputChildComponent } from './demo-comp-input/
6
7 export const DEMO2_COMPONENTS = [
8   Demo2InterpolationComponent,
9   NgIfCompComponent,
10  MycompComponent,
11  DemoCompInputComponent,
12  Comp1Component,
13  DemoCompInputComponent, DemoCompInputChildComponent,
14];
15
```

```
app.module.ts
import { DEMO2_COMPONENTS } from './demo2/demo2-features';
@NgModule({
  declarations: [
    AppComponent,
    ...DEMO2_COMPONENTS,
  ],
  imports: [
```

# Using repeated (static) components ... need dynamic binding



# Outline

templateHtml {{ expr }}  
\*ngFor  
\*ngIf  
Chrome Debugger  
@Component  
 [] .. @Input field  
() .. @Output changed  
[()] .. Bidirectional Binding  
router

# Binding between components : pass values / capture changes

- [] One-Way data binding
  - = pass value from parent to child @input field
- ( ) change events binding
  - = handle child @output change by parent code
- [()] input+output = 2-Way data binding
  - = bind parent field to child field

# Declare @Input() field ... then use [field]=«expr»

## Child component.ts

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

@Component({
  selector: 'app-demo-comp-input-child',
  templateUrl: './demo-comp-input-child.component.html'
})
export class DemoCompInputChildComponent {

  @Input()
  field: string = '';

}
```

## Parent component.html



```
<app-demo-comp-input-child  
  [field]="parentValue1"  
></app-demo-comp-input-child>
```

passing parent value to child component:  
`<child [childField]=« parentValue »></child>`

### Parent component.html

```
parentValue1: {{parentValue1}}
<app-demo-comp-input-child [field]="parentValue1">
</app-demo-comp-input-child>
```

### Parent component.ts

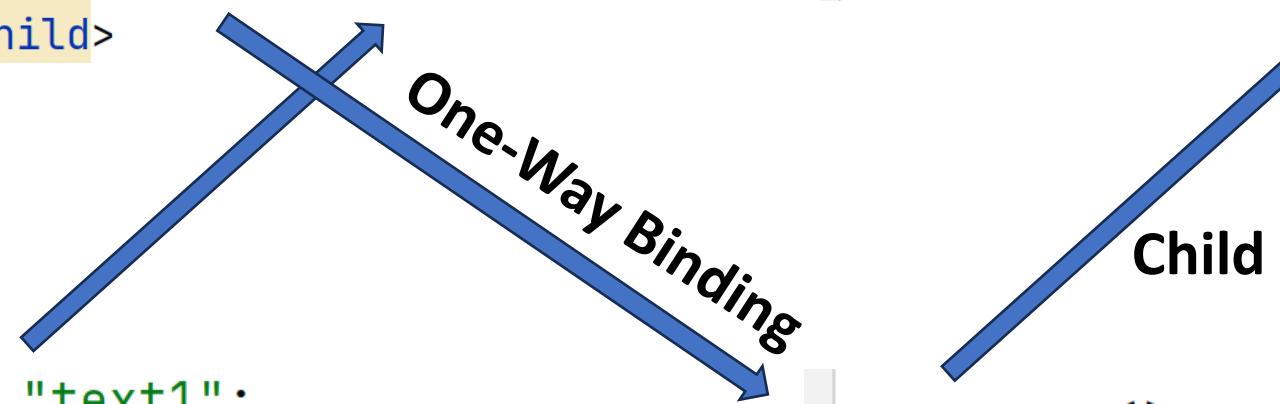
```
parentValue1 = "text1";
```

### Child component.html

```
child component, field: {{field}}
```

### Child component.ts

```
@Input()
field: string = '';
```



# Demo One-Way binding [childField]=« expr »

## Passing value from parent to child component: [childField]="parentValue"

```
change *child.component.ts to declare @Input field
import { Component, Input } from '@angular/core'; .. @Input() field: string = '';
```

```
change *child.component.html to use field
child component, field: {{field}}
```

```
comp using field set to parent 'parentValue1' expression:
```

```
parentValue1: text1
child component, field: text1
```

```
comp using field set to parent 'parentValue2' expression:
```

```
parentValue2: text2
child component, field: text2
```

```
comp without field set:
```

```
child component, field:
```

```
comp with field set to constant text:
```

```
child component, field: constant text
```

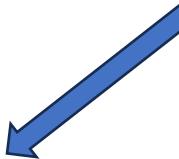
# Outline

templateHtml {{ expr }}  
\*ngFor  
\*ngIf  
Chrome Debugger  
@Component  
[] .. @Input field  
 () .. @Output changed  
[()] .. Bidirectional Binding  
router

# Using button callback: (click)=« code...»

```
<button (click)="onClick()">Click</button>
```

```
onClick() {  
  console.log("button clicked!");  
}
```

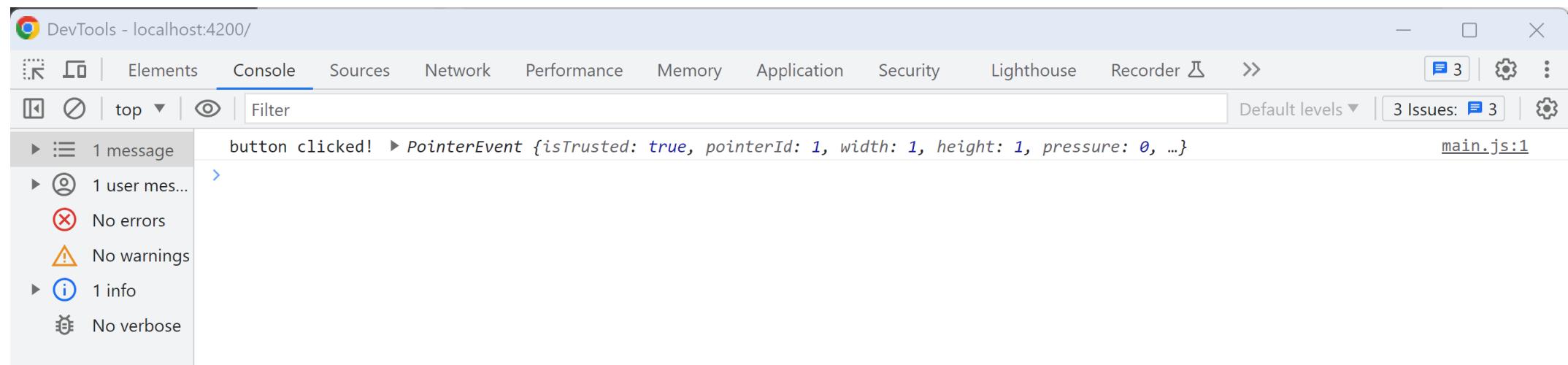


# (click) using implicit parameter « \$event »

```
<button (click)="onClick($event)">Click</button>
```

```
onClick($event: Event) {  
  console.log("button clicked!", $event);  
}
```

# Click Demo ... log in Chrome F12 > Console



# \$event detail

The screenshot shows the Chrome DevTools interface with the 'Console' tab selected. The left sidebar displays a summary of log entries: 9 messages, 9 user messages, No errors, No warnings, 9 info, and No verbose. The main content area shows a series of 'button clicked!' events, each represented as a `PointerEvent` object. The first few events are expanded to show their properties. The properties listed include: `isTrusted: true`, `pointerId: 1`, `width: 1`, `height: 1`, `pressure: 0`, `...}`, `button: 0`, `buttons: 0`, `cancelBubble: false`, `cancelable: true`, `clientX: 224`, `clientY: 86`, `composed: true`, `ctrlKey: false`, `currentTarget: null`, `defaultPrevented: false`, `detail: 3`, `eventPhase: 0`, `fromElement: null`, `height: 1`, `isPrimary: false`, `layerX: 224`, `layerY: 86`, `metaKey: false`, `movementX: 0`, `movementY: 0`, `offsetX: 155`, `offsetY: 7`, `pageX: 224`, `pageY: 86`, `pointerId: 1`, `pointerType: "mouse"`, `pressure: 0`, `relatedTarget: null`, `returnValue: true`, `screenX: 738`, `screenY: 182`, `shiftKey: false`, `sourceCapabilities: InputDeviceCapabilities {firesTouchEvents: false}`, `srcElement: button`, `tangentialPressure: 0`, `target: button`, `tiltX: 0`, `tiltY: 0`, `timeStamp: 157343.9000000596`, `toElement: null`, `twist: 0`, `type: "click"`, and `view: Window {window: Window, self: Window, document: document, name: "", location: Location, ...}`. The right sidebar shows 'Default levels' and '3 Issues' with links to 'main.js:1'. The bottom right corner shows the file path 'main.js:1'.

```
button clicked! > PointerEvent {isTrusted: true, pointerId: 1, width: 1, height: 1, pressure: 0, ...}
button clicked! > PointerEvent {isTrusted: true, pointerId: 1, width: 1, height: 1, pressure: 0, ...}
button clicked! > PointerEvent {isTrusted: true, pointerId: 1, width: 1, height: 1, pressure: 0, ...}
button clicked! > PointerEvent {isTrusted: true, pointerId: 1, width: 1, height: 1, pressure: 0, ...}
    isTrusted: true
    altKey: false
    altitudeAngle: 1.5707963267948966
    azimuthAngle: 0
    bubbles: true
    button: 0
    buttons: 0
    cancelBubble: false
    cancelable: true
    clientX: 224
    clientY: 86
    composed: true
    ctrlKey: false
    currentTarget: null
    defaultPrevented: false
    detail: 3
    eventPhase: 0
    fromElement: null
    height: 1
    isPrimary: false
    layerX: 224
    layerY: 86
    metaKey: false
    movementX: 0
    movementY: 0
    offsetX: 155
    offsetY: 7
    pageX: 224
    pageY: 86
    pointerId: 1
    pointerType: "mouse"
    pressure: 0
    relatedTarget: null
    returnValue: true
    screenX: 738
    screenY: 182
    shiftKey: false
    sourceCapabilities: InputDeviceCapabilities {firesTouchEvents: false}
    srcElement: button
    tangentialPressure: 0
    target: button
    tiltX: 0
    tiltY: 0
    timeStamp: 157343.9000000596
    toElement: null
    twist: 0
    type: "click"
    view: Window {window: Window, self: Window, document: document, name: "", location: Location, ...}
[[Prototype]]: PointerEvent
```

Default levels ▾ 3 Issues: 3 | main.js:1  
main.js:1  
main.js:1  
main.js:1  
main.js:1

# Chrome Debug breakpoint

The screenshot shows the Google Chrome DevTools interface with the Sources tab selected. On the left, the file tree displays the project structure under the 'localhost:4200' node. The 'demo-comp-output-child.component.ts' file is open, showing a code snippet with a breakpoint at line 10. The right side of the screen features the Breakpoints sidebar, which indicates that the script is 'Paused on breakpoint'. The sidebar lists several configuration options and a detailed breakdown of the current pause point, including the stack trace and the specific line of code where the event occurred.

DevTools - localhost:4200/

Elements Console Sources Network Performance Memory Application Security Lighthouse Recorder Performance insights

Page Filesystem Overrides Content scripts > :

demo-comp-output-child.component.ts

```
1 import { Component, Output } from '@angular/core';
2
3 @Component({
4   selector: 'app-demo-comp-output-child',
5   templateUrl: './demo-comp-output-child.component.html'
6 })
7 export class DemoCompOutputChildComponent {
8
9   onClick($event: Event) { $event = PointerEvent {isTrusted: true, pointerId: 1, width: 1, height: 1}; console.log("button clicked!", $event);
10 }
11
12
13 }
14
```

top

localhost:4200

- (index)
- main.js
- polyfills.js
- runtime.js
- styles.js
- vendor.js
- styles.css

Selenium IDE

webpack://

- node\_modules
- src
- app
- demo2
- comp1
- demo2-interpolation
- demo-comp-input
- demo-comp-output
  - demo-comp-output-child.component.html
  - demo-comp-output-child.component.ts
  - demo-comp-output.component.html
  - demo-comp-output.component.ts
- mycomp
- ng-if-comp
- app-routing.module.ts
- app.component.html
- app.component.ts
- app.module.ts

main.ts

styles.css

styles.css

webpack

Paused on breakpoint

Breakpoints

- Pause on uncaught exceptions
- Pause on caught exceptions

demo-comp-output-child.component.ts

console.log("button clicked!", \$event);

Scope

Local

- this: DemoCompOutputChildComponent
- \$event: PointerEvent {isTrusted: true, pointerId: 1, width: 1, height: 1}

Closure (274)

Script

Global

Call Stack

Show ignore-listed frames

onClick

DemoCompOutputChildComponent\_Template\_button\_click\_6\_listener

DemoCompOutputChildComponent\_Template

Zone - HTMLButtonElement.addEventListener:click (async)

DemoCompOutputChildComponent\_Template

Promise.then (async)

4913

\_webpack\_exec\_

(anonymous)

(anonymous)

(anonymous)

XHR/fetch Breakpoints

DOM Breakpoints

Global Listeners

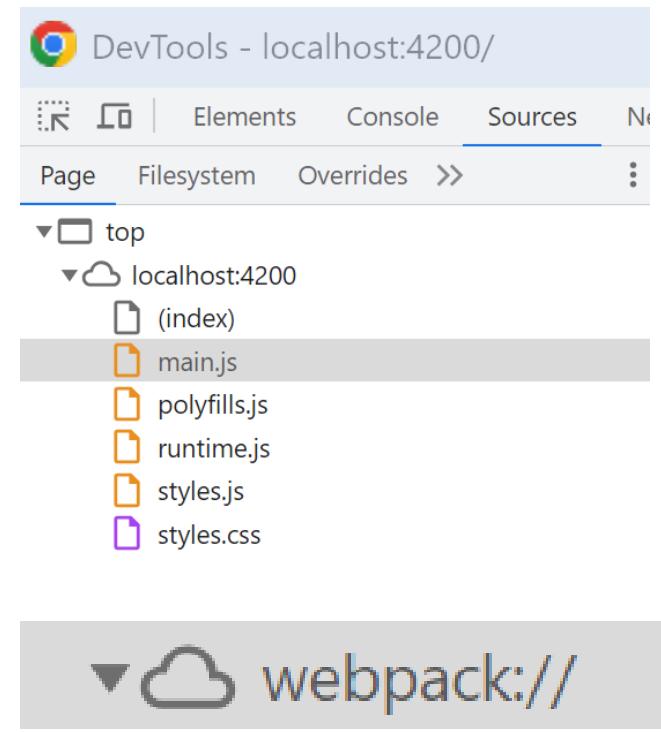
Event Listener Breakpoints

CSP Violation Breakpoints

Notice .. « ng serve -c production »  
=> NO source « .map » code !

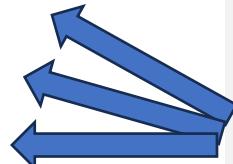
Can not browse to put breakpoint (obfuscated code)

Remark  
Source .map



# (click) on button is built-in ... you can define your own `@Output()`

`@Output()` =  
**Observable**  
**subscribed**  
from parent  
components



```
import { Component, Output } from '@angular/core';
import { Observable, Subject } from 'rxjs';

export interface FieldEvent {
  message: string;
}

@Component({
  selector: 'app-demo-comp-output-child',
  templateUrl: './demo-comp-output-child.component.html'
})
export class DemoCompOutputChildComponent {

  @Output()
  fieldChange = new Subject<FieldEvent>();

  onClickCustomEmit() {
    this.fieldChange.next({ message: "event message" });
  }

  onClick($event: Event) {
    console.log("button clicked!", $event);
  }
}
```

.next(event)

Triggered in child component (click)

# Subscribe to @Output : (childFieldChange)=« parentCallback (\$event) »



```
demo-comp-output.component.html
1
2 <app-demo-comp-output-child
3   (fieldChange)="parentCallback($event)"
4 ></app-demo-comp-output-child>
5
```

```
demo-comp-output.component.ts
1 import { Component } from '@angular/core';
2 import { FieldEvent } from './demo-comp-output-child.component';
3
4 @Component({
5   selector: 'app-demo-comp-output',
6   templateUrl: './demo-comp-output.component.html'
7 })
8 export class DemoCompOutputComponent {
9
10   parentCallback(event: FieldEvent) {
11     console.log("DemoCompOutputComponent.parentCallback", event);
12   }
13
14 }
```

# Chrome Debugger > « Call Stack » (parent ts)

The screenshot shows the Chrome Debugger interface with the 'Call Stack' panel highlighted by a green box. A blue arrow points from the left margin of the code editor towards the Call Stack panel.

**Code Editor (demo-comp-output.component.ts):**

```
1 import { Component } from '@angular/core';
2 import { FieldEvent } from './demo-comp-output-child.component';
3
4 @Component({
5   selector: 'app-demo-comp-output',
6   templateUrl: './demo-comp-output.component.html'
7 })
8 export class DemoCompOutputComponent {
9
10   parentCallback(event: FieldEvent) { event = {message: 'event message'}
11     console.log("DemoCompOutputComponent.parentCallback", event);
12   }
13
14 }
15
16
17
18 // parentCallback2(event: FieldEvent) {
19 //   console.log("DemoCompOutputComponent.parentCallback2", event);
20 // }
```

**Call Stack Panel:**

- Paused on breakpoint
- ▶ Watch
- ▼ Breakpoints
- Pause on uncaught exceptions
- Pause on caught exceptions
- ▼ demo-comp-output.component.ts
- console.log("DemoCompOutputComponent.parentCallback", event); 11
- ▼ Scope
- ▼ Local
  - ▶ this: DemoCompOutputComponent
  - ▶ event: {message: 'event message'}
  - ▶ Closure (8371)
  - ▶ Script
  - ▶ Global
- ▼ Call Stack
- Show ignore-listed frames
- ▶ parentCallback demo-comp-output.component.ts:11
  - DemoCompOutputComponent\_template\_app\_demo\_comp\_output\_child\_fieldChange\_0\_lis... demo-comp-output.component.html:3
  - onClickCustomEmit demo-comp-output.component.ts:18
  - DemoCompOutputChildComponent\_Template\_button\_click\_4\_listener demo-comp-output.component.html:5
- Zone - HTMLButtonElement.addEventListener:click (async)
  - DemoCompOutputChildComponent\_Template demo-comp-output.component.html:5
  - Promise.then (async)
  - 4913 main.ts:8
  - \_webpack\_exec\_ main.ts:9
  - (anonymous) main.ts:9
  - (anonymous) main.ts:9
  - (anonymous) main.js:2
- ▶ XHR/fetch Breakpoints
- ▶ DOM Breakpoints
- ▶ Global Listeners
- ▶ Event Listener Breakpoints
- ▶ CSP Violation Breakpoints

# Debugger Call Stack .. Parent caller[1] (parent html)

The screenshot shows a debugger interface with two main panes. On the left is a code editor with two tabs: `demo-comp-output.component.ts` and `demo-comp-outpu...component.html`. The `demo-comp-outpu...component.html` tab contains the following template code:

```
1<app-demo-comp-output-child
2  (fieldChange)="parentCallback($event)"
3  ></app-demo-comp-output-child>
4
5
6
7
8<!--
9<app-demo-comp-output-child
10   (fieldChange)="parentCallback2($event)"
11  ></app-demo-comp-output-child>
12-->
13
```

The line `(fieldChange)="parentCallback($event)"` is highlighted in blue. On the right is a debugger sidebar with the following sections:

- Paused on breakpoint**: Shows a yellow bar at the top.
- Breakpoints**: A list of breakpoints, with one entry for `demo-comp-output.component.ts` checked.
- Scope**: Shows the variable `$event` with the value `{message: 'event message'}`.
- Call Stack**: A list of frames:
  - `parentCallback` at `demo-comp-outpu...omponent.ts:11`
  - `DemoCompOutputComponent_Template_app_demo_comp_output_child_fieldChange_0_lis...` at `demo-comp-outpu...mponent.html:3` (highlighted with a green box and a blue arrow pointing from the code editor to it)
  - `onClickCustomEmit` at `demo-comp-outpu...omponent.ts:18`
  - `DemoCompOutputChildComponent_Template_button_click_4_listener` at `demo-comp-outpu...mponent.html:5`
- Zone - HTMLButtonElement.addEventListener:click (async)**: Shows the zone where the event listener was added.
- Promise.then (async)**: Shows the promise chain.

# Debugger Call Stack .. Parent[2] caller (child ts)

The screenshot shows a debugger interface with two main panes. On the left is a code editor for `demo-comp-output-child.component.ts`, displaying TypeScript code for a child component. On the right is a call stack window.

**Code Editor (Left):**

```
1 import { Component, Output } from '@angular/core';
2 import { Observable, Subject } from 'rxjs';
3
4 export interface FieldEvent {
5   message: string;
6 }
7
8 @Component({
9   selector: 'app-demo-comp-output-child',
10  templateUrl: './demo-comp-output-child.component.html'
11})
12 export class DemoCompOutputChildComponent {
13
14   @Output()
15   fieldChange = new Subject<FieldEvent>();
16
17   onClickCustomEmit() {
18     this.fieldChange.next({ message: "event message" });
19   }
20
21   onClick($event: Event) {
22     console.log("button clicked!", $event);
23   }
24
25 }
26
```

**Call Stack Window (Right):**

- Paused on breakpoint
- Watch
- Breakpoints
  - Pause on uncaught exceptions
  - Pause on caught exceptions
  - demo-comp-output.component.ts
    - console.log("DemoCompOutputComponent.parentCallback", event); 11
- Scope
- Local
  - this: DemoCompOutputChildComponent
- Closure (274)
- Script
- Global
- Call Stack
  - Show ignore-listed frames
  - parentCallback demo-comp-output.component.ts:11
  - DemoCompOutputComponent\_Template\_app\_demo\_comp\_output\_child\_fieldChange\_0\_listener demo-comp-output-component.html:2
  - onClickCustomEmit demo-comp-output.component.ts:18
  - DemoCompOutputChildComponent\_Template\_button\_click\_4\_listener demo-comp-output-component.html:5
- Zone - HTMLElement.addEventListener:click (async)
- DemoCompOutputChildComponent\_Template demo-comp-output-component.html:5
- Promise.then (async)
- 4913 main.ts:8
- \_webpack\_exec\_ main.ts:9
- (anonymous) main.ts:9
- (anonymous) main.ts:9
- (anonymous) main.js:2

# Debugger Call Stack ... Parent[3] caller (child html)

The screenshot shows a debugger interface with two main panes. The left pane displays the code for `demo-comp-output-child.component.ts`, which contains a template with a container, a row, a label, and a button. The right pane shows the debugger controls and the call stack.

**Call Stack:**

- parentCallback demo-comp-output...component.ts:11
- DemoCompOutputComponent\_Template\_app\_demo\_comp\_output\_child\_fieldChange\_0\_lis... demo-comp-output...ponent.html:3
- onClickCustomEmit demo-comp-output...ponent.ts:18
- ▶ DemoCompOutputChildComponent\_Template\_button\_click\_4\_listener demo-comp-output...ponent.html:5
- Zone - HTMLElement.addEventListener:click (async)
- DemoCompOutputChildComponent\_Template demo-comp-output...ponent.html:5
- Promise.then (async)
- 4913 main.ts:8
- \_webpack\_exec\_ main.ts:9
- (anonymous) main.ts:9
- (anonymous) main.ts:9
- (anonymous) main.js:2

A green box highlights the entry point in the call stack, and a blue arrow points from the left pane towards the highlighted entry point in the call stack.

```
1<div class="container">
2  <div class="row">
3    <label class="col-md-4">click to emit change</label>
4    <button type="button" class="col-md-2 btn btn-primary" (click)="onClickCustomEmit()">Emit Event</button>
5  </div>
6</div>
7<p>demo-comp-output-child works!</p>
```

() = Change binding  
reverse way: child -> parent

### Parent component.html

```
<app-demo-comp-output-child  
  (fieldChange)="parentCallback($event)"  
></app-demo-comp-output-child>
```

### Parent component.ts

```
parentCallback(event: FieldEvent) {  
  console.log("parentCallback()", event);  
}
```

*Change Binding*

### Child component.html

```
<button (click)="onClickCustomEmit()">(
```

### Child component.ts

```
onClickCustomEmit() {  
  this.fieldChange.next({ message: "event message" });  
}  
  
@Output()  
fieldChange = new Subject<FieldEvent>();
```

# (Reminder) [] = One-Way data binding

## Parent component.html

```
parentValue1: {{parentValue1}}  
<app-demo-comp-input-child [field]="parentValue1">  
</app-demo-comp-input-child>
```

## Parent component.ts

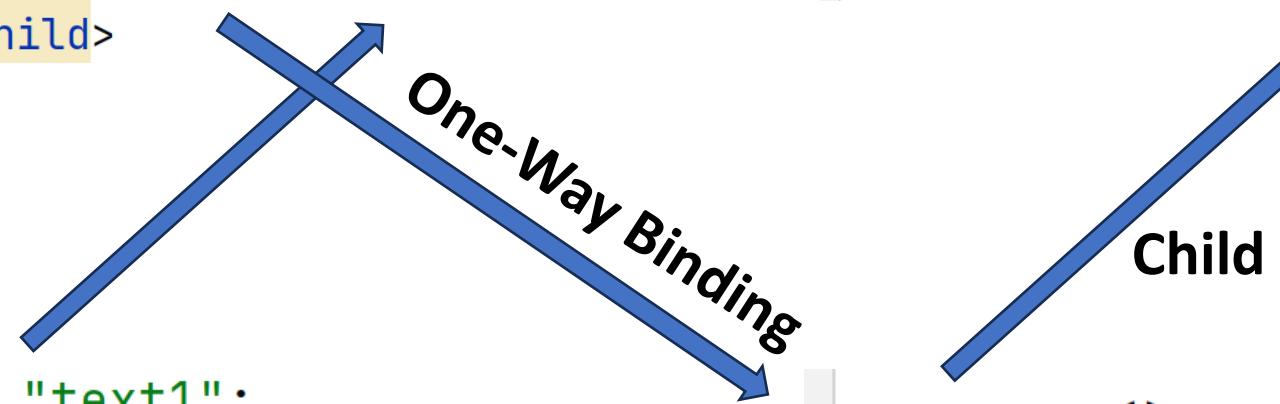
```
parentValue1 = "text1";
```

## Child component.html

```
child component, field: {{field}}
```

## Child component.ts

```
@Input()  
field: string = '';
```

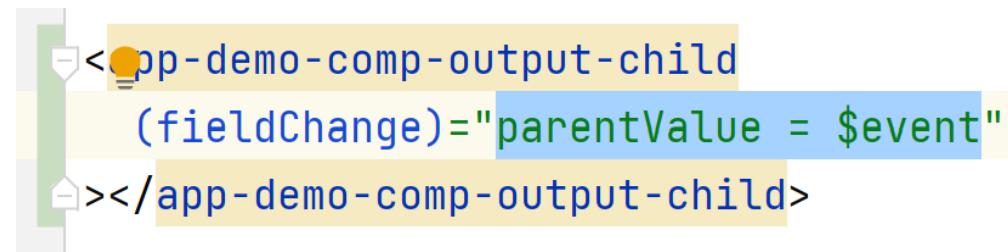


# Outline

templateHtml {{ expr }}  
\*ngFor  
\*ngIf  
Chrome Debugger  
@Component  
[] .. @Input field  
() .. @Output changed  
[()] .. Bidirectional Binding  
router



When event is value type (string .. From Subject<string>)  
Using callback as setter method



# [()] = 2-Way Data Binding

1

[]

[childField]=« parentField »

+

+

+

1

()

(childFieldChange)=« parentField = \$event; »

=

=

=

2

[()]

[(childField)]=« parentField »

# 2-Way Data Binding

naming Convention constraint:  
« field » <-> « fieldChange »

event type constraint:  
\$event is field value

# Mnemonic Tips: [()] instead of ([])

[()] is named = « Banana Box »

... = banana inside a box



# `[(childField)]="parentField"`

## Parent component.html

```
<app-demo2-bidirectionbinding-child  
  [(childField)]="parentField"  
></app-demo2-bidirectionbinding-child>
```

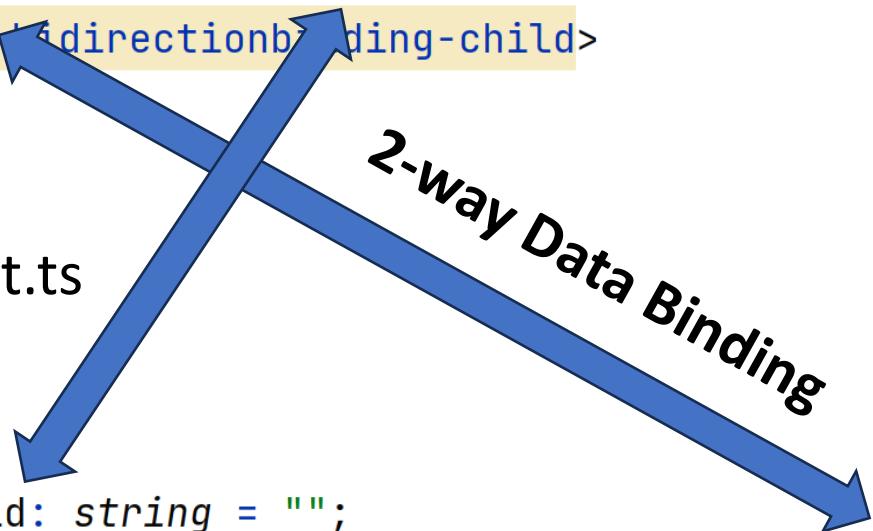
## Parent component.ts

```
parentField: string = "";
```

## Child component.html

## Child component.ts

```
@Input()  
childField: string = "";  
  
@Output()  
childFieldChange = new Subject<string>();
```



# [(ngModel)] ... using FormsModule

```
import { FormsModule } from '@angular/forms';
```

```
@NgModule({  
  declarations: [  
    AppComponent,  
    ...DEMO2_COMPONENTS,  
,  
  imports: [  
    BrowserModule,  
    AppRoutingModule,  
     FormsModule,  
  ]  
})
```

```
<input type="text" [(ngModel)]="field">
```

```
<input type="text" [(ngModel)]="field"  
      (ngModelChange)="onInputChange()">
```

# Demo 2-way data binding

## Parent .html

```
<input class="col-md-8" type="text"  
      [(ngModel)]="parentField">
```

```
<app-demo2-bidirectionbinding-child  
  [(childField)]="parentField"  
></app-demo2-bidirectionbinding-child>
```

## Parent .ts

```
parentField: string = "";
```

## Child .html

```
<label class="col-md-2">childField </label>  
<input class="col-md-8" type="text"  
      [(ngModel)]="childField" (ngModelChange)="onInputChange()">
```

## Child .ts

```
import { Component, Input, Output } from '@angular/core';  
import {Subject} from 'rxjs';  
  
@Component({  
  selector: 'app-demo2-bidirectionbinding-child',  
  templateUrl: './demo2-bidirectionbinding-child.component.html'  
})  
export class Demo2BidirectionbindingChildComponent {  
  
  @Input()  
  childField: string = "";  
  
  @Output()  
  childFieldChange = new Subject<string>();  
  
  onInputChange() {  
    this.childFieldChange.next(this.childField);  
  }  
}
```

# Demo 2-way data binding

A screenshot of a web browser window displaying a 2-way data binding demonstration. The address bar shows the URL `localhost:4200`. The page content includes a form field labeled `parentField:` containing the value `some value`. Below this, there is a list with two items: `child component (1)` and `child component (2)`. To the right of the list is a table structure with two rows. Both rows have two columns: `childField` and `some value`. The entire application interface is styled to look like a standard web browser.

child component (1)	childField	some value	
child component (2)	childField	some value	

# Outline

templateHtml {{ expr }}  
\*ngFor  
\*ngIf  
Chrome Debugger  
@Component  
[] .. @Input field  
() .. @Output changed  
[()] .. Bidirectional Binding  
router



# Routes

URL **/pageX** => **@Component X**

URL **/pageY** => **@Component Y**

May use « hash »:

URL **/#/pageX** => **@Component X**

May use Child-Path and Param:

URL **/#/pageX/childY/id1234**

# RouterModule.forRoot(routes)

```
@NgModule({
  declarations: [
    AppComponent,
    ...DEMO2_COMPONENTS,
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
  ]
})
```

```
app-routing.module.ts ×
1 import { NgModule } from '@angular/core';
2 import { RouterModule, Routes } from '@angular/router';
3
4 import { Page1Component } from './demo2/page1/page1.component';
5 import { Page2Component } from './demo2/page2/page2.component';
6 import { Comp1Component } from './demo2/comp1/comp1.component';
7
8 const routes: Routes = [
9   {path: "page1", component: Page1Component },
10  {path: "page2", component: Page2Component },
11  {path: "demo2-comp1", component: Comp1Component },
12];
13
14 @NgModule({
15   imports: [RouterModule.forRoot(routes)],
16   exports: [RouterModule]
17 })
18 export class AppRoutingModule { }
```

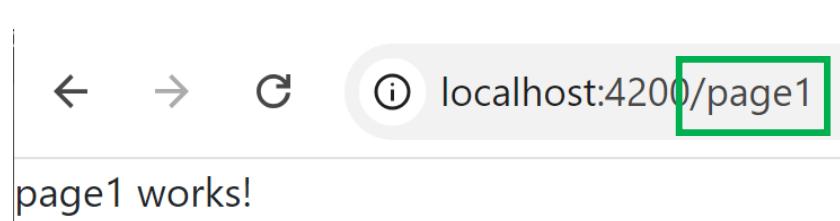
```
app.component.html ×
1 <router-outlet></router-outlet>
```

# Demo Router

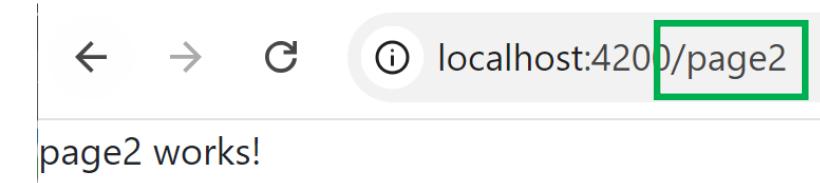
```
$ ng g c demo2/page1
CREATE src/app/demo2/page1/page1.component.html (20 bytes)
CREATE src/app/demo2/page1/page1.component.spec.ts (552 bytes)
CREATE src/app/demo2/page1/page1.component.ts (198 bytes)
CREATE src/app/demo2/page1/page1.component.css (0 bytes)
UPDATE src/app/app.module.ts (792 bytes)
```

```
$ ng g c demo2/page2
CREATE src/app/demo2/page2/page2.component.html (20 bytes)
CREATE src/app/demo2/page2/page2.component.spec.ts (552 bytes)
CREATE src/app/demo2/page2/page2.component.ts (198 bytes)
CREATE src/app/demo2/page2/page2.component.css (0 bytes)
UPDATE src/app/app.module.ts (876 bytes)
```

```
const routes: Routes = [
  {path: "page1", component: Page1Component },
  {path: "page2", component: Page2Component },
];
```



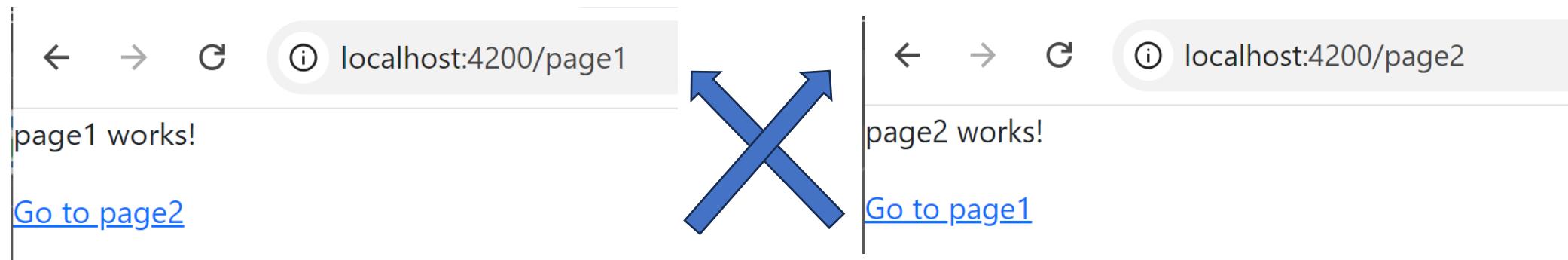
```
const routes: Routes = [
  {path: "page1", component: Page1Component },
  {path: "page2", component: Page2Component },
];
```



# Navigation : <a routerLink=<< /page >>>

```
page1.component.html ×  
1 <p>page1 works!</p>  
2  
3 <a routerLink="/page2">Go to page2</a>  
4
```

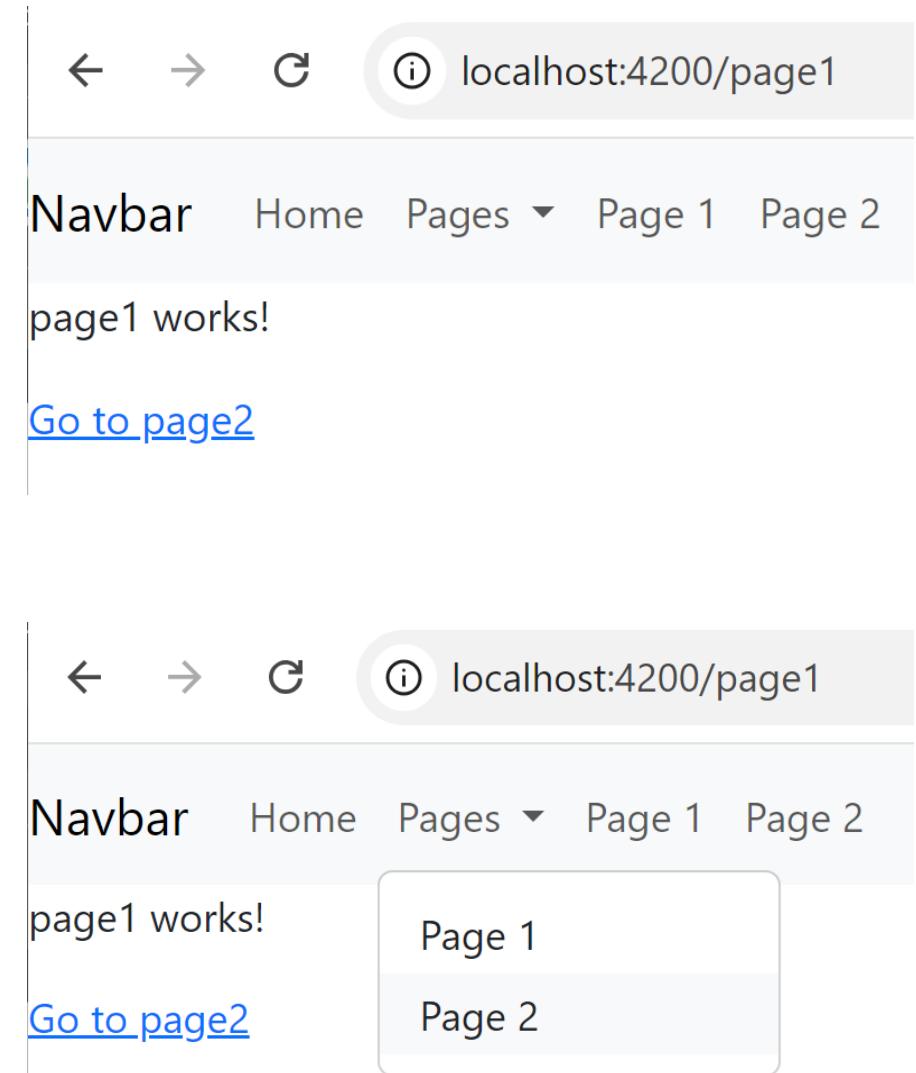
```
page2.component.html ×  
1 <p>page2 works!</p>  
2  
3 <a routerLink="/page1">Go to page1</a>  
4
```



# Main page = NavBar menu + <router-outlet> (+ footer)

```
app.component.html
```

```
16  <nav class="navbar navbar-expand-lg navbar-light bg-light">
17    <a class="navbar-brand" href="#">Navbar</a>
18    <ul class="navbar-nav mr-auto">
19      <li class="nav-item active">
20        <a class="nav-link" href="#">Home</a>
21      </li>
22      <li ngbDropdown class="nav-item">
23        <a ngbDropdownToggle id="navbarMenu1" class="nav-link" role="button">
24          Pages
25        </a>
26        <div ngbDropdownMenu aria-labelledby="navbarMenu1" class="dropdown-menu" >
27          <a class="dropdown-item" routerLink="/page1">Page 1</a>
28          <a class="dropdown-item" routerLink="/page2">Page 2</a>
29        </div>
30      </li>
31      <li class="nav-item">
32        <a class="nav-link" routerLink="/page1">Page 1</a>
33      </li>
34      <li class="nav-item">
35        <a class="nav-link" routerLink="/page2">Page 2</a>
36      </li>
37    </ul>
38  </nav>
39
40  <router-outlet></router-outlet>
```



# Get ActivatedRoute path & param from URL

```
import { ActivatedRoute, Router } from '@angular/router';
import { Subscription } from 'rxjs';

@Component({
  selector: 'app-page3',
  templateUrl: './page3.component.html',
  styleUrls: ['./page3.component.css']
})
export class Page3Component {

  initialId: string|null;
  currentId: string|null;
  paramSubscription!: Subscription;

  constructor(private activatedRoute: ActivatedRoute, private router: Router) {
    this.initialId = activatedRoute.snapshot.paramMap.get('id');

    console.log('Page3.constructor, initialId', this.initialId);
    this.currentId = this.initialId;

    this.paramSubscription = activatedRoute.paramMap.subscribe(paramMap => {
      this.currentId = paramMap.get('id');
      console.log("param change, currentId", this.currentId);
    });
  }
}
```

# Navigate to Route & change param

```
import { Router } from '@angular/router';

constructor(private activatedRoute: ActivatedRoute, private router: Router) {
    ...
}

onChangeValue() {
    this.router.navigate(['/page3', this.value]);
}

onClickGotoPage2() {
    this.router.navigate(['/page2']);
}
```

# Storing data in @Service ... to keep value while changing page

\$ ng generate service

```
$ ng g s appStore
CREATE src/app/app-store.service.spec.ts (368 bytes)
CREATE src/app/app-store.service.ts (137 bytes)
```



The image shows a code editor window with the tab 'app-store.service.ts' selected. The code inside the file is:

```
1 import { Injectable } from '@angular/core';
2
3 @Injectable({
4   providedIn: 'root'
5 })
6 export class AppStoreService {
7
8   globalValue: string = 'global text';
9
10  constructor() { }
11}
```

# @Injectable Service in @Component constructor

```
service-page4.component.ts ×
1 import { Component } from '@angular/core';
2 import { AppStoreService } from '../../../../../app-store.service';
3
4 @Component({
5   selector: 'app-service-page4',
6   templateUrl: './service-page4.component.html',
7   styleUrls: ['./service-page4.component.css']
8 })
9 export class ServicePage4Component {
10
11   constructor(protected appStoreService: AppStoreService) {
12   }
13
14 }
```

# Outline

- templateHtml {{ expr }}
- \*ngFor
- \*ngIf
- Chrome Debugger
- @Component
- [] .. @Input field
- () .. @Output changed
- [()] .. Bidirectional Binding
- router



Next steps ?

Next steps...